Conformance Checking and Simulation-based Evolutionary Optimization for Deployment and Reconfiguration of Software in the Cloud

Sören Frey

Dissertation
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.)
der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel
eingereicht im Jahr 2013

Kiel Computer Science Series (KCSS) 2014/1 v1.0 dated 2014-01-22

ISSN 2193-6781 (print version)
ISSN 2194-6639 (electronic version)

Electronic version, updates, errata available via https://www.informatik.uni-kiel.de/kcss

Published by the Department of Computer Science, Kiel University

Software Engineering Group

Please cite as:

▻ Sören Frey. *Conformance Checking and Simulation-based Evolutionary Optimization for Deployment and Reconfiguration of Software in the Cloud.* Number 2014/1 in Kiel Computer Science Series. Department of Computer Science, 2014. Dissertation, Faculty of Engineering, Kiel University.

```
@book{Frey2014,
  author    = {S\"oren Frey},
  title     = {Conformance Checking and Simulation-based Evolutionary Optimization
               for Deployment and Reconfiguration of Software in the Cloud},
  publisher = {Department of Computer Science, CAU Kiel},
  year      = {2014},
  number    = {2014/1},
  isbn      = {9783732297344},
  series    = {Kiel Computer Science Series},
  note      = {Dissertation, Faculty of Engineering, Kiel University.}
}
```

© 2014 by Sören Frey

Herstellung und Verlag: BoD – Books on Demand, Norderstedt

About this Series

The Kiel Computer Science Series (KCSS) covers dissertations, habilitation theses, lecture notes, textbooks, surveys, collections, handbooks, etc. written at the Department of Computer Science at Kiel University. It was initiated in 2011 to support authors in the dissemination of their work in electronic and printed form, without restricting their rights to their work. The series provides a unified appearance and aims at high-quality typography. The KCSS is an open access series; all series titles are electronically available free of charge at the department's website. In addition, authors are encouraged to make printed copies available at a reasonable price, typically with a print-on-demand service.

Please visit `http://www.informatik.uni-kiel.de/kcss` for more information, for instructions how to publish in the KCSS, and for access to all existing publications.

1. Gutachter:	Prof. Dr. Wilhelm Hasselbring Christian-Albrechts-Universität Kiel
2. Gutachter:	Prof. Dr. Gerti Kappel Technische Universität Wien
3. Gutachter:	Prof. Dr. Schahram Dustdar Technische Universität Wien

Datum der mündlichen Prüfung: 13. Dezember 2013

Zusammenfassung

Der verstärkte Einsatz von Cloud-basierten Technologien bei der Neuentwicklung moderner Softwaresysteme demonstriert in jüngster Zeit das Potential von Cloud Computing zur Realisierung einer verbesserten Skalierbarkeit und Kosteneffizienz. Viele Software as a Service (SaaS) Anbieter sind bestrebt, dieses Potential auch für bestehende Anwendungen zu erschließen und erwägen eine Migration zu Infrastructure as a Service (IaaS) und Platform as a Service (PaaS)-basierten Cloud Umgebungen.

Bei einer solchen Migration existieren jedoch vielfältige Herausforderungen. (1) Die potenziellen Vorteile einer Cloud Umgebung, wie etwa eine dynamische Skalierung der Ressourcen oder eine häufig eingesetzte nutzungsbasierte Abrechnung, können ohne eingehende Migrationsplanung nicht optimal genutzt werden. Darüber hinaus weisen Cloud Umgebungen oftmals umfangreiche Restriktionen auf, z.B. bei direkten Dateisystemzugriffen oder dem Öffnen von bestimmten Netzwerk-Sockets durch ihre Gastanwendungen. Derartige Restriktionen bezeichnen wir als *Cloud Environment Constraints (CECs)*. Eine Anwendung verursacht etwa hinsichtlich der zuvor erwähnten CEC Typen sogenannte *CEC Violations*, falls sie auf das Dateisystem schreibt oder den betreffenden Netzwerk-Socket öffnet. (2) Generell werden CEC Violations bei einer Migration meistens nicht systematisch überprüft. (3) Des Weiteren existiert eine Vielzahl an unterschiedlichen *Cloud Deployment Optionen (CDOs)*, für einen Vergleich fehlt jedoch eine geeignete Unterstützung. Eine CDO legt z.B. fest, welche Cloud Umgebung, Cloud-basierten Ressourcen, Architektur und Laufzeitrekonfigurationsregeln verwendet werden sollen, um die Elastizität der Cloud Umgebung ausnutzen zu können. Die Performanzeigenschaften und Kosten von CDOs können hierbei um Größenordnungen variieren.

Zur Bewältigung der vorgenannten Herausforderungen schlägt diese Dissertation den Ansatz *CloudMIG* vor. Dieser unterstützt SaaS Anbieter bei der Migration von Unternehmensanwendungen zu IaaS und PaaS-basierten

Cloud Umgebungen. CloudMIG basiert auf Meta-Modellen der Architecture-Driven Modernization (ADM) Initiative der OMG und verwendet beispielsweise das Knowledge Discovery Meta-Model (KDM) der ADM zur Repräsentation von Code-Modellen einer Anwendung. Diese KDM Modelle werden mittels Reverse Engineering extrahiert. Cloud Umgebungen werden in wiederverwendbaren *Cloud Profilen* modelliert. Diese beschreiben z.B. die spezifischen Cloud-basierten Ressourcen, Preismodelle und CEC Definitionen. CloudMIG setzt folgende zwei Schwerpunkte.

Erstens umfasst es einen *automatischen Conformance Checking* Ansatz zur Erkennung von CEC Violations in extrahierten KDM Modellen hinsichtlich eines bestimmten Cloud Profils. Die Erkennung erfolgt mit Hilfe wiederverwendbarer *Constraint Validatoren*. Zusätzliche Constraint Validatoren können bei Bedarf in den Erkennungsprozess integriert werden.

Zweitens ermöglicht CloudMIG eine automatische *Erstellung und Optimierung von CDOs* mit Hilfe eines *simulationsbasierten genetischen Algorithmus* namens *CDOXplorer*. CDOXplorer verwendet unseren Simulator CDOSim als Fitnessfunktion. CDOSim simuliert CDOs und berechnet potenzielle Kosten, Antwortzeiten und die Anzahl an SLA Verletzungen. CDOXplorer liefert eine pareto-optimale Menge an CDOs, aus der ein SaaS Anbieter die jeweils am besten geeignete CDO auswählen kann.

CloudMIG vereinfacht die Erkennung von CEC Violations und die Erstellung geeigneter CDOs beträchtlich. SaaS Anbieter müssen keine zeitaufwendigen und teuren Code-Reviews mehr durchführen oder Fehlfunktionen riskieren, weil CEC Violations unerkannt bleiben. Anstatt CDOs manuell implementieren, bewerten und vergleichen zu müssen, ermöglicht CloudMIG eine automatische Generierung optimierter CDOs.

Umfangreiche Experimente zeigen die Eignung und Praktikabilität von CloudMIG und seiner zwei Kernbestandteile. Zum einen wird der Conformance Checking Ansatz mit drei Fallstudien evaluiert, die Laborexperimente und Experimente in einem industriellen Kontext umfassen. Die Evaluation zeigt die hohe Präzision des Erkennungsmechanismus. Zum anderen wird unser genetischer Algorithmus CDOXplorer mit drei etablierten multi-kriteriellen Such- und Optimierungsalgorithmen verglichen. Die

Evaluation zeigt, dass CDOXplorer Lösungen erstellen kann, die denen der anderen Algorithmen um bis zu 60% überlegen sind. Eine Proof of Concept Implementierung von CloudMIG existiert in Form der Anwendung *CloudMIG Xpress*. Diese ist als Open Source Software verfügbar.

Abstract

Newly created software systems that were built on a cloud computing basis from the ground up recently demonstrated the cloud's capabilities for enabling sound scalability and cost-effectiveness. Many Software as a Service (SaaS) providers want to leverage this potential for existing software systems as well and consider a migration to Infrastructure as a Service (IaaS) and Platform as a Service (PaaS)-based cloud environments.

However, the migration of existing software systems to a cloud computing basis often faces severe difficulties. (1) Migrations are often performed in an ad-hoc manner and the migrated systems therefore do often not leverage the cloud's capabilities, such as its dynamic resource scaling mechanisms and the frequently employed pay-per-use pricing model. Many cloud environments also impose restrictions to deployed applications, such as prohibiting directly writing to the filesystem or opening a specific network socket. We call those restrictions *cloud environment constraints (CECs)*. According to the two exemplary CEC types mentioned before, an application provokes corresponding *CEC violations* if it writes to the filesystem or opens the specific network socket, respectively. (2) CEC violations are most often not systematically evaluated before initiating a migration. (3) There also exist billions of different *cloud deployment options (CDOs)*, but appropriate support for comparing CDOs is missing. For example, a CDO determines which cloud environment, cloud resource types, deployment architecture, and runtime reconfiguration rules for exploiting the cloud's elasticity should be used. The performance and costs associated with diverse CDOs can differ in orders of magnitude.

To cope with these challenges, this thesis proposes the approach *CloudMIG* that supports SaaS providers to migrate existing enterprise software systems to IaaS and PaaS-based cloud environments. CloudMIG builds on metamodels from OMG's Architecture-Driven Modernization (ADM) initiative and employs, for example, reverse-engineered code models that correspond

to ADM's Knowledge Discovery Meta-Model (KDM). Each cloud environment candidate is modeled in a reusable *cloud profile* that includes the corresponding cloud resources, pricing model, and CEC definitions, for instance. CloudMIG focuses on two core components.

First, it includes an *automatic conformance checking* approach for detecting CEC violations in extracted KDM models regarding a specific cloud profile. It employs reusable *constraint validators* for detecting CEC violations concerning particular CEC types. Additional constraint validators can be plugged into the conformance checking process as needed.

Second, CloudMIG enables automatically *creating and optimizing CDOs* with a *simulation-based genetic algorithm* called *CDOXplorer*. CDOXplorer uses our simulation tool CDOSim as a fitness function. CDOSim simulates CDOs and computes potential costs, response times, and number of SLA violations. CDOXplorer delivers a pareto-optimal set of CDOs from which a SaaS provider can select the CDO that best satisfies its specific needs.

The approach CloudMIG substantially simplifies the detection of CEC violations and the creation of optimized CDOs. Regarding the former, it relieves SaaS providers from manually performing time-consuming and costly source code reviews or risking system malfunctions due to undetected CEC violations during operation. Regarding the latter, CloudMIG enables to automatically create optimized CDOs instead of having to actually implement, assess, and compare the CDOs manually.

Extensive experiments show the feasibility and practicality of CloudMIG and both of its core components. First, the conformance checking approach is evaluated with three different case studies covering lab experiments and experiments in an industrial context. The evaluation shows the high precision of CloudMIG's detection capabilities. Second, our genetic algorithm CDOXplorer is compared with three other state-of-the-art multi-objective search and optimization algorithms. The evaluation shows that CDOXplorer can produce solutions that surpass those of the other approaches by up to 60%. A proof of concept implementation of CloudMIG, that is called *CloudMIG Xpress*, is available as open source software.

Preface

by Prof. Dr. Wilhelm Hasselbring

Migrating existing on-premises software applications from your own data center towards the operation on off-premises software applications on cloud platforms constitutes a great challenge for many enterprises and organizations. Such migrations are often motivated by business decisions such as intended cost reduction, but also accompanied by various organizational and technical hurdles to be conquered.

Sören Frey presents his new, innovative CloudMIG approach to migrating existing software applications into the cloud. CloudMIG focuses on two main tasks when migrating existing software applications to some cloud platform: checking whether the existing software may be executed in a specific cloud environment and optimizing the deployment and automatic reconfiguration (adaptation) of software systems in the cloud. Besides designing CloudMIG, Sören implemented the tool CloudMIG Xpress to support the CloudMIG method that includes the CDOXplorer algorithm. CDOXplorer is a genetic algorithm that aims to automatically identify a near-optimal set of cloud deployment options by considering multiple objectives, i.e., costs, response time and service levels. The tool has been extensively evaluated in several experiments. The experimental evaluation is based on case studies that have been conducted for both the automatic conformance checking approach and the CDOXplorer algorithm.

The technical design and the implementation re-uses and integrates many software components and frameworks from various domains and sources. The re-use of such powerful components and frameworks relieves from building the respective functions, but imposes the challenge to check their fitness for purpose and to integrate diverse architectural styles into a coherent whole. The implementation of CloudMIG Xpress constitutes a remarkable engineering achievement. Besides the conceptual and the technical design, this engineering thesis provides an extensive experimental evaluation, including experiments with software from an industrial partner.

If you are interested in migrating software to the cloud, this is a recommended reading for you.

Wilhelm Hasselbring
Kiel, December 2013

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Problem Statement . . . 7
1.3 Scientific Contributions . . . 12
1.4 Preliminary Work . . . 18
1.5 Structure . . . 27

I Foundations **33**

2 Cloud Computing **35**
2.1 Overview . . . 37
2.1.1 Development . . . 37
2.1.2 Benefits . . . 38
2.1.3 Concerns . . . 40
2.2 Definition by NIST . . . 42
2.2.1 Essential Characteristics . . . 43
2.2.2 Service Models . . . 45
2.2.3 Deployment Models . . . 47
2.3 NIST Cloud Computing Reference Architecture . . . 48
2.3.1 Overview . . . 48
2.3.2 Actors in Cloud Computing . . . 50
2.4 Summary . . . 51

3 Software Modernization **53**
3.1 Reengineering . . . 55
3.1.1 Overview . . . 56
3.1.2 The Horseshoe Model . . . 58
3.1.3 Reverse Engineering . . . 59

3.2 Migration . . . 63
3.2.1 Overview . . . 63
3.2.2 Migration Approaches . . . 66
3.2.3 ISO/IEC 14764 Standard . . . 70
3.3 Architecture-Driven Modernization . . . 72
3.3.1 Overview . . . 73
3.3.2 Knowledge Discovery Meta-Model . . . 74
3.3.3 Structured Metrics Metamodel . . . 77
3.4 Summary . . . 79

4 Search-Based Software Engineering 81
4.1 Overview . . . 83
4.1.1 Classic Optimization Techniques . . . 84
4.1.2 Meta-heuristic Optimization Techniques . . . 86
4.1.3 Multi-Objective Optimization . . . 89
4.2 Genetic Algorithms . . . 93
4.3 Simulation-Based Optimization . . . 100
4.4 Summary . . . 101

II The CloudMIG Approach 105

5 Research Design and Methods 107
5.1 WP1: Problem Analysis . . . 109
5.2 WP2: CloudMIG Method Foundations . . . 112
5.3 WP3: Conformance Checking . . . 115
5.4 WP4: Deployment and Reconfiguration Optimization . . . 116
5.5 WP5: Tool Architecture . . . 118
5.6 WP6: Evaluation . . . 119
5.7 Summary . . . 121

6 The CloudMIG Method 123
6.1 Overview . . . 126
6.1.1 Addressed Challenges . . . 128
6.1.2 CloudMIG and the ISO/IEC 14764 Standard . . . 138
6.2 Cloud Migration Types . . . 141

6.3 Fundamental Approach . . . 145
6.3.1 Basic Design . . . 146
6.3.2 Activities . . . 151
6.3.3 Roles . . . 167
6.3.4 Cloud Environment Model . . . 170
6.4 Summary . . . 175

7 Conformance Checking **177**
7.1 Overview . . . 179
7.1.1 Basic Concepts . . . 182
7.1.2 Exemplary Constraints and Constraint Violations . . . 187
7.1.3 Detectability of Constraint Violations . . . 190
7.1.4 Addressed Constraints . . . 201
7.2 Constraint Modeling . . . 202
7.2.1 The Constraints Package of CEM . . . 203
7.2.2 Generic Constraint Modeling with SMM . . . 212
7.3 Automatic Constraint Validation . . . 215
7.3.1 Constraint Validation Process . . . 216
7.3.2 Constraint Validators . . . 221
7.4 Impact and Handling of Constraint Violations . . . 226
7.4.1 CSA Hierarchy . . . 226
7.4.2 Prioritization of Constraint Violations . . . 229
7.5 Summary . . . 231

8 Deployment and Reconfiguration Optimization **233**
8.1 Overview . . . 235
8.1.1 Basic Concepts . . . 238
8.1.2 Assumptions . . . 246
8.1.3 Involved CEM Packages . . . 248
8.2 The Simulator CDOSim . . . 251
8.2.1 Fundamental Concepts . . . 251
8.2.2 Accuracy of CDOSim . . . 254
8.3 CDO Model . . . 257
8.3.1 Scaling Types . . . 258
8.3.2 Phenotype Model . . . 261

8.3.3 Genotype Model . . . 265
8.3.4 Feasibility of CDOs . . . 266
8.4 Crossover and Mutation Operators . . . 269
8.4.1 Crossover Operator . . . 270
8.4.2 Mutation Operator . . . 271
8.5 Adaptivity and Hybridity Characteristics . . . 273
8.5.1 Adaptivity . . . 274
8.5.2 Hybridity . . . 276
8.6 Search Space Analysis . . . 277
8.6.1 Search Space Characteristics . . . 277
8.6.2 Example Scenario . . . 278
8.7 Summary . . . 279

9 Tool Architecture 281
9.1 Architectural Strategy . . . 283
9.1.1 Architectural Drivers . . . 283
9.1.2 Architectural Patterns and Styles . . . 289
9.2 Architecture Overview . . . 291
9.3 Summary . . . 297

III Evaluation 301

10 Tool Implementation 303
10.1 KDM Model Extraction . . . 305
10.2 MAMBA . . . 309
10.3 Constraint Validation . . . 311
10.4 CDOXplorer . . . 315
10.5 Summary . . . 318

11 Evaluation of the Conformance Checking Approach 321
11.1 Methodology . . . 323
11.1.1 GQM Method . . . 324
11.1.2 Binary Classification . . . 326
11.2 GQM Planning Phase . . . 330
11.2.1 Case Studies Overview . . . 331

11.2.2 Applications . . . 333
11.2.3 Cloud Environments . . . 335
11.3 GQM Definition Phase . . . 338
11.3.1 Case Study CC-1 . . . 339
11.3.2 Case Study CC-2 . . . 343
11.3.3 Case Study CC-3 . . . 346
11.4 GQM Data Collection Phase . . . 348
11.4.1 Case Study CC-1 . . . 348
11.4.2 Case Study CC-2 . . . 351
11.4.3 Case Study CC-3 . . . 352
11.5 GQM Interpretation Phase . . . 354
11.5.1 Case Study CC-1 . . . 354
11.5.2 Case Study CC-2 . . . 362
11.5.3 Case Study CC-3 . . . 368
11.6 Threats to Validity . . . 370
11.6.1 Case Study CC-1 . . . 370
11.6.2 Case Study CC-2 . . . 374
11.6.3 Case Study CC-3 . . . 375
11.7 Summary . . . 376

12 Evaluation of the Deployment and Reconfiguration Optimization Approach 379
12.1 Methodology . . . 381
12.1.1 Search and Optimization Techniques . . . 381
12.1.2 Performance Assessment of Multi-Objective Optimizers 385
12.2 GQM Planning Phase . . . 389
12.2.1 Overview . . . 389
12.2.2 Apache OFBiz . . . 391
12.2.3 Cloud Environments . . . 392
12.2.4 Search Space Analysis . . . 394
12.3 GQM Definition Phase . . . 394
12.3.1 Basic Definitions . . . 395
12.3.2 GQM Model . . . 396
12.4 GQM Data Collection Phase . . . 402
12.5 GQM Interpretation Phase . . . 404

12.6 Threats to Validity . . . 408
12.7 Summary . . . 409

13 Related Work **411**
13.1 Cloud Migration . . . 412
13.1.1 Cloud Migration Approaches . . . 413
13.1.2 Cloud Suitability Analysis . . . 430
13.1.3 Cloud Environment Modeling . . . 439
13.1.4 Cloud Application Modeling . . . 449
13.2 Conformance Checking . . . 462
13.2.1 Conformance Checking in the Context of Software Evolution . . . 465
13.2.2 Conformance Checking in the Context of Cloud Computing . . . 472
13.3 Deployment and Reconfiguration Optimization . . . 480
13.3.1 Deployment Optimization in Non-Cloud Scenarios . . . 482
13.3.2 Non-Evolutionary Cloud Deployment Optimization . . . 490
13.3.3 Evolutionary Cloud Deployment Optimization . . . 499
13.4 Summary . . . 502

IV Conclusions and Future Work **507**

14 Conclusions and Future Work **509**
14.1 Conclusions . . . 509
14.2 Future Work . . . 516

A The Packages of the Cloud Environment Model **521**

B Cloud Profile Excerpts **539**

Bibliography **549**

List of Figures

1.1 Example application that violates restrictions of a cloud environment . . . 5
1.2 CloudMIG Xpress synthetic workload profile creation 17

2.1 Cloud computing allows to dynamically provision resources 36
2.2 NIST cloud computing reference architecture overview . . . 49
2.3 Cloud computing role model . . . 51

3.1 Software modernization in the context of the Simple Staged Software Lifecyle Model . . . 54
3.2 Horseshoe model of reengineering . . . 59
3.3 Reverse engineering concepts . . . 61
3.4 Chicken little migration approach - gateway types . . . 68
3.5 Chicken little migration approach - architecture example . . 69
3.6 ISO/IEC 14764 maintenance process . . . 71
3.7 Items in a migration plan . . . 72
3.8 OMG ADM standards . . . 73
3.9 KDM overview . . . 75
3.10 KDM example from C# source code . . . 76
3.11 Core classes of the SMM meta-model . . . 78
3.12 SMM module count example . . . 79

4.1 Exemplary multi-objective search space . . . 82
4.2 Gradient ascent example . . . 85
4.3 Pareto-optimal front example . . . 92
4.4 Improvement of a pareto front over time . . . 93
4.5 Selection and recombination of parents per generation . . . 95
4.6 Non-domination rank example . . . 97
4.7 Crowding distance calculation . . . 98

4.8 NSGA-II procedure . . . 99
4.9 Simulation-based optimization . . . 101

5.1 Work packages overview . . . 108
5.2 Work packages contents . . . 110
5.3 Method engineering process . . . 113
5.4 Basic CDO optimization feedback loop . . . 118

6.1 CloudMIG overview . . . 124
6.2 Inter-arrival time function of an initial experiment . . . 131
6.3 Choice of VM instance types leading to SLA violations . . . 132
6.4 Response times and CPU utilization for each VM instance type used in initial experiment . . . 133
6.5 Characteristics of system configurations that utilize the minimum nr. of VM instances necessary to satisfy the SLA . . . 134
6.6 Addressed items in a migration plan . . . 138
6.7 Cloud migration types . . . 144
6.8 Exemplary cloud migration type SN2I . . . 145
6.9 The CloudMIG method . . . 147
6.10 CloudMIG and the horseshoe model of reengineering . . . 152
6.11 Benefit of using KDM models for conformance checking . . . 154
6.12 Target architecture generation process G2 . . . 159
6.13 Example scenario for phase P3 of CloudMIG's activity A3 (approach G2) . . . 162
6.14 Potential target architecture of the example scenario for phase P3 . . . 165
6.15 Different roles in CloudMIG . . . 168
6.16 Packages of CEM . . . 171
6.17 Cloud Profile package of CEM . . . 173

7.1 Conformance checking overview . . . 178
7.2 Google plugin for Eclipse detects an unsupported type . . . 190
7.3 Google plugin for Eclipse erroneously accepting a statement 191
7.4 Google App Engine error message . . . 192
7.5 StorageUnit KDM instance example . . . 198

7.6 KDM instance excerpt . . . 200
7.7 Constraints package of CEM . . . 205
7.8 Meta-model elements of KDM for modeling a class instantiation . . . 206
7.9 SMM file count measure . . . 214
7.10 Constraint validation process . . . 217
7.11 Conformance checking states . . . 220
7.12 Important classes for checking the conformance of CECs . . . 222
7.13 Validation of a SocketOpeningConstraint . . . 224
7.14 The CSA hierarchy . . . 227

8.1 Under- and over-provisioning example . . . 234
8.2 Input and output of CDOXplorer . . . 239
8.3 Iterative simulation-based CDO optimization . . . 241
8.4 Mapping on-premise servers and deployed components to atomic services in a basic CDO example . . . 242
8.5 IBM's MAPE-K reference model for autonomic control loops 245
8.6 Logical structure for dynamic resource scaling with reconfiguration rules . . . 248
8.7 CEM packages relevant for CDO optimization . . . 249
8.8 Integration of CDOSim with CloudMIG Xpress . . . 254
8.9 Day-night-cycle workload intensity used for evaluating CDOSim's accuracy . . . 256
8.10 Average CPU utilization of allocated nodes in SingleCore.1 . . . 257
8.11 Median of response times in SingleCore.1 . . . 258
8.12 Examples of the different scaling types . . . 259
8.13 Emulated scaling up . . . 260
8.14 Basic structure of CDOs . . . 262
8.15 Effect of different MIPIPS multiples . . . 263
8.16 Shift of RPVMI's VM instance type . . . 264
8.17 Extended CDO example . . . 265
8.18 Compound chromosome overview . . . 267
8.19 CDO examples encoded as genotypes . . . 268
8.20 CDO constraint example . . . 269
8.21 Crossover operator examples . . . 271

8.22 Mutation operator examples 273
8.23 Progress of pareto optimum quality improvement 275

9.1 The ADD-based architecture design process 282
9.2 CloudMIG Xpress conceptual architecture overview 292

10.1 CloudMIG Xpress logo 304
10.2 CloudMIG Xpress start page 305
10.3 KDM model extraction basic design 307
10.4 Three-phase transformation of C# to KDM 308
10.5 Design of the Mamba Execution Engine 309
10.6 CEC violation visualization in CloudMIG Xpress 312
10.7 CloudMIG Xpress constraint validation design 313
10.8 Basic design of CDOXplorer using the Opt4J framework . . 316

11.1 Important components of the conformance checking evaluation 322
11.2 The four phases of the GQM method 324
11.3 Binary contingency table 328
11.4 Receiver Operating Characteristic (ROC) example 330
11.5 Overview on case study CC-1's GQM model 340
11.6 Overview on case study CC-2's GQM model 344
11.7 Overview on case study CC-3's GQM model 346
11.8 SMM import coupling measure 351
11.9 Percentage of classes that cause constraint violations 355
11.10 Detected constraint violation densities per application 356
11.11 Detected constraint violation densities per class 357
11.12 Distribution of detected violation types 358
11.13 Partitioning of classes that raise CEC violations through applying the function prio 360
11.14 Classes being responsible for an amount of constraint violations and the relation to their size 362
11.15 Distribution of the constraint violations' origin 363

12.1 Important components of the CDO optimization evaluation 380
12.2 Pareto fronts PF_{Known}/PF_{True} example 387

12.3 Hypervolume indicator example . . . 389
12.4 Measures IGD and HV . . . 396
12.5 GQM model for evaluating the deployment and reconfiguration approach . . . 397
12.6 Simulated annealing reuses mutation sub operators . . . 402
12.7 Best known pareto front for SC_S: $OPF_{Best}(SC_S)$ . . . 403
12.8 CDOXplorer advantage relative to other approaches . . . 407

13.1 Related work overview . . . 412
13.2 Cloudstep process . . . 417
13.3 CMotion process . . . 418
13.4 MOCCA method . . . 421
13.5 REMICS approach . . . 423
13.6 Seven-step model of migration into the cloud . . . 424
13.7 Cloud migration approach that uses extensions of the Darwin framework . . . 426
13.8 Cloud migration approach based on OMG's ADM initiative 428
13.9 Cloud adoption toolkit . . . 433
13.10 Magic Matrices Method . . . 435
13.11 Cloud Computing Reference Architecture overview . . . 444
13.12 Cloud OS overview . . . 445
13.13 MODAClouds overview . . . 452
13.14 Reservoir's elasticity rules syntax . . . 456
13.15 SPOSAD architectural style . . . 459
13.16 Deployment optimization using UML models and MILP . . 486
13.17 CloudGuide overview . . . 493

14.1 Core aspects of the CloudMIG approach investigated in this thesis . . . 511

A.1 The *Core* Package of CEM . . . 523
A.2 The *Mapping* Package of CEM . . . 526
A.3 The *Constraints* Package of CEM . . . 527
A.4 The *Usage* Package of CEM . . . 531
A.5 The *IaaS* Package of CEM . . . 532

A.6 The *PaaS* Package of CEM 534
A.7 The *Pricing* Package of CEM 535
A.8 The *Cloud Profile* Package of CEM 537

List of Tables

6.1 Eucalyptus hardware configuration . . . 129
6.2 VM instance types . . . 130
6.3 VM instance type price model . . . 130
6.4 Naming scheme of cloud migration types . . . 143
6.5 Activities to challenges mapping . . . 149

7.1 The CECs of CEM's Constraints package . . . 204

8.1 Design of the used genes . . . 266

11.1 Overview of the conformance checking evaluation case studies 332
11.2 Applications analyzed in the case studies - basic characteristics I . . . 334
11.3 Applications analyzed in the case studies - basic characteristics II . . . 334
11.4 CECs modeled in the GAE for Java cloud profile for CC-1 . . . 350
11.5 CEC violation inspection template . . . 352
11.6 Inspection template of CEC violation V-102 . . . 364
11.7 Inspection template of CEC violation V-117 . . . 365
11.8 Inspection template of CEC violation V-2081 . . . 367

12.1 VM instance types of Eucalyptus for evaluating CDOXplorer 393
12.2 Configuration of CDOXplorer in the context of the multi cloud scenario . . . 395
12.3 Coefficient of variation for analyzed scenarios regarding CDOXplorer . . . 405
12.4 Performance of the search methods regarding the single cloud scenario (SC_S) . . . 406

12.5 Performance of the search methods regarding the multi cloud scenario (SC_M) 406

13.1 Related work for cloud migration approaches 414
13.2 Related work for cloud suitability analysis 431
13.3 Related work for cloud environment modeling 440
13.4 Related work for cloud application modeling 450
13.5 Related work for conformance checking in the context of software evolution 466
13.6 Related work for conformance checking in the context of cloud computing 473
13.7 Related work from approaches for deployment optimization in non-cloud scenarios 483
13.8 Related work from approaches for non-evolutionary deployment optimization in cloud scenarios 491
13.9 Related work from approaches for evolutionary deployment optimization in cloud scenarios 500

List of Acronyms

ADD Attribute-Driven Design

ADM Architecture-Driven Modernization

API Application Programming Interface

ARIMA Autoregressive Integrated Moving Average

ASP Application Service Provider

AST Abstract Syntax Tree

ASTM Abstract Syntax Tree Metamodel

ATL Atlas Transformation Language

BLOB Binary Large Object

BPMN Business Process Model and Notation

CCEL C++ Constraint Expression Language

CAA Cloud Application Architecture

CAAM Cloud Computing Adoption Assessment Model

CAMP Cloud Application Management for Platforms

CCOA Cloud Computing Open Architecture

CCRA Cloud Computing Reference Architecture

CCS Conformance Checking State

CDO Cloud Deployment Option

CDOXplorer Cloud Deployment Option Explorer

List of Acronyms

CEC Cloud Environment Constraint

CEM Cloud Environment Model

CIMI Cloud Infrastructure Management Interface

CMMI Capability Maturity Model Integration

CMP Cloud Migration Point

COTS Commercial Off-The-Shelf

CPA Cloud Platform Architecture

CPC Cloud Profile Contributor

CQL Code Query Language

CRM Customer Relationship Management

CSA Cloud Suitability and Alignment

CVC CEC Validation Contributor

DC Detectability Category

DMTF Distributed Management Task Force

DNS Domain Name System

DSL Domain-specific Language

EMF Eclipse Modeling Framework

ERP Enterprise Resource Planning

FCL Framework Class Library

FGD Final Generational Distance

FLOPS Floating Point Operations Per Second

FPR False Positive Rate

GAE Google App Engine

GPGPU General-Purpose Graphics Processing Unit

GQM Goal Question Metric

HPC High Performance Computing

HV Hypervolume Indicator

IaaS Infrastructure as a Service

IDE Integrated Development Environment

IGD Inverted Generational Distance

IIS Internet Information Services

JRE Java Runtime Environment

JSP Java Server Pages

JVM Java Virtual Machine

KDC KDM Discoverer Contributor

KDM Knowledge Discovery Meta-Model

LCOM Lack of Cohesion of Methods

LOC Lines of Code

MAMBA Measurement Architecture for Model-Based Analysis

MAPE-K Monitor, Analyze, Plan, Execute, Knowledge

MDE Model-Driven Engineering

MDL Metrics Definition Language

MDMIPS Mega Double Multiply Instructions per Second

MEE MAMBA Execution Engine

List of Acronyms

MFRC Monitoring Format Reader Contributor

MILP Mixed Integer Linear Programming

MIPIPS Mega Integer Plus Instructions per Second

MOF Meta Object Facility

MVC Model View Controller

NIC Network Interface Card

NIST National Institute of Standards and Technology

OCL Object Constraint Language

OMG Object Management Group

OSS Open Source Software

OWL Web Ontology Language

PaaS Platform as a Service

PIM Platform-Independent Model

PSM Platform-Specific Model

QoS Quality of Service

RCP Rich Client Platform

RDF Resource Description Framework

ROC Receiver Operating Characteristic

RPVMI Reference Point VM Instance

SaaS Software as a Service

SBSE Search-Based Software Engineering

SLA Service Level Agreement

SLO Service Level Objective

SMM Structured Metrics Metamodel

SOA Service-Oriented Architecture

SOCCA Service-Oriented Cloud Computing Architecture

SQL Structured Query Language

SUA System Under Analysis

SYBL Simple-Yet-Beautiful Language

TNR True Negative Rate

TPR True Positive Rate

UCI Unified Cloud Interface

UDP User Datagram Protocol

UML Unified Modeling Language

VM Virtual Machine

VPN Virtual Private Network

WMC Weighted Methods per Class

XMI XML Metadata Interchange

XML Extensible Markup Language

Chapter 1

Introduction

Migrating software systems to infrastructure and platform cloud services promises to alleviate resource over-provisioning and to lower corresponding expenses as cloud resources can be flexibly allocated and deallocated due to varying user demand. Many enterprises, academic institutions, and public authorities consider the migration of existing software systems to the cloud to benefit from dynamic resource scaling capabilities and the frequently employed pay-per-use model. The introduction described in this chapter starts with a motivation in Section 1.1 regarding the incentives for migrating software systems to the cloud and also for developing the CloudMIG approach in this thesis that supports corresponding migration projects. Major challenges that accompany the migration of software systems to the cloud are described in the problem statement in Section 1.2. Section 1.3 summarizes the scientific contributions of this thesis, which build on several publications. These publications are summarized in Section 1.4. Finally, Section 1.5 describes the structure of this thesis.

1.1 Motivation

In recent years, cloud computing has gained considerable attention in the industry, academia, and the public sector. It can be seen as a realization of the utility computing paradigm [Buyya et al. 2008] that establishes the notion of computing services that are delivered on demand like utilities, such as water, gas, and electricity. Cloud computing provides infrastructure, platform, and software services over a network connection. For example, it

allows to rapidly develop and provision applications on remote servers due to pre-built software platforms that are offered, operated, and maintained by cloud providers. Cloud computing combines several existing technologies to form a new computing paradigm, for example, it relies on a broad network access and most often employs virtualization technology. It is the combination of those technologies that facilitates the emergence of a central new characteristic of cloud computing as it allows to dynamically and rapidly scale resources based on the actual user demand.

This so-called *elasticity* allows to cope with the problem of over-provisioning server hardware that is frequently found in common data centers. For example, Armbrust et al. [2009] report on real world estimates [Rangan 2008; Siegele 2008], which state that the server utilization in data centers ranges from 5% to 20%. As cloud resources can be released when they are no longer required, there is no need to provision hardware resources for peak load and to make huge up front capital investments. Moreover, as cloud resource usage is often charged on a pay-per-use basis, releasing resources allows to only pay for actually used infrastructure, platform, or software services.

A plethora of success stories regarding the adoption of cloud computing technologies were recently reported by enterprises, IT departments, and cloud providers[1] and numerous case studies from academia report on migration experiences [Palankar et al. 2008; Khajeh-Hosseini et al. 2010; King and Ganti 2010; Thakar and Szalay 2010; Babar and Chauhan 2011; Chauhan and Babar 2011; Rajan et al. 2011; Tran et al. 2011a]. The anticipated benefits, like the mentioned resource efficiency and cost-effectiveness, were materialized in real world scenarios where infrastructures could be consumed as services for dynamically scaling out and in as a result of varying workloads, for instance. Especially the operation of software systems whose usage patterns exhibit these inconstancies can profit by running them on a cloud computing basis.

[1] For example: `https://cloud.google.com/customers/`
`https://aws.amazon.com/solutions/case-studies/`
`http://www.windowsazure.com/en-us/home/case-studies/`
`http://www.rackspace.com/blog/category/customer-stories/`

Those types of software systems are often provided as services themselves and the volatile load factor results from changing user behaviors. The service model of providing software in the form of a service is called Software as a Service (SaaS) in the context of cloud computing, whereas it is not relevant whether the services are offered in-house or to external customers. An example for a successful application of cloud technologies is the online service Animoto for creating and sharing videos. Their online service faced a tremendous increase in user demand in 2008 when its service was integrated in the popular social network Facebook and could handle additional 750,000 users in three days by using VM instances of the Amazon EC2 cloud environment.[2]

The authors in Armbrust et al. [2009] provide a role model that distinguishes those SaaS providers that supply the services, such as the online video service provided by Animoto, from the SaaS users who consume the services. On the other hand, SaaS providers consume infrastructures or whole platforms as services from cloud providers and therefore act as *cloud users*. A corresponding service model that describes the provisioning of infrastructure resources—such as computing resources, virtual machines (VMs), storage, and network connections—is called Infrastructure as a Service (IaaS). If complete platforms are provisioned to cloud users—such as a software stack running on a computing node—the corresponding service model is called Platform as a Service (PaaS, cf. Section 2.2.2).

Not least because of the aforementioned published success stories, many traditional SaaS providers are encouraged and begin to search for ways to employ an IaaS or PaaS cloud computing foundation for their own offered services. Additional business drivers may also be the goals to put a stronger focus on their core business of developing and delivering services instead of operating and administering the underlying hardware infrastructures, or to fulfill the often-cited CapEx to OpEx transition [Leymann 2009] and therefore achieve an enhanced flexibility, for instance.

From a SaaS provider perspective, cloud computing technologies provide means to reduce over- and also under-provisioning for supplying their ser-

[2] http://aws.amazon.com/solutions/case-studies/animoto/

vices through enabling a highly flexible resource allocation. SaaS providers can smoothly improve resource efficiency and scalability in a greenfield project through aligning their emerging application with a specific cloud provider's environment. However, there often exist many software assets which cannot easily be rebuilt from scratch to benefit from cloud computing's capabilities. Migrating those systems to a solid cloud computing foundation and therefore preserving a company's investments constitutes a worthwhile approach. Nevertheless, running an existing software system on a cloud computing basis may involve considerable reengineering activities during the migration. The magnitude of adaptation being necessary to migrate a software system depends on various influencing factors.

For example, SaaS providers may emulate the previous on-premise deployment structure in the cloud to attain rapid results and to minimize reengineering needs. This approach presumes a general cloud compatibility of the existing software system, i.e., conformance with a specific cloud environment's imposed constraints, through, e.g., not using unsupported network protocols or operating systems. For example, the cloud environment Google App Engine for Java defines several restrictions applications have to obey. Access to types of the Java Runtime Environment (JRE) is restricted to a limited subset of those types and the applications are not allowed to write to the filesystem and to spawn standard Java threads, for instance. Hence, corresponding source code statements of existing systems that violate those restrictions have to be found and modified prior to an actual migration.

Those incompatible statements are not uncommon in enterprise software systems. For example, experiments in the context of this thesis with five well-known web-based Java applications and Google App Engine for Java show that up to 33% of the applications' classes contain those incompatible statements (cf. Section 11.5.1). However, those statements can be hard to detect due to a system's complexity and size. Furthermore, it is often not clear which elements of a software system actually violate given restrictions, as those restrictions are frequently stated by cloud providers in a rather informal way.

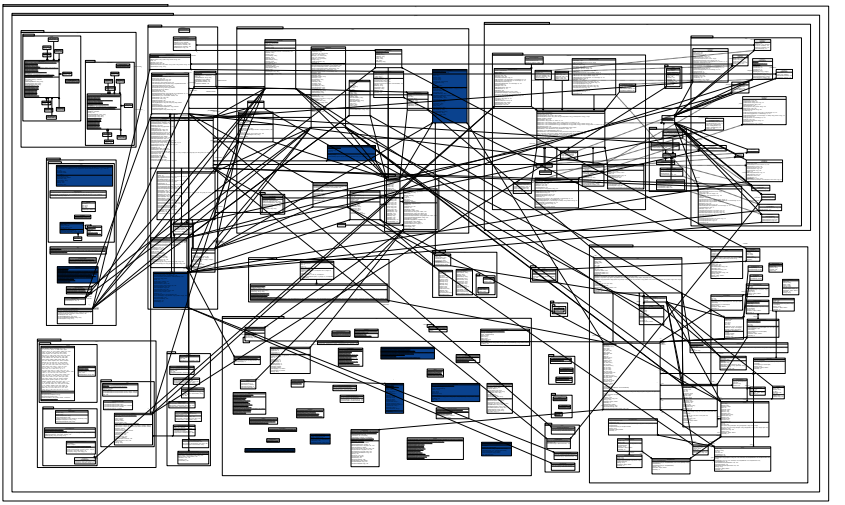


Figure 1.1. Reverse-engineered UML class diagram of the web-based forum software JavaBB V.0.99. The blue-colored classes violate restrictions of the cloud environment Google App Engine for Java.

Figure 1.1 shows an example regarding the Java-based forum software JavaBB V.0.99 that is also analyzed in the context of the experiments mentioned above. The figure highlights those classes of JavaBB that actually violate given restrictions of the Google App Engine for Java cloud environment. Though, even after detecting and correcting all of those kinds of statements, a cloud compatible system is not yet improved for running in the cloud and therefore often does not leverage the cloud's elasticity or is not aligned to a specific cloud provider's resource pricing model. Moreover, taking the line of the least resistance and merely virtualizing applications to hand them over to VM instances of an IaaS provider to lower migration efforts may be counterproductive in several cases and yield unsolicited results.

For example, running VM instances over longer time periods when they are not needed might foil an expected cost reduction. The same negative effect can be provoked when choosing inappropriate hardware profiles as

a basis for running VM instances. Over- and under-provisioning can be easily migrated to the cloud in conjunction with the legacy system. For example, initial experiments in the context of this thesis that pursue such a simplistic migration approach result in a CPU over-provisioning ranging from 41%-84%, implicating up to doubled operational costs compared to the possible minimum (cf. Section 6.1.1). However, when properly exploiting the dynamic resource scaling capabilities of a target cloud environment, the possible savings are even bigger. A further case study in the context of this thesis shows that achievable response times, costs, and number of violations regarding Service Level Agreements (SLA) vary in orders of magnitude due to the employed dynamic resource scaling rules and chosen deployment aspects, such as mapping of components to cloud resources (cf. Chapter 12). But finding well-suited cloud deployment strategies is a challenging, nontrivial task, because of the huge number of potential cloud environments, heterogeneity of cloud resources, and cloud system architectures [Grundy et al. 2012], for instance. Unfortunately, it is all too easy to choose solutions that are far from optimal.

Avoiding pitfalls might require to increase the degree of parallelization and the introduction of a management layer to handle a cloud environment's potential elasticity, for instance. Furthermore, worst case scenarios might not be limited by inadvertent inefficiencies but even entail disasters like the loss of data, if the mentioned restrictions of a cloud environment are ignored. To get around these menaces and to leverage the cloud-specific aspects like enabling elasticity or increasing resource efficiency, we argue that it is most often inevitable to conduct further methodical adaptations.

For adapting and leveraging existing software systems to provide a credible basis for using cloud computing technologies, various methods and techniques originating from the software modernization and reengineering domain can be utilized. For example, a system model up to the architectural level should ideally be present serving as a foundation for the further restructuring, optimization, and evaluation activities. However, a description of the actual architecture is often missing or is only available in an outdated version. Therefore, the relevant models have to be recovered in a reverse engineering step. Other relevant criteria to take into account

while planning a migration can, for example, be seen in the targeted cloud computing service model, namely PaaS or IaaS, and the conformance with a cloud environment's imposed constraints. The optimization of several further aspects should also be considered, such as the mapping of system artifacts to cloud resources, the types of those resources, and the rules for dynamic resource-scaling with regard to the present usage patterns.

1.2 Problem Statement

Migrating existing enterprise applications to infrastructure and platform cloud services has become a worthwhile option for many software service providers. Nevertheless, besides potential issues regarding security, legislation, and governance, there also exist several hurdles that impede *cloud migration* projects from a solely technical point of view. Those difficulties have already been mentioned in the context of the previous Section 1.1. They are summarized in the three problems P1-P3 that are stated below:

P1: Limited Support for Planning the Migration of Enterprise Software Systems to the Cloud

P2: Detecting Constraint Violations is Complex and Cumbersome

P3: Vast Amount of Cloud Deployment Options of Unknown Quality

These problems P1-P3 are detailed in the following.

P1: Limited Support for Planning the Migration of Enterprise Software Systems to the Cloud

Each project that considers the migration of existing software systems to a new environment typically starts with a planning phase (see Section 3.2.3). This is also true for cloud migration projects that cover the migration of

software systems to the cloud. However, there is a lack of approaches that provide support for SaaS providers to (1) systematically prepare the migration of enterprise software systems to a cloud computing basis, (2) evaluate the conformance of an existing system regarding cloud environment candidates and imposed technical constraints, and (3) thoroughly reason about potential migration options based on project-specific characteristics. Those characteristics include the actual usage patterns of existing systems, for instance. The cloud's dynamic resource scaling capabilities can be leveraged best if a system architecture gets aligned with a specific cloud environment.

Potential target architectures should be assessed under consideration of realistic workload patterns prior to an actual migration. Though, performing comprehensive comparisons among a wide spectrum of potential cloud solutions is often not possible. On the one hand, there exists no prevalent cloud computing standard at the moment [Jr. 2011]. On the other hand, most existing approaches concentrate on particular activities for migrating applications to the cloud or focus on specific cloud environments (cf. Section 13.1). Furthermore, analyzing existing systems—for example, regarding potential constraint violations and cloud deployment options—should be possible regardless of used technologies, such as specific programming languages. Hence, a cloud migration approach should enable to extract corresponding models that allow to perform those analyses on a language-agnostic and model-based foundation. For performing those analyses, an appropriate meta-model is also required that allows to describe the specifics of arbitrary cloud environments.

P2: Detecting Constraint Violations is Complex and Cumbersome

The restrictions imposed by most cloud environments (cf. Section 1.1) form a substantial challenge to SaaS providers that consider the migration of existing applications to the cloud. Especially cloud environments that provide complete software platforms to their cloud users enforce considerable constraints regarding the deployed applications, for example, to enable transparent resource scaling. Proactively detecting incompatible program statements currently either requires (1) to perform a manual collection of

a cloud environment's constraint descriptions followed by a review of an application's source code, or (2) to deploy and execute the system while trying to uncover potential violations during operation.

Both methods have considerable drawbacks. The constraints are most often described by cloud providers in a rather informal way, for example, in textual documentations and web logs. Hence, they have to be manually distilled from such sources. Uncovering incompatible source code elements in the course of a code review can be a time-consuming and costly endeavor. Moreover, the internal structures of software systems tend to erode over time [Lehman 1980]. Thus, the resulting complexity and also the sheer system sizes frequently found can often render a manual detection approach unreasonable. Due to the prevalent informality of constraint descriptions, it is often even challenging to map the descriptions to specific program statements. Furthermore, each cloud environment exhibits a different set of constraints and a specific code review procedure would therefore have to be performed for each cloud environment candidate.

There currently exists no approach that aims to systematically detect program statements of existing systems that violate constraints of arbitrary cloud environments (cf. Sections 13.1 and 13.2). Hence, those program statements are often detected following a trial-and-error method that deploys, executes, and observes the system. That means, they are uncovered during runtime when they are executed the first time and the system exhibits a malfunction. Existing test cases can be employed to support this procedure, however, requiring a high level of test coverage. This approach is therefore also rather inappropriate. For each cloud environment that should be considered as a potential cloud candidate, the application has to be packaged, deployed, and tested. Often, considerable configuration or even modification effort is required to initially deploy and run an application in a cloud computing environment. Hence, comparing the suitability of different cloud environment candidates is also rather costly and tedious following this approach.

P3: Vast Amount of Cloud Deployment Options of Unknown Quality

A central challenge in migrating software systems to the cloud is the comparison of different cloud deployment options and the selection of the best suited candidate [Grundy et al. 2012]. For example, SaaS providers initially have to choose from hundreds of available cloud environments. Afterwards, adequate cloud resources have to be selected, for example, specific VM instance types that should be used for launching virtual machines. Then, a deployment architecture has to be defined that maps existing components to those resources. Moreover, to exploit the cloud's dynamic resource scaling capabilities, a dynamic resource scaling strategy has to be defined that often involves the configuration of several rules for controlling the allocation and deallocation of cloud resources. Those rules are usually composed of various parameters that have to be configured carefully. For example, an included condition could specify a CPU utilization threshold (e.g., 70%) that is used to trigger a cloud resource reconfiguration action at runtime, for instance, to start or stop VM instances of certain VM instance types.

Finding well-suited configurations of those reconfiguration rules can have a huge impact on performance and costs. For example, Marshall et al. [2012] report on a queued job time performance characteristic that could be improved by up to 58% and costs that could be reduced by up to 38% by provisioning cloud resources on a flexible basis incorporating dynamic provisioning policies. Adapting the composition of service-based applications at runtime was also shown to be a worthwhile means for reducing the number of occurring SLA violations [Leitner et al. 2011].

However, comparing different cloud deployment options and finding a well-suited candidate is non-trivial, costly, and time-consuming, due to the following reasons.

1. *Size of the search space:* There exists a vast amount of potential cloud deployment options. Even when considering a single cloud environment, there usually exist hundreds of millions potential deployment options (cf. Section 8.6). On the one hand, this is due to the number of combinations resulting from mapping existing software components to different cloud

resource types and also from considering different numbers of instances that can be initially started from those mappings.

On the other hand, the high number of possibilities for configuring the reconfiguration rules contributes to a considerable extent to the search space explosion. For example, it has to be decided when to scale, how to scale, and which scaling strategy should be used [Suleiman et al. 2012]. In summary, evaluating all of the potential cloud deployment options is, even for a single cloud environment candidate, most often not a viable option.

2. *Complex and costly assessment:* For assessing the quality of potential cloud deployment options, the most reliable and accurate way is to actually implement, observe, and evaluate them. However, due to the vast number of potential cloud deployment options, this is most often only a feasible option for a small amount of them. Though, it is hard to estimate the resulting service response times and costs that incur due to the frequently employed pay-per-use pricing model without actually implementing a specific option. For being able to compare and evaluate a greater amount of cloud deployment options, an alternative approach is needed. For example, there exist several approaches in the related area of software architecture optimization [Aleti et al. 2013] that allow to iteratively evaluate and improve software architectures using model-based techniques.

   However, none of those provides comprehensive support for assessing and optimizing different runtime reconfiguation rules to exploit the cloud's dynamic resource scaling capabilities (cf. Section 13.3), for instance. Furthermore, to provide realistic estimations of aspects such as the future costs and performance characteristics, it has to be possible to take actual workload patterns into account.

   Though, it is still hard to infer those future characteristics on the basis of the monitored behavior of an existing software system, as the future characteristics are influenced by a completely different environment, deployment architecture, resource set, and scaling opportunities.

1.3 Scientific Contributions

This thesis makes the following five scientific contributions SC1-SC5 to the research area of cloud computing:

SC1: An approach called CloudMIG that supports software service providers to plan the migration of existing enterprise software systems to a cloud environment.

SC2: An automatic conformance checking approach for detecting a software system's violations of constraints that are imposed by cloud environments.

SC3: An approach for optimizing the deployment and reconfiguration of software systems in the cloud that uses techniques from the field of search-based software engineering. Specifically, it includes an adaptive, hybrid, and simulation-based genetic algorithm termed CDOXplorer for optimizing the deployment and reconfiguration.

SC4: A proof of concept regarding SC1, SC2, and SC3 that includes (1) the construction of a software architecture following a structured architecture design process and (2) the implementation of the tool *CloudMIG Xpress* that constitutes a realization of that architecture.

SC5: An experimental evaluation of SC1, SC2, and SC3.

These contributions SC1-SC5 are detailed in the following.

SC1: The Approach CloudMIG for Supporting the Migration of Enterprise Software Systems to the Cloud

Our approach CloudMIG supports software service providers to plan the migration of existing enterprise software systems to a cloud computing basis (cf. problem P1 in the previous Section 1.2). Cloud environments that offer infrastructure and platform services can be considered as potential cloud candidates. Software systems have to be reengineered to fully ex-

ploit the cloud's merits, such as its dynamic resource scaling capabilities and the frequently offered pay-per-use pricing model. CloudMIG follows a model-based approach to enable reasoning about possible migration options. Hence, a model describing the software system that should be migrated to the cloud has to be available. However, those models most often first have to be recovered and, therefore, CloudMIG comprises a reverse engineering step that extracts an architectural model from an existing software system. It builds on corresponding meta-models from the Architecture-Driven Modernization (ADM) initiative of the Object Management Group (OMG). In this context, the Knowledge Discovery Meta-Model (KDM) [Pérez-Castillo et al. 2011] is used to represent existing software systems on different levels of abstraction and the Structured Metrics Metamodel (SMM) [Object Management Group 2012] is utilized to model and apply software metrics.

CloudMIG also considers the present usage of software systems that should be migrated to the cloud. This constitutes an important precondition for enabling realistic projections regarding the future behavior of a migrated system. The actual system usage is described in so-called *workload profiles* that can be extracted from historic monitoring log data, for instance. Workload profiles can also be used as drivers for simulating specific mappings of the extracted KDM elements to a set of cloud resources. This enables to evaluate and compare so-called Cloud Deployment Options (CDOs) that describe, among others, particular deployment architectures using certain cloud environments and runtime reconfiguration rules for exploiting the cloud's elasticity.

To describe specifics of arbitrary cloud environments, CloudMIG contributes a meta-model called Cloud Environment Model (CEM). The CEM allows to describe aspects such as a cloud environment's offered resource types, availability in certain geographical regions, or employed pricing model. An instance of CEM that describes a specific cloud environment is called a *cloud profile*. Cloud profiles also include the constraints that are imposed by the cloud environment, such as a restriction regarding the use of specific types of a programming language. In combination with the mentioned workload profiles, KDM, and SMM models, the cloud profiles form the basis for the application of the automatic conformance checking process and the

optimization of the deployment and reconfiguration rules as described in the contributions SC2 and SC3, respectively.

SC2: Automatic Conformance Checking for Detecting Constraint Violations

Detecting constraint violations during the migration of enterprise software systems to the cloud is complex and cumbersome (cf. problem P2 in the previous Section 1.2). We contribute an automatic conformance checking approach that allows to detect a software system's violations of constraints that are imposed by cloud environments. As a basis for this conformance checking approach, the Cloud Environment Model (CEM) provides elements for modeling the constraints of a specific cloud environment in a corresponding cloud profile. These constraints are called Cloud Environment Constraints (CECs). Consequently, if those CECs are violated—for example, due to a program statement in the system that should be migrated to the cloud that does not conform with a specific CEC—a corresponding violation is called a *CEC violation*.

So-called *constraint validators* are used to automatically check the conformance of a software system regarding one or more CECs. Additional constraint validators can be plugged into the conformance checking process as needed. The constraint validators and the cloud profiles along with the included CEC models can be reused among software service providers. Hence, compared to the prevalent manual detection approach, our automatic conformance checking process considerably accelerates and simplifies the detection of CEC violations.

SC3: An Approach for Optimizing the Deployment and Reconfiguration of Software in the Cloud

When considering a migration of an existing software system to the cloud, there usually exists a vast amount of cloud deployment options that have to be compared for being able to select a well-suited solution (cf. problem P3 in the previous Section 1.2). We contribute an approach for automatically

optimizing the deployment and reconfiguration of software in the cloud that (1) copes with the involved search space explosion and that also (2) enables to assess specific solution candidates. For the latter aspect, our simulator CDOSim enables to simulate so-called Cloud Deployment Options (CDOs).[3] CDOs describe deployment aspects, such as the mapping of software components to VMs and the selection of specific VM instance types, and also involve the configuration of reconfiguration rules.

Our approach uses techniques from the field of search-based software engineering [Harman 2007] to generate and optimize CDOs. Using search-based techniques for solving problems in the context of cloud computing is a reasonable approach that is also proposed by other authors, e.g., see [Harman et al. 2012a]. With our approach, a future SaaS provider does not longer have to manually design a CDO and hope that it exhibits desired quality characteristics when it is actually implemented, such as low costs and response times. Instead, a set of near-optimal CDOs is automatically created and assessed. Then, the future SaaS provider can select the candidate that constitutes the best trade-off solution and best suits its specific needs.

The approach for optimizing CDOs uses our genetic algorithm called CDOXplorer. It is a simulation-based optimization algorithm (e.g., see [Law and McComas 2000]) as it uses our simulator CDOSim to evaluate (i.e., to simulate) CDO candidates. CDOXplorer is also an evolutionary multi-objective optimization algorithm (cf. Sections 4.1.2 and 4.1.3) that optimizes the overall costs, average response times, and number of SLA violations of a CDO. CDOXplorer employs adaptive crossover and mutation rates and is combined with a local search (cf. Section 8.5). Hence, CDOXplorer is actually an adaptive, hybrid, and simulation-based genetic algorithm.

SC4: Proof of Concept - Design and Implementation

A proof of concept is created for the CloudMIG approach and the incorporated automatic conformance checking and deployment and reconfiguration

[3] CDOSim was developed in a master's thesis by Fittkau [2012]. The master's thesis was co-supervised by the author of this thesis.

optimization approach. The corresponding tool architecture and implementation are briefly described below.

1. *Software architecture:* A structured iterative architecture design process is used for creating the proof of concept's software architecture. The architecture design process follows the Attribute-Driven Design (ADD) method from Bass et al. [2002]. Architectural drivers are mapped to quality characteristics and subcharacteristics of the ISO/IEC 9126 standard [ISO/IEC 2001] that are used as a basis for selecting adequate architectural styles. The resulting architecture can be extended with plugin components. For example, further constraint validators can be plugged into the conformance checking process, additional KDM discoverers can provide support for further programming languages, and workload profiles can be created with the help of plugins that can process further monitoring log formats.

2. *Implementation:* The proof of concept implements the designed software architecture in the tool *CloudMIG Xpress*. *CloudMIG Xpress* is implemented in Java and builds upon the Eclipse Rich Client Platform (RCP) [McAffer et al. 2010] for creating cross-platform GUI-based applications. It implements the components defined in the corresponding software architecture and therefore allows software service providers to model cloud profiles, detect CEC violations, model the current deployment of the application that is to be migrated to the cloud, and to generate and optimize CDOs, for instance. Besides creating workload profiles from historic monitoring log data, *CloudMIG Xpress* also allows to create synthetic workload profiles for modeling arbitrary usage patterns. Figure 1.2 depicts a screenshot of *CloudMIG Xpress* that shows the creation of an exemplary synthetic workload profile.

SC5: Experimental Evaluation

The approach CloudMIG and its core components are assessed in the context of an extensive experimental evaluation. The experiments are designed, executed, and analyzed following the Goal Question Metric (GQM)

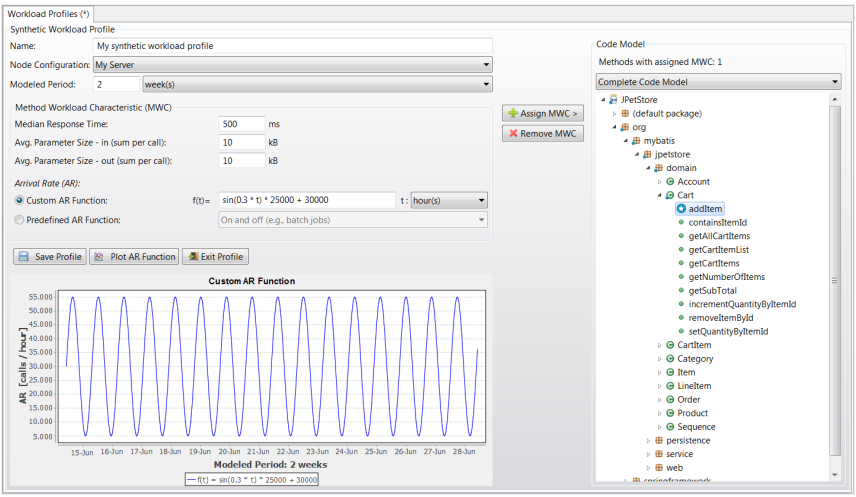


Figure 1.2. CloudMIG Xpress synthetic workload profile creation

method [Basili and Rombach 1988]. For evaluating the conformance checking approach and investigating patterns of detected CEC violations, three case studies are performed. Two case studies use a set of open source enterprise software systems, the third case study utilizes a software library from an industrial partner to assess CloudMIG's capabilities for detecting CEC violations. The experiments employ two well-known cloud environments (Google App Engine for Java[4] and Microsoft Windows Azure[5]) for detecting corresponding CEC violations and performing a CEC violation distribution analysis and a CEC violation density analysis, for instance. Furthermore, binary classification techniques are used to determine the approach's precision, which is excellent (cf. Section 11.5.2).

To evaluate the deployment and reconfiguration optimization approach, we employ the well-known public cloud environments Microsoft Windows

[4] http://developers.google.com/appengine/docs/java/overview/

[5] http://www.windowsazure.com/

Azure and Amazon EC2[6] and also utilize a private Eucalyptus[7] cloud environment. CDOXplorer is compared with three other state-of-the-art search and optimization algorithms. Our corresponding experiments comprise almost one million simulations with our simulation tool CDOSim. The experiments show that CDOXplorer can find solutions that are up to 60% better than those of the other search and optimization techniques (cf. Section 12.5).

1.4 Preliminary Work

This thesis builds on preliminary work that was already published in several research papers. Furthermore, this thesis is also based on various other theses, such as Diploma Theses and Master's Theses, which were co-supervised by the author. The corresponding research papers are categorized and briefly described along with the student theses in the following.

Foundations The three research papers listed below describe important foundations of the CloudMIG approach, such as its rationale, basic structure, and involved activities.

- [Frey and Hasselbring 2010a] Frey, Sören and Hasselbring, Wilhelm, Model-Based Migration of Legacy Software Systems into the Cloud: The CloudMIG Approach, *Softwaretechnik-Trends, 30 (2). pp. 84-85,* 2010.

  This paper introduces the CloudMIG approach. The problems tackled by the approach are explained by stating major challenges software service providers face when they migrate existing applications to the cloud. The paper also outlines the basic structure of CloudMIG.

- [Frey and Hasselbring 2010b] Frey, Sören and Hasselbring, Wilhelm, Model-Based Migration of Legacy Software Systems to Scalable and Resource-Efficient Cloud-Based Applications: The CloudMIG Approach,

[6] http://aws.amazon.com/ec2/

[7] http://www.eucalyptus.com/

Proceedings of the First International Conference on Cloud Computing, GRIDs, and Virtualization (Cloud Computing 2010), pp. 155-158, Lisbon, Portugal, November, 2010.

CloudMIG addresses two basic approaches for generating target architectures regarding a specific cloud environment. The focus is on the approach for generating and optimizing CDOs with our genetic algorithm CDOXplorer. The other approach is covered in this paper and describes a rule-based heuristic for creating a resource-efficient allocation of architectural features to cloud resources (cf. Section 6.3.2).

▷ [Frey and Hasselbring 2011b] Frey, Sören and Hasselbring, Wilhelm, The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications, *International Journal on Advances in Software, 4 (3 and 4). pp. 342-353,* 2011.

This journal article is an extended version of the previous paper. It adds (1) experiments showing the limitations of simplistic migration approaches (cf. Section 6.1.1) and (2) proposes the Cloud Suitability and Alignment hierarchy (CSA hierarchy) (cf. Section 7.4.1) for assessing existing applications regarding their suitability for specific cloud environments. Furthermore, the CSA hierarchy is also intended to judge the degree of a system's alignment regarding a cloud environment after performing a migration.

Automatic Conformance Checking The detection of a software system's CEC violations with our automatic conformance checking approach is covered in the following two research papers.

▷ [Frey and Hasselbring 2011a] Frey, Sören and Hasselbring, Wilhelm, An Extensible Architecture for Detecting Violations of a Cloud Environment's Constraints During Legacy Software System Migration, *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR 2011), pp. 269-278, Oldenburg, Germany,* March, 2011. doi: 10.1109/CSMR.2011.33

Our approach for modeling CECs and detecting CEC violations is introduced in this paper. The included experiments cover the cloud environment Google App Engine for Java and analyses regarding the CEC violations found in five web-based Java applications.

▷ [Frey et al. 2013a] Frey, Sören and Hasselbring, Wilhelm and Schnoor, Benjamin, Automatic Conformance Checking for Migrating Software Systems to Cloud Infrastructures and Platforms, *Journal of Software: Evolution and Process, 25.10, pp. 1089-1115*, 2013. doi: 10.1002/smr.582

This journal article is an extended version of the previous paper. It adds means for prioritizing detected CEC violations (cf. Section 7.4.2) and describes additional SMM-based violation detection capabilities (cf. Section 7.2.2). It also introduces the precursor of our Measurement Architecture for Model-Based Analysis (MAMBA) framework (cf. Section 10.2) for applying SMM-based measures.

CDO Optimization The research papers listed below address the optimization of the deployment and reconfiguration of software in the cloud.

▷ [Fittkau et al. 2012a] Fittkau, Florian and Frey, Sören and Hasselbring, Wilhelm, CDOSim: Simulating Cloud Deployment Options for Software Migration Support, *IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), pp. 37-46, Riva del Garda, Italy*, September, 2012.
doi: 10.1109/MESOCA.2012.6392599

This paper presents our simulation tool CDOSim. CDOSim is used as a fitness function by our genetic algorithm CDOXplorer. The experiments described in this paper demonstrate that CDOSim can produce sufficiently accurate simulation results.

▷ [Fittkau et al. 2012b] Fittkau, Florian and Frey, Sören and Hasselbring, Wilhelm, Cloud User-Centric Enhancements of the Simulator CloudSim

to Improve Cloud Deployment Option Analysis, *Proceedings of the European Conference on Service-Oriented and Cloud Computing (ESOCC), pp. 200-207, Bertinoro, Italy,* September, 2012. doi: 10.1007/978-3-642-33427-6_15

This paper describes CDOSim and focuses on the cloud user-centric perspective that is followed for simulating CDOs. For example, CDOSim abstracts away elements that are usually only known to cloud providers. CDOSim is used as a fitness function by our genetic algorithm CDOXplorer.

▻ [Frey et al. 2013b] Frey, Sören and Fittkau, Florian and Hasselbring, Wilhelm, Search-Based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud, *35th Int'l Conference on Software Engineering (ICSE 2013), pp. 512-521, San Francisco, USA,* May, 2013.

Our genetic algorithm CDOXplorer is introduced in this paper that also provides a comparison with three other state-of-the-art search and optimization methods.

Support Projects The research papers listed below do not cover CloudMIG, but, nevertheless, address subjects that are also relevant in the context of migrating software systems to the cloud.

▻ [Eysholdt et al. 2009] Eysholdt, Moritz and Frey, Sören and Hasselbring, Wilhelm, EMF Ecore Based Meta Model Evolution and Model Co-Evolution, *Softwaretechnik-Trends, 29 (2). pp. 20-21,* 2009.

This paper addresses the evolution of meta-models that are created with the Eclipse Modeling Framework (EMF). The proof of concept implementation of CloudMIG (*CloudMIG Xpress*) builds on EMF and offers EMF-based meta-models. For evolving those meta-models and the corresponding model instances, this paper can provide assistance.

▻ [van Hoorn et al. 2009] van Hoorn, André and Rohr, Matthias and Hasselbring, Wilhelm and Waller, Jan and Ehlers, Jens and Frey, Sören and Kieselhorst, Dennis, Continuous Monitoring of Software Services:

Design and Application of the Kieker Framework, *Department of Computer Science, Kiel University, Kiel, Germany, Technical Report, TR-0921,* 2009.

The Kieker monitoring framework is described in this technical report. *CloudMIG Xpress* can create workload profiles from Kieker monitoring log data. Corresponding workload profiles play an essential role as they constitute realistic drivers for simulations with our tool CDOSim.

▷ [Efftinge et al. 2011] Efftinge, Sven and Frey, Sören and Hasselbring, Wilhelm and Köhnlein, Jan, Einsatz domänenspezifischer Sprachen zur Migration von Datenbankanwendungen, *Datenbanksysteme für Business, Technologie und Web (BTW 2011), pp. 554-573, Kaiserslautern, Germany,* March, 2011 (in German).

This paper investigates the use of Domain-specific Languages (DSLs) for migrating database-centric software systems to new environments. CloudMIG also supports the migration of software systems to new environments but focuses on the systems' application logic and software architecture. The approach presented in this paper could therefore be worthwhile for further extensions of CloudMIG with regard to database migration support.

▷ [van Hoorn et al. 2011] van Hoorn, André and Frey, Sören and Goerigk, Wolfgang and Hasselbring, Wilhelm and Knoche, Holger and Köster, Sönke and Krause, Harald and Porembski, Marcus and Stahl, Thomas and Steinkamp, Marcus and Wittmüss, Norman, DynaMod Project: Dynamic Analysis for Model-Driven Software Modernization, *Proceedings of the 1st International Workshop on Model-Driven Software Migration (MDSM 2011), Project Presentations Track, Oldenburg, Germany,* March, 2011.

This paper reports on the DynaMod project that covered the creation of methods and techniques for supporting software modernization scenarios with dynamic analyses. As described in Section 7.1.3, some CEC violations cannot be detected using solely static analysis techniques. Hence, incorporating dynamic analyses in the conformance checking process would promise to further refine its detection capabilities.

▷ [Frey et al. 2011] Frey, Sören and van Hoorn, André and Jung, Reiner and Hasselbring, Wilhelm and Kiel, Benjamin, MAMBA: A Measurement Architecture for Model-Based Analysis, *Department of Computer Science, Kiel University, Kiel, Germany, Technical Report, TR-1112,* December, 2011.

The technical report introduces our Measurement Architecture for Model-Based Analysis (MAMBA) framework (cf. Section 10.2) for applying SMM-based measures. For example, our CloudMIG approach also uses MAMBA for detecting CEC violations with a specific constraint validator that can process SMM-based measures.

▷ [Frey et al. 2012] Frey, Sören and van Hoorn, André and Jung, Reiner and Kiel, Benjamin and Hasselbring, Wilhelm, MAMBA: Model-Based Software Analysis Utilizing OMG's SMM, *Proceedings of the 14. Workshop Software-Reengineering (WSR '12), pp. 37-38, Bad Honnef, Germany,* May, 2012.

This paper presents advancements of our MAMBA framework. For example, it presents the concept of the Metrics Definition Language (MDL) for simplifying the construction of SMM measures. The MDL also promises to simplify the construction of SMM-based constraint validators in the context of CloudMIG.

▷ [Wulf et al. 2012] Wulf, Christian and Frey, Sören and Hasselbring, Wilhelm, A Three-Phase Approach to Efficiently Transform C# into KDM, *Department of Computer Science, Kiel University, Kiel, Germany, Technical Report, TR-1211,* August, 2012.

CloudMIG Xpress provides support for extracting KDM models from C#-based software systems. This is due to the three-phase transformation of C# source code to KDM that is described in this technical report.

Student Theses The student theses listed below were co-supervised by the author of this thesis. They address subjects that are relevant in the context of migrating software systems to the cloud.

▷ Fittkau, Florian, Reconstructing Software Architectures using the Code- and Structure Package of the Knowledge Discovery Meta-Model, Bachelor's Thesis, Kiel University, Kiel, Germany, February, 2010.
http://eprints.uni-kiel.de/16427/

This Bachelor's Thesis covers the design and implementation of a software for extracting KDM-based architectural models from Java source code. CloudMIG uses KDM models to represent existing software systems.

▷ Zimmermann, Daniel, Eigenschaften, Entwurf und Evaluation ressourceneffizienter Softwarearchitekturen, Bachelor's Thesis, Kiel University, Kiel, Germany, September, 2010 (in German).
http://eprints.uni-kiel.de/16428/

Considering resource efficiency when creating cloud-based target architectures of existing software systems is essential. This Bachelor's Thesis investigates the possibilities for facilitating resource efficiency on an architectural level.

▷ Schnoor, Benjamin, Modeling Usage and Architecture Metrics for Software Systems Applying OMG's KDM and SMM, Bachelor's Thesis, Kiel University, Kiel, Germany, September, 2010.
http://eprints.uni-kiel.de/16429/

This Bachelor's Thesis investigates the use of SMM for modeling and applying measures. It also provides the predecessor of our MAMBA framework as described above. SMM-based measures are applied in the context of CloudMIG's conformance checking process, for instance.

▷ Tietjens, Björn-Peter, Modellbasierte Architektur-Rekonstruktion mit Hilfe von KDM und EMF, Diploma Thesis, Kiel University, Kiel, Germany, November, 2010 (in German). http://eprints.uni-kiel.de/15430/

This Diploma Thesis covers the extraction of KDM models from Java. The developed solution builds on Eclipse mechanisms and wraps an existing reverse engineering framework. The software developed in the context of this Diploma Thesis constitutes the foundation for CloudMIG's KDM extraction capabilities regarding Java-based software systems.

- Löffler, Pascal, Migration von Softwaresystemen auf IaaS-basierte Cloud Umgebungen, Bachelor's Thesis, Kiel University, Kiel, Germany, March, 2011 (in German). http://eprints.uni-kiel.de/16430/

  This Bachelor's Thesis investigates challenges when migrating software systems to cloud environments that offer infrastructure services, for example, for provisioning virtual machines. The collected challenges contribute to clarifying the basic assumptions that underlie the CloudMIG approach regarding those kinds of cloud environments.

- Fenner, Sören, Migration of Software Systems to Platform as a Service based Cloud Environments, Diploma Thesis, Kiel University, Kiel, Germany, October, 2011. http://eprints.uni-kiel.de/16431/

  Similar to the thesis above, this Diploma Thesis investigates challenges when migrating software systems to cloud environments. However, it focuses on cloud environments that offer comprehensive platform services, e.g., complete software stacks. Besides helping to clarify the basic assumptions that underlie the CloudMIG approach regarding those kinds of cloud environments, the Diploma Thesis also contributes experiments that evaluate CloudMIG's conformance checking approach (cf. Section 11.3.2).

- Fittkau, Florian, Simulating Cloud Deployment Options for Software Migration Support, Master's Thesis, Kiel University, Kiel, Germany, March, 2012. http://eprints.uni-kiel.de/16432/

  This Master's Thesis designs, implements, and evaluates the CDO simulation tool CDOSim. CDOSim is used in our tool *CloudMIG Xpress* for evaluating CDOs and also as part of the simulation-based genetic algorithm CDOXplorer.

- Wulf, Christian, Automatic Conformance Checking of C#-based Software Systems for Cloud Migration, Master's Thesis, Kiel University, Kiel, Germany, March, 2012. http://eprints.uni-kiel.de/16433/

  The three-phase approach for transforming C# to KDM that was mentioned above is designed, implemented, and evaluated in this Master's

Thesis. The developed software constitutes the foundation for *CloudMIG Xpress*' capabilities to extract KDM models from C#-based software systems.

▷ Prinz, Oliver, Transformation of Java Bytecode to KDM Models as a Foundation for Dependency Analysis, Diploma Thesis, Kiel University, Kiel, Germany, July, 2012. http://eprints.uni-kiel.de/16434/

This Diploma Thesis investigates the extraction of KDM models from Java bytecode. Extending *CloudMIG Xpress*' capabilities for extracting KDM models from Java bytecode is reasonable as often software systems that were developed a long time ago miss some source code artifacts. Furthermore, the Diploma Thesis explores ways for using KDM models as a basis to perform dependency analyses that could be utilized, for instance, to further refine CloudMIG's conformance checking process.

▷ Lübbe, Kim Yannik, Improving a Transformation of Java Models to KDM, Bachelor's Thesis, Kiel University, Kiel, Germany, September, 2012. http://eprints.uni-kiel.de/16435/

This Bachelor's Thesis investigates ways to improve *CloudMIG Xpress*' process for extracting KDM models from large Java-based software systems.

▷ Clausen, Alexander, Transforming Python into KDM to Support Cloud Conformance Checking, Bachelor's Thesis, Kiel University, Kiel, Germany, September, 2012. http://eprints.uni-kiel.de/16436/

This Bachelor's Thesis also addresses the extraction of KDM models from software systems to extend *CloudMIG Xpress*' corresponding capabilities. It covers a new KDM discoverer that extracts KDM models from Python source code.

▷ Kund, Simon, Design and Implementation of an Eclipse P2-based Online Repository for Exchanging Cloud Profiles, Bachelor's Thesis, Kiel University, Kiel, Germany, March, 2012. http://eprints.uni-kiel.de/21123/

This Bachelor's Thesis covers the design and implementation of a repository for exchanging cloud profiles, which can be created with *CloudMIG Xpress*.

- Kiel, Benjamin, Investigating the Use of Graph Databases for Large Model Repositories, Master's Thesis, Kiel University, Kiel, Germany, June, 2013.

  This Master's Thesis explores means for storing, managing, and processing large models. It also investigates whether and to which extent the use of graph databases can improve, for example, the performance of processing large models. As KDM models that are extracted by *CloudMIG Xpress* from software system artifacts frequently become very large, corresponding means promise to improve *CloudMIG Xpress*' performance.

1.5 Structure

This thesis consists of the following four parts:

- **Part I** describes the foundations of this thesis to clarify the context, to introduce central concepts, and to provide basic definitions that are relevant in the following parts.
  - **Chapter 2** gives an overview on cloud computing. It starts with summarizing the development of the cloud computing paradigm, its core benefits, and concerns that are sometimes associated with using cloud computing technologies, such as security and compliance issues. Then, the chapter presents the most frequently used definition of cloud computing that comes from the National Institute of Standards and Technology (NIST). Based on this definition, the chapter describes NIST's cloud computing reference architecture that explains many central cloud computing concepts as well as a model of involved actors, for instance.
  - **Chapter 3** describes the research area of software modernization. CloudMIG employs several methods and techniques from the subareas software reengineering and software migration. Hence, the chapter gives an overview on these two subareas and also describes

software reverse engineering that provides means for analyzing software systems and extracting representations on a higher level of abstraction, for instance. CloudMIG uses the meta-models KDM and SMM from OMG's Architecture-Driven Modernization initiative. Hence, Architecture-Driven Modernization and these two meta-models are also covered in the chapter.

- **Chapter 4** provides an overview on the field of Search-Based Software Engineering (SBSE). SBSE aims to apply search-based optimization techniques to problems in software engineering. The chapter describes core concepts of meta-heuristic optimization and also of multi-objective optimization. CloudMIG proposes the genetic algorithm CDOXplorer that also optimizes multiple objectives, i.e., costs, response times, and the number of SLA violations. The chapter also overviews genetic algorithms and describes simulation-based optimization. CDOXplorer falls under this category of optimization approaches, as the corresponding genetic algorithm uses our simulator CDOSim as a fitness function.

- **Part II** describes the CloudMIG approach and the research design and methods chosen for its development. Besides the general concepts and structure of the CloudMIG approach, a focus is on the included automatic conformance checking approach and the optimization of deployment and reconfiguration of software in the cloud.

  - **Chapter 5** describes the research design and methods used to develop and evaluate the CloudMIG approach. Utilized research methods comprise action research, literature review, and metamodeling, for instance. The research project is structured into six work packages. Each work package defines a set of research questions that are derived from the problems that are stated in Section 1.2. Furthermore, each work package defines (1) the research methods used to approach the particular research questions and also (2) the results that are produced through applying these research methods.

  - **Chapter 6** introduces the general concepts, structure, and activities of the CloudMIG approach for supporting the migration of enterprise

software systems to the cloud. The chapter starts with detailing the challenges that software service providers face when considering a migration to the cloud. To simplify reasoning about different migration options, the concept of cloud migration types is introduced. Then, the fundamental components of the approach are explained, i.e., its basic design, included activities, roles, and models.

- **Chapter 7** details the automatic conformance checking approach for detecting CEC violations. The chapter first presents several exemplary constraints and constraint violations and reasons about the general detectability of CEC violations. Afterwards, the chapter shows how CECs can be modeled and how SMM can be used as a part of those CEC definitions to enable a generic constraint modeling. Based on these models, the chapter then details the conformance checking process and describes how CECs are processed by constraint validators. The chapter finishes with a description of the CSA hierarchy and by explaining how the detected CEC violations can be prioritized.
- **Chapter 8** covers the deployment and reconfiguration optimization of software in the cloud. After clarifying basic concepts and assumptions that underlie our genetic algorithm CDOXplorer, the chapter presents our simulation tool CDOSim for simulating CDOs. The construction of the crossover and mutation operations constitutes an essential part in the overall design of any genetic algorithm. Hence, the chapter also details CDOXplorer's crossover and mutation operations. As described before, CDOXplorer is also an adaptive and hybrid genetic algorithm. Its specific adaptivity and hybridity characteristics are therefore also described in the chapter. The chapter finishes with a search space analysis that examines characteristics and the size of CDOXplorer's search space.
- **Chapter 9** describes the software architecture of the tool that constitutes a proof of concept realization of CloudMIG (*CloudMIG Xpress*). The tool architecture is created with the help of a structured, iterative architecture design process that follows the Attribute-Driven Design (ADD) method from Bass et al. [2002]. This ADD method is also summarized in the chapter. It uses architectural drivers as a basis

for selecting architectural styles that themselves support accomplishing certain non-functional quality characteristics. The chapter then describes the resulting plugin-based architecture that allows to add specific functionalities, such as additional KDM discoverers.

- **Part III** describes the evaluation of the CloudMIG approach. Among others, the evaluation includes experiments that are performed with the help of the proof of concept implementation *CloudMIG Xpress*.
  - **Chapter 10** covers implementation details of the tool *CloudMIG Xpress*. *CloudMIG Xpress* constitutes an implementation of the software architecture that is described in Chapter 9. Specific emphasis is put on the extraction of KDM models from existing software system artifacts and on our Measurement Architecture for Model-Based Analysis (MAMBA) framework for applying SMM-based measures. Furthermore, the chapter elaborates on implementing constraint validators and finally describes implementation details of our genetic algorithm CDOXplorer.
  - **Chapter 11** describes the evaluation of CloudMIG's automatic conformance checking approach. The included experiments are planned, executed, and analyzed following the Goal Question Metric (GQM) method from Basili and Rombach [1988], refining the research questions described in Chapter 5. Three case studies CC-1, CC-2, and CC-3 address different aspects of the conformance checking approach. CC-1 aims to show the applicability of the approach and investigates patterns of detected CEC violations. CC-2 and CC-3 build on experiments from student theses and assess the approach's performance and its feasibility for detecting CEC violations in an industrial case study, respectively. Furthermore, the chapter reports on threats to validity.
  - **Chapter 12** covers the evaluation of CloudMIG's deployment and reconfiguration optimization approach. The included experiments are also planned, executed, and analyzed following the GQM method. They assess CDOXplorer's performance and compare its results with three other state-of-the-art search and optimization algorithms. Furthermore, as the optimization of the deployment and reconfiguration of

software in the cloud usually involves various distinct cloud environment candidates, the experiments aim to demonstrate CDOXplorer's scalability in this regard. Finally, the chapter reports on threats to validity.

- ▻ **Chapter 13** presents other work that is related to CloudMIG. Related work comes from three different areas. As CloudMIG basically addresses the migration of software systems to the cloud, the chapter starts with reporting on corresponding related work (1). Afterwards, the chapter describes related work regarding conformance checking (2) and deployment and reconfiguration optimization (3). Each area is refined to several subareas. Then, characteristics of each subarea are identified for comparing the other work with our approaches. The characteristics are evaluated regarding the related work and the results are presented in a tabular overview. Based on each overview table, the related work of each subarea is then described in detail.

▻ **Part IV** concludes the thesis and presents future work.

- ▻ **Chapter 14** presents the conclusions of the thesis and describes future work. The conclusions summarize the challenges software service providers face when considering a migration to the cloud and provide an overview on CloudMIG's core concepts, structure, and activities, highlighting its conformance checking and deployment and reconfiguration optimization approach. A special emphasis is on the benefits that are provided by CloudMIG as shown by the extensive experiments. However, there are also some open issues that are worthwhile to be addressed in future work. Hence, the chapter also reports on the corresponding future work.

Part I

Foundations

Chapter 2

Cloud Computing

Cloud computing [Buyya et al. 2008; Foster et al. 2008; Hayes 2008; Birman et al. 2009; Dikaiakos et al. 2009; Grossman 2009; Dillon et al. 2010; Sriram and Khajeh-Hosseini 2010; Marston et al. 2011; Phaphoom et al. 2012; Murugesan 2013] provides applications and computing resources, such as storage and compute capabilities, as services over the internet. Starting from 2008, it attracted considerable attention in both industry and academia. At the time of writing, information technology research firm Gartner estimates that global end user spending on public cloud services amount to $110.3 billion in 2012 [Gartner, Inc. 2013].

The services offered by cloud providers span various abstraction levels. On the one end, their customers may merely use web-based software through a web browser, such as email or Customer Relationship Management (CRM) applications. On the other end, cloud providers may offer raw virtual machines (VMs) in several flavors including different types of CPUs and amount of memory and local storage, for instance. In this case, customers have the full flexibility to run any software system, but at the same time need to deploy, operate, and maintain the system on their own.

However, it is possible to browse web pages since the early days of the world wide web [BenMrad 1994; Leiner et al. 1997]. Likewise, mainframe VMs were invented even several decades ago [Waldspurger and Rosenblum 2012] before the term cloud computing was coined. Therefore, the enormous popularity of cloud computing is accompanied with a confusion regarding its novelty and whether cloud computing really constitutes a

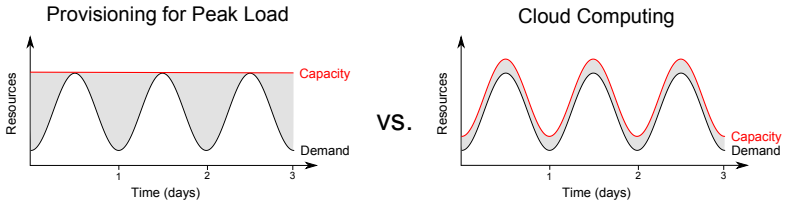


Figure 2.1. Cloud computing allows to dynamically provision resources (based on [Armbrust et al. 2009; Löffler 2011])

"game-changer."[1] As many IT companies try to benefit from the current sky-rocketing expectations, plenty of them seem to *cloud wash* (e.g., see [Adamov and Erguvan 2009]) their product portfolio, i.e., basically re-brand their offerings for being able to provide a "cloud computing product." This chapter elucidates on the cloud computing paradigm and shows that there exist characteristics that are actually new and powerful. For example, cloud computing enables to rapidly provision computing resources based on the current demand. This is illustrated in Figure 2.1. Instead of provisioning resources for peak load, cloud computing's so-called *elasticity* [Dustdar et al. 2011; Kuperberg et al. 2011; Galante and de Bona 2012; Islam et al. 2012; Suleiman et al. 2012] enables to quickly allocate and de-allocate resources on demand for reducing the waste of unused resources.

The chapter is organized as follows. Section 2.1 provides an overview on cloud computing. Section 2.2 describes the most recognized and accepted definition of cloud computing that originates from the US National Institute of Standards and Technology (NIST). The NIST also proposes a reference architecture of cloud computing that clarifies many basic concepts. This reference architecture is described in Section 2.3. Finally, Section 2.4 sums up this chapter.

[1] Neelie Kroes, European Commissioner for Digital Agenda called cloud computing a "game-changer for the EU economy" while presenting EU's cloud computing strategy on 27 September 2012, see `http://euobserver.com/news/117695`.

2.1 Overview

This section provides an overview on the cloud computing field. We start by recapitulating the development of the field in Section 2.1.1. The next Section 2.1.2 motivates the use of cloud computing technologies, i.e., it reports on incentives and benefits of employing cloud computing. However, the introduction of cloud computing technologies is sometimes accompanied by particular concerns. Those concerns are described in Section 2.1.3.

2.1.1 Development

Cloud computing is often referred to as a realization of the idea of *utility computing* [Parkhill 1966] or at least as a big step in its direction [Buyya et al. 2008; Dikaiakos et al. 2009; Weinhardt et al. 2009; Armbrust et al. 2010; Wang et al. 2010; Phaphoom et al. 2012]. The notion of utility computing dates back to the 1960s where John McCarthy anticipated [Foster et al. 2008; Younge et al. 2010] that computing services might someday be delivered on demand in a similar way as is known for public utilities, such as water, gas, and electricity. In essence, computing services were envisioned to be available to anyone to the extent required at any point in time. However, the term cloud computing was coined later.

Sriram and Khajeh-Hosseini [2010] state that the term seems to originate from network diagrams that often represent the Internet as a cloud shape. Though, according to Zhang et al. [2010] "it was after Google's CEO Eric Schmidt used the word [cloud] to describe the business model of providing services across the Internet in 2006, that the term [cloud computing] really started to gain popularity." Such a business model was already pursued, at this time, for several years by Application Service Providers (ASPs) [Tao 2001]. Customers can use applications over the internet that are hosted and maintained by ASPs.

Grid and cluster computing also share some commonalities with cloud computing. However, there exist considerable differences, for example, regarding their programming models, applications, scalability, abstractions,

resource access, and compute models [Buyya et al. 2008; Foster et al. 2008; Youseff et al. 2008; Wang et al. 2010].

A major contribution in initiating and shaping the cloud computing movement was made by Amazon [Armbrust et al. 2009]. Amazon.com, Inc. started as an online bookseller. In 2006, their subsidiary Amazon Web Services, Inc. offered an online data storage service termed Amazon Simple Storage Service (Amazon S3) and launched the Amazon Elastic Compute Cloud (Amazon EC2). Over the course of time, Amazon had gained considerable experience in the efficient operation of computing resources. Advances in server technology and software facilitated the efficient use of VMs on commodity hardware [Buyya et al. 2009]. Amazon EC2 was initially planned as an internal platform that should enable rapid provisioning of VMs to corporate development teams.[2] Amazon realized that provisioning resources in an on demand and self-service manner was a veritable business model on its own. As a consequence, Amazon offered these services publicly and thus also launched one of the first publicly available cloud environments.

The rise of cloud computing was also made possible by increasing Internet bandwidth that made it economically reasonable to lease computing resources over broadband WAN connections [Bojanova et al. 2013]. Furthermore, as cloud computing relies on the provisioning of services, it is also closely related to service-oriented computing [Kappel et al. 2011]. For example, computing services are offered by cloud providers to cloud users. Corporate cloud users may themselves offer services to end users based on the rented computing services. Hence, combining cloud computing and service-oriented methods and technologies seems to be natural and worthwhile (e.g., see [Zhang and Zhou 2009]).

2.1.2 Benefits

The dynamic resource provisioning that is facilitated by cloud computing is appealing to many customers, developers, companies, organizations,

[2] http://www.zdnet.com/how-amazon-exposed-its-guts-the-history-of-awss-ec2-3040155310/

researchers, and governments. For figuring out the concrete benefits of using cloud computing technologies, it is useful to emphasize cloud characteristics that are actually new or that could not be adequately realized before the emergence of the cloud computing paradigm. According to Armbrust et al. [2009], the following three aspects of cloud computing are new:

1. "*The illusion of infinite computing resources available on demand*, thereby eliminating the need for Cloud Computing users to plan far ahead for provisioning.

2. *The elimination of an up-front commitment by Cloud users*, thereby allowing companies to start small and increase hardware resources only when there is an increase in their needs.

3. *The ability to pay for use of computing resources on a short-term basis as needed* (e.g., processors by the hour and storage by the day) and release them as needed, thereby rewarding conservation by letting machines and storage go when they are no longer useful."

The third aspect stated above is also related to the problem of under- and over-provisioning of traditional server resources. It is common that resources are provisioned for peak load (cf. Figure 2.1), hence, data centers often exhibit a rather low average utilization, especially in the case of fluctuating workload patterns, such as those of enterprise applications. For example, Armbrust et al. [2009] also report on real world estimates [Rangan 2008; Siegele 2008], which state that the server utilization in data centers ranges from 5% to 20%. On the other hand, if user demand unexpectedly ascends steeply, for example, because of social media news coverage, a fixed amount of server resources may not suffice. Hence, services may by unavailable due to temporal under-provisioning causing users to permanently abandon the offered services [Armbrust et al. 2010].

In general, cloud computing allows companies to switch the focus of IT spending from capital expenditures (CapEx) to operational expenditures (OpEx) [Leymann 2009; Pocuca et al. 2012; Baglietto et al. 2012]. That means, server hardware may not longer be purchased. Instead, computing resources may rather be rented when they are actually needed and in use, for example,

VMs may be started and stopped dynamically for adapting to a varying demand. Furthermore, data center operation may not be a core expertise of an organization. Hence, it may be reasonable to outsource the operation and maintenance of specific IT services to companies which are experienced in that domain [Zhang et al. 2010] and that are able to realize an economy of scale [Dikaiakos et al. 2009; Grossman 2009; Rafique et al. 2011; Wang et al. 2012].

Such an outsourcing of services may also allow a company to focus on its core business processes that actually distinguish them from competitors [Armbrust et al. 2010]. Because of the illusion of infinite computing resources that was mentioned above, cloud computing can also enable solving specific uses cases or implementing specific services that were not possible before. For example, the document-sharing website Scribd was able to migrate from Adobe Flash to HTML5 in just a few weeks by converting its documents with the help of a batch processing job that ran on up to 2,000 Amazon EC2 spot instances in parallel.[3]

2.1.3 Concerns

Along with the recent considerable interest in cloud computing technologies, there exist, however, several concerns and reservations that are frequently quoted. These concerns are briefly described below.

- **Compliance**: Software systems that are deployed to cloud environments may not correspond to certain regulations and it may not be possible to achieve legal compliance. Several domains have to comply with a wealth of standards, rules, and laws. For example, storing and processing electronic health records (EHRs) in cloud-based systems require sophisticated precautions for enforcing privacy and data security [Zhang and Liu 2010] and meeting requirements, e.g., regarding the Sarbanes-Oxley Act and Health and Human Services Health Insurance Portability and Accountability Act (HIPAA) regulations [Armbrust et al. 2010].

[3] https://aws.amazon.com/solutions/case-studies/scribd/

Furthermore, transferring sensitive information to data centers in countries that allow, under certain conditions, data access to domestic law enforcement agencies—such as the US Patriot Act [Pearson and Benameur 2010; Zhou et al. 2010b]—might disqualify the use of those cloud services because of legislation issues.

- **Loss of governance**: Operating an on premise infrastructure enables a fine-grained control of aspects such as maintenance windows, availability, application performance, security mechanisms, employed middleware components, and procured server hardware. Considering a public cloud environment, those aspects may no longer be in the sole responsibility of a software service provider, but may be in a large part determined by the cloud provider. Hence, potential cloud users may fear losing a part of present control while at the same time remaining fully responsible regarding their own customers (e.g., see [Catteddu and Hogben 2009]).
- **Performance variability**: Cloud environments may exhibit a varying performance. This may be caused by workload variability or resource time-sharing [Li et al. 2010a; Iosup et al. 2011], for instance.
- **Security**: Security is often seen as a major concern when considering the use of IT cloud services [Gruschka and Jensen 2010; Popovic and Hocenski 2010; Rocha et al. 2011; Subashini and Kavitha 2011; Ren et al. 2012; Tchifilionova 2011]. On the other hand, it could also be argued that established cloud providers can offer a level of security that cannot be ensured in many cases, for example, if security and hosting in general does not constitute a small company's core domain of expertise. The security aspect subsumes several specific security concerns, such as data protection, access control, and identity management.
- **Software license incompatibility**: Software installed to on premise machines may be restricted to run on specific servers [Armbrust et al. 2009]. Furthermore, existing licensing agreements may forbid to install software in virtual machines in general.
- **Vendor lock-in**: Building cloud-based applications with proprietary cloud services may impede portability, i.e., the transfer of an application to another cloud environment or to an on premise infrastructure [Leavitt

2009; Teckelmann et al. 2011; Yu 2012]. Hence, cloud users may be forced to accept changing contract components, for example, adverse SLAs or increasing prices.

2.2 Definition by NIST

Especially in the early days of cloud computing, when the field was in its infancy and the interest in cloud computing technologies started to experience a giant boost, there was much confusion on what cloud computing actually stood for. In the context of this thesis, we build on the cloud computing definition by the US National Institute of Standards and Technology (NIST) [Mell and Grance 2011] that is widely adopted [Binz et al. 2011; Zhang et al. 2010; Höfer and Karagiannis 2011; Nasir and Niazi 2011; Silva and Lucredio 2012; Folkerts et al. 2013] and that seems to be the one that is most respected [Dillon et al. 2010; Pallis 2010; Sriram and Khajeh-Hosseini 2010; Phaphoom et al. 2012].

Definition: Cloud Computing (Mell and Grance [2011])

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.

The main part of the definition is structured into three sections and describes the essential characteristics, service models, and deployment models of cloud computing. This section follows the structure of the NIST definition and briefly explains cloud computing's essential characteristics in the next Section 2.2.1. The service models and deployment models are then described in the Sections 2.2.2 and 2.2.3, respectively.

2.2.1 Essential Characteristics

The NIST definition of cloud computing describes the following five essential characteristics.

- **On-demand self-service**: Customers can manage the usage of cloud services—such as resource allocation and de-allocation—in a self-service manner. That means, instead of purchasing hardware and software via traditional procurement processes, compute and software services can be leased by cloud users on demand and even without requiring manual intervention.
- **Broad network access**: Capable network connections make it economically reasonable to outsource computations to remote locations, i.e., to transfer input and output data to and from geographically distributed cloud data centers. Resources can be accessed via a broad range of devices, such as PCs, laptops, and mobile devices.
- **Resource pooling**: Resources, such as network connections, storage, and CPU capabilities, both physical and virtual [Luo 2010; Waldspurger and Rosenblum 2012], are offered in the sense of resource pools. Users of those resources are not aware of the exact location of these resources. However, in some cases, the location can be specified on a higher level of abstraction. For example, users may be able to select the country where VMs should reside.
- **Rapid elasticity**: Resources can be dynamically and automatically allocated and de-allocated based on fluctuating demand. Besides counteracting over- and under-provisioning [Meng et al. 2010], users can benefit from seemingly unlimited resources. There exists a wide range of technologies for specifying the rules for elastically allocating and de-allocating resources [Galán et al. 2009; Chapman et al. 2010; Dustdar et al. 2012; Galante and de Bona 2012; Huber et al. 2012]. In the following, we briefly present two examples from Dustdar et al. [2012] and Chapman et al. [2010].

```
1 @SYBLLang ("MONITORING SPOT_PRICE=get_env(AVG_SPOT_PRICE)")
2 @SYBLLang ("CONSTRAINT LOW_SPOT_PRICE =(SPOT_PRICE < 1.2)")
3 @SYBLLang ("STRAGEGY EXECUTE(ExecutionStrategy.WAIT_UNTIL,
        LOW_SPOT_PRICE)")
4 Solution s = solveOnSpotInstance(OptimizationProblem p);
```

Listing 2.1. SYBL directives for postponing a computation (optimization problem `p`) until the price for a spot instance is lower than the given threshold 1.2 (from [Dustdar et al. 2012])

```
1  <elr:ElasticityRule name="DI_Increase">
2    <elr:Trigger>
3      <TimeConstraint unit="ms">5000</TimeConstraint>
4      <elr:Expression>
5        kpis.totalUsers / components.DI.replicas.amount > 200
6        && components.DI.replicas.amount < 5
7      </Expression>
8    </Trigger>
9    <elr:Action run="deployVM(components.DI)"/>
10 <elr:ElasticityRule>
```

Listing 2.2. An elasticity rule for scaling an SAP ERP system. Dialog instance (DI) components are added when the workload increases. A new DI is deployed for every 200 users, up to 5 DI replicas can be deployed (from [Chapman et al. 2010]).

Listing 2.1 from Dustdar et al. [2012] shows directives of the Simple-Yet-Beautiful Language (SYBL) for managing elasticity in cloud-based applications. The example considers a Java-based system. The SYBL directives are implemented with Java annotations [Bloch 2008], they abstract from cloud API usage details. Line 1 initiates monitoring the spot prices of VM instances. A defined strategy (Line 3) postpones a Java-based computation of an optimization problem (Line 4) until a specific constraint (Line 2) matches. In this example, the constraint ensures that the VM instance spot price is below a given threshold before the computation of the optimization problem is started.

Listing 2.2 from Chapman et al. [2010] shows an elasticity rule in XML for scaling an SAP ERP system. The SAP system employs so-called dialog

instance (DI) components for handling business logic and generating web-based user interfaces for the interaction with SAP users. The elasticity rule contains an action (Line 9), which describes that a new DI is added if a specific trigger (Lines 2-8) fires. The described trigger ensures that for every 200 users a new DI is added where a maximum of 5 replicas are allowed.

- **Measured service**: This essential characteristic of cloud computing describes the metering of utilized computing resources and software services. A cloud platform keeps track of the amount of used resources and software services and also provides these information to the cloud users. Hence, from a cloud user perspective, it is possible to optimize the employed type and quantity of resources in a fine-grained manner, for example, regarding costs and QoS properties.

2.2.2 Service Models

The NIST definition of cloud computing describes the following three service models.

- **Software as a Service (SaaS)**: Applications that run on a cloud platform are provided to cloud users over a network connection. Examples for those applications are web-based email, CRM, or ERP systems such as Gmail,[4] Salesforce Sales Cloud,[5] and SAP Business ByDesign,[6] respectively. Cloud users can access these applications via various devices such as PCs, laptops, and mobile devices through web browsers, client applications, or provided APIs.

  A SaaS application and the underlying hardware infrastructure and software stack is operated and maintained by a cloud provider. Hence, a SaaS application user does not need to update the application or install security fixes to the underlying operating system on her own, for instance.

[4] https://mail.google.com/

[5] https://www.salesforce.com/sales-cloud/

[6] https://www.sapbydesign.com/

Most SaaS applications allow their users to perform, to a limited extent, configurations to customize specific settings. However, the SaaS service model does not allow users to deploy customer-specific applications.

- **Platform as a Service (PaaS)**: This service model allows users to deploy own applications, both self-made applications or third-party applications. The cloud platform that hosts these applications includes a set of software components—e.g., operating systems, middleware components, and libraries—that run on a provided hardware infrastructure. The cloud platform is operated and maintained by the cloud provider. For being able to run in a specific PaaS cloud platform, an application has to comply with the restrictions induced by the platform, for example, a cloud platform may only support Windows programs or Java-based applications. Most PaaS-based cloud platforms offer auto-scaling capabilities that are transparent to cloud users. Examples for PaaS cloud environments are Google App Engine (GAE),[7] Heroku,[8] and Engine Yard.[9]
- **Infrastructure as a Service (IaaS)**: IaaS-based cloud environments provide fundamental computing resources to their users, such as storage, networks, and processing capabilities. Many IaaS cloud platforms allow the users to package arbitrary software stacks to VM images and run VM instances from those VM images [Schmidt et al. 2010]. Hence, users can also employ operating systems and middleware components of their choice, for instance. However, in IaaS-based cloud environments, cloud providers only manage and maintain the underlying hardware infrastructure and basic software components, such as hypervisors. The software stacks running in VMs have to be kept current by cloud users. Furthermore, auto-scaling capabilities are often not fully transparent to cloud users. For example, they might have to define elasticity rules (cf. Section 2.2.1) by themselves to fully leverage dynamic resource scaling capabilities. Examples for IaaS-based cloud environments are Amazon

[7] https://developers.google.com/appengine
[8] https://www.heroku.com/
[9] https://www.engineyard.com/

EC2,[10] Microsoft Windows Azure (VM role),[11] and HP Cloud.[12] Most IaaS-based cloud environments allow their users to select the geographical locations where VMs are run. For example, Amazon EC2 offers several data centers (so-called *regions*) in locations such as US East (Northern Virginia) and Asia Pacific (Singapore). Each region also contains several *availability zones*. These are distinct locations in Amazon EC's regions that are built to isolate each availability zone from failures in other availability zones.

All of the aforementioned service models have in common that cloud service users have only limited control and knowledge over the underlying hardware and software infrastructure. For example, users of IaaS-based cloud environments most often have no control on which host their VM instance actually runs or which VM instances are also executed on the hosts of their VM instances.

2.2.3 Deployment Models

The NIST definition of cloud computing describes the following four deployment models.

- **Private cloud**: Only a single organization uses the cloud infrastructure. The cloud infrastructure may be provisioned by the organization itself or by a third party. Popular open source software for building private (but also public) IaaS-based clouds are Eucalyptus,[13] OpenStack,[14] and Cloud Stack.[15]
- **Community cloud**: The cloud infrastructure is solely used by a community of consumers that share some common interests, such as a common business domain.

[10] https://aws.amazon.com/ec2/
[11] https://www.windowsazure.com/
[12] https://www.hpcloud.com/
[13] http://www.eucalyptus.com
[14] https://www.openstack.org/
[15] https://cloudstack.apache.org

- **Public cloud**: The offered cloud services are available to the general public.
- **Hybrid cloud**: The hybrid cloud model combines at least two of the other deployment models (private cloud, community cloud, and public cloud). For example, a cloud user may host sensitive data in a private cloud and perform batch processing jobs on anonymized data with the help of public cloud resources.

2.3 NIST Cloud Computing Reference Architecture

This section gives an overview on the NIST cloud computing reference architecture [Liu et al. 2011]. It was built to support the adoption of cloud computing technologies into the US Government. However, due to its general and vendor-neutral nature, it subsumes many central concepts of cloud computing and serves well to concisely explain those concepts that also form cornerstones of our CloudMIG method. This section is structured as follows. Section 2.3.1 gives an overview on the NIST cloud computing reference architecture. Then, Section 2.3.2 describes the actors in cloud computing.

2.3.1 Overview

Figure 2.2 shows an overview of the reference architecture that includes the major actors, activities, and functions in cloud computing. The actors are described in the next Section 2.3.2 in greater detail, some of them were already mentioned before. The reference architecture aligns with the NIST definition of cloud computing (cf. Section 2.2). For example, the *cloud provider* actor is responsible for orchestrating the system components to provide services, according to the service models of the NIST definition (SaaS, PaaS, and IaaS), to *cloud consumers*. SaaS-based services can build on PaaS

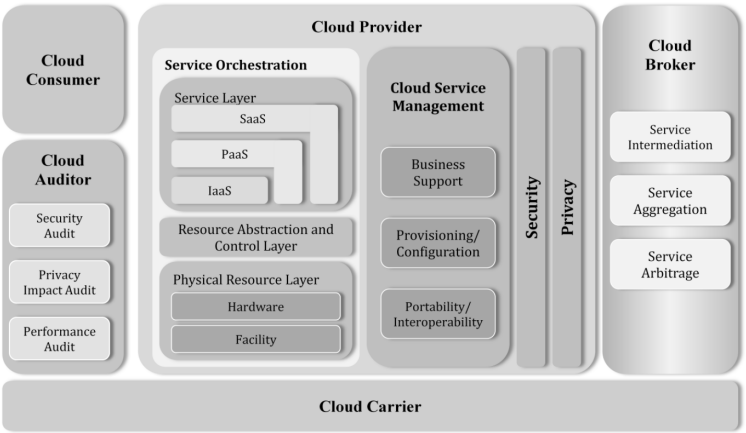


Figure 2.2. NIST cloud computing reference architecture overview (based on [Liu et al. 2011])

and IaaS-based services. For example, they can themselves use VMs from an IaaS-based cloud environment. However, this is not a prerequisite. The same is true for PaaS-based services, which may be constructed using services from an IaaS-based cloud environment.

The *service layer* comprises the SaaS, PaaS, and IaaS services models and builds upon the *resource abstraction and control layer*. This layer comprises, among others, hypervisors and means for access control. The service layer uses resources of the *physical resource layer*. The latter includes the actual hardware, such as servers and routers, and the data center facilities, for instance. Cloud providers use *cloud service management* functions to compose the services that are offered to cloud consumers. Typical functions include *customer management*, *accounting and billing*, and *metering*, for instance. Furthermore, implementing means for ensuring *security* and *privacy* also constitute central responsibilities of a cloud provider (cf. Figure 2.2). Considering those activities and functions of a cloud provider, we can now define the notion of a *cloud environment*.

Definition: Cloud Environment

A cloud environment constitutes all SaaS, PaaS, and IaaS services that are offered by a cloud provider. Furthermore, a cloud environment includes all components of the cloud provider actor of the NIST cloud computing reference architecture that support the provisioning and management of these services.

Please note that the cloud environment definition builds upon the NIST cloud computing reference architecture. It is used in the context of this thesis, however, it is not part of the NIST cloud computing reference architecture.

2.3.2 Actors in Cloud Computing

As can be seen in Figure 2.2, the NIST cloud computing reference architecture includes five participants for the cloud computing domain that it calls *actors*. The reference architecture defines the actors as follows (cf. [Liu et al. 2011]).

- "*Cloud Consumer*: A person or organization that maintains a business relationship with, and uses service from, *Cloud Providers*.
- *Cloud Provider*: A person, organization, or entity responsible for making a service available to interested parties.
- *Cloud Auditor*: A party that can conduct independent assessment of cloud services, information system operations, performance and security of the cloud implementation.
- *Cloud Broker*: An entity that manages the use, performance and delivery of cloud services, and negotiates relationships between *Cloud Providers* and *Cloud Consumers*.
- *Cloud Carrier*: An intermediary that provides connectivity and transport of cloud services from *Cloud Providers* to *Cloud Consumers*."

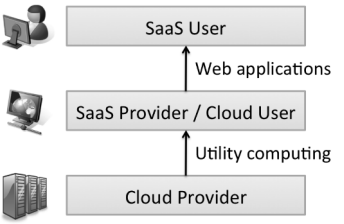


Figure 2.3. Cloud computing role model (based on [Armbrust et al. 2009])

Besides this actor model of the NIST cloud computing reference architecture, Armbrust et al. [2009] contribute a further model that describes common roles of cloud computing. This model is briefly described below.

Cloud Computing Role Model by Armbrust et al. [2009]

Figure 2.3 shows the role model from Armbrust et al. [2009]. The *cloud provider* role corresponds to the cloud provider actor from the NIST cloud computing reference architecture. Two distinct roles (*SaaS user* and *SaaS provider/ cloud user*) can represent the cloud consumer actor. A *SaaS user* utilizes applications that are provided according to the SaaS service model. Those SaaS applications are provided by *SaaS providers*. If the SaaS providers themselves utilize resources from PaaS or IaaS-based cloud environments (cf. Section 2.3.1), those *SaaS providers* can also be called *cloud users*.

2.4 Summary

This chapter gave an overview on cloud computing. Cloud computing is often seen as an implementation of the utility computing paradigm that suggests considering computing services as utilities—in an analogy to, e.g., electricity and gas—that are delivered to customers on demand and billed in a pay-per-use manner. It enables users to benefit from seemingly unlimited computing resources (e.g., servers, applications, and services) that are

accessed over a network connection. Cloud computing also eliminates the necessity of huge up-front investments by shifting the focus from capital expenditures (CapEx) to operational expenditures (OpEx). Computing resources can be paid on a short-term basis according to the actual usage (e.g., processors per hour). However, the adoption of cloud technologies is sometimes accompanied by concerns such as compliance and security issues.

This thesis builds upon the NIST definition of cloud computing that is widely adopted and accepted. The definition by NIST describes the following five essential characteristics: On-demand self-service, broad network access, resource pooling, measured service, and rapid elasticity. The latter enables to rapidly allocate and de-allocate resources to counter server under- and over-provisioning, respectively. The definition by NIST also defines the three service models Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). The service models differ in the type of provided computing resources and in the scope of controls the users and providers of the cloud services have. Furthermore, the definition by NIST describes the four deployment models private cloud, community cloud, public cloud, and hybrid cloud. They determine how the infrastructure for provisioning the cloud services is shared.

Finally, the NIST cloud computing reference architecture was described. It covers several fundamental concepts of cloud computing, such as relevant actors and their activities and functions. Based on the NIST cloud computing reference architecture, the notion of a *cloud environment* was defined. The actor model of the NIST cloud computing reference architecture overlaps with the role model from Armbrust et al. [2009] that is also used in the context of this thesis. This role model distinguishes cloud providers, SaaS providers/ cloud users, and SaaS users.

Chapter 3

Software Modernization

This chapter describes the field of software modernization. CloudMIG addresses the modernization of software systems by facilitating the migration to a platform or infrastructure cloud environment. Considering the classic software lifecycle definition described by Boehm [1976], any adaptation of an already implemented, tested, and deployed software system takes place in a phase called *software maintenance*. Therefore, the field of software maintenance [Yau et al. 1988; Hale et al. 1990; Haziza et al. 1992; Pigoski and Nelson 1994] is closely related to software modernization. The IEEE Standard 1219-1998 [IEEE 1998] defines software maintenance as follows.

Definition: Software Maintenance (IEEE [1998])

Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment.

Four types of software maintenance are being distinguished in this standard. Adaptive maintenance activities (1) aim at modifying a software system to conform to new business rules or to new requirements. As opposed to this, corrective maintenance activities (2) fix defects. The emergency maintenance activities (3) are themselves corrective maintenance activities of exceptional urgency. Perfective maintenance activities (4) can be applied to improve internal and external quality attributes. The field of software maintenance is also closely related to the research area of *software evolution* [Kemerer and Slaughter 1999; Cook et al. 2001; Mens et al. 2005;

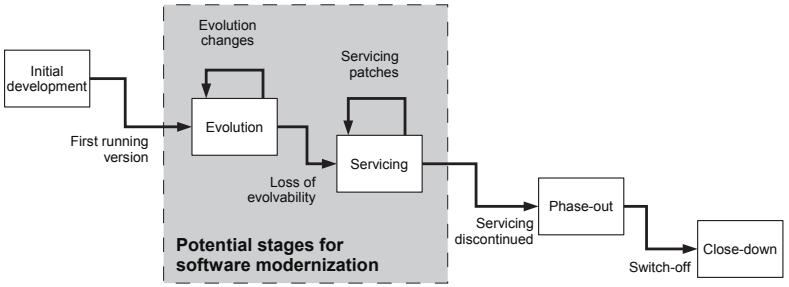


Figure 3.1. Software modernization in the context of the Simple Staged Software Lifecyle Model (based on [Bennett and Rajlich 2000])

Godfrey and German 2008]. Lehman [1980] introduced the so-called *laws of software evolution* that describe observed phenomena of evolving systems, such as a steadily increasing complexity, for instance. However, in contrast to software maintenance there exists no common definition for the notion of software evolution. Bennett and Rajlich [2000] state that "[...] *some researchers and practitioners use it as a preferable substitute for maintenance.*" Nevertheless, attempts have been made to distinguish the involved activities (e.g., see [Chapin et al. 2001]).

Furthermore, the term *evolution* can refer to a stage in the staged model of the software lifecycle as described by Bennett and Rajlich [2000]. Figure 3.1 illustrates the simple staged version of this model and highlights the stages that are common candidates for starting software modernization activities. After the initial development, software maintenance activities as well as pervasive adaptations can be performed in the *evolution* stage, whereas in the *servicing* stage the system has lost evolvability and merely emergency and corrective maintenance activities can be performed.

Evolution and *servicing* are potential stages for modernizing a software system. Ulrich [2004] states that "*modernization examines, exposes and facilitates the refactoring, redesign and redeployment of core application architectures with the intent of meeting critical business requirements in a way that lowers risks, costs and delivery timeframes.*" As servicing is abandoned in the *phase-out*

stage and software is replaced in the *close-down* stage, no extensive software modernization activities will typically be initiated in these stages.

In the rest of this chapter, we describe the subjects of the software modernization research area that are relevant in the context of CloudMIG. The field of reengineering provides important methods and techniques for modernizing software, as for example stated by Comella-Dorda et al. [2000]. Therefore, reengineering is explained in Section 3.1. CloudMIG is an approach for supporting the migration of software to the cloud. Hence, section 3.2 describes the field of software migration. Furthermore, CloudMIG utilizes various standards of the Architecture-Driven Modernization initiative from the OMG. The initiative and the corresponding standards are described in Section 3.3, before Section 3.4 summarizes this chapter.

3.1 Reengineering

Reengineering denotes a methodology that addresses the analysis of existing software systems and their reuse, complete or in parts, to build adapted systems where the purpose or the construction of the new systems differ to a great extent. Existing software artifacts may have to be transformed to be usable in a changed context. In this thesis, the most common definition of reengineering given by Chikofsky and Cross [1990] is used.

Definition: Reengineering (Chikofsky and Cross [1990])

Reengineering, also known as both renovation and reclamation, is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form.

The reengineering definition names as an essential characteristic the intended *new form* of the system. The aimed new form may be represented through a restructured software architecture, new technical foundation, or a novel followed development paradigm, for instance. In the case of migrating

software systems to the cloud, the new form can be seen as the system that is adapted up to a certain point until reaching cloud compatibility or meeting a defined alignment level (see Section 7.4.1).

This section is structured as follows. Section 3.1.1 gives an overview on reengineering. A central model of this field is the horseshoe model of reengineering [Kazman et al. 1998]. It is described in Section 3.1.2. The definition of reengineering stated above also covers the examination of software systems. The field of reverse engineering provides corresponding methods and techniques and is therefore described in Section 3.1.3.

3.1.1 Overview

The technical difficulties one faces when migrating software systems to the cloud overlap to some extent with those of general reengineering efforts. For example, software ages over time as described by Parnas [1994] and the present design documentation might no longer cover the current status quo. Furthermore, a system's internal structure may be eroded and therefore modifying system components can become cumbersome because the risk of inadvertently inducing side effects increases.

Moreover, because of steadily increasing complexity (e.g., see [Lehman 1980]) the program structures tend to become less comprehensible and modifiable while evolving. Hence, it is useful to apply software reengineering methods like redocumentation [Freeman and Munro 1992] and design recovery [Biggerstaff 1989] to restore the evolvability. A further common reenigneering method is reverse engineering. As this discipline is of special interest in the context of CloudMIG, it is described separately in Section 3.1.3

Referring to the software reengineering definition from Chikofsky and Cross [1990] stated above, Byrne [1992] proposes a conceptual foundation for software reengineering and a general software reengineering model that builds upon this foundation. The conceptual foundation consists of basic properties, assumptions, and principles regarding the software reengineering

domain. For example, starting from the definition of different abstraction levels of system artifacts, one property states that "*as the level of abstraction decreases, the amount of information describing a system increases*" [Byrne 1992, p. 228]. Byrne [1992] considers the following three principles as key concepts of software reengineering.

1. Abstraction
2. Alteration
3. Refinement

According to the general software reengineering model that is also described by Byrne [1992], these principles are applied successively in respective steps. The abstraction step involves the application of reverse engineering techniques. Alteration operations can be conducted along the aforementioned different abstraction levels. These operations produce new or modified system artifacts. For altering system artefacts as well as detailing them in the refinement step, forward engineering techniques are being utilized. Forward engineering is defined by Chikofsky and Cross [1990] as follows.

Definition: Forward Engineering (Chikofsky and Cross [1990])

Forward engineering is the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a system.

In every reengineering project it should be decided whether the three reengineering principles abstraction, alteration, and refinement have to be applied in full breadth. An essential distinction can be drawn through deciding on a black-box or a white-box approach as described by Seacord et al. [2003]. The first aims at wrapping the existing system's functionality by utilizing provided interfaces. For example, features can be altered by incorporating transformations of the input and output data. This strategy promises an

overall reduced initial effort compared to accomplishing an in-depth investigation of a legacy system and altering established structures. Nevertheless, it is often necessary to conduct a white-box approach, i.e., to investigate and modify internal program structures, to actually improve maintainability in a sustainable way, for instance.

A postulation that distinguishes reengineering from general software modernization is posed by Sneed [1995]. The author states that no new business functionality should be added while reengineering a software system as the accumulated complexity poses a serious threat to project success. Further foundations of reengineering stem from the research area of software reuse that is, for example, surveyed by Krueger [1992] and Frakes and Kang [2005]. Baumöl et al. [1996] distinguish *planned* from *unplanned* software reuse and point out that software reengineering always addresses the latter. For example, in contrast to the field of software product lines (e.g., see [Clements and Northrop 2001]), the specific way software artifacts are reused in a software reengineering project was not planned during construction.

3.1.2 The Horseshoe Model

The horseshoe model of reengineering was introduced by Kazman et al. [1998]. It represents a vivid metaphor for describing typical software reengineering processes, an illustration is shown in Figure 3.2. The horseshoe model corresponds to the general reengineering model from Byrne [1992] that was outlined above (cf. Section 3.1.1). It adopts the abstraction, alteration, and refinement principles with a focus on software architectures [Bass et al. 2003] as an abstraction level that forms the connecting link between the aged and the new form of a software system.

The reverse engineering activities are arranged along the left side of an upright standing horseshoe. While ascending this side of the horseshoe, deeper knowledge about the software system is recovered and software artifacts are transferred and combined to more abstract representations revealing novel insights. As mentioned before, the topmost abstraction level is represented by means of a software architecture. The horseshoe

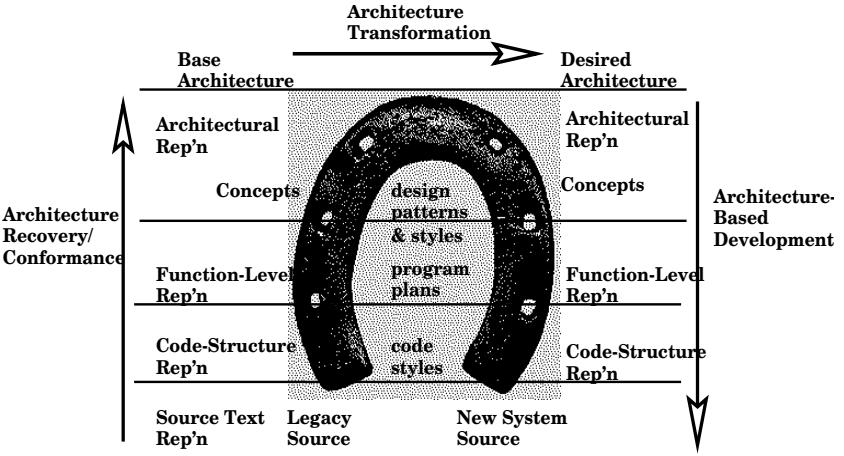


Figure 3.2. The horseshoe model of reengineering (from [Kazman et al. 1998])

model comprises the transformation to a new target architecture through illustrating the transition to the right side of the horseshoe. The subsequent implementation is then accomplished by descending the horseshoe's right side and detailing the abstract specification of the target architecture. Corresponding to the general reenginering model, the more detailed artifacts are produced through applying forward engineering techniques.

3.1.3 Reverse Engineering

Carrying out pervasive modifications to complex software systems in reengineering efforts ought to start with the examination of existing system artifacts such as source code, runtime log information, or code repository meta-data. Referring to the conceptual foundation of software reengineering and its corresponding principles that were presented in Section 3.1.1, those activities constitute abstraction steps to reveal hidden dependencies or to reconstruct a software architecture that was specified once but that eroded over time, for instance.

When reengineering a system, the gained knowledge forms a starting point to plan the subsequent transformation process. For example, a target architecture and a corresponding mapping often have to be defined that restructure the system and eliminate unintended dependencies that were discovered in the reverse engineering step. In the context of this thesis, we use the most common definition of reverse engineering from Chikofsky and Cross [1990].

Definition: Reverse Engineering (Chikofsky and Cross [1990])

Reverse engineering is the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction.

Considering the horseshoe model of reengineering that was described in Section 3.1.2 and that is illustrated in Figure 3.2, the reverse engineering activities, like distilling more abstract representations that unveil inherent structures after the initial development of a system, are arranged along the left side of the upright standing horseshoe.

The definition of reverse engineering from Chikofsky and Cross [1990] that is mentioned above includes as an essential part the analysis of software systems. In reviewing the field of software reverse engineering and listing research challenges, Müller et al. [2000] divide this field in two fundamental categories that correspond to the subject of an analysis. The authors distinguish the *reverse engineering of code* from the *reverse engineering of data*. The first category, reverse engineering of code, comprises techniques to extract knowledge from system artifacts that address the processing of data or that provide a more abstract view on those artifacts. In contrast to this, reverse engineering of data is concerned with understanding the data of information systems itself, as well as the underlying data schemata and data documentation.

Canfora et al. [2011] give an overview on the field of reverse engineering as well. In this context, the authors propose a conceptualization of the

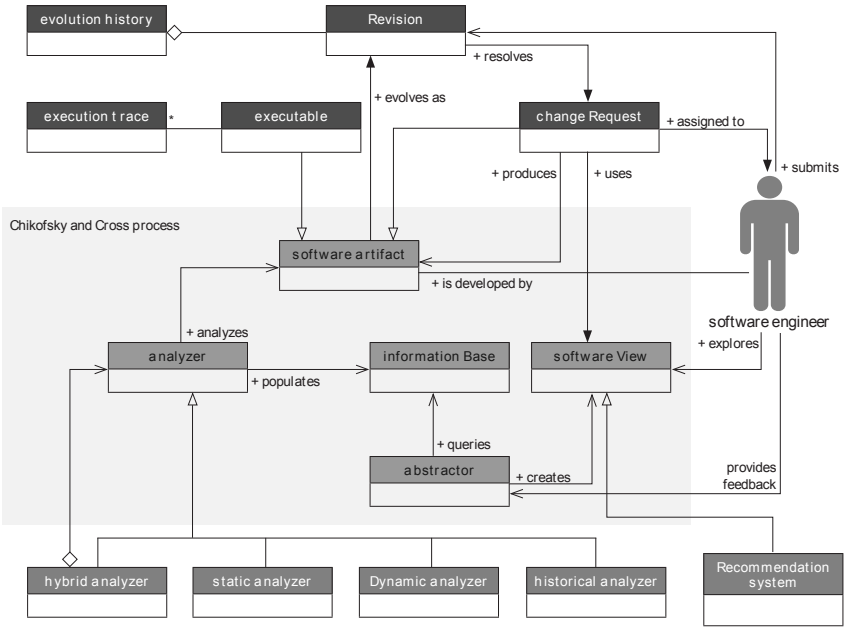


Figure 3.3. Reverse engineering concepts (based on [Canfora et al. 2011])

fundamental terms related to reverse engineering that is illustrated in Figure 3.3. The depicted UML class diagram from Canfora et al. [2011] uses and extends basic concepts already described by Chikofsky and Cross [1990], such as the utilization of *analyzer* components that inspect *software artifacts* and the notion of *information bases* that store the information that is extracted from these artifacts. For example, the information can be represented in the information base using relations or graphs as demonstrated by Chen et al. [1990] and Ebert et al. [2008], respectively. Furthermore, so-called *abstractor* components are used to query the information base and to create corresponding *views* on the system. This process does not incorporate any modification of the system under analysis. Chikofsky and Cross [1990] highlight this important characteristic of reverse engineering and state that "*reverse engineering in and of itself does not involve changing the subject*

system. It is a process of examination, not change or replication." As opposed to this, forward engineering modifies a system and adds new features or improves the product's quality attributes, for instance. Thus, the order of implementing a system and subsequently building abstractions is inverted (cf. Section 3.1.1).

However, software reverse engineering techniques can be of great value in forward engineering projects as well. For example, architecture consistency checking and the detection of code clones can be worthwhile activities when developing a software system [Canfora and Di Penta 2007; Canfora et al. 2011]. Regarding the different software views that can be created by reverse engineering abstractor components that are shown in Figure 3.3, the views mainly differ in three dimensions: (1) The subjects of the view, i.e., the system artifacts of interest. (2) The abstraction level that is used to present consolidated or inferred information. (3) The chosen type of representation. Visualizations of certain aspects of a software system can be seen as a special type of representation.

A considerable body of work exists in this area. For example, approaches and tools to view those aspects that are recovered with the use of reverse engineering techniques and that employ two-dimensional graphics are described by Brade et al. [1992]; Consens et al. [1992]; Eick et al. [1992]; Price et al. [1993]; Storey and Müller [1995], and Lanza and Ducasse [2003]. Using a graph representation is widespread. Software entities are depicted as nodes and the edges show associations between those entities. Through varying, for example, the width, the height, or the color code of the nodes and assigning appropriate metrics, the graphs can then be enriched with additional semantics.

Furthermore, three-dimensional views are also used, for example by Maletic et al. [2003] and Wettel and Lanza [2007]. Other approaches place special emphasis on illustrating the evolution of software systems. For example, changes to the systems that happen over time, such as the addition of modules or a shift of the modules' cyclomatic complexities, are tracked and visualized by Holt and Pak [1996] and Lanza [2001]. An overview of the field of software evolution visualization is given by Voinea [2007].

Considering the analyzer components that are shown in Figure 3.3, there exist *historical analyzers* that examine evolution repositories, as well as general *static* and *dynamic analyzers* that extract information using static (e.g., see [Ferenc et al. 2002; Tonella 2005; Binkley 2007; Melski et al. 2009]) and dynamic analyses (e.g., see [Systä 2000; Ernst et al. 2001; Stroulia and Systä 2002; Cornelissen et al. 2009]). Static analyses do not require information from the execution of a software system, whereas dynamic analyses require those type of information, i.e., *execution traces* [Canfora et al. 2011]. *Hybrid analyzers* constitute a combination of both static and dynamic analyzers.

3.2 Migration

This section describes the field of software migration. It is organized as follows. Section 3.2.1 gives an overview on software migration. Common migration approaches are described in Section 3.2.2. Software migration can be seen as a specific activity in the context of software maintenance. The ISO/IEC 14764 Standard defines a software maintenance process and also considers software migration as a particular activity. The ISO/IEC 14764 Standard along with the embedded software migration activity is described in Section 3.2.3.

3.2.1 Overview

Software migration [Brodie and Stonebraker 1993; Bisbal et al. 1997; Richardson et al. 1997; Bisbal et al. 1999a; Bergey et al. 2001; Wu et al. 2005] is a subdiscipline of software reengineering (see Section 3.1). It therefore includes activities that comprise reverse and forward engineering techniques to evolve and transform an existing software system. Software migration projects aim to move a legacy software system to a new target platform or, more generally, to a new environment. A corresponding definition is given by Winter and Ziemann [2007]:

Definition: Software Migration (Winter and Ziemann [2007])

Software migration is viewed as a transformation of software systems into a new environment without changing its functionality.

Considering CloudMIG, the *new environment* that constitutes a target for the transformation of an existing software system comprises platform and infrastructure cloud services. Furthermore, CloudMIG facilitates reasoning about suitable compositions of a target architecture but does not incorporate changes to a system's functionality. Richardson et al. [1997] remark that software migration projects may include comprehensive restructuring and adaptation activities, but also underline that the business feature set should be kept stable during the migration. Hence, evaluating the functional equivalence and analyzing the incorporated risks is simplified.

In contrast to migration projects, other reengineering projects often solely improve internal quality attributes such as the availability or the maintainability by restructuring the software system's composition. However, software migration shares, of course, some commonalities with general reengineering. For example, if a system has lost evolvability a migration is often considered as part of general reengineering measures to benefit from new technologies. In contrast to this, the need to migrate a system may be a direct result of an abandoned support of underlying technologies, for instance.

Referring to those difficulties, Bisbal et al. [1997] highlight the main stimuli of legacy information system migration and state that "*[...] many organisations now wish to move their legacy systems to new environments which allow information systems to be easily maintained and adapted to new business requirements but retain functionality of existing information systems without having to completely redevelop them.*" However, software migration is, as any reengineering discipline, error-prone and complex. To cope with the complexity of legacy system migration, Wu et al. [2005] propose to apply a combination of techniques such as dynamic program analysis, software visualization, and knowledge recovery. A relevant factor when reasoning about the complexity of a software migration project is the type of the migration.

Migration Types A way to categorize software migration projects is by taking into account the type of the targeted new environment. Following such a categorization, examples for common migration types are listed below.

- ▻ Migration to a new framework, fundamental library, or middleware
- ▻ Migration to a new operating system
- ▻ Migration to a new development tool
- ▻ Migration to a new database management system
- ▻ Migration to a new hardware platform
- ▻ Migration to a new hosting provider
- ▻ Migration to a new programming language
- ▻ Migration to a new programming paradigm

As described by Wu et al. [2005], there exist common migration issues that accompany any legacy system migration project, for example, the necessity to schedule testing activities. Moreover, each migration type exhibits characteristic challenges that need to be addressed. For instance, considering the migration to new programming paradigms, various combinations have been examined, such as

- ▻ Migration of legacy systems to object-oriented platforms (e.g., see [De Lucia et al. 1997; Zou and Kontogiannis 2002])
- ▻ Migration of object-oriented to component-based systems (e.g., see [Lee et al. 2003])
- ▻ Migration of legacy systems to Service-Oriented Architectures (SOA) (e.g., see [Canfora et al. 2008; Fuhr et al. 2010])

Most migration projects incorporate more than one migration type. The decision to follow one migration type can entail the need to apply activities of other migration types. A migration project may therefore be heterogeneous and entail overlapping inherent migration types. For example, a migration from a mainframe architecture to commodity servers most often implies the usage of a different operating system. Considering CloudMIG and the migration types listed above, a migration to platform and infrastructure cloud services is supported as long as the involved set of migration types does not include the migration to other programming languages (cf. Section 7.1).

Software migration can be further categorized by the extent of system adaptation that has to be conducted. Here, possible metrics can include the number of changed classes, altered lines of code, or introduced adapter components, for instance. To alter the legacy software system, the source code has to be under one's own control or it must be legally changeable. Third-party components call for a special treatment. For example, a licensing agreement may have to be adjusted to utilize a component within the new environment. Furthermore, it is frequently seen that moving to a new environment often includes the migration to alternative database management systems (cf. the migration types above) and therefore the migration of the data has to be considered. The migration can span different database models. Fahrner and Vossen [1995] and Jahnke et al. [1996] consider the migration from relational to object-oriented databases, for instance. From CloudMIG's point of view, a specific database model constitutes a restriction and a mismatch with provided cloud opportunities has to be reported.

3.2.2 Migration Approaches

There exist several cut-over strategies to hand over the reengineered software system to the production environment. According to Bisbal et al. [1999b], these cut-over strategies include the *cut-and-run*, *phased interoperability*, and *parallel operations* strategy. Compared with the cut-and-run strategy, which turns off the old and starts the new system, the phased interoperability strategy hands over the new system in incremental steps. The

parallel operations strategy runs the legacy system and the reengineered system, that is equipped with the complete feature set, side by side until the exclusive operation of the new system is considered to be reliable. During the parallel operation the new system is evaluated continuously.

To structure the actual transformation activities there exist a number of *migration approaches*. For example, the *Chicken Little* approach described by Brodie and Stonebraker [1993] uses gateway components that are placed between components of an existing system. The gateways isolate system parts and enable to migrate the system in a stepwise fashion. Components of the existing system are therefore wrapped and as changes to these components are transparent to unmodified dependent components through the use of these gateways, the migration can be carried out iteratively. Figure 3.4 depicts the gateway types and their placements following the *Chicken Little* approach. Three gateway types are considered that describe distinct scenarios corresponding to different degrees of decomposability of the legacy information system (IS). These scenarios are briefly summarized in the following.

- **Non-Decomposable Legacy IS:** This scenario is depicted in the left part of Figure 3.4 and is considered for monolithic legacy IS that cannot be separated in different components. The user interfaces (UIs), system level interfaces (SIs), and the application logic are tightly coupled with a database management component. Hence, the *IS gateway* has to wrap the complete system and must provide the previous interfaces to end users and other ISs.

- **Semi-Decomposable Legacy IS:** System level interfaces (SIs) and UIs can be separated in this scenario that uses an *application gateway* and that is illustrated in the middle portion of Figure 3.4. Therefore, the application gateway can wrap changes to the application logic and database management system.

- **Decomposable Legacy IS:** This scenario is depicted in the right part of Figure 3.4 and considers a system that is ideally suited for migration as, besides the SIs and UIs, the application logic can be separated and

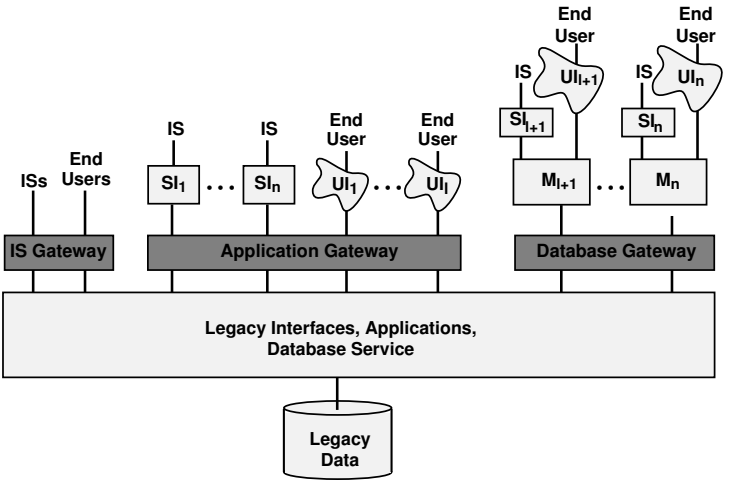


Figure 3.4. Chicken little migration approach - gateway types and placements (from [Brodie and Stonebraker 1993])

modularized in application modules (MIs). Thus, the *database gateway* wraps modifications to the data layer.

Following the *Chicken Little* approach, parts of the old and the new system are operated in parallel and can interoperate during the incremental migration. The general gateway types mentioned above (IS gateway, application gateway, and database gateway) are constructed using several further components. *Forward Gateways* are utilized to allow unmodified components to access modified components. The opposite direction is facilitated through the use of *Reverse Gateways*. An example from Brodie and Stonebraker [1993] that considers the Semi-Decomposable Legacy IS scenario from above is illustrated in Figure 3.5. SIs and UIs of the old system (depicted using light gray) can be separated and access the new system (dark gray) through a *Forward Gateway*. Furthermore, a *coordinator* component manages the linkage and conversion using mapping information that is stored in a *mapping table* and enables the new system components to access the legacy application

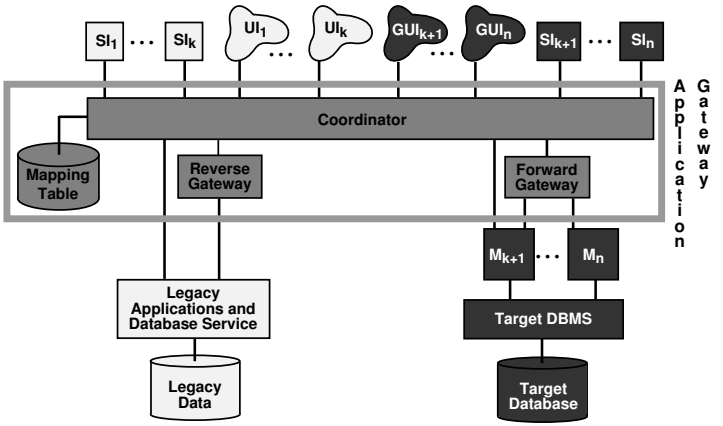


Figure 3.5. Chicken little migration approach - semi-decomposable legacy IS migration architecture. Components of the old system are colored light gray, those of the new system are colored dark gray (from [Brodie and Stonebraker 1993]).

logic and data through a *Reverse Gateway*. Operating parts of the old and the new system in parallel facilitates the incremental substitution of legacy components.

Another approach that addresses the parallel operation of new and unmodified system parts during a migration is the *Dublo* (dual business logic) architectural pattern. The core concept of the original version of *Dublo* that is presented by Hasselbring et al. [2004] is to enable a smooth migration by implementing new business logic in a new architectural middle tier such as an application server. The legacy code is accessed through an adapter. In terms of the *Chicken Little* approach, the adapter constitutes a *Reverse Gateway*. The legacy database is accessed through existing business logic in the legacy code. Two further versions of *Dublo* are added by Hasselbring et al. [2008]. The first version enables the new business logic to access the legacy database directly, the second addresses the construction of a completely new infrastructure via the usage of a new database.

The gateway-free migration approach *Butterfly* is presented by Wu et al. [1997]. It aims to reduce the complexity of gateway-based approaches that is implied by the concurrent access of legacy components and components of the new system. The *Butterfly* approach extracts legacy data samples. A new data schema is constructed and the legacy data samples are being transformed. A newly build system architecture is then implemented on the basis of the transformed data samples using the new data schema. Data modifications of the production system that occur in the meantime are then incrementally integrated by transforming the deltas.

However, if reengineering and migrating an existing system ultimately turns out to be an unfeasible or too costly option, a system might have to be rebuilt from scratch following the *Cold Turkey* strategy as described by Brodie and Stonebraker [1993]. As remarked by Richardson et al. [1997], this strategy is often referred to as the *Big Bang* approach as well.

3.2.3 ISO/IEC 14764 Standard

The ISO/IEC 14764:2006 Standard [ISO/IEC/IEEE 2006] (denoted as ISO/IEC 14764 in the following, omitting edition number 2006 for brevity) describes a software maintenance process and common involved activities and tasks. An overview of this maintenance process is shown in Figure 3.6.

The *process implementation* activity determines the plans and actions that are executed during the maintenance process. If a problem occurs or a modification request demands a maintenance action, the *problem and modification analysis* activity verifies if a modification is necessary and feasible. If this is the case, the modification is executed during the *modification implementation* activity. The *maintenance review/acceptance* activity checks whether the modification was performed correctly. If a software product reaches its end of life and it is closed down (cf. Figure 3.1), it is retired with the help of the *retirement* activity.

Considering our cloud migration approach CloudMIG, the further *migration* activity (cf. Figure 3.6) is of particular interest. It covers the general process

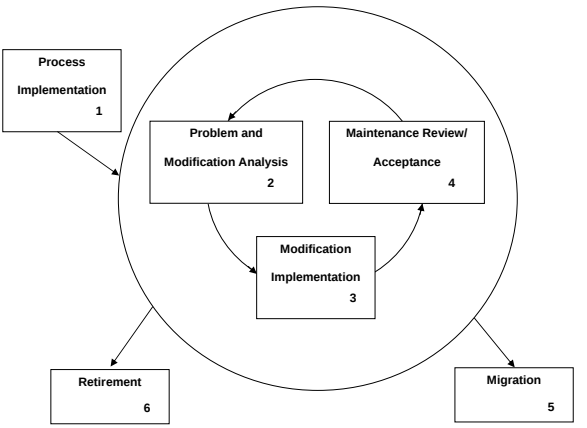


Figure 3.6. ISO/IEC 14764 maintenance process (from [ISO/IEC/IEEE 2006])

for migrating a software system to a new environment and describes the structure of a corresponding *migration plan*. Such a migration plan consists of several coarse-grained items that are illustrated in Figure 3.7. The items cover essential parts of full migration projects, such as planning, executing, and verifying a migration (items *requirements analysis and definition of migration*, *migration execution*, and *migration verification*, respectively). The *migration* activity of ISO/IEC 14764 also defines several task-steps that further detail the items of a migration plan. As several of these task-steps are also relevant for CloudMIG and are revisited in later chapters, they are listed below.

- Analyze the migration requirements
- Determine the impact of migrating the software product
- Establish a schedule for performing the migration
- Identify data collection requirements for post-operation review
- Define and document the migration effort
- Determine and mitigate risks

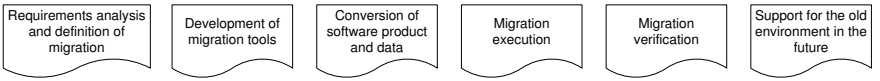


Figure 3.7. Items in a migration plan (based on [ISO/IEC/IEEE 2006])

- Identify needed migration tools
- Identify support for the old environment
- Develop and/or acquire migration tools
- Incrementally decompose software products and data for conversion
- Prioritize conversion of software products and data
- Convert software products and data
- Migrate software products and data to new environment
- Run parallel operations
- Verify migration through testing
- Provide support for old environment

3.3 Architecture-Driven Modernization

This section describes the Architecture-Driven Modernization (ADM) initiative[1] from the Object Management Group (OMG) that aims to provide standards for supporting the modernization of existing software systems. The section is structured as follows. Section 3.3.1 gives an overview on the ADM initiative and the specifications that are being developed under its umbrella. As CloudMIG utilizes the two particular ADM standards Knowledge Discovery Meta-Model (KDM) and Structured Metrics Metamodel (SMM), KDM and SMM are described separately in the Sections 3.3.2 and 3.3.3, respectively.

[1] http://adm.omg.org/

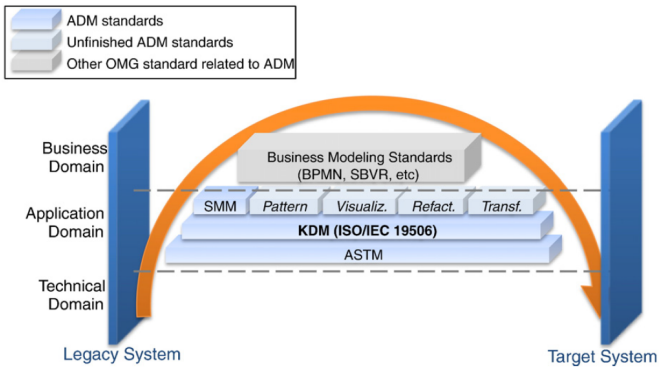


Figure 3.8. OMG ADM standards (from [Pérez-Castillo et al. 2011])

3.3.1 Overview

OMG's ADM initiative (also called ADM task force) develops standards for supporting the modernization of existing software systems. Ulrich [2004] lists several exemplary scenarios that form potential use cases for employing those standards. For example, those scenarios include tasks for the analysis of business processes, recovery of software architectures, extraction of data definitions, and transformation of source code.

In general, ADM follows the concept of the horseshoe model of reengineering (cf. Section 3.1.2) as it comprises reverse engineering, transformation, and forward engineering steps for modernizing software. This is also illustrated in Figure 3.8 from Pérez-Castillo et al. [2011] that depicts several of the OMG ADM standards and links them to the horseshoe model. The transformation of system artifacts can be pursued according to several abstraction levels. On the lowest level of abstraction, the Abstract Syntax Tree Metamodel (ASTM) [Object Management Group 2011b] constitutes a specification for modeling low-level Abstract Syntax Trees (ASTs). This abstraction level provides a fine-grained source code representation and is therefore suited for source code transformations, for instance. KDM models software artifacts at a higher abstraction level. It is described in greater detail in Section 3.3.2.

Likewise, the Structured Metrics Metamodel (SMM) is covered separately in Section 3.3.3. Several other standards are available in draft status or as white papers only, while others lack a public release at all. Those further envisioned OMG ADM standards include the Software Patterns Analysis Package, Visualization Package, Refactoring Package, and Transformation Package. The standards provided by OMG ADM also aim to provide the foundations for extracting and processing instances of established, higher-level OMG meta-models, such as the Business Process Model and Notation (BPMN) [Grosskopf et al. 2009], for instance.

3.3.2 Knowledge Discovery Meta-Model

The Knowledge Discovery Meta-Model (KDM) [Object Management Group 2011a] aims at modeling software systems on various abstraction levels. However, KDM is not intended to model source code below the sub-statement level, as this is the purpose of ASTM. KDM was adopted as standard ISO/IEC 19506 by the International Standards Organization. A core goal of KDM is representing software systems in a way that is independent of the specific technologies and programming languages used to build the systems. KDM models that are serialized to XMI can also function as an interchange format between arbitrary software modernization tools. Though, those tools have to conform to KDM's compliance levels.

An overview of KDM is given in Figure 3.9 from Pérez-Castillo et al. [2011]. KDM is structured into four layers that consist of several packages. These layers and their packages are briefly described below.

- **Infrastructure Layer:** This layer comprises the *Core*, *KDM*, and *Source* packages.

  The *Core* package contains meta-model elements that build the foundations for all other packages, i.e., elements from other packages derive from meta-model elements of the *Core* package.

  The *KDM* package provides elements for assembling concrete KDM instances. For example, `KDMModels` can be consolidated in a `Segment` con-

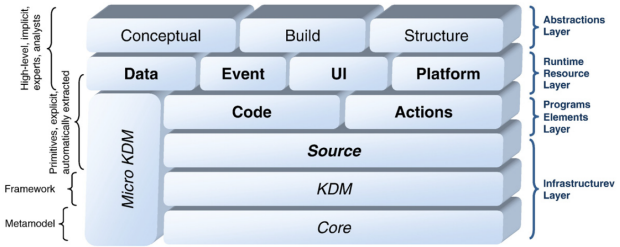


Figure 3.9. KDM overview (from [Pérez-Castillo et al. 2011])

tainer element. The KDM package also provides means for KDM's so-called *light-weight extension mechanism* that is used by CloudMIG. From a high-level view, this mechanism enables to enhance the semantics of KDM models. Instead of introducing new meta-model elements that would, for instance, break the compatibility among KDM-based tools, it is possible to augment KDM models with elements from the *KDM* package. This mechanism is further detailed in Section 6.3.4.

The *Source* package includes elements for representing physical artifacts, such as images, source code files, configuration files, and directories. Those elements that derive from the classes `InventoryItem` or `InventoryContainer` are grouped in an `InventoryModel` element.

▻ **Program Elements Layer:** This layer comprises the *Code* and *Action* packages.

The *Code* package provides means for representing program elements on an implementation level. A basic example from Wulf et al. [2012] is shown in Figure 3.10. The C# class called *Example* is represented by a KDM `ClassUnit` element. The member *degree* and method *test* are modeled by a `MemberUnit` and `MethodUnit` element, respectively.

The *Action* package includes meta-model elements that describe the behavior of program elements, for example, associations and control flows. The *Action* package describes those kinds of behavior on a rather high level of abstraction. A more fined-grained level of abstraction can be expressed with the help of the so-called *Micro KDM* (cf. Figure 3.9). *Micro*

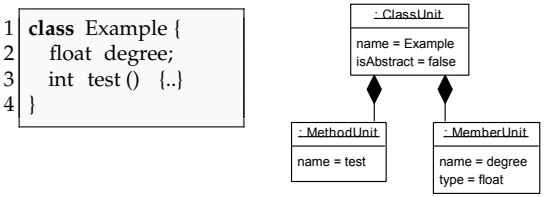


Figure 3.10. A C# example and a corresponding, simplified KDM instance (based on [Wulf et al. 2012])

KDM constitutes a compliance level and provides means for specifying precise semantics regarding a program's behavior.

- **Runtime Resource Layer:** This layer comprises the *Data*, *Event*, *UI*, and *Platform* packages.

  The *Data* package provides elements for representing the organization of data, such as a relational schema.

  The *Event* package includes elements that describe high-level behavior. A focus is on event-driven state transitions.

  The *UI* package includes elements for describing user interfaces (UIs), such as elements for screens and layouts.

  The *Platform* package provides elements for describing the runtime operating environment of a software system, for example, its resources, streams, sockets, operating systems, or application servers.

- **Abstractions Layer:** This layer comprises the *Conceptual*, *Build*, and *Structure* packages.

  The *Conceptual* package includes elements that enable building a conceptual model of a software system. Such a conceptual model may comprise domain terms and business rules that are implemented in an application, for instance.

  The *Build* package includes elements for describing the build process of a software system, such as elements for representing the generated artifacts and workflow of a build process.

> The *Structure* package provides elements for describing high-level, architectural components of a software system, such as layers and subsystems.

A tool that enables the extraction of KDM models from software systems with so-called *discoverers* is MoDisco [Bruneliere et al. 2010], for instance. Our tool *CloudMIG Xpress* builds on MoDisco (cf. Chapter 9).

3.3.3 Structured Metrics Metamodel

The Structured Metrics Metamodel (SMM) [Object Management Group 2012] is a meta-model for describing measures, such as McCabe's cyclomatic complexity [Kan 2002], measurement processes, and measurement results. In the context of SMM, the terms *measure* and *metric* are used as synonyms. SMM enables to apply measures on the basis of arbitrary MOF-based metamodels [Object Management Group 2011c]. In combination with KDM (cf. Section 3.3.2), this enables to define a measure only once but compute the measure for every software system that employs technologies from which KDM models can be extracted, i.e., where corresponding discoverers are available (cf. Section 3.3.2).

Core classes of SMM are shown in Figure 3.11. An `SmmModel` contains several `MeasureLibrary` elements that are used for grouping measures. A `Measure` is applied to model elements that are determined through a `Scope` element. The `Scope` element itself can use an `Operation` for defining the exact set of elements that should be used for applying the measure. `Measure`, `Scope`, and `Operation` are all `AbstractMeasureElements`. Examples for specific measures are the `DimensionalMeasure` and the `Ranking` element. The former assigns a numeric value, the latter determines if a measured element complies to a specific class, for example, to a *high complexity* or *normal complexity* class.

Applying `Measures` to a model results in a set of `Measurements`. Those `Measurements` represent the results of a measurement process. Exemplary, specific results that correspond to the aforementioned `DimensionalMeasure` and `Ranking` measures are the `DimensionalMeasurement` and `Grade` measurements, respectively. They result through applying a `Measure` to an element

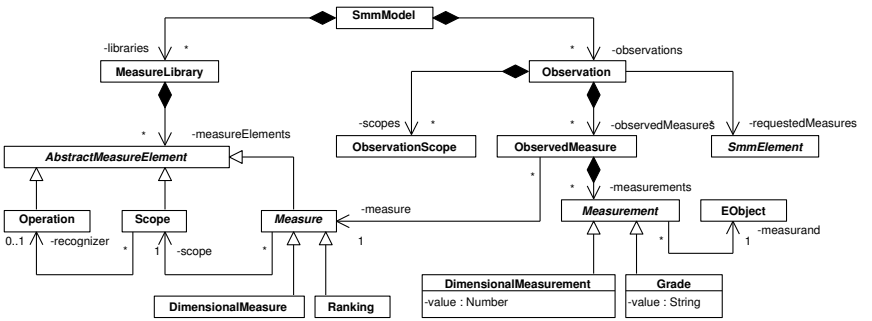


Figure 3.11. Core classes of the SMM meta-model (using the `EObject` class for measurands instead of a `MofElement`) (from [Frey et al. 2011])

that is referenced via the `measurand` relationship. For grouping `Measurements`, so-called `Observations` are used. They reference the `Measures` that could actually be computed via `ObservedMeasure` elements. The scope that was actually used for calculating the measures is determined through `ObservationScopes`.

An adapted example from Object Management Group [2012] that shows a simple SMM measure is depicted in Figure 3.12. The example counts the KDM modules, i.e., the instances of KDM's class `Module`, that are found in a specific KDM `CodeModel`. The `CodeModel` element is a container for elements from KDM's *Code* package (cf. Section 3.3.2). The `CollectiveMeasure` depicted in Figure 3.12 is a specific `Measure` that is used for aggregating the `Measurements` of a further `Measure` that is called the *base measure*. In this example, the base measure is a `Counting` measure that returns "1" for each KDM `Module` it discovers. Hence, as the `CollectiveMeasure` has the accumulator set to *sum*, it adds all results from `Counting` that are represented by the specific `Count` measurement. Given a particular KDM model, the SMM instance is applied to a KDM `CodeModel` that represents the *measurand*. Hence, the SMM instance delivers the number of all KDM `Modules` that are included in the `CodeModel` (1 in this example).

However, it should be noted that the OMG SMM does only constitute a specification for modeling concepts from the measurement domain. For

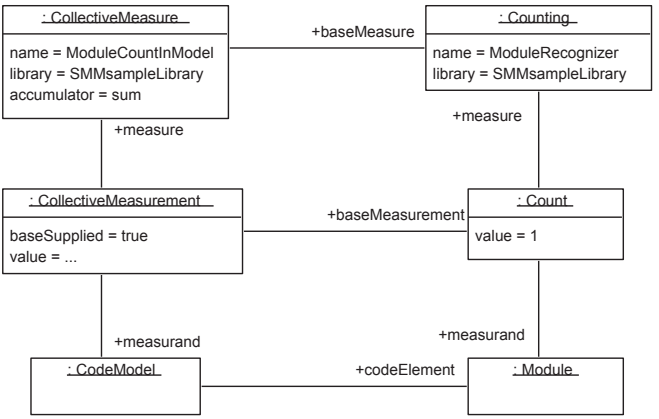


Figure 3.12. SMM module count example (based on [Object Management Group 2012])

actually computing measures that are modeled with SMM, further tool support is needed. At the time of writing, corresponding tool support is very limited. Hence, we contribute our framework *MAMBA (Measurement Architecture for Model-Based Analysis)* [Frey et al. 2011] that is described in Section 10.2 and that allows, among others, to apply SMM measures to KDM models that are extracted from existing software systems.

3.4 Summary

This chapter gave an overview on the software modernization field and highlighted the specific areas that are relevant in the context of Cloud-MIG. Software modernization pervasively modifies a software system so it maintains or regains evolvability in a technical or financial sense. Software modernization shares several commonalities with software reengineering, where the concepts of abstraction, alteration, and refinement constitute core principles. Those principles also manifest in the well-known horseshoe

model of reengineering that involves reverse engineering, transformation, and forward engineering steps.

A specific form of software reengineering is software migration that transfers an existing software system to a new environment while leaving its functionality unchanged. The body of work in the software migration field comprises several migration approaches that aim to streamline the migration process. We also described the ISO/IEC 14764 standard that covers a common software maintenance process in general and the (potentially) included software migration activity in particular. The description of this activity also addresses several aspects that are valuable for explaining and classifying CloudMIG in later chapters.

Finally, this chapter elucidated on the Architecture-Driven Modernization (ADM) initiative from the Object Management Group (OMG). This initiative (often also called *task force*) aims to provide standards for supporting the modernization of software systems. Among the most mature standards that were created under the umbrella of the ADM initiative are the Knowledge Discovery Meta-Model (KDM) and the Structured Metrics Metamodel (SMM). Both standards are also utilized by CloudMIG. KDM was adopted as ISO/IEC 19506 by the International Standards Organization. It provides means for modeling software systems on various levels of abstraction in a way that is independent from a specific programming language. SMM is a meta-model for describing concepts and activities from the domain of measuring software quality attributes, i.e., computing software measures, such as McCabe's cyclomatic complexity. For example, SMM enables to model measures, measurement processes, and measurement results.

Chapter 4

Search-Based Software Engineering

The use of optimization techniques, such as linear programming [Schrijver 1998], gradient descent [Avriel 2003], and meta-heuristics [Luke 2011], is a common means in various engineering disciplines to cope with complex problems and to efficiently find well-suited solutions. For example, meta-heuristic optimization algorithms—that iteratively try to improve potential solutions—are used to optimize water distribution networks [Montesinos et al. 1999], to improve chemical processes [Lin and Miller 2004], and to locate structural damages that may be caused, e.g., by material corrosion [Chou and Ghaboussi 2001]. The field of *Search-Based Software Engineering (SBSE)* aims to exploit the potential of those optimization techniques for problems that emerge in the software engineering domain [Harman and Jones 2001; Clarke et al. 2003; Harman 2007; Harman et al. 2009; Harman 2011; Vergilio et al. 2011; Harman et al. 2012b]. In the context of this thesis, we use the following definition of SBSE.

Definition: Search-Based Software Engineering (based on Harman [2011])

The field of *Search-Based Software Engineering (SBSE)* addresses the application of search-based optimization techniques to problems in software engineering.

Many software engineering problems are complex and incorporate manual intervention. Hence, finding adequate solutions is often time-consuming

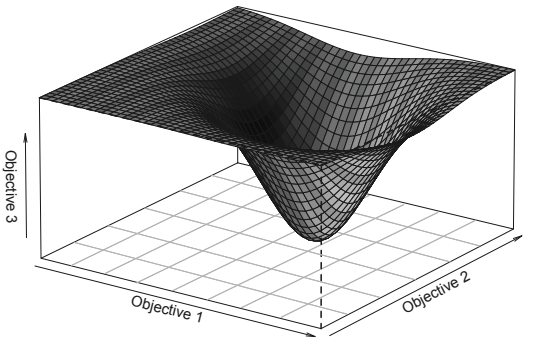


Figure 4.1. Exemplary multi-objective search space. Three objectives correspond to three dimensions in this example.

and costly. SBSE can help to effectively approach common software engineering problems. For example, a wide range of works demonstrated that SBSE techniques can be successfully applied to problems from the software testing, refactoring, design, maintenance, verification, and requirements engineering areas [Harman et al. 2009].

The involved search spaces often involve multiple competing, often conflicting objectives and are therefore explored with multi-objective optimization algorithms [Harman et al. 2012b]. Figure 4.1 illustrates such an exemplary multi-objective search space that consists of three objectives.

This chapter describes the Search-Based Software Engineering (SBSE) field and is structured as follows. Section 4.1 provides an overview on SBSE. Genetic algorithms constitute a particular optimization technique that is most-frequently applied in SBSE [Harman 2011] and that is also used by CloudMIG (cf. Chapter 8). Genetic algorithms are described in Section 4.2. As CloudMIG uses a genetic algorithm in combination with a discrete event simulator (cf. Chapter 8), our corresponding optimization of deployment and reconfiguration of cloud-based applications falls into the category of simulation-based optimization. Simulation-based optimization is described in Section 4.3. Finally, Section 4.4 sums up this chapter.

4.1 Overview

Many of the problems prevalent in software engineering can be traced back to optimization problems [Harman et al. 2012b]. For example, a software architect who designs a new system actually strives to optimize quality attributes when developing the architecture, e.g., regarding maintainability, security, and availability. Those quality properties determine the *fitness* of a candidate solution, i.e., the degree it meets the given requirements. Hence, following an SBSE-based approach would enable to explore the search space of all potential software architectures and evaluate elements of that search space (e.g., see Räihä [2010] for a survey of search-based approaches that optimize software design and architecture). As remarked by Clarke et al. [2003], it is often easier to evaluate the quality of software processes, their activities, and artifacts such as software architectures instead of actually engineering well-suited solutions. Thus, SBSE leverages this observation and emphasizes the means for (1) efficiently searching a given solution space and (2) assessing solution candidates in order to actually engineer capable solutions.

As noted by Harman et al. [2009], the term "search" in Search-Based Software Engineering should not be confused with textual or hypertextual searching. In the context of SBSE, the term is used to describe the application of optimization techniques to problems from the software engineering domain in the sense that *optimal* or *near-optimal solutions* (cf. Section 4.1.3) are searched. However, because of the vast number of choices and degrees of freedom that are often involved, the search spaces are frequently huge and it is hard to actually find the needle(s) in the haystack, i.e., the well-suited solution(s). According to Harman [2007], SBSE has become popular because there exist only the following two key aspects for applying optimization techniques to software engineering problems.

1. The choice of the representation of the problem
2. The definition of the fitness function

In general, a fitness function assesses the quality of a candidate solution. Harman and Clark [2004] show that software metrics, such as McCabe's

cyclomatic complexity [Kan 2002], can be used as fitness functions for evaluating candidate solutions, for instance.

This section provides an overview on the foundations of SBSE. It is structured as follows. We describe classic optimization techniques with a focus on those that have been employed in an SBSE context in Section 4.1.1. Then, Section 4.1.2 elaborates on meta-heuristic optimization techniques, before Section 4.1.3 describes important fundamentals regarding multi-objective optimization.

4.1.1 Classic Optimization Techniques

This section briefly describes classic optimization techniques that also were applied to software engineering problems. For example, Heričko et al. [2008] propose an approach that uses gradient-based optimization for finding well-suited software development team sizes. A classic mathematical method for finding the minimum of a function is *gradient descent*. If the goal is to maximize a function, the method is called *gradient ascent*.

Gradient ascent The slope of a mathematical function is used to find the maximum of the function. Luke [2011] describes a basic form of gradient ascent with the help of a simple function $f(x)$. $f'(x)$ has to be known. Gradient ascent starts with an arbitrary value of x. Then, x is repeatedly altered in the form $x \leftarrow x + \alpha f'(x)$. α is a small positive value. Figure 4.2 from Luke [2011] illustrates an example where the slope is negative and x decreases. Hence, this procedure will ultimately reach the peak where the slope is zero. In the simplest form of gradient ascent that does not consider local optima, the procedure then stops. For multi-dimensional functions, x is replaced with the vector $\vec{x}$ and the slope is replaced with the *gradient* of $\vec{x}, \bigtriangledown f(\vec{x})$. As noted by Luke [2011], "the gradient is simply a vector where each element is the slope of $\vec{x}$ in that dimension, that is, $\langle \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n} \rangle$."

Harman [2007] states *linear programming* and *branch and bound* as further techniques that were applied to software engineering problems.

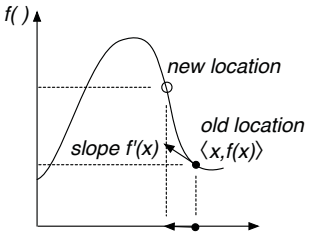


Figure 4.2. Gradient ascent with a negative slope. x is decreasing (from [Luke 2011]).

Linear programming Linear programming has been used in the requirements engineering domain for problems regarding release planning [Carlshamre 2002; Ruhe and Saliu 2005], for instance. Harman [2007] describes linear programming as follows.

Definition: Linear programming (Harman [2007])

Linear programming (LP) is a mathematical optimization technique that is guaranteed to locate the global optimum solution. The inputs to a linear programming model are a set $\{x_1, ..., x_n\}$ of n real, non-negative values, called the decision variables. The goal is to maximize the value of some linear expression in these decision variables subject to a set of constraints, expressed as linear equations in the decision variables. That is

$$\text{Maximize} \sum_{i=1}^{n} c_i x_i$$

Where $\{c_1, ..., c_n\}$ is a set of problem-specific coefficients, subject to a set of m constraints of the form

$$\sum_{i=1}^{n} a_{1i} x_i \leqslant b_1$$
$$\vdots$$
$$\sum_{i=1}^{n} a_{mi} x_i \leqslant b_m$$

Where a_{ij} and b_i are problem determined constants. The constraints can also be expressed using $\geqslant$ and $=$ in place of $\leqslant$ and the goal can be minimization rather than maximization.

Branch and bound A further classic optimization technique is branch and bound [Harman 2007]. To find an optimal solution, branch and bound follows an iterative approach that divides a set of solutions in a specific number of subsets according to a particular strategy. Then, it computes for each subset a lower bound and compares it with a known upper bound (or vice versa). If a lower bound exceeds the upper bound (or conversely), the corresponding subset is discarded. Hence, if corresponding strategies for a given problem can be found, this approach enables to quickly dismiss large parts of the search space that do not seem to include promising candidate solutions.

We refer to Rardin [1997]; Brusco and Stahl [2005] for further details regarding the branch and bound algorithm and to Harman [2007]; Burke and Kendall [2010]; Harman et al. [2012b] for further classic optimization techniques.

4.1.2 Meta-heuristic Optimization Techniques

For some classes of problems it is not possible to use classic optimization techniques, e.g., those that were introduced in the previous Section 4.1.1. For example, gradient ascent can only be employed if the first derivation of a function can be computed. However, often not even the actual function that is to be maximized or minimized is known. In the context of those types of problems, it is not possible to create a mathematical model that takes some input parameters for solving a set of functions and producing the output of the model.

For example, often there exist many variables that interact in non-linear and unforeseen ways. There may also exist hidden dependencies and interrelations that are hard to be expressed accurately with mathematical functions. In those types of situations, meta-heuristic techniques (or just *meta-heuristics*) [Osman and Kelly 1996; Jones et al. 2002; Blum and Roli 2003; Talbi 2009; Luke 2011; Gendreau and Potvin 2012] provide means for optimizing the corresponding problems. They rather focus on the assessment of candidates and approximation of optimal solutions instead of

providing procedures for determining exact solutions (e.g., see [Luke 2011]). When considering hard combinatorial problems, it is often not clear how to build well-suited solutions, but it is possible to evaluate and compare candidate solutions. Nevertheless, a brute force approach that enumerates and tests all candidates is still often inapplicable as the number of potential solutions (i.e., the search space) is just too big.

Meta-heuristics are problem-independent and involve some sort of randomness when iteratively approximating optimal solutions. Hence, finding an optimal solution can usually not be guaranteed. Luke [2011] distinguishes *single-state* and *population-based* meta-heuristic optimization methods. Both are briefly described in the following, we begin with single-state methods.

Single-state methods Single-state meta-heuristic optimization methods keep a single candidate solution per iteration that is tweaked and assessed. According to Luke [2011], it has to be possible to perform the following steps for optimizing such a candidate solution:[1]

- "Provide one or more initial candidate solutions. This is known as the **initialization procedure**.
- Assess the `Quality` of a candidate solution. This is known as the **assessment procedure**.
- Make a `Copy` of a candidate solution.
- `Tweak` a candidate solution, which produces a *randomly slightly different* candidate solution. This, plus the `Copy` operation, are collectively known as the **modification procedure**."

Furthermore, a **selection procedure** often exists that determines which potential solution to keep and which to refuse [Luke 2011]. Examples for those kinds of optimization methods are *hill-climbing*, *iterated local search*, *tabu search*, and *simulated annealing* (e.g., see [Talbi 2009; Gendreau and Potvin 2012]). The latter is explained in Section 12.1.1. In the following, we use hill-climbing as a basic example. Listing 4.1 shows the corresponding algorithm as described by Luke [2011]. The copy of a candidate solution is iteratively

[1] The stated `Quality`, `Copy`, and `Tweak` operations constitute corresponding operators that are used in the following.

```
1 S ← some initial candidate solution          ▷ The Initialization Procedure
2 repeat
3     R ← Tweak(Copy(S))                       ▷ The Modification Procedure
4     if Quality(R) > Quality(S) then          ▷ The Assessment and Selection Procedures
5         S ← R
6 until S is the ideal solution or we have run out of time
7 return S
```

Listing 4.1. Hill-climbing (from [Luke 2011])

modified with the help of some `Tweak` procedure (Line 3). If the quality of the resulting candidate solution is higher, than the initial candidate solution is replaced by the new one (Lines 4 and 5). This procedure is repeated until some stopping criterion is satisfied (Line 6). Then, the candidate with the best quality is considered as the optimal solution.

Population-based methods In contrast to the single-state methods described before, population-based methods maintain several candidate solutions for tweaking and assessment. Most population-based methods use an analogy with concepts from biology. Luke [2011] describes those concepts as follows. *Populations*—i.e., a sample of candidate solutions—are evolved over several iterations, the iterations are then called *generations*. So-called *individuals* are elements of a population. Specific individuals are *childs* (or *offspring*) and *parents*. The procedure of producing offspring from parents is called *breeding*. This concept of *reproduction* plays an important role, as it mimics the process of transmitting some characteristics of individuals to individuals of subsequent generations.

Furthermore, referring to the term *fitness* that was coined in the context of Darwin's theory of evolution [Bowler 2009], the quality of individuals is also denoted as fitness. The data structure of individuals that is used during breeding is called *genotype* or *genome* [Luke 2011]. It consists of single properties (*genes*) that together form a *chromosome*. The data structure of individuals that is used during assessment is called *phenotype*.

Using those concepts, **Evolutionary Algorithms** determine the fitness of each generation's individuals and breed new offspring from selected parents.

In each generation, μ parents get selected and together produce λ childs. The reproduction most often involves the two basic operations *recombination* (also called *crossover*) and *mutation*. Crossover mixes parts of the parents' genotype for sharing and combining properties. The mutation operation alters one or more specific genes for searching the surrounding of an individual independent of the characteristics of other individuals [Deb 2011]. Those concepts again have their analogy in biology, for example, mutation mimics occasional modifications to the genome. Furthermore, crossover and mutation operations are only performed with specific probabilities that are determined by a *crossover rate* and *mutation rate*. Each subsequent generation is built by combining the offspring with the individuals of the existing population. In each generation, the best solution found so far is tracked. After a termination criterion is satisfied, the best solution found at this time is considered the (near-)optimal solution.

4.1.3 Multi-Objective Optimization

A multi-objective optimization problem includes two or more objectives that ought to be optimized. We refer to Zitzler and Thiele [1999] and formalize the notion of a *multi-objective optimization problem* as follows.

Definition: Multi-objective optimization problem (based on Zitzler and Thiele [1999])

A *(general) multi-objective optimization problem* can be described as a vector function f that maps a tuple of p parameters (decision variables) to a tuple of n objectives. Formally:

$$\begin{aligned} &min/max\ y = f(x) = (f_1(x), f_2(x), ..., f_n(x)) \\ &subject\ to\ x = (x_1, x_2, ..., x_p) \in X \\ &\qquad\qquad y = (y_1, y_2, ..., y_n) \in Y \end{aligned}$$

where x is called the *decision vector*, X is the *parameter space*, y is the *objective vector*, and Y is the *objective space*.

f is a vector of *objective functions* that have to be either minimized or maximized. The corresponding *objective functions* f_1 to f_n are also called *objectives*, criteria, payoff functions, cost functions, or value functions [Marler and Arora 2004]. A multi-objective optimization problem is often constrained by m inequality constraints (say $g_j(x) \leqslant 0,\ j = 1, 2, ..., m$) and e equality constraints (say $h_l(x) = 0,\ l = 1, 2, ..., e$) [Marler and Arora 2004]. Using these notions of equality and inequality constraints, we also refer to Marler and Arora [2004] in defining the terms *feasible design space* and *feasible criterion space* as follows.

Definition: Feasible design space (based on Marler and Arora [2004])

Let the functions $g_j(x) \leqslant 0,\ j = 1, 2, ..., m$ and $h_l(x) = 0,\ l = 1, 2, ..., e$ determine the inequality and equality constraints of a multi-objective optimization problem, respectively. The *feasible design space* S (often called the feasible *decision space* or *constraint set*) is defined as the set $\{x | g_j(x) \leqslant 0,\ j = 1, 2, ..., m;\ and\ h_l(x) = 0,\ l = 1, 2, ..., e\}$.

Hence, the feasible design space includes all decision vectors that do not violate given constraints. The *feasible criterion space* is defined as follows.

Definition: Feasible criterion space (based on Marler and Arora [2004])

Let f be a vector of objective functions and S the feasible design space. The *feasible criterion space* Z (also called the feasible *cost space* or the *attainable set*) is defined as the set $\{f(s) | s \in S\}$.

A decision vector d_1 clearly outperforms a decision vector d_2 if each component of d_1's objective vector ($o(d_1)$) is superior to the corresponding component of d_2's objective vector ($o(d_2)$). However, most often no single optimal solution is present that outperforms all other potential solutions in all objectives [Coello Coello 2006]. Thus, existing solutions often constitute trade-off solutions. A further central concept in the context of multi-objective

optimization is the notion of *pareto optimality*. A decision vector is *pareto optimal* if the corresponding objective vector "cannot be improved in any dimension without degradation in another" [Zitzler and Thiele 1999]. The notion of a *pareto-optimal front* (also called a *pareto-optimal set* or *pareto optimum*) is formally defined below on the basis of Zitzler and Thiele [1999]. Please note that this definition is a streamlined version that includes only the parts that are relevant in this thesis.

Definition: Pareto-optimal front (based on Zitzler and Thiele [1999])

Assume, without loss of generality, a maximization problem, a vector f that contains n objective functions, a parameter space X, and consider two decision vectors $a, b \in X$. Then, a is said to *dominate* b (also written as $a \succ b$) iff

$$\forall i \in \{1, 2, ..., n\} : f_i(a) \geqslant f_i(b) \quad \wedge$$
$$\exists j \in \{1, 2, ..., n\} : f_j(a) > f_j(b)$$

All decision vectors which are not dominated by any other decision vector of a given set are called *non-dominated* regarding this set. The decision vectors that are non-dominated within the entire search space are denoted as *pareto optimal* and constitute the so-called *pareto-optimal front* or *pareto-optimal set*.

Figure 4.3 shows an example that uses a two-objective space with given candidate solutions (cf. Figure 4.3 (a)), where bigger values are better for f_1 but worse for f_2. The objective values 3, 5, and 6 constitute the pareto-optimal front (cf. Figure 4.3 (b)).

Employing the concepts of *domination* and *non-domination*, that are introduced in the definition above, is beneficial as they allow to combine several objective functions. Usually, different objective functions (also denoted *fitness functions*) are used for calculating the components of an objective vector. Harman et al. [2012b] notes that the fitness functions and the corresponding fitness values often cannot be easily aggregated into a single fitness function

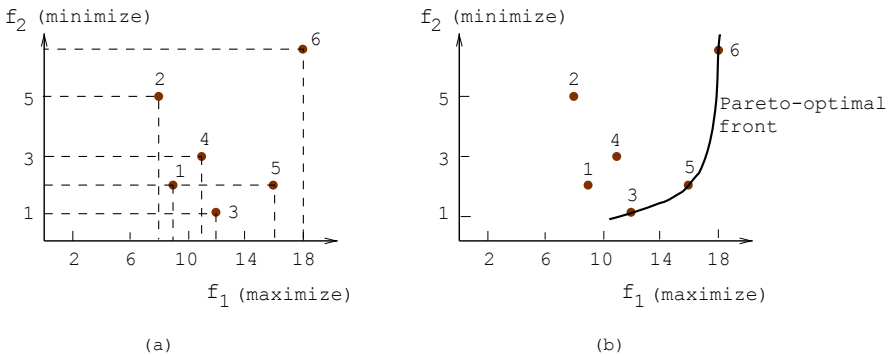


Figure 4.3. (a) A set of points in a two-objective space and (b) the corresponding pareto-optimal front (based on [Deb 2012])

for assessing the overall quality of a decision vector. This is due to the fact that metrics are often employed as fitness functions that are measured on an ordinal scale (cf. [Harman et al. 2012b]).

As the search spaces are often huge, it is frequently not possible to visit and assess each candidate solution. Hence, multi-objective algorithms may often rather find *near-optimal* solutions instead of the pareto-optimal front. In this case, the known non-dominated candidate solutions do not constitute the pareto-optimal front, but rather an approximation that is then simply called *pareto front* [Harman 2007]. Figure 4.4 illustrates the improvement of a pareto front over time. The longer a multi-objective optimization runs, the better its approximation of the pareto-optimal front (also called *true pareto front*) becomes.

The quality of multi-objective optimizers—i.e., the quality of the approximations they produce—can be assessed with specific performance measures [Zitzler et al. 2003]. For example, in Section 12.1.2 we describe the performance measures Inverted Generational Distance (IGD) and Hypervolume Indicator (HV) in detail that are used for evaluating CDOXplorer and for comparing it with other multi-objective optimization approaches.

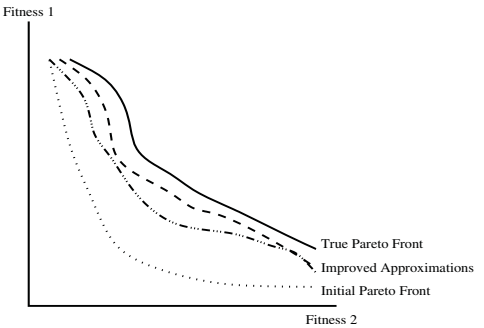


Figure 4.4. Improvement of a pareto front over time (from [Harman 2007])

4.2 Genetic Algorithms

Genetic algorithms [Goldberg 1989; Whitley 1994; Srinivas and Patnaik 1994b; Koza 1995; Mitchell 1998; Haupt and Haupt 2004; Konak et al. 2006] constitute a specific population-based meta-heuristic optimization technique (cf. Section 4.1.2). As mentioned before, genetic algorithms are also the most-frequently applied optimization technique in the context of SBSE [Harman 2011].

As most population-based meta-heuristics, genetic algorithms build on analogies from biology and employ concepts such as evolution, generations, fitness, and genes, and organize the individuals' reproduction with the help of crossover and mutation operations (cf. Section 4.1.2), for instance. Likewise, they involve some sort of randomness in exploring the search space and usually cannot guarantee that an existing global optimum is actually found. For explaining the functional principle of genetic algorithms, often the *schema theorem* from Holland [1975] is used that builds on the sampling of hyperplane partitions [Whitley 1994].

Basic operations A generic genetic algorithm from Harman [2011] is shown in Listing 4.2. After creating and evaluating the initial population (Line 2 and 3), the reproduction of each generation is accomplished with the

```
1  Set generation number, m := 0
2  Choose the initial population, P(0)
3  Evaluate fitness P(0), F(P_i(0))
4  loop
5  Recombine: P(m) := R(P(m))
6  Mutate: P(m) := M(P(m))
7  Evaluate: F(P(m))
8  Select: P(m + 1) := S(P(m))
9  m:=m+1
10 exit when goal or stopping condition is satisfied
11 end loop;
```

Listing 4.2. A generic genetic algorithm (from [Harman 2011])

help of four basic steps (the loop in Lines 4-10). The reproduction procedure includes the application of the `crossover` (`recombine`) and `mutate` operations, as well as the repeated calculation of the individuals' fitness (`evaluate`) and selection of the individuals that constitute the new generation (`select`). However, actual implementations of genetic algorithms include at least two different `selection` operations. One that selects the parent individuals (i.e., the individuals that are allowed to breed) and another one that selects the individuals of the new generation from the set of individuals that is formed from the old generation and the newly spawned children. Finally, if a stopping criterion is satisfied in Line 10 (e.g., a specific number of generations are processed, an optimum is found, or a time is elapsed), the genetic algorithm terminates.

An example adapted from Whitley [1994] that illustrates the selection and recombination of parents is shown in Figure 4.5. In this example, the individuals are encoded as strings and in each generation, a set of pairs of parents are chosen (the intermediate generation). Each pair of this intermediate generation breeds two children through applying the crossover operator. In the new generation, the offspring completely substitutes all previous individuals (other variants are possible). In the following, we present a further example from Whitley [1994] that illustrates a possible usage of a crossover operator. Binary strings are considered as the individuals' genotype. The following two individuals are used.

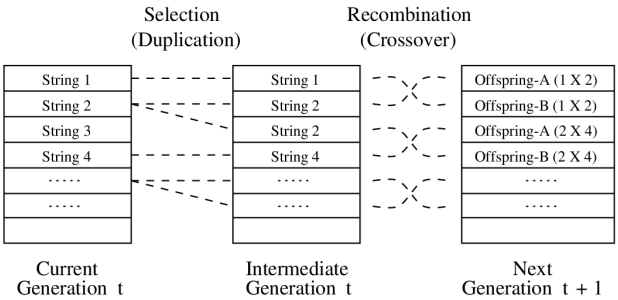


Figure 4.5. Selection and recombination of parents per generation (based on [Whitley 1994])

$$\texttt{11010} \bigvee \texttt{01100101101}$$
$$\texttt{yxyyx} \bigwedge \texttt{yxxyyyxyxxy}$$

The signs $\bigvee$ and $\bigwedge$ denote a randomly chosen *crossover point*. A crossover point determines the parts of the genotype that are mixed. Hence, a 1-point crossover (application of the crossover operator using a single crossover point) would produce the following two children:

$$\texttt{11010yxxyyyxyxxy} \quad \text{and} \quad \texttt{yxyyx01100101101}$$

The single crossover point simply splits the genes of each individual in two groups. Then, the crossover operator swaps the corresponding groups of genes. It should be noted that there often exist constraints in choosing the position of crossover points and intermixing arbitrary groups of genes might result in *infeasible candidate solutions*, i.e., candidate solutions that are not included in the feasible design space (cf. Section 4.1.3). Furthermore, there also may exist multi-point versions of the crossover operator that mix the parents in more than one point [Mitchell 1998]. Considering the child individuals from above, after applying the crossover operator, a simple mutation operation might change an arbitrary gene. For example, if the

second last gene of the second individual was mutated to "1", the following individual would result:

```
yxyyx01100101111
```

When using genetic algorithms to optimize multiple objectives, it is common to apply so-called *pareto-ranking approaches* [Konak et al. 2006] for comparing the fitness of individuals or selecting parents that are allowed to breed. Those approaches utilize the concept of pareto fronts (cf. Section 4.1.3) for evaluating the individuals and provide a way for combining independent objective functions. A popular approach that uses pareto-ranking is *NSGA-II* [Deb et al. 2002]. It is briefly described below.

NSGA-II NSGA-II is an improved (second) version of the original *non-dominated sorting genetic algorithm* [Srinivas and Deb 1994]. NSGA-II has an overall complexity of $O(MN^2)$ with N being the population size and M the number of objectives [Deb et al. 2002]. A further improvement regarding its first version concerns the application of *elitism*, i.e., the preservation of the best individuals across generations [Srinivas and Patnaik 1994b]. Elitism has been found to be beneficial and is used by many modern evolutionary algorithms [Zitzler et al. 2000]. Two variants are common. The best found individuals may be maintained within the population itself, or within a separate set that is then called an *archive* [Jensen 2003].

A further characteristic of NSGA-II is its preservation of *diversity* with a specific *crowded-comparison operator*. According to Zitzler et al. [2000], it is important to ensure diversity in the pareto-optimal front, i.e., to aim for a well-suited distribution of individuals to prevent unbalanced or biased results, for instance. The crowded-comparison operator utilizes pareto fronts and corresponding *non-domination ranks*. Figure 4.6 illustrates an example of non-domination ranks with four pareto fronts. All individuals of the first pareto front F_1 (the pareto-optimal front) are non-dominated (cf. Section 4.1.3) regarding the individuals of all other pareto fronts and therefore have non-domination rank 1. Likewise, individuals of pareto front F_2 are non-dominated regarding the individuals of the pareto fronts $F_3 - F_4$ and therefore exhibit non-domination rank 2 and so forth.

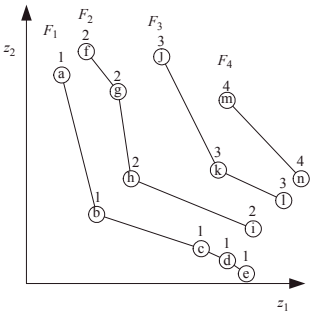


Figure 4.6. Non-domination rank example. Two minimizing objectives z_1 and z_2 and individuals $a - n$ form four pareto fronts ($F_1 - F_4$). Each pareto front F_x corresponds to non-domination rank x (based on [Konak et al. 2006]).

In the context of the crowded-comparison operator, the non-domination rank of an individual i is termed i_{rank}. The operator employs a further measure that estimates the density of candidates in the surrounding of an individual. For an individual i it is called $i_{distance}$ (or *crowding distance*). An example is shown in Figure 4.7. The crowding distance $i_{distance}$ is given by the average side length of the cuboid that is marked with dashed lines in Figure 4.7. The cuboid is formed using the nearest neighbors of i [Deb et al. 2002]. With the help of the crowding distance and non-domination rank concepts, the crowded-comparison operator ($\prec_n$) can now be defined for two individuals i and j as follows [Deb et al. 2002].

$i \prec_n j$:
$i_{rank} < j_{rank}$ OR
$(i_{rank} = j_{rank}$ AND $i_{distance} > j_{distance})$

NSGA-II uses both, *non-dominated sorting* that relies only on non-domination ranks and *crowding distance sorting* that utilizes the operator $\prec_n$. The NSGA-II procedure is illustrated in Figure 4.8 and it basically works as follows [Deb et al. 2002]. After a set of children (Q_t) were produced from a set of parents (P_t) in the t-th generation, the population R_t initially consists of both sets

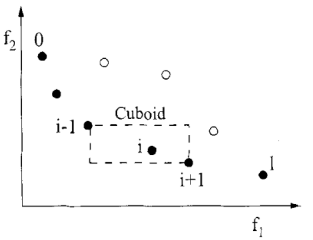


Figure 4.7. Crowding distance calculation. Points marked in filled circles are solutions of the same non-dominated front (from [Deb et al. 2002]).

($R_t = Q_t \cup P_t$). The new individuals are evaluated so the pareto fronts (F_1, F_2, ...) can be determined. The maximum number of individuals per population is set to N. As there exist $2N$ individuals at this stage in R_t, N individuals have to be removed, i.e., not all existing individuals can be transferred to the new generation. The selection operation of NSGA-II considers the individuals according to their pareto fronts, where the pareto fronts that correspond to lower (better) ranks are regarded primarily.

In the example of Figure 4.8, the individuals of the pareto fronts F_1 and F_2 can be transferred completely to the new generation, as $|F_1| + |F_2| < N$ in this example. However, the individuals of F_3 cannot be transferred completely as $|F_1| + |F_2| + |F_3| > N$. Besides considering non-dominated sorting (the ranks of the individuals according to their pareto fronts) for F_1 and F_2, the individuals of F_3 are therefore considered according to their crowding distance. Hence, F_3 is sorted according to $\prec_n$ and the best individuals (i.e., the individuals with the highest crowding distance) are chosen primarily for filling the remaining slots.

Variants Genetic algorithms are determined by their genotype and phenotype models, implementations of crossover and mutation operations, used selection operation, and chosen *control parameters*, such as the population size, mutation rate, and crossover rate [Srinivas and Patnaik 1994b], for instance. However, there exist further characteristics that form specific variants and types of genetic algorithms. For example, referring to the mutation

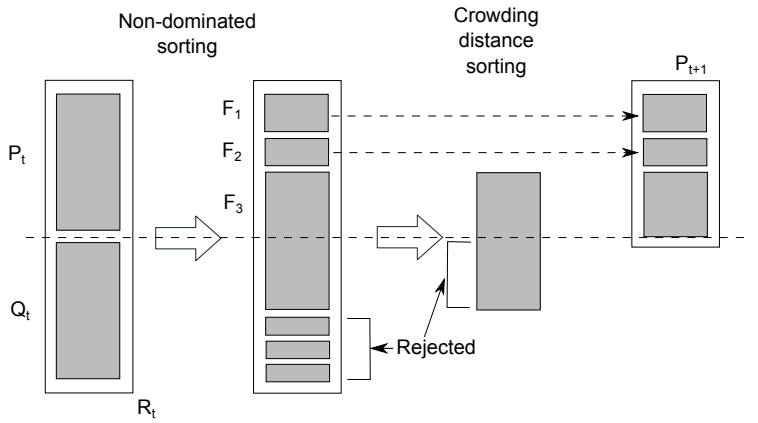


Figure 4.8. NSGA-II procedure (based on [Deb et al. 2002])

rate and crossover rate mentioned before, those rates can either be fixed or be modified by the genetic algorithms in a self-adaptive manner. The rates can be adapted to modify the speed used for exploring the search space, i.e., to adjust the corresponding step sizes, for instance. Such *adaptivity* has proven to be beneficial in several scenarios [Srinivas and Patnaik 1994a; Yun and Gen 2003; Law and Szeto 2007].

Further optimization techniques may also be combined with genetic algorithms to exploit their particular strengths. For example, a local search may be used to preferentially inspect only the promising candidates and their neighborhoods in a fine-grained way to reduce computational cost [Bhuvana and Aravindan 2011]. Genetic algorithms that incorporate further optimization techniques are called *hybrid* [Whitley 1994]. Those hybrid algorithms also have shown their ability to improve the performance of stand-alone evolutionary algorithms [Buckley 1996; Ishibuchi and Murata 1999; Nie and Deng 2008; Man et al. 2008; Bhuvana and Aravindan 2011; Sha and Xu 2011].

4.3 Simulation-Based Optimization

Simulation-based optimization [Law and McComas 2000; Al-Aomar 2002; Guikema et al. 2004; Deng 2007] aims to approach optimization problems though combining the application of optimization techniques and simulation models. Similar notions that are often used are *optimization via simulation* [Hong and Nelson 2009], *simulation optimization* [Azadivar 1999], or *optimization for simulation* [Fu 2002]. Those approaches all include both components, simulation (most often relating to discrete-event simulation [Zeigler et al. 2000]) and optimization techniques, but differ in the degree the components are weighted and emphasized [Fu 2002]. For example, optimization techniques can be seen as an extension to simulators for efficiently sweeping the possible simulation input parameters, or vice versa, simulation can be regarded as an objective function for a given optimization problem.

The approaches have in common that they are used for problems that usually cannot be solved with the help of a mathematical model. Hence, simulation is employed to approximate one or more solutions. However, to obtain the best-suited output of the simulation, it is necessary to explore the search space that spans through all potential simulation input parameters. As testing all combinations is most often too (computationally) expensive, an optimization technique is employed to efficiently explore the search space and to find promising candidate solutions. This general procedure is illustrated in Figure 4.9. An *optimizer* proposes candidate solutions that are then simulated by a *simulator*. The obtained fitness values (cf. Section 4.1) are then fed back to the optimizer to help producing improved candidates in the next iteration.

In the following we briefly summarize specific issues of simulation-based optimization as described by Azadivar [1999].

Specific issues (based on Azadivar [1999]):

- The objective function usually cannot be expressed with the help of exact mathematical functions.

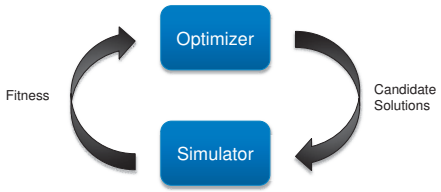


Figure 4.9. Simulation-based optimization (based on [Gao and Wang 2008])

- The decision variable search space is most often huge and neighborhood relations are often blurred or absent, as small changes to the decision variables may lead to big changes in the objective space.
- Simulations are most often computationally more expensive than evaluating analytical functions.
- Simulation programs often use other programming languages than optimization tools.
- The complexity of a simulation is not directly related to the complexity of a simulated system.
- Testing arbitrary combinations of parameters is most often only feasible with the help of simulation, but not with real systems.

As simulation is a versatile technique that is well-known in several engineering disciplines, simulation-based optimization is successfully applied to problems of diverse domains. For example, it is employed to optimize green building designs [Wang et al. 2005], the management of chemical supply chains [Mele et al. 2006], and communication protocols for large-scale wireless sensor networks [Simon et al. 2003].

4.4 Summary

This chapter gave an overview on Search-Based Software Engineering (SBSE). SBSE addresses the application of search-based optimization techniques

to problems in software engineering. Many of those problems in software engineering can be traced back to search and optimization problems. Among others, SBSE has become popular in recent years due to its versatility. Instead of requiring engineers to know *how to build* high-quality solutions, SBSE enables them to focus on describing *what characteristics* well-suited solutions should have.

As complex problems in SBSE often cannot be optimized with traditional techniques such as gradient descent, meta-heuristic optimization techniques are often employed, e.g., hill-climbing, simulated annealing, and tabu search are used. Meta-heuristics are not tied to a specific problem domain and involve some sort of randomness when iteratively approximating optimal solutions. The most popular meta-heuristics in SBSE are *genetic algorithms*. They follow analogies from biology and mimic evolutionary processes. For example, they emulate the reproduction of a population's individuals over several generations for inheriting and reinforcing well-suited characteristics and for improving the overall *fitness* of candidate solutions. A specific optimization approach that employs simulation to evaluate candidate solutions is called *simulation-based optimization*.

Part II

The CloudMIG Approach

Chapter 5

Research Design and Methods

This chapter describes the research design and methods that were used to tackle the complex challenges that accompany the migration of existing enterprise software to infrastructure and platform cloud services as outlined in the Sections 1.2 and 1.3. *Qualitative* and *quantitative* research methods [Sjoberg et al. 2007] were employed to examine the field of cloud migration and to design and evaluate the corresponding approach CloudMIG with a focus on conformance checking and optimization of deployment and runtime reconfiguration. In general, those *mixed-method* approaches [Creswell 2008], that combine qualitative and quantitative research methods, are beneficial as all methods alone are known to exhibit limitations [Easterbrook et al. 2008]. Mixed-method approaches aim at mitigating those limitations through providing a broader perspective and additional facts and insights that can serve, for example, as a corrective. The mainly utilized research methods are sketched below.

- **Action Research:** Iteratively improve a theory through incorporating practical phases that include representative actions [Potts 1993; Avison et al. 1999].
- **Case Study:** Initial investigation of facts (*exploratory*) or testing of a hypothesis (*confirmatory*) within a realistic setting [Easterbrook et al. 2008; Flyvbjerg 2011].
- **Literature Review:** A systematic review of the body of knowledge regarding a specific research area [Cooper 1998; Jesson et al. 2011].
- **Metamodeling:** Creation of meta-models facilitates uncovering the relevant linked concepts of a research area or method [Gonzalez-Perez and Henderson-Sellers 2008; Jeusfeld et al. 2009].

Figure 5.1. Work packages overview

- **Method Engineering:** Development of a single-purpose method, containing, e.g., domain-specific concepts and processes [Mayer et al. 1995; Brinkkemper et al. 1996].
- **Observation:** An empirical method for observing real-world phenomena that fosters building or testing of hypotheses [Seaman 1999; Sjoberg et al. 2007].

The research project is structured in the six work packages WP1-WP6 that are illustrated in Figure 5.1. Please note that the indicated order of the work packages should not be understood as a strict chronological sequence but rather as a conceptual sequence. Even though the conceptual

sequence largely maps to a corresponding chronological sequence, there exist activities from distinct work packages that were executed in parallel. For example, WP3 investigates conformance checking regarding CECs and CloudMIG's capabilities for detecting *CEC violations*. The corresponding models have to be integrated into the fundamental CEM that results from work package WP2. Hence, the construction of CEM is not completed after WP2. Instead, it has to be refined with new concepts arising from WP3.

Each work package poses a set of research questions and uses a combination of research methods for accomplishing the work packages' goals. Furthermore, every work package delivers artifacts that constitute the resulting outcomes of that specific work package.

Figure 5.2 summarizes the work packages WP1-WP6. It follows the structure introduced by Giesecke 2008, p.70. As indicated in Figure 5.2, the contribution SC1 of this thesis (cf. Section 1.3) is addressed in the work packages WP1 and WP2, as those cover the clarification of basic preconditions for building the CloudMIG method and also its actual construction. The contributions SC2-SC5 map to the work packages WP3-WP6, whereas a part of contribution SC4 (tool implementation) is covered by WP6. In the following, the work packages WP1-WP6 are described in the corresponding Sections 5.1-5.6. Finally, Section 5.7 sums up this chapter.

5.1 WP1: Problem Analysis

The work package WP1 prepares the ground for tackling challenges regarding the migration of enterprise software to the cloud by providing a thorough problem analysis. In the first place, the relevant research fields are identified that form the underlying theoretical foundation.

As CloudMIG addresses the migration of software systems to cloud environments, investigating the area of cloud computing is essential, because specific migration challenges are raised by this type of target environment. Furthermore, the field of software modernization plays a vital role. For example, Section 7.4.1 shows that the degree of modification can be used to classify a

Work Package	Research Questions	Research Methods	Results
WP1: Problem Analysis *(Contribution SC1)*	Q1.1: What is the current state of research and which semi-automated and automated approaches exist for migrating a software system to a cloud environment? Q1.2: Which challenges regarding a migration to the cloud are unique with respect to other migration scenarios? Q1.3 What are major technical challenges when migrating software to a cloud environment?	Literature Review, Exploratory Case Study, Observation	- Review on the current state of research on software migration towards cloud environments - Report on unique characteristics of migrating software to cloud environments - Observation report on challenges in cloud migration
WP2: CloudMIG Method Foundations *(Contribution SC1)*	Q2.1: Which activities can be performed to tackle identified cloud migration challenges? Q2.2: Which meta-models have to be provided for CloudMIG and what information do they have to include? Q2.3: Which existing meta-models can be utilized?	Method Engineering, Metamodelling, Action Research	- CloudMIG method description - Cloud Environment Model (CEM) - Software architecture meta-model - Utilization meta-model - Overview constraint violation detection - Overview cloud deployment option generation and improvement
WP3: Conformance Checking *(Contribution SC2)*	Q3.1: Can arbitrary cloud environment constraints be modeled? Q3.2: Can cloud environment constraints be described in a form that is agnostic to a particular cloud environment? Q3.3: Which constraint violations can be detected when using solely the code models extracted via a static analysis?	Literature Review, Metamodelling, Action Research	- Cloud environment constraint meta-model - Code model suitability analysis for conformance checking - CSA hierarchy
WP4: Deployment and Reconfiguration Optimization *(Contribution SC3)*	Q4.1: How can cloud deployment options be described and created? Q4.2: Is it possible to automatically assess cloud deployment options? Q4.3: How can cloud deployment options be automatically optimized?	Literature Review, Metamodelling, Action Research	- Cloud deployment option meta-model - Simulator CDOSim - Simulation-based genetic algorithm CDOXplorer
WP5: Tool Architecture *(Contribution SC4)*	Q5.1: How can the elements of the CloudMIG method be modularized? Q5.2: Which quality characteristics are required in particular? Q5.3: Which architectural styles facilitate the implementation of these quality characteristics?	Metamodelling, Action Research	- Plugin-based software architecture for CloudMIG Xpress - Quality characteristics and architectural styles report
WP6: Evaluation *(Contributions SC4 and SC5)*	Q6.1: Is the approach feasible and applicable for diverse cloud environments? Q6.2: Is the approach feasible for different programming languages? Q6.3: Can the conformance evaluation automatically detect a wide range of constraint violations in enterprise software? Q6.4: Can CDOXplorer find near-optimal cloud deployment options? Q6.5: Can the approach practically be applied within an industrial case study?	Tool Development, Experimental Evaluation through Confirmatory Case Studies	- Cloud profiles for Amazon EC2, Google App Engine for Java, Microsoft Windows Azure, and Eucalyptus cluster - Tool CloudMIG Xpress - KDM discoverers for Java (including bytecode), C#, and Python - Report on characteristics of constraint violations in enterprise software and concerning the precision of detection - Report on CDOXplorer's performance - Feasibility evaluation with an industrial partner

Figure 5.2. Work packages contents

software system's suitability and alignment regarding a specific target cloud environment. Hence, diverse modernization methods and techniques—such as reengineering, architecture-driven modernization, and restructuring, for instance—are relevant and have to be reviewed. Moreover, the search-based software engineering field contributes suitable methods and techniques for optimizing cloud deployment architectures and runtime reconfiguration rules. As a consequence, the three mentioned fundamental fields of cloud computing, software modernization, and search-based software engineering are elucidated based on literature studies in the Chapters 2, 3, and 4, respectively. Based on these fundamental overviews, research question *Q1.1* addresses a combination of the corresponding fields and is processed by compiling the current state of research regarding software migration to the cloud with the help of a literature review. Please note that conformance checking does not constitute a research field on its own. Thus, we merely describe corresponding related work in Section 13.2. Furthermore, as CloudMIG strives for raising the degree of automation during the migration, the research question *Q1.1* consequently differentiates the approaches according to their degree of automation. The corresponding related work is described in Section 13.1.1.

Research question *Q1.2* aims at investigating the unique characteristics of software migration to infrastructure and platform cloud services. That means, it aims at identifying the challenges of this migration type that are unique compared with migration approaches that focus on other target environments, such as newer versions of a middleware or more advanced programming languages. Besides performing a literature review, we employed the method of observation [Endres and Rombach 2003] in an exploratory case study to find additional challenges. Here, in the context of a class, a group of master students was asked to migrate a Java-based webshop to the PaaS cloud environment Google App Engine for Java within a 1.5 months timeframe. The challenges raised by this migration scenario were then distilled by means of observation and by letting the students complete questionnaires.

The challenges were then analyzed for extracting the specific characteristics of a cloud migration that distinguish it from other migration types, i.e., that

would not have occured considering other target environments. These specific characteristics are described in Chapter 6. Furthermore, the mentioned questionnaires asked for the most pressing challenges. The corresponding answers together with the observations and the problems stated in the literature were used to prioritize the challenges. Hence, the major technical challenges when migrating software to the cloud (*Q1.3*) could be identified.

5.2 WP2: CloudMIG Method Foundations

This work package aims at providing the foundations for the CloudMIG method that supports (future) cloud users in the planning phase of a cloud migration. WP2 builds upon the identified state of research and the challenges that are investigated in WP1. In general, the structure and basic models for the CloudMIG method are created in a similar way as proposed by the method engineering process introduced by Mayer et al. [1995] that is shown in Figure 5.3. The process starts with a documentation of the motivation and a search for existing methods. These activities are already addressed in WP1. The next activities in the process of Mayer et al. [1995] aim at reusing existing methods as far as possible. Depending on the extent existing methods can be reused, it is possible to adopt complete methods, tailor existing methods, or develop a new method while considering best practices. In this thesis, we follow the last alternative in WP2 and build a new method while considering some best applicable practices from the approaches explored in WP1 that fit our needs.

Research question *Q2.1* aims at specifying the activities that can generally be performed to tackle the cloud migration challenges from WP1. Please note that the conformance checking capability and the cloud deployment architecture and runtime reconfiguration optimization capability are defined in detail in WP3 and WP4, respectively. Furthermore, WP2 addresses the method engineering process' activity six (cf. Figure 5.3) and designs basic method application techniques. In this context, research question *Q2.2* is posed to investigate the fundamental meta-models that describe the types, relationships, and workflow process of the CloudMIG method. For

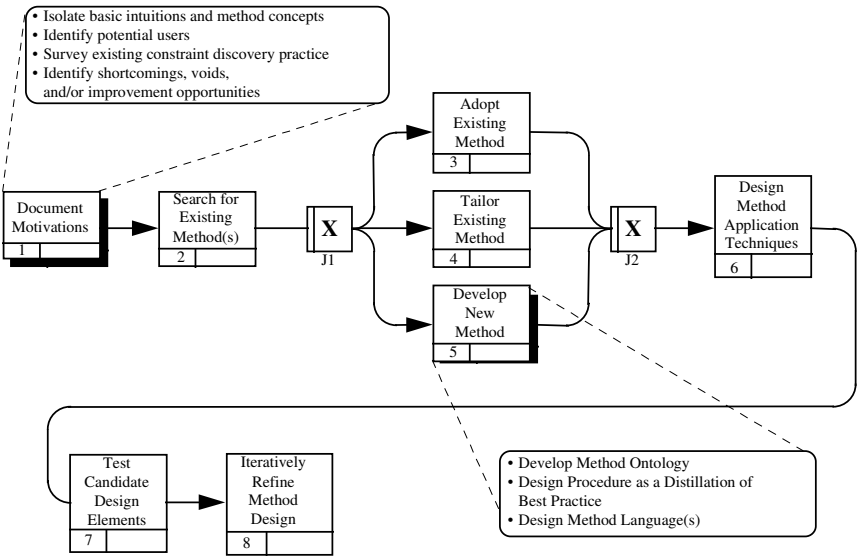


Figure 5.3. Method engineering process (from [Mayer et al. 1995])

example, the following basic elements have to be described in order that the conformance checking capability and cloud deployment architecture and reconfiguration optimization capability can be specified.

- **CloudMIG Method Description:** The method description forms the basic framework that defines CloudMIG's components and their interactions. It outlines all meta-models, stakeholders, artifacts, roles, activities, preconditions, and assumptions relevant for planning a cloud migration with the help of CloudMIG. These constituents are described in a varying degree of detail as, for example, the specifics regarding the conformance checking capability are investigated in greater detail in a subsequent work package.
- **Cloud Environment Model:** This meta-model defines the elements, structures, and service models of a cloud environment. For example, it

distinguishes between IaaS- and PaaS-based cloud environments, describes offered services, and specifies pricing models. Furthermore, several elements are necessary as a foundation for checking the conformance of a software system in relation to a particular cloud environment. For example, the CEM provides means to integrate specific CECs and to incorporate an existing system's source code. When considering the optimization of cloud deployment architectures and runtime reconfiguration rules, the basic CEM packages can be used to define VM instance types and performance capabilities of a specific cloud environment, for instance.

- **Software Architecture Meta-Model:** To create and compare various cloud deployment architectures and to search a system's source code for CECs, it is necessary to provide a sound model of the system's software architecture. For example, the corresponding software components can be mapped to cloud containers with the help of CEM's mapping model. Here, some requirements considering the software architecture meta-model have to be met. For example, the software architecture description has to be available as an Ecore-based model for integrating it with the help of the mapping model. Moreover, it should be possible to allow for a large degree of automation when extracting the architectural model using static analysis techniques.
- **Utilization Meta-Model:** To reason about a potential over- and under-provisioning of a specific cloud deployment architecture or to optimize runtime reconfiguration rules, it is necessary to take the previous utilization of a software system into account. In the context of this thesis, we propose to follow a measurement-based approach to provide actual and precise information regarding, for example, the CPU utilization of an on-premise server and usage patterns from historical log files.

To further leverage the reuse principle that is also stressed by the method engineering process of Mayer et al., research question *Q2.3* aims at investigating existing meta-models that can be utilized completely or be tailored to provide a basis for the meta-models sketched above. The research questions *Q2.1*-*Q2.3* and the resulting meta-models, concepts, and activities approached by the work package WP2 are described in Chapter 6.

5.3 WP3: Conformance Checking

The previous work package WP2 laid the foundations for, among others, building and integrating the conformance checking capability. The subsequent work package WP3 investigates related solution approaches and the construction of conformance checking concepts and techniques in greater detail. Several challenges of CECs have to be taken into account. For example, despite the significant differences between IaaS- and PaaS-based cloud environments, it has to be possible to cover both service models. A small excerpt of diverse CECs is listed below for demonstration purposes.

- No threads can be spawned
- The firewall cannot be configured to permit outgoing traffic for specific port ranges
- The instance storage of a VM instance is not saved persistently
- Merely the use of white-listed types is allowed
- Calls to a set of API methods are forbidden and produce an exception
- Only 20 CPU cores can be utilized in parallel considering a specific price category

Those CECs differ to a great extent and require a completely separate handling as well as varying mechanisms for tailoring the CECs for particular cloud environments. Hence, research question *Q3.1* addresses the challenge regarding the broad diversity among CECs.

The plethora of existing cloud providers further adds to the complexity of building a generic meta-model for conformance checking. Even if considering cloud environments with identical service models, most cloud providers do not follow a common cloud standard [Miranda et al. 2012] and therefore provide different services, resource configurations, or sandbox environments, for instance. Nevertheless, an appropriate meta-model has to cope with those challenges and has to offer powerful and expressive

modeling capabilities that are practicable for a variety of cloud environments. Consequently, the research question *Q3.2* aims at ensuring that the conformance checking approach is (1) not biased towards a particular cloud environment and that it is (2) applicable for a broader range of cloud environments instead.

The research question *Q3.3* investigates the suitability of the meta-model used for modeling an existing software system (KDM) regarding the detection of *CEC violations*. This suitability is of particular importance because the information concerning a software system's source code, architecture, and artifacts is solely described in a model that conforms to KDM. Though conceptually being extensible with information that stems from dynamic analyses, in the context of this thesis, a KDM-based model is extracted by exclusively using a static analysis. Hence, *Q3.3* analyzes (1) which constraint violations can be detected when utilizing the KDM-based models resulting from static analysis, and (2) what type of *CEC violations* require additional information from some kind of dynamic analyses.

5.4 WP4: Deployment and Reconfiguration Optimization

The work package WP4 covers the optimization of cloud deployment architectures and runtime reconfiguration rules. It builds upon the work package WP2 and extends the meta-models with a specification of the CDO element. Furthermore, it provides means for creating, assessing, and optimizing CDOs. These aspects are detailed in the following.

- **CDO Meta-Model:** The basic structure of CDOs was already sketched in Chapter 2. WP4 provides the corresponding meta-models for the elements forming a CDO, i.e., cloud deployment architectures and runtime reconfiguration rules, and integrates these meta-models with the CEM. Challenges regarding the specification of the CDO meta-model and the integration with the CEM are addressed by the research question *Q4.1*.

- **Creation of CDOs:** It should be possible to create CDOs manually as well as automatically. Manually constructing CDOs is appropriate for future SaaS providers who have a clear picture of the specific cloud environment, deployment architecture, and runtime reconfiguration rules that should be used for a (final or candidate) CDO. For example, this can be based on expert knowledge or recommendations. In contrast, automatically creating CDOs is appropriate if the structure of potentially well-suited candidates is unknown or if a great number of CDOs is supposed to be evaluated and compared.

  Considering the CDO construction process, particular challenges arise for the restructuring and mapping of the extracted KDM models to the target software architecture and cloud deployment architecture. For example, the latter has to incorporate the specific cloud container structures that are available in a cloud environment, i.e., in a CEM-based model. The approach for creating CDOs and corresponding challenges are also addressed by the research question *Q4.1*.

- **Assessment of CDOs:** Both manually and automatically created CDOs have to be assessed to judge about their suitability and fitness compared to other CDO candidates. A straightforward approach for assessing the appropriateness of a CDO could employ architectural quality characteristics. For example, through considering the number of average classes per package (ACP), abstractness (ABS), and normalized distance (NOD) (e.g., as described by Hansen et al. [2011]). However, solely relying on those architectural metrics does leave out CDO properties that are of great interest when deploying existing software systems to a cloud environment, for example, the resulting costs and QoS characteristics [Yusoh and Tang 2012]. Hence, the research question *Q4.2* aims at the construction of an automated assessment approach that also incorporates such properties.

- **Optimization of CDOs:** The aforementioned automated assessment approach lays the foundation for the automatic optimization of candidate CDOs. As there exists a huge number of possible CDO candidates, such an optimization approach has the potential of releasing SaaS providers from the burden of manually creating and evaluating CDOs.

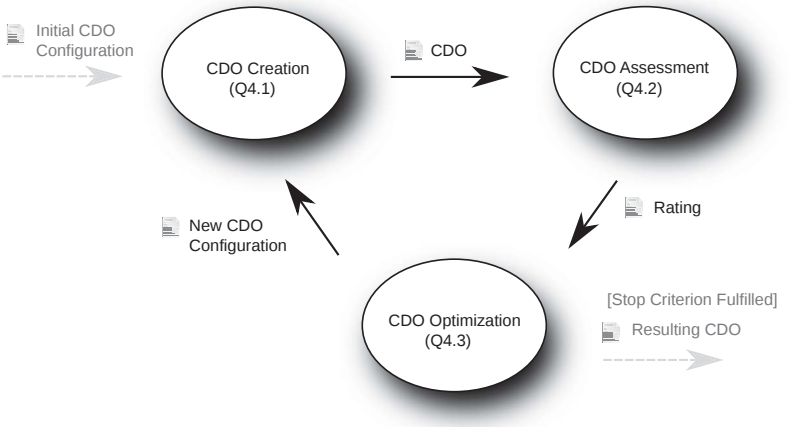


Figure 5.4. Basic CDO optimization feedback loop

In general, the optimization procedure conceptually follows the basic feedback loop that is illustrated in Figure 5.4. An initial CDO configuration is used to create a first CDO with the mechanism addressed by the research question *Q4.1*. This CDO is then assessed by means of the assessment approach created in the context of the research question *Q4.2*, before the optimization procedure, that is addressed by the research question *Q4.3*, produces a new CDO configuration. This new CDO configuration is then used for the next iteration of the feedback loop. The feedback loop ends and delivers the resulting CDO when a stopping criterion is fulfilled. For example, when the maximum number of iterations is reached or a specific quality attribute achieves a target value.

5.5 WP5: Tool Architecture

This work package WP5 creates a software architecture for an application that provides tool support for the CloudMIG method. Furthermore, this

application is intended to provide means for evaluating the core contributions of this thesis, i.e., the approaches for conformance checking and optimization of deployment architectures and runtime reconfiguration rules in the context of a cloud migration.

The preceding work packages deliver resulting meta-models, algorithms, and concepts that form the approach CloudMIG. These results have to be structured to connected components and in some cases have to be further detailed to obtain realizable requirements, for example, from high-level concepts. As a consequence, the research question *Q5.1* aims at finding a well-suited modularization of CloudMIG's elements. Hence, this research question addresses the structuring of required functionalities.

However, the quality of a software architecture is mainly determined by its non-functional properties—for example, considering its maintainability, performance, and security—that in turn can be stipulated by selecting and combining proper architectural styles [Bass et al. 2002].

The research question *Q5.2* addresses the identification of quality characteristics that are of particular importance for potential users of the envisioned tool. Consequently, the research question *Q5.3* aims for investigating architectural styles that facilitate achieving these quality characteristics.

5.6 WP6: Evaluation

The work packages WP1-WP5 deliver the individual building blocks of the approach CloudMIG. After producing and examining the respective results on their own, this work package WP6 evaluates several important aspects of the assembled and completed approach CloudMIG. The common high-level goals are (1) to validate CloudMIG's applicability regarding the fundamental challenges that were identified in the WP1 during the detailed problem analysis. Furthermore, (2) it has to be verified whether CloudMIG's methods and techniques can tackle identified challenges in accordance with expected quality requirements. Considering this last high-level goal, the focus on quantifiable experiments is on the contributions

SC2 and SC3 of this thesis (cf. Section 1.3), i.e., regarding conformance checking and optimization of cloud deployment architectures and runtime reconfiguration rules, respectively.

A shortcoming of many current cloud migration approaches is their concentration on specific cloud environments. As CloudMIG supports SaaS providers to migrate enterprise software to the cloud, it facilitates a dedicated cloud user perspective by enabling unbiased comparisons of several cloud environment candidates. Thus, the research question *Q6.1* evaluates CloudMIG's respective capabilities, i.e., whether the approach is feasible and applicable for diverse cloud environments. Therefore, the software architecture introduced in the work package WP5 is implemented in the tool *CloudMIG Xpress* and is supplemented with various cloud profiles that demonstrate CloudMIG's suitability regarding the diversity of cloud environments.

Similar to the diversity of cloud environments, it is important that diverse programming languages can be integrated in the CloudMIG approach. Hence, the research question *Q6.2* aims at extending *CloudMIG Xpress* with several discoverers that extract KDM-based models from source code that stems from a range of different programming languages.

Confirmatory case studies are used to evaluate the contributions SC2 and SC3 of this thesis (cf. Section 1.3). Firstly, the detection of *CEC violations* is addressed by the research question *Q6.3*. Here, a focus is on characteristics of *CEC violations* in current enterprise software and concerning the precision of the detection mechanism. Secondly, the research question *Q6.4* aims for evaluating the simulation-based genetic algorithm CDOXplorer, i.e., the performance of the CDO optimization procedure in comparison to other state-of-the-art search and optimization approaches.

Another aspect that is evaluated by the work package WP6 is the feasibility of several of CloudMIG's techniques in an industrial context. Together with an industrial partner, the KDM extraction and conformance checking capabilities are evaluated to answer the research question *Q6.5*.

5.7 Summary

In this chapter, the research design and methods were described that are used to prepare, construct, and evaluate the approach CloudMIG. The included activities are structured into six work packages WP1-WP6. For each work package, accompanying research questions inquire about ways to achieve one or more goals of a work package. To obtain the desired outcome of each work package, a range of research methods are employed, for example, literature reviews, metamodeling, and case studies.

The first work package WP1 conducts a detailed problem analysis and uncovers the current state of research and major challenges when migrating enterprise software to a cloud environment. Based on these findings, the work package WP2 builds the foundations of the CloudMIG method, e.g., elementary meta-models and activities. The central contributions SC2 and SC3 of this thesis (cf. Section 1.3), i.e., the conformance checking approach and the optimization of cloud deployment architectures and runtime reconfiguration rules, are separately detailed in the work packages WP3 and WP4, respectively.

To provide tool support for the CloudMIG method, the software architecture for a corresponding application is specified in the work package WP5. The last work package WP6 covers the evaluation of the CloudMIG method. A focus lies on the evaluation of the conformance checking approach and the optimization procedure for cloud deployment architectures and runtime reconfiguration rules. Confirmatory case studies are employed in both cases to provide thorough insights, to confirm the approach's applicability, and to obtain quantifiable and comparative analyses.

Chapter 6

The CloudMIG Method

This chapter describes the CloudMIG method that supports (future) cloud users to migrate existing enterprise software to IaaS- and PaaS-based cloud environments. The CloudMIG method focuses on SaaS providers and targets the planning phase of a cloud migration project. It addresses SaaS providers that formerly utilized an on premise infrastructure to provide the services of an enterprise software system—such as web shops and Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM) systems—to internal or external customers. Those SaaS providers typically operate and maintain complete hardware and software stacks on their own. Hence, routine activities of SaaS providers include procuring new server machines, installing additional switches, replacing faulty power adapters, installing operating systems, and laying network cables, for instance. Migrating existing software systems to the cloud now becomes compelling for more and more SaaS providers for various reasons. For example, it enables the SaaS providers to increase their flexibility and agility, to exploit the cloud's elasticity, to focus on their core business, and to cut the capital expenditures [Goyal 2010; Ward et al. 2010; Marston et al. 2011; Kaisler and Money 2011; Phaphoom et al. 2012]. As briefly outlined below, CloudMIG supports the SaaS providers in multiple dimensions.

Figure 6.1 shows an overview of the method. CloudMIG helps to identify the IaaS- or PaaS-based cloud environment that is best suited for hosting the software system that is to be migrated to the cloud. For example, cloud environments differ regarding the provided performance, costs, and offered services. Hence, picking a suboptimal cloud environment is an indeed costly but nevertheless frequently found endeavor because of the

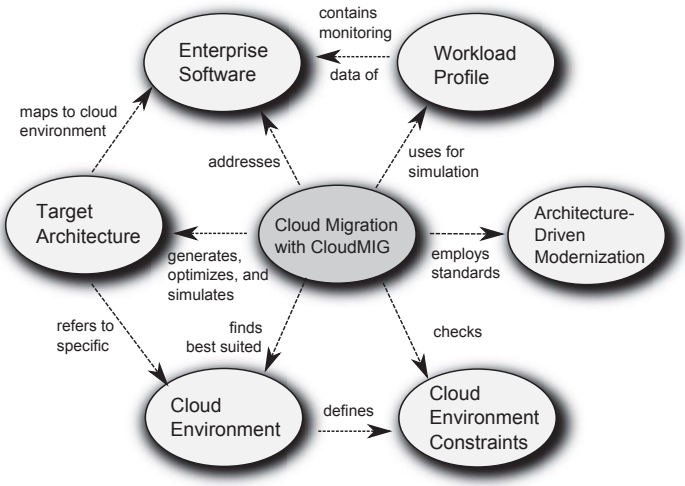


Figure 6.1. CloudMIG overview

multitude of existing cloud providers [Prodan and Ostermann 2009]. Besides supporting the cloud environment selection procedure, the CloudMIG method produces a target architecture for the enterprise software system that aligns with the selected cloud environment's specific characteristics, for example, provided VM instance types, services, and charged prices. Such a target architecture describes a beneficial structuring and deployment in the selected target cloud environment.

Moreover, a target architecture constitutes a trade-off solution, as, on the one hand, SaaS providers strive for reducing the migration effort through minimizing the required modifications. On the other hand, a more sophisticated alignment of the software system to a specific cloud environment allows to leverage its particular capabilities and to reduce future operational expenditures, for instance. The CloudMIG method helps to generate and optimize target architectures. For assessing the target architectures, the method enables to employ various evaluation mechanisms, for example, several software metrics or a simulation. For the latter, it is possible to uti-

lize an SMM-based workload profile that consists of monitored or synthetic usage data.

A further applied standard from OMG's ADM initiative is KDM. CloudMIG mainly employs KDM to model a system's source code and artifacts. These models can, to a great extent, be reverse-engineered automatically. Furthermore, both KDM and SMM-based models, that describe properties of an existing software system, are used to analyze its suitability regarding specific cloud environments by means of the cloud environments' CECs and the system's corresponding *CEC violations*.

The CloudMIG method is structured according to several activities that form the overall approach. For example, reverse engineering a KDM-based model from a system's source code is part of the activity *Extraction* and generating a target architecture is part of the activity *Generation*. Furthermore, CloudMIG supports a range of specific cloud migration types that are described later in Section 6.2. For example, merely substituting applications with existing web-based solutions is out of CloudMIG's scope.

This chapter utilizes and builds upon the following previously published work:

1. *Frey, Sören and Hasselbring, Wilhelm, "The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications," in International Journal on Advances in Software, ISSN 1942-2628, Vol. 4, Nr. 3 and 4, pp. 342-353, 2011.*
2. *Frey, Sören and Hasselbring, Wilhelm and Schnoor, Benjamin, "Automatic Conformance Checking for Migrating Software Systems to Cloud Infrastructures and Platforms," in Journal of Software: Evolution and Process, Vol. 25, Nr. 10, pp. 1089–1115, 2013.*
3. *Frey, Sören and Schulz, Eike and Rau, Malin and Hesse, Kevin, "CloudMIG Xpress 0.5 Beta User Guide," 2012.*

The chapter is organized as follows. Section 6.1 gives an overview of our cloud migration method CloudMIG. The next Section 6.2 explains the range of cloud migration types that are addressed by CloudMIG. Section 6.3 details our fundamental approach for supporting the planning phase of

cloud migrations that comprises, among others, CloudMIG's activities. This section also describes how the conformance checking approach and the deployment and reconfiguration optimization are embedded in the overall method. Finally, Section 6.4 sums up this chapter.

6.1 Overview

The general decision to migrate an existing enterprise software to the cloud should already have been made before utilizing the CloudMIG method. That means, an upfront analysis should have been conducted for investigating fundamental legal, organizational, and economic implications. Those are potential areas that may reveal prohibiting obstacles. For example, the jurisdiction of a country may forbid transferring any business data to foreign cloud providers or a company's financial structure may be incompatible with the commonly found pay-per-use model, as it relies on the tax depreciations that are realized through recurrently procuring IT hardware. Notwithstanding the above, there may indeed exist challenges that can impede the migration to a cloud environment and that can only be uncovered by actually applying CloudMIG, for example, a very high number of *CEC violations* or inappropriate future expenditures. But first ensuring the general suitability of migrating an existing system to the cloud is a reasonable approach.

The CloudMIG method concentrates on migrating enterprise systems to the cloud as those are primary candidates that can benefit from smoothly scaling up and down in the cloud due to varying workload patterns. As the focus of CloudMIG is on the planning phase of a cloud migration, it does not cover the actual migration of an enterprise software system to a cloud environment. Hence, the actual transition from a chosen target architecture to the cloud has to be implemented manually. This will become evident when we describe how CloudMIG can be mapped to the ISO/IEC 14764 standard (cf. Section 3.2.3) in the later Section 6.1.2. Furthermore, the CloudMIG method does not address migration projects that solely substitute modern applications for legacy software. For example, projects are out of

CloudMIG's scope that solely substitute modern web-based human resource management systems for other human resource management software products that were previously hosted on premise.

We mentioned before that CloudMIG requires the source code of an existing enterprise software that is to be migrated to the cloud. Thus, it addresses those SaaS providers that are at the same time the producers of the software. If a service developer S_D outsources the operation of the service to a SaaS provider S_P, CloudMIG can still be applied by S_P if S_D agrees to make the relevant source code available, for example, by corresponding contractual commitments. Alternatively, if a SaaS provider wants to migrate an open source enterprise system to the cloud, CloudMIG can be used as well, as the source code is available, too.

In addition to it, SaaS providers have to overcome numerous further challenges and shortcomings of current approaches (see Section 13.1) when migrating existing software systems to the cloud. Organizational implications might include a reshaping of internal divisions as responsibilities of IT service management or software maintenance departments shift, for instance. In addition, new liability or auditing issues may arise because sensible data is no longer stored exclusively on premise. Along with the mentioned problem to identify data assets that can be moved to the cloud comes the increased need to encrypt this data. However, recent advancements in the cryptography domain in achieving fully homomorphic encryption might eventually enable practicable arbitrary computation on encrypted data without the necessity to decrypt it beforehand and therefore mitigate data security concerns to a great extent [Gentry 2010].

Compared with the aforementioned organizational, legal, and economical challenges, this securing of the data access in cloud computing environments is a technical problem. CloudMIG mainly targets the technical perspective of migrating enterprise software to cloud environments by checking CECs and generating target architectures, for instance. Examining the challenges that emerge from migrating software to the cloud, these challenges can, of course, be partitioned in *specific* and *unspecific* cloud migration challenges (cf. research question Q1.2 in Section 5.1). That means,

the challenges can arise only in the course of a cloud migration (specific) or during migrations to arbitrary target platforms (unspecific), such as a SOA or a particular object-oriented middleware. For example, challenges addressing liability and governance issues may occur in any migration scenario that incorporates outsourcing a system's operation. Hence, those (broadly formulated) challenges are unspecific to a cloud migration. However, as the elasticity, combined with its availability in the form of a commodity service, is a core benefit and technical innovation of the cloud computing paradigm, reengineering a software system towards enabling elasticity is an exemplary specific challenge of a cloud migration.

All specific cloud migration challenges that are addressed by CloudMIG are described in Section 6.1.1. Then, Section 6.1.2 shows how CloudMIG integrates with the ISO/IEC 14764 standard (see Section 3.2.3).

6.1.1 Addressed Challenges

As described before, SaaS providers that migrate enterprise software systems to the cloud have to consider various dimensions that can impose potential hurdles, for example, organizational, legal, and economical challenges. However, CloudMIG focuses on technical challenges, such as detecting *CEC violations*. This section further narrows down those challenges that are tackled by CloudMIG, to provide a solid ground for describing the design of the CloudMIG method in the following sections.

To investigate some elementary assumptions, we begin with examining basic challenges in the context of a cloud migration with the help of an initial experiment (cf. research question Q1.3 in Section 5.1). Simplistic migration approaches try to minimize the effort for migrating software to the cloud while accepting potential imperfections and inefficiencies during operation. As CloudMIG improves the efficiency of a target architecture prior to the actual migration, the experiment aims at verifying that simplistic migration approaches exhibit substantial room for improvement.

Table 6.1. Eucalyptus hardware configuration

Component	Variant
CPU type	2x AMD Opteron 2384 2.7GHz (4 cores)
RAM	16 GB DDR2-667
Network	1 Gbit/s

An Initial Experiment for Verifying Basic Addressed Challenges

We investigated the deployment of Apache OFBiz 9.04[1] into an installation of the cloud software Eucalyptus [Nurmi et al. 2009]. Apache OFBiz is a Java-based open source E-Commerce/ ERP system. For instance, it provides several modules for accounting, order processing, and human resource management that are accessible via a web-based Graphical User Interface (GUI).

The hardware listed in Table 6.1 was utilized for Eucalyptus' cluster and node controllers responsible for allocating and controlling the cluster of VMs. The superordinate cloud controller node was installed on an identically equipped machine. However, that second machine did not provide dedicated resources for VM allocation. In typical IaaS offerings as well as with Eucalyptus, a cloud user can choose between different VM instance types as basic building blocks. A VM instance type determines the hardware configuration that is available for running the user's virtual machine. With every start of a VM, an appropriate instance type can be assigned according to the user's current needs.

To evaluate the implications of VM instance type selection we configured the six different VM instance types that are listed in Table 6.2. In many cloud offerings, the VM instance types are priced on a pay-per-use basis and proportional to supplied resources (cf. [Armbrust et al. 2009; Vaquero et al. 2009]). The selection of proper VM instance types may therefore have a substantial impact on overall operational costs. Since in real cloud offerings the amply equipped VM instance types are more costly by tendency, we

[1] http://ofbiz.apache.org/

Table 6.2. VM instance types

Name	#CPU cores	RAM (MB)
Standard.M	1	512
Standard.L	2	1,024
Memory.M	2	2,048
Memory.L	2	3,584
Compute.M	4	2,048
Compute.L	6	2,048

Table 6.3. VM instance type price model

VM instance type	Costs/hour ($)
Standard.M	0.3
Standard.L	0.4
Memory.M	0.5
Memory.L	0.75
Compute.M	0.6
Compute.L	1.15

used the price model shown in Table 6.3 that follows this principle. The inter-arrival time function illustrated in Figure 6.2 was applied to simulate a typical day night cycle usage pattern where the experiment minutes map to the hours of a day. Our employed user behavior emulated customers visiting the web store and browsing a product category. The number of user requests exhibits two peaks, one in the morning and one in the evening hours.

It should be noted that the demo installation of Apache OFBiz 9.04 was used that applies the rather slow embedded Java database Derby to deliver the demo catalog products. However, as the focus of our experiment was to compare the implications resulting from different VM instance types, this does not affect the results' validity. We were particularly interested in the resulting variations concerning the response times and the observed CPU utilization. Regarding the response times we defined a limit of 1.5s that should not be exceeded for our test user sequence and which can be seen as a part of a virtual SLA [Iqbal et al. 2009]. As illustrated in Figure 6.3, the usage of one single instance of a VM instance type was not always sufficient to fulfill the SLA. Here, one Standard.L instance provokes an SLA violation in the evening hours (Figure 6.3a). The single Standard.M instance (Figure 6.3b) exhibits an even more distinctive under-provisioning, as the CPU was often used up to the limit. As a consequence, Apache OFBiz repeatedly just returned error messages after experiencing a massive increase in response times up to minute 19, and therefore, caused the test to stop. Hence, in the following, we also investigated the minimum number of instances concerning each VM instance type that were necessary to satisfy the SLA. Here, we always maximized the Java Virtual Machine

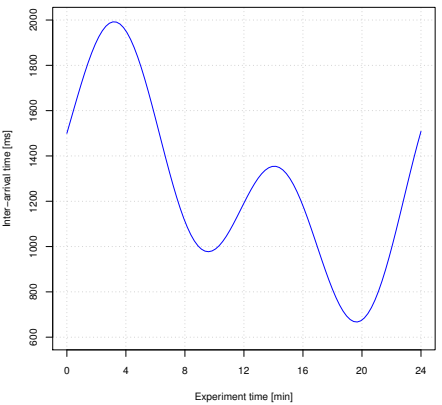


Figure 6.2. Inter-arrival time function (cf. [Frey and Hasselbring 2011b])

(JVM) heap size that could be configured according to the VM instance type specifications and that was available to Apache OFBiz.

Experiment Results

An overview regarding the measured response times and CPU utilization following the varying load for each applied VM instance type is presented in Figure 6.4. To stay below the 1.5s SLA response time limit, two instances of the Standard.M and Standard.L VM instance types were required in each case. The corresponding Figure 6.4a and Figure 6.4b therefore show the average response times and CPU utilization for both instances. The response times and CPU utilization generally followed the usage pattern with a rise during peak times and exhibiting lower phases otherwise. Nevertheless, considering the response times this effect manifests more blurred for the aggregated measurements of the two Standard.M and Standard.L instances. Regarding the Standard.M instances the overall CPU utilization was still rather high. An interesting detail can be noticed in Figure 6.4c - 6.4f as there are short bursts around the 14th minute when the number of user requests leaves behind a local minimum.

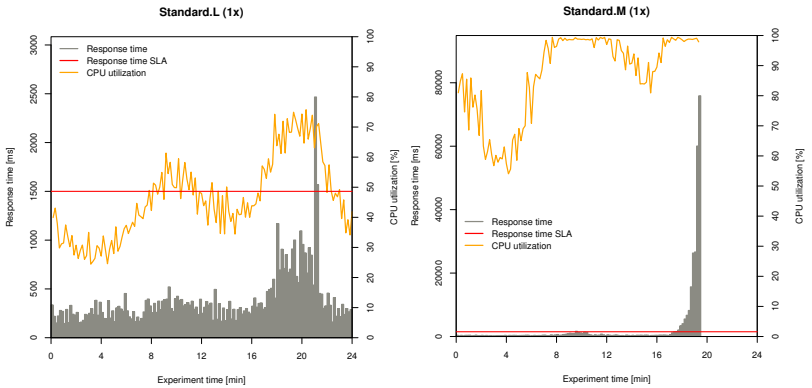


Figure 6.3. SLA violation when using a single instance of the Standard.L (a) or Standard.M (b) VM instance type (cf. [Frey and Hasselbring 2011b])

Besides for the Standard.M VM instance type, the CPU utilization fluctuates at a rather low level. Figure 6.5a underlines this observation by showing the average CPU utilization for each experiment. Incorporating the Standard.M VM instance type, the avg. CPU utilization ranges from 16%-59%, which translates to an avg. CPU over-provisioning ranging from 41%-84% at the same time. As mentioned before, we assume presence of a pay-per-use billing model. Considering our defined VM instance type price model (see Table 6.3) the resulting operational costs being extrapolated for one month are presented in Figure 6.5b. Here, we simplifying presume that the usage pattern repeats each day and therefore the number of the minimally required instances remains stable. The cost minimum is reached by utilizing one Memory.M instance.

Challenges Revealed by the Experiment

The previously described example scenario reveals several challenges considering simplistic approaches for the migration of software systems to a

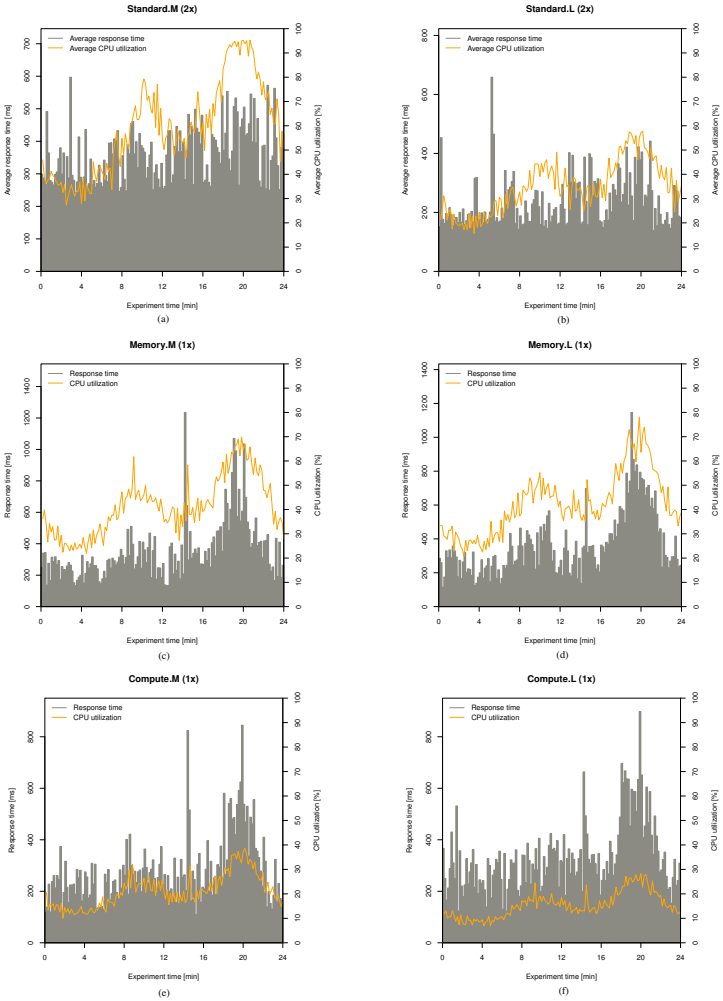
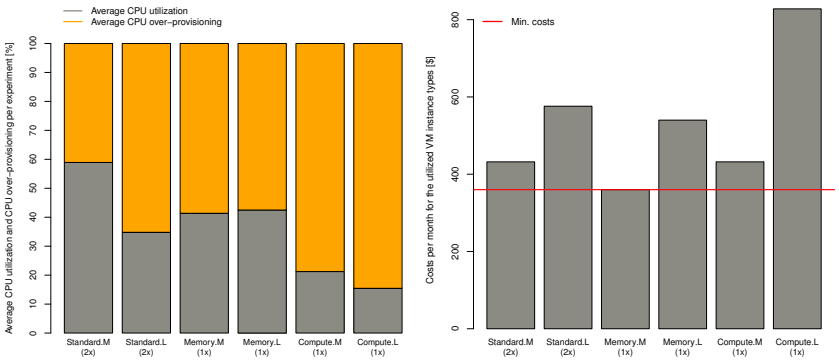


Figure 6.4. Response times and CPU utilization for each VM instance type. Two instances were used for Standard.M (a) and Standard.L (b) (cf. [Frey and Hasselbring 2011b])



(a) Average CPU utilization per conducted experiment

(b) Extrapolated operational costs per month

Figure 6.5. Characteristics of system configurations that utilize the minimum nr. of VM instances necessary to satisfy the SLA (cf. [Frey and Hasselbring 2011b])

cloud environment. These challenges form additional basic technical difficulties of cloud migration projects that also need to be addressed by SaaS providers when migrating existing systems to the cloud and reworking them for optimized alignment. The example scenario emulates a common approach to minimize the migration effort and to obtain working results in a short period. It deploys the corresponding software system to coarse grained IaaS building blocks (VMs). After altering the persistency layer, the existing system can be used in a cloud environment.

However, the presented experiment highlights several open issues. Running an existing application in the cloud does not imply relief of under- and over-provisioning concerns as such. Instead of supplying inappropriate physical on premise hardware configurations, the under- and over-provisioning of resources can easily be migrated to a cloud environment itself. For example, an inappropriate number of VM instances or unsuitable VM instance types could be employed. Figure 6.3 demonstrates the resource under-provisioning in our example scenario. The hardware configuration of Standard.M and Standard.L VM instance types is too restricted for utilizing

just a single instance. In this case the response times exceed the defined limit and cause a violation of the SLA.

Moreover, this scenario shows the constrained scalability of an application running in a cloud environment. The operation in a cloud does not solve scalability issues per se. For example, an IaaS-based application often needs to have built-in self-adaptive capabilities for leveraging a cloud environment's elasticity. In contrast to the former example, Figure 6.5a gives evidence for the over-provisioning of cloud resources. Our experiment resulted in a maximum of average 84% over-provisioning of CPU resources for the Compute.L VM instance type implicating more than doubled operational costs compared to the possible minimum (see Figure 6.5b).

Nevertheless, the effects on additional expenditures cannot simply be evaluated according to the over-provisioning of resources. They depend on other factors, such as the selected VM instance type, and do not necessarily scale linearly, as can be seen in Figure 6.5a, for instance. Comparing different cloud vendors would additionally complicate a cost estimation, as the different price models and VM instance type configurations impede the assessment of real world usage scenarios as well. Hence, a better support for anticipating the operational costs without limiting the modeling capabilities to, for example, a set of specific cloud environments, a set of particular configurations, or resource types as VMs is needed. This is especially the case when incorporating PaaS cloud environments, which follow other design paradigms and offer basic building blocks that differ from the VMs used in IaaS-based clouds. Furthermore, our example scenario utilizes a repeating usage pattern as well as homogeneous VM instance types and a constant number of VM instances during an experiment run.

This is likely to change in real world scenarios and adds additional complexity in evaluating migration alternatives and estimating the related costs. Further difficulties may arise considering architectural limitations of an existing system. For example, if distribution and parallelization is omitted in the present system design, there may emerge data inconsistency issues when scaling up horizontally while joining the VM instances to an existing data persistency layer. Moreover, exhibiting a reproducible short burst in

response times after leaving behind a local minimum in the number of requests (see Figure 6.4), the experiment revealed an unexpected behavior of the application running in the cloud. Hence, some effects may generally be hard to predict and therefore require profound evaluation.

As mentioned before, the scalability issues as well as challenges regarding under- and over-provisioning are most often not solved by merely deploying an existing software system in a virtual machine and running it in an IaaS cloud environment. Therefore, we argue that migrating typical enterprise software to a cloud-based application usually implies an architectural restructuring step for aligning it with a cloud environment and exploit the cloud's offered advantages. However, knowledge about the internal structure of an existing software system is often not available and therefore an architectural model has to be reconstructed first. The architectural model serves as a starting point for restructuring activities towards a cloud-optimized target architecture, which at the moment most often has to be created manually. This often is not an easy task, as construction of the advanced architecture usually presumes profound comprehension of the existing one. Furthermore, the target architecture must comply with the specific cloud environment's offered resources and imposed CECs, for example application frameworks and limitations of programming interfaces in PaaS-based cloud environments, respectively.

Besides the need for an automated detection of the *CEC violations*, a mapping model that describes the relationships between system parts of the status quo and a target architecture is required as well. Future workload in combination with the target architecture arrangement will determine resource utilization of the cloud environment during operation. As most cloud providers follow the paradigm of utility computing, and therefore, charge resource utilization on a pay-as-you-use basis, the arrangement of the target architecture has a direct impact on the operational costs.

Identified Challenges Addressed by CloudMIG

To identify the eventual set of the cloud migrations' difficulties that are addressed by CloudMIG and that subsume the problem descriptions from

Section 1.2, on the one hand, we condense the challenges described before. On the other hand, the set of addressed challenges is completed with the help of a literature review. For a detailed investigation of related migration approaches and corresponding challenges, we refer to Section 13.1. In summary, the challenges that are addressed by the cloud migration method CloudMIG can be subsumed as follows:

C1 **Applicability [Greenwood2010; Kim et al. 2009; Ward et al. 2010; Khajeh-Hosseini et al. 2010; Tran et al. 2011a; Zardari and Bahsoon 2011]:** Solutions for migrating and aligning enterprise software systems to cloud-based applications are limited to particular cloud providers or particular system technologies.

C2 **Level of automation [Ward et al. 2010; Zhou et al. 2010a; Tran et al. 2011a; Babar and Chauhan 2011]:** To align existing systems with a cloud environment and to enable them to exploit the cloud's offered advantages, a reengineering step is required. Here, a target architecture and a mapping model currently often have to be built manually. Additionally, the target architecture's violations against the cloud environment's constraints are not identified automatically at design time.

C3 **Resource efficiency [Khajeh-Hosseini et al. 2010; Hajjat et al. 2010; Shimba 2010; Tran et al. 2011a; Misra and Mondal 2011]:** Various migrated software systems are not designed to be resource-efficient and do not leverage the cloud environments' elasticity, because even transfering an established application to a new cloud environment can be a cumbersome task itself. Over- and under-provisioning of resources is a challenge in cloud environments, too. Furthermore, means for evaluating a target architecture's dynamic resource utilization at design time are most often inadequate. This even strengthens the general problem that estimating the future operational costs for arbitrary cloud environments is difficult.

C4 **Scalability [Greenwood2010; Ward et al. 2010; Misra and Mondal 2011]:** Scalability remains a concern in cloud environments as well. Automated support for evaluating a target architecture's scalability at design time is rare in the cloud computing context.

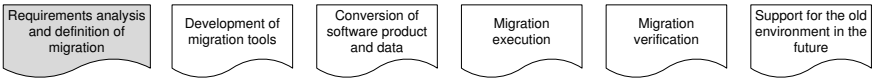


Figure 6.6. Single addressed item (gray) in a migration plan that follows the ISO/IEC 14764 standard (based on [ISO/IEC/IEEE 2006])

6.1.2 CloudMIG and the ISO/IEC 14764 Standard

CloudMIG addresses the planning phase of a cloud migration. Hence, typical concerns at this stage include the comparison of various cloud environments, the selection of the best suited cloud environment candidate, and the definition of a mapping to a new target architecture, for instance. In Section 3.2.3, the ISO/IEC 14764 Standard was described that covers common software maintenance activities. As planning the migration of enterprise systems to the cloud is a specific maintenance activity, this section relates the CloudMIG method with the ISO/IEC 14764 Standard to explain the broader context of maintenance activities and the overlapping with this standard. Regarding general migration projects, the ISO/IEC 14764 Standard employs the notion of a *migration plan* that can be used to guide and manage the migration. The migration plan includes coarse-grained *items* (see Section 3.2.3 for further details). As illustrated in Figure 6.6, the CloudMIG method addresses, deliberately, just one out of six items.

The corresponding definition of a cloud migration and the accompanying requirements analysis are in the scope of the CloudMIG method. The definition of a cloud migration can be seen as one or more planning activities, that are, in general, also addressed by CloudMIG. Consequently, creating an aligned software architecture and deployment model for a specific cloud environment falls into the category of defining an element of the migration. The same applies to CloudMIG's conformance checking approach. The detected *CEC violations* regarding the selected cloud environment induce specific requirements, as those uncovered *CEC violations* should—optionally or mandatory, see Chapter 7—be fixed during the conversion of the software product (see the corresponding item in Figure 6.6).

The ISO/IEC 14764 Standard also describes task-steps that should be addressed by a migration. They were already listed in Section 3.2.3. In the following, we revisit these task-steps and highlight the ones that are relevant in the context of the CloudMIG method. Those task-steps are marked bold in the following list.[2]

1. Analyze the migration requirements

2. Determine the impact of migrating the software product

3. Establish a schedule for performing the migration

4. Identify data collection requirements for post-operation review

5. Define and document the migration effort

6. Determine and mitigate risks

7. Identify needed migration tools

8. Identify support for the old environment

9. Develop and/or acquire migration tools

10. Incrementally decompose software products and data for conversion

11. Prioritize conversion of software products and data

12. Convert software products and data

13. Migrate software products and data to new environment

14. Run parallel operations

15. Verify migration through testing

16. Provide support for old environment

The requirements analysis was already covered in the context of a migration plan's items. Therefore, the similar task-step 1 is likewise addressed by

[2]The numbering was introduced by us. It is not used for ordering, but for referencing purposes only.

CloudMIG. However, for this and the following addressed task-steps, there, obviously, exist further aspects that play a role during an actual migration project, but those are out of CloudMIG's scope. For example, identifying all stakeholders is an essential component for requirements analysis [Sharp et al. 1999], but is not covered by the CloudMIG method.

Though, the method can contribute to determine the impact of migrating a software product (task-step 2) to a new cloud computing environment. For example, the simulation can indicate future costs and the conformance checking approach is able to reveal faults and supports preventing potential errors [Avizienis et al. 2004] before the actual migration.

The migration effort is influenced by the automated creation and evaluation of potential target architectures and the efficient identification of *CEC violations*, for instance. Hence, the conformance checking approach and the deployment and reconfiguration optimization can support the definition of the migration effort, as is involved in the task-step 5.

Similarly, an early identification of *CEC violations* and the evaluation of different target architectures mitigates the related risks of a migration project. Thus, CloudMIG assists during the task-step 6 as well. An actual migration is usually executed with the help of an incremental strategy [Richardson et al. 1997]. CloudMIG can be employed to identify the corresponding increments—e.g., by means of simulation or identification of interdependent *CEC violations*—as is involved in the task-step 10.

Analyzing the task-step 11, CloudMIG can also be used to support the prioritization of the software product and data conversion. For example, system parts that exhibit a higher number of *CEC violations* induce, by tendency, a higher level of uncertainty. To mitigate risks as early as possible, those parts should usually be converted primarily (see Section 7.4).

The remaining task-steps are out of the method's scope. For example, they cover the development of other tools (task-step 9), the actual conversion activity (task-step 12), or the quality assurance process posterior to the actual conversion (task-step 15).

6.2 Cloud Migration Types

As stated in previous sections, CloudMIG does not address migrations to pure SaaS-based cloud environments through merely substituting local software with leased, web-based applications, for instance. To facilitate reasoning about characteristics regarding the targeted forms of cloud migrations, this section introduces the notion of *cloud migration types* and shows, which of these types are supported by CloudMIG.

Before we detail the concept of cloud migration types, we introduce three basic terms that play an important role in the context of the CloudMIG method in general, and that are relevant for explaining cloud migration types, too. These terms are *system under analysis*, *status quo deployment*, and *status quo node*. They relate to the current on premise deployment of a software that should be migrated to a cloud environment.

Definition: System Under Analysis (SUA)

A system under analysis (SUA) is an existing software system that ought to be migrated to a cloud environment. An SUA is analyzed with respect to its suitability for a deployment in the cloud.

Regarding a *status quo deployment*, the phrase "status quo" is used, as this deployment describes the status quo situation of a SaaS provider, i.e., the starting point of a migration.

Definition: Status Quo Deployment

A status quo deployment describes how the SUA is currently deployed, i.e., which parts of the existing software currently run on which server machines.

Similarly to the contents of UML deployment diagrams [Object Management Group 2010], relevant information regarding a status quo deployment is given by the mapping of software components (manifested in software artifacts) to computing nodes, hardware details of these computing nodes,

and interconnections between these computing nodes, for instance. Consequently, these computing nodes are denoted *status quo nodes*.

Definition: Status Quo Node

A status quo node is a computing node in a status quo deployment that hosts some or all components of the SUA.

Incorporating the introduced notions of SUAs, status quo deployments, and status quo nodes, the definition of a cloud migration type reads as follows.

Definition: Cloud Migration Type

A cloud migration type describes three characteristics of a specific cloud migration: 1. The *scope* of a cloud migration, i.e., if an entire SUA or only some parts of it are supposed to be migrated to the cloud. 2. Whether the status quo nodes in a status quo deployment already utilize virtualization technology for server consolidation purposes. 3. The service model of the selected target cloud environment, i.e., PaaS or IaaS.

The cloud migration types are termed according to these three characteristics that form their specific properties. The naming scheme is defined according to the Table 6.4, where the name of a cloud migration type is given by the character combination of the first column (read top-down). For positions where no character is used, we utilize $\varnothing$.

CloudMIG supports eight cloud migration types that are illustrated in Figure 6.7. We demonstrate the use of the naming scheme with the help of the cloud migration type *SN2I*. An example is shown in Figure 6.8. In this example, a system with a common three-tier architecture should be migrated to the cloud. Accordingly, the status quo deployment consists of three types of status quo nodes (*Presentation Tier*, *Logic Tier*, and *Data Tier*), that host the components *Rich Interface*, *Application Logic*, and *Database*, respectively.

In the example of Figure 6.8, only the Logic Tier and Data Tier ought to be migrated (S**N**2I) to an IaaS-based cloud environment (SN2**I**) and the Logic Tier and Data Tier do not already utilize virtualization technology (*V* is

Table 6.4. Naming scheme of cloud migration types

Character	Description
S	Fixed character that stands for "status quo deployment."
$C\|N$	If the status quo deployment ought to be completely migrated to the cloud environment, this is indicated by `C`. If it should not be migrated completely (only parts), this is indicated by `N`.
$V\|\varnothing$	Indicates if the status quo deployment already utilizes virtualization technology (`V`) or not ($\varnothing$).
2	Fixed character that stands for "to" (a migration of the status quo deployment **to** a target cloud environment).
$I\|P$	The service model of the target cloud environment. CloudMIG supports IaaS- (`I`) and PaaS-based (`P`) cloud environments.

omitted). The naming scheme reveals that CloudMIG does not support the migration to pure SaaS-based clouds and also not to hybrid clouds. Hence, multi-cloud provider scenarios are left for future work. Furthermore, Figure 6.8 shows another important concept of the CloudMIG method. CloudMIG does not address the actual migration of an enterprise system to the cloud, but investigates competing target architectures with the help of corresponding models. As detailed in Chapter 8 and specified in the naming scheme of cloud migration types above, some parts of an enterprise system can remain deployed on previously used on premise machines. Figure 6.8 demonstrates a corresponding use case for such a scenario, where a rich client interface component, that provides the frontend to an enterprise application, remains deployed on premise. This corresponds to the concept of *attached services* that describes a local application that uses remote services or applications in the context of a cloud migration [Kaisler and Money 2011].

The CloudMIG method assumes the implicit creation of appropriate connector components if only a subset of an enterprise application's components are deployed to a cloud environment. Hence, for this purpose, the example of Figure 6.8 adds an *Application Logic Stub* and an *Application Logic Skeleton* to the *Presentation Tier* and the *Logic Tier*, respectively.

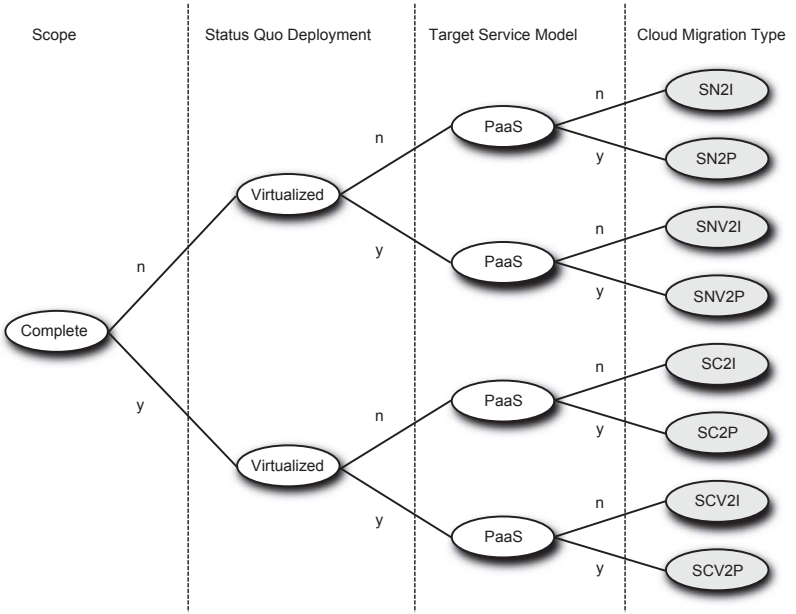


Figure 6.7. Cloud migration types

The name and purpose of the new components draw on the *stub* and *skeleton* components that are often used in the context of web services [Alonso et al. 2004]. A stub is utilized to marshal information that is to be transferred from a local machine to components running in the cloud environment. Datagrams that are received on premise server machines from software that is deployed to a cloud environment is unmarshaled and forwarded for local processing. A skeleton component is used for the corresponding tasks on the side of components running in the cloud.

Those stub and skeleton components are implicitly included in the model of a target architecture to allow remote communication. For executing the actual migration, a suitable technology for implementing those components has to be selected and integrated.

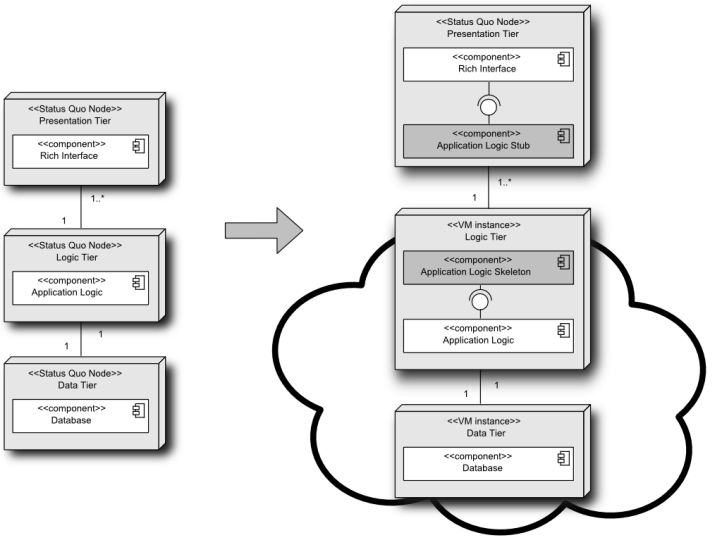


Figure 6.8. Exemplary cloud migration type *SN2I*. The logic tier and the data tier of an existing three-tier architecture (left) are migrated to an IaaS-based cloud environment (lower right). As the presentation tier that hosts a rich interface remains deployed on premise, a stub- and a skeleton component have to be added to allow remote communication.

6.3 Fundamental Approach

This section details the fundamental approach for supporting SaaS providers during the planning phase of a cloud migration. It includes, among others, CloudMIG's basic design, its activities, roles, important concepts, and elementary models. This section also describes how the conformance checking approach and the deployment and reconfiguration optimization are embedded in the overall method. The remainder of this section is organized as follows.

Section 6.3.1 describes the basic design of CloudMIG that is structured in six activities. These activities are detailed in Section 6.3.2. There exist different

incentives and views in utilizing CloudMIG that are addressed with the help of individual roles. These roles are introduced in Section 6.3.3. Finally, Section 6.3.4 describes the Cloud Environment Model (CEM) that enables specifying cloud environments and their characteristics, and that lays the foundations for the definition and detection of CECs and the generation and optimization of target architectures.

6.3.1 Basic Design

Central challenges that should be tackled with the help of the CloudMIG method are defined in Section 6.1.1 (challenges C1-C4). Hence, the overall design of CloudMIG has to reflect these challenges, i.e., it has to include models and arrange mechanisms that enable to approach the challenges.

Before we explain CloudMIG's basic design, we introduce the essential concept of a *cloud profile*. A basic requirement for tackling the identified challenges stems from the principal conception of the conformance checking approach and of the optimization of target architectures: To specify CECs and to enable the definition of a deployment architecture, a suitable meta-model is needed. For this purpose, CloudMIG defines the CEM that can describe the characteristics of a specific cloud environment. Such specifications of particular cloud environments are called *cloud profiles*.

Definition: Cloud Profile

Cloud profiles describe the specifics of a particular cloud environment from the perspective of a cloud user. For example, they include the available services, a pricing model, and a list of CECs.

For example, the cloud environments Amazon EC2[3] and Microsoft Windows Azure[4] can be described by specific cloud profiles. The underlying meta-model CEM is detailed in Section 6.3.4.

[3] http://aws.amazon.com/ec2/

[4] http://www.windowsazure.com/

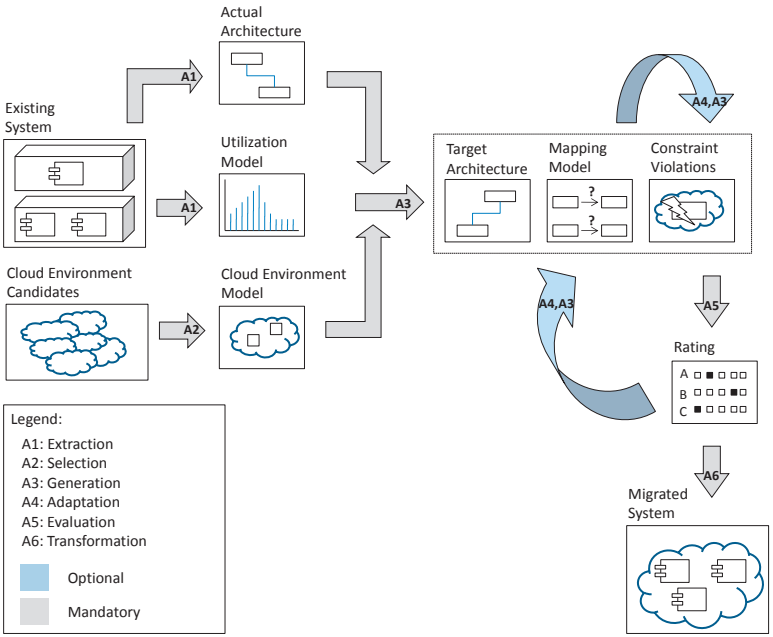


Figure 6.9. The CloudMIG method (based on [Frey and Hasselbring 2011b])

Turning to CloudMIG's basic design, Figure 6.9 illustrates the method's six major activities A1-A6. These activities are briefly described in the following and will be detailed later in Section 6.3.2.

- ***A1* Extraction:** Includes the extraction of architectural and utilization models of the existing enterprise system (SUA). They base upon the ADM efforts from the OMG and utilize KDM and SMM.
- ***A2* Selection:** Selection of an appropriate *CEM*-compatible cloud profile candidate. Relevant criteria can be the preference of renowned cloud providers, the number of defined CECs that are included in a cloud profile, or a defined cost structure, for instance. This activity can be performed in parallel with A1.

- ***A3* Generation:** The generation activity produces the target architecture and a mapping model. Furthermore, a model describing the target architecture's *CEC violations* is created. The violations are detected with the help of the conformance checking approach that is described in Chapter 7. The target architecture itself comprises the target software architecture, deployment model, and runtime reconfiguration rules. The generation and optimization of those models is described in greater detail in Chapter 8.
- ***A4* Adaptation:** The adaptation activity enables a SaaS provider to manually adjust the target architecture.
- ***A5* Evaluation:** The evaluation activity involves static analyses and a runtime simulation of the target architecture.
- ***A6* Transformation:** The actual manual migration towards the target architecture that was generated, adapted, and evaluated in the previous activities A3-A5. CloudMIG provides no further support for performing A6. It is very likely that A6 will include substantial quality assurance activities in real migration projects.

Within the frame of this section, we will now connect the activities with the challenges C1-C4 that were described in Section 6.1.1, as each activity and related model has to (1) support meeting the challenges, or (2) support the integration of CloudMIG into the general cloud migration process. For convenience, we briefly replicate the challenges below.

C1: Applicability

C2: Level of automation

C3: Resource efficiency

C4: Scalability

Table 6.5 traces CloudMIG's activities A1-A6 to the challenges C1-C4. The construction of an activity can be motivated by several challenges, however, activity A6 is an exception. As can be seen in Table 6.5, it is not traced back

Table 6.5. Activities to challenges mapping

Activity		
Symbol	**Name**	**Addressed Challenge(s)**
A1	Extraction	C1,C2,C3,C4
A2	Selection	C1,C2
A3	Generation	C1,C2,C3,C4
A4	Adaptation	C1,C3,C4
A5	Evaluation	C1,C2,C3,C4
A6	Transformation	-

to any challenge because we, as mentioned before, deliberately omitted the actual conversion task there. Nevertheless, we included A6 in the method to clearly indicate the phase in a cloud migration process when applying CloudMIG is sensible, i.e., during the planning phase of a cloud migration before the actual conversion of a software product. Therefore, we omit A6 in the following explanation of the mappings.

The challenges C1 and C2 are relevant for each of the activities A1-A5. All activities contribute a particular facet that facilitates the independence from specific cloud environments or specific technologies (C1). For example, A1 covers the extraction of language-agnostic code models, A2 enables the utilizability for diverse cloud environments, and A3 can optimize target architectures regarding arbitrary resource configurations, workload patterns, and pricing models. Similarly, all activities except A4 provide or foster some sort of automatism to create associated models (C2). For example, A1 can automatically reverse-engineer source code to KDM models, A2 frees SaaS providers from manually investigating all cloud environment candidates, and A5 provides means for simulating the consequences of different runtime reconfiguration rules. Hence, we concentrate on explaining the activities' mappings to C3 and C4 in the following.

The activity A1 (*Extraction*) includes the provisioning of usage data from monitoring log files. This provisioning does not launch an assessment or improvement of resource efficiency (C3) or scalability (C4) itself. How-

ever, that information can effectively support assessing and improving resource efficiency and scalability in subsequent activities based on realistic assumptions concerning the expectable load. For example, the results from simulating a target cloud deployment are far less useful when assuming a widely differing demand, even if the simulation is accurate. A deployment architecture and reconfiguration rules that exhibit resource efficiency concerning a considerable load may yet result in over-provisioning and wasting of resources during slack periods, for instance.

Consequently, the generation of a potent target architecture and the mapping of existing components to this new architecture (A3) are crucial components of CloudMIG. Here, resource efficiency (C3) and scalability (C4) have to be considered. For example, resource efficiency can be improved by exploiting the elasticity of a cloud environment through adequate runtime reconfiguration rules. Furthermore, a suitable way to scale existing software in a new cloud computing environment is to distribute its components to appropriate resources and to trigger automatic reconfiguration if variations in resource demand exceed certain thresholds. Moreover, the activity A3 includes the detection of *CEC violations*. As CECs can model restrictions of arbitrary cloud environments and can, to a great extent (see Chapter 7), be detected automatically, the conformance checking approach also contributes tackling C1 and C2.

Similarly, as SaaS providers may want to manually adapt generated target architectures (A4). The possibility to change cloud deployment models and runtime reconfiguration rules provides means to further improve resource efficiency (C3) and scalability (C4). This is especially the case when combined with CloudMIG's evaluation (A5) capabilities for assessing adapted target architecture candidates. Hence, C3 and C4 constitute strong stimuli for including A5 in the CloudMIG method as well.

CloudMIG and Common Migration Methodology The Section 3.2.2 describes the concept of migration approaches such as *Chicken Little* and *Butterfly*. Those approaches define a framework for actually converting a legacy system towards a new target environment. The focus is on the transi-

tion phase itself. For example, the Chicken Little approach uses gateway components that are placed between components of an existing application to enable an incremental migration. However, those approaches often do not detail the reverse engineering step that is needed to obtain a realistic architectural model that can serve as a starting point for restructuring activities.

In contrast, CloudMIG focuses on those activities that are relevant before performing an actual conversion. For example, it focuses on reverse-engineering static and dynamic aspects of an SUA, generating and optimizing a resource-efficient target architecture model, and searching potential pitfalls through detecting CECs. Because of this orientation, CloudMIG does not further specify activity A6. SaaS providers applying CloudMIG can choose an appropriate migration method for the actual conversion. Here, best practices should be considered in any case, e.g., splitting the migration process in manageable increments and employing appropriate quality assurance techniques. Nevertheless, already Section 6.1.2 described CloudMIG's embedding in a universal migration process. Similarly, Figure 6.10 illustrates the mapping of the introduced basic activities A1-A6 to a migration according to the general structure of the Horseshoe Model of Reengineering (see Section 3.1.2). Here, CloudMIG mainly covers the reverse engineering step and the model-based architectural transformation.

6.3.2 Activities

The CloudMIG method comprises the six activities A1-A6. They are detailed in the following.

A1: Extraction

CloudMIG aims at the migration of established enterprise applications. It is far from uncommon that those applications are used over many years or even decades [Seacord et al. 2001]. This is especially the case if the applications are business-critical. Their development might have been started

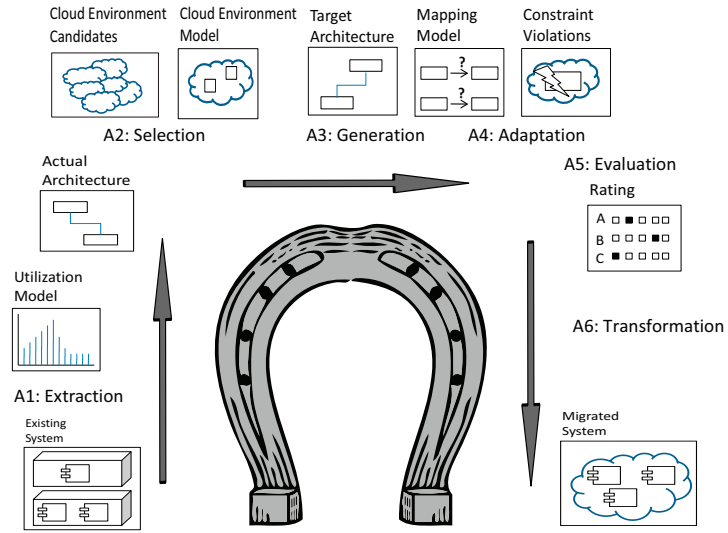


Figure 6.10. CloudMIG and the horseshoe model of reengineering

long ago and possibly the initial developers retired or moved to another employer, or the support for applied technologies ended in the meantime, for instance. Usually, the inherent complexity continually increases during software development [Mens 2012], while the architecture of software systems tends to simultaneously erode over time [Bennett and Rajlich 2000]. Therefore, initially envisioned architectures frequently diverge from actual implementations and the knowledge about the internal structure is often incomplete, erroneous, or even missing. As CloudMIG generates and optimizes target architectures on the basis of a status quo deployment model, a representation of the software system's actual architecture has to be available early in the course of a cloud migration.

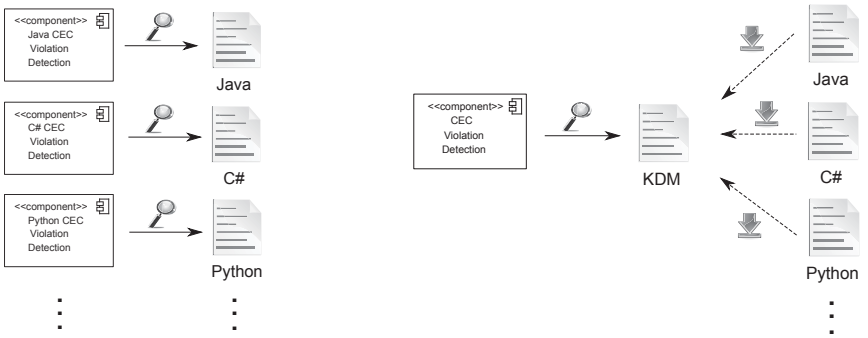
A corresponding software architecture meta-model has to be aligned with the challenges C1 and C2. Regarding C1, it must be possible to reverse-engineer the software to a model that includes its artifacts, source code, and architectural elements in a widely technology- and language-independent

way. This enables generic analyses and the creation of target architecture candidates regardless of specific techniques. For example, instead of implementing CEC violation detection capabilities for each relevant programming language, it is reasonable to transform the systems' source code to abstract models that can be used as a basis for generic detection mechanisms. Employing abstract models in modernization efforts is a widespread approach. For example, this is demonstrated by the DynaMod project [van Hoorn et al. 2011] that addresses the model-driven modernization of software systems and augments the abstract models with information resulting from a dynamic analysis. In general, an accompanying benefit of a common format and joint analysis techniques includes, of course, raising the potential level of automation (C2).

We propose using OMG's KDM as a corresponding software architecture metamodel. Figure 6.11 illustrates the above-mentioned simplification of the *CEC violation* detection mechanism when transitioning from a per-language conformance checking (Figure 6.11a) to a generic KDM-based variant (Figure 6.11b), where the number of required detection components is finally reduced to a single instance.[5] In return, language-specific KDM extractors have to be present. Though, this is justified as the KDM models are also used as a foundation for generic restructuring operations, generically measurable system descriptions, and consolidated input format for simulating software that is deployed to cloud environments.

KDM was especially designed with respect to the domain of architecture-driven software modernization (see Section 3.3) and for representing all aspects of a software system that may be relevant in that regard. Furthermore, there already exists an appropriate framework for extracting basic KDM models from Java-based software systems. Support for further programming languages can be incorporated with the help of plugins. This framework, support for further programming languages via plugins, improvements, and integration into *CloudMIG Xpress* are described in Chapters 9 and 10.

[5] Indeed, lean language-dependent detection components are needed anyway. However, as described in Section 7.2, those components are required only for a limited number of specific CECs.



(a) Per-language conformance checking based on source code

(b) Conformance checking based on extracted KDM models

Figure 6.11. Benefit of using KDM models for conformance checking. Instead of providing a full-fledged component for the detection of *CEC violations* for each specific programming language (a), a generic single component can be reused when employing extracted KDM models (b).

The extracted KDM-based software architecture models that serve as a basis for CloudMIG's target architecture generation and conformance checking approach mainly rely on KDM's *Source*, *Code*, and *Action* packages. This provides sufficient information for modeling the necessary level of detail. To give a simple example, methods and their defining class can be represented with the *Code* package's `MethodUnit` and `ClassUnit` elements, respectively. The file that contains these source code elements is modeled with the help of the *Source* package's `SourceFile` elements. Furthermore, method calls among the aforementioned methods can be represented using the *Action* package's `Calls` element.

Considering the basic requirements that are posed by the CloudMIG method on a possible software architecture model candidate, the information included in KDM's *Source*, *Code*, and *Action* packages provide an adequate level of detail. For example, the conformance checking approach makes an extensive use of type information and elements that describe the usage of

this type information under certain conditions, for instance, the occurrence of a type in the signature of a method. Here, the *Code* and *Action* packages are ideally suited for modelling those information. Additional information below the sub-statement level—such as the indentation of text blocks or number of blanks between tokens—is often missing in KDM. However, as those information that is missing in standard ASTs is also out of CloudMIG's scope, this does not impede employing KDM as a central foundation for the CloudMIG method.

Besides using the extracted KDM models for conformance checking purposes, they form the basis for restructuring software system models towards potential cloud deployment models. On the one hand, these models comprise a KDM-based model that represents the newly generated software architecture. On the other hand, a corresponding instance of CEM is used for describing the mapping of this software architecture to resources of a specific cloud environment. Hence, the newly created software architecture candidate is then interleaved with elements describing a concrete cloud environment. For further details regarding this mapping mechanism, we refer to Section 6.3.4.

As described before, employing KDM enables the utilization of language-independent analysis and restructuring operations, for instance. However, only programming languages that follow the object-oriented paradigm are currently integrated. It remains to investigate potential limitations in future work when expanding CloudMIG's scope to languages that follow other paradigms, such as functional programming.

For leveraging the commonly applied utility computing paradigm, the target architectures generated by the CloudMIG method have to be laid out resource-efficient and elastic. Therefore, besides the extraction of a KDM-based software architecture model, the method includes the extraction of an SUA's utilization model acting as a starting point.

The utilization model (resp. its meta-model) includes statistical properties concerning user behavior like service invocation rates over time or average submitted datagram sizes per request. Relevant information can be retrieved from various sources. For example, considering log files or

instrumenting the given system with our tool Kieker [Rohr et al. 2008; van Hoorn et al. 2009; van Hoorn et al. 2012] for setting up a monitoring step constitute possible techniques. Furthermore, the utilization model contains application-inherent information related to proportional resource consumption. Metrics of interest include a method's cyclomatic complexity or memory footprint.

We propose OMG's Structured Metrics Metamodel (SMM) [Object Management Group 2012] as a foundation for building the related meta-model (see Section 3.3.3 for a description of SMM). The current version of our tool *CloudMIG Xpress* merely incorporates information regarding a system's actual usage, i.e., the extent and characteristics of the load a system was exposed to over a particular timeframe. This usage data is included in so-called *workload profiles*.

Definition: Workload Profile (based on CloudMIG Xpress Project [2012])

A workload profile describes the usage pattern of a client/ server system in terms of client requests. The clients call methods. A workload profile contains a list of records, each record representing characteristics of a single method call. The record includes a timestamp, the fully qualified method name, a response time, and an identifier specifying the host where a specific method was executed. It is also possible to link several method calls to represent complete traces.

Compared to *operational profiles*, a workload profile includes more detailed information regarding a set of method calls. Operational profiles merely contain aggregated probabilities of usage events, for example, method calls [Musa 1993]. Those probabilities can, of course, be computed on the basis of workload profiles.

Using SMM to specify the utilization model and an included workload profile aligns with the meta-model KDM that is chosen for representing a system's actual software architecture. Both come from OMG's ADM initiative and are well integrated. Furthermore, SMM is also used in the context of

CEC violation detection. Hence, available means for metrics execution and integration can be reused.

A2: Selection

Common properties of cloud environments are described in the meta-model CEM. Selecting a cloud provider specific environment as a target platform for planned migration activities therefore implies the selection of the cloud profile that is the corresponding instance of the CEM. For example, the CEM comprises entities like VM instances or worker threads for IaaS and PaaS-based cloud environments, respectively. As a result, for every cloud environment, which shall be targeted with CloudMIG, a corresponding instance of CEM has to be created once beforehand. Mapping rules define possible relationships to the architecture meta-model KDM for specifying diverse deployment architectures.

Furthermore, the cloud profiles can include constraints imposed by cloud environments restricting the reengineering activities (CECs). For example, the opening of sockets or the access to the file system are often constrained. The CEM is described in Section 6.3.4. If near-optimal CDOs ought to be searched among various cloud environments with the simulation-based genetic algorithm CDOXplorer (cf. activity A3 below), multiple cloud profiles can be selected in A2 as well.

A3: Generation

The generation activity produces three artefacts, namely a target architecture, a mapping model, and a model characterizing the target architecture's violations of the CECs. These constraint violations explicitly highlight the target architecture's parts which have to be redesigned manually by the reengineer (cf. A6). The mapping model assigns elements from the actual architecture to those included in the target architecture. The latter comprises a unification of the restructured actual software architecture and the specifically utilized cloud resources together with the means used for dynamic

resource scaling. Regarding the 4+1 architectural view model [Kruchten 1995], the target architecture therefore provides information regarding the development and logical view (restructured actual software architecture) and regarding the physical view (deployment model including instructions for dynamic resource scaling).

We define the two following approaches G1 and G2 that enable the generation of target architectures that are improved regarding C3 and C4:

G1: Simulation-based evolutionary optimization

G2: Rule-based restructuring

The generation of a resource-optimized target architecture with the help of G1 is a core contribution of this thesis. Hence, the deployment and reconfiguration optimization approach with the help of an evolutionary algorithm that employs simulation is described in greater detail later in Chapter 8. Likewise, a further core contribution of this thesis that covers the detection of CEC violations is also settled in this activity A3 and is described in Chapter 7.

Besides these core contributions, the other approach G2 for generating target architectures uses a rule-based heuristic to steer the way towards a well-suited solution. In contrast to G1 that is extensively described, analyzed, and validated, we just briefly outline G2 below. For the generation of the target architecture, G2 proposes the three phases P1-P3 that are illustrated in Figure 6.12. The phases are constructed as follows.

P1 - Model transformation: The phase P1 produces an initial assignment from elements of the existing architecture to cloud-specific elements available in the CEM. The initial assignment is created applying a model-to-model transformation according to the transformation rules that could be included—in the case G2 would be used—in the selected CEM instance (cf. activity A2). Hence, a cloud profile would also specify the basic permitted mapping from software architecture elements to cloud resources.

P2 - Configuration: The phase P2 serves as a configuration of the algorithm used for obtaining an actual mapping of architectural elements in the phase

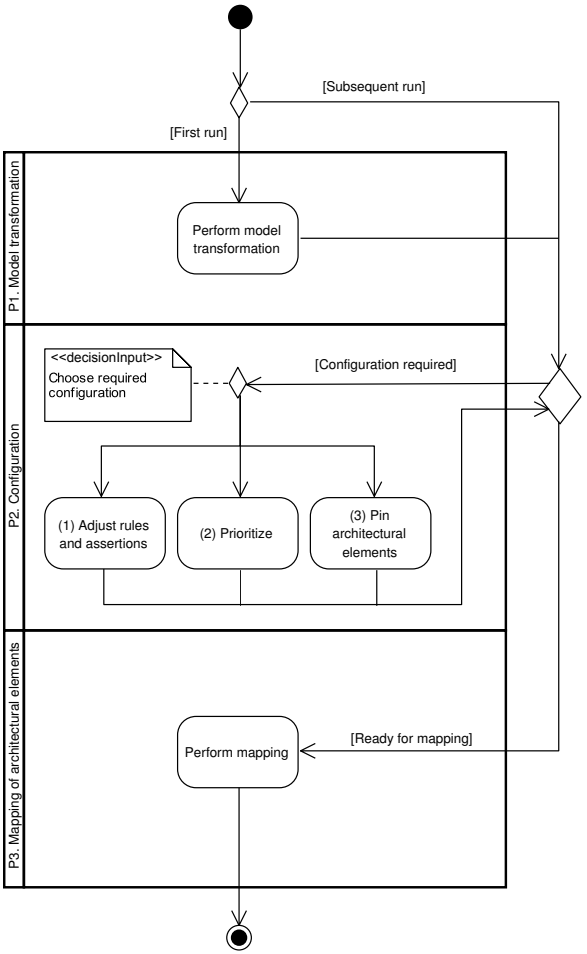


Figure 6.12. Target architecture generation process G2 (cf. [Frey and Hasselbring 2011b])

P3. During P2, a reengineer may adjust rules and assertions for heuristic computation (cf. P3). A rule could be formulated like the following ex-

amples: “Distribute the five most frequently used services to own virtual machines” or “The server methods responsible for at least 10% of overall consumption of the CPU time shall be moved to client side components if they do not need access to the database.” An exemplary assertion could be: “An existing component must not be divided in more than 3 resulting components.” In addition to a set of default rules and assertions, the reengineer is given the possibility to modify those rules and assertions either via altering the respective numerical values or applying a corresponding DSL. In both cases, the rules and assertions have to be prioritized after their selection. Hereby, the reengineer determines their significance during execution of P3. This means that architectural elements which are related to higher-weighted rules are considered primarily for assignment and therefore have a stronger impact on the further composition of the target architecture. Furthermore, a reengineer may pin architectural elements. This prevents the rearrangement of previously assigned architectural elements to other target architecture components in the phase P3.

P3 - Mapping of architectural elements: The phase P3 improves the initial assignment of architectural elements generated in phase P1 referring to resource-efficiency. Therefore, the formulated rules are utilized and the compliance of the resulting architecture with the defined assertions is considered. There exists an enormous number of possible combinations for assigning architectural elements. Efficiency improvements for one resource can lead to degradation for other resources or impair some design quality attributes. For example, splitting a component’s parts towards different virtual machines can improve relative CPU utilization, but may lead to increased network traffic for intra-component communication and a decreased cohesion. Additionally, those effects do not necessarily have to move on linearly and moreover, the interrelations are often ambiguous as well. Therefore, we propose application of a heuristic rule-based approach to achieve an overall improvement. A potential algorithm is sketched in Algorithm 1 and it works as follows.

The rules are considered successively according to their priority (Algorithm 1 iterates over a list of rules (R_{Sort}) in Line 6. The list is sorted descending by the rules’ priority). Thus, in the following, rules with higher

Algorithm 1: Rule-based heuristic for creating a mapping of architectural elements that improves resource efficiency

1: $E_{Pinned} \leftarrow$ Pinned architectural elements
2: $R \leftarrow$ All rules
3: $A \leftarrow$ All assertions
4: $R_{Sort} \leftarrow$ Sort R descending by priority
5: $E_{AllAffected} \leftarrow E_{Pinned}$
6: **for all** r in R_{Sort} **do**
7: $E_r \leftarrow$ All architectural elements delivered by r's selection criterion
8: $P_r^E \leftarrow$ Power set of E_r
9: $Score \leftarrow$ New associative array
10: **for all** p_r^E in P_r^E **do**
11: $Score[p_r^E] \leftarrow$ Rate p_r^E
12: **end for**
13: $Score_{Sort} \leftarrow$ Sort $Score$ descending by score
14: $Score_{Sort}^{Keys} \leftarrow$ Keys of $Score_{Sort}$
15: **for all** p_r^E in $Score_{Sort}^{Keys}$ **do**
16: $E_{FormerlyAffected} \leftarrow p_r^E \cap E_{AllAffected}$
17: $E_{NeedReassignment} \leftarrow$ Elements of $E_{FormerlyAffected}$ that need reassignment conc. r
18: **if** $E_{NeedReassignment} == \varnothing$ **then**
19: $A_{HigherPrio} \leftarrow$ All $a \in A$ with higher priority than r
20: **if** $\nexists a \in A_{HigherPrio}$ with r violates a **then**
21: Apply rule r to all elements in p_r^E
22: $E_{AllAffected} = E_{AllAffected} \cup p_r^E$
23: **end if**
24: **end if**
25: **end for**
26: **end for**

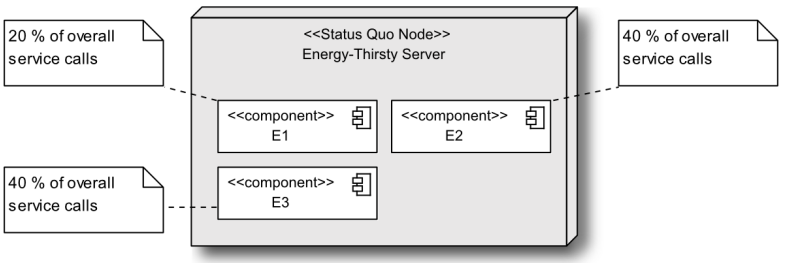


Figure 6.13. Example scenario for phase P3 of CloudMIG's activity A3 (approach G2)

priorities are weighted higher and have a stronger impact on the generated target architecture. The selection criterion of a rule is defined to deliver a set of architectural elements (e.g., the "five most frequently used services"). All possible subsets of the set are rated respective to the quality of the target architecture that would result, if the elements in the subset would be assigned correspondingly (Lines 10 and 11). This aims at considering interdependencies at the level of a single rule. For regarding interdependencies on an inter-rule level, the formulated assertions are taken into account (Lines 19-23). A rule is only applied if the reengineer did not formulate an assertion with a higher priority that would be violated after the rule's execution (Lines 20 and 21). Furthermore, the rule is applied to all mentioned subsets in order of their score (Lines 15-25). However, the rule is only utilized if no rearrangement of elements is necessary whose subset was rated higher (Line 18). The same applies to assignments that would lead to rearrangement of elements that were placed by rules of higher priority or formerly pinned elements (Lines 17 and 18).

P3 Example To demonstrate the functioning of the rule-based heuristic sketched in Algorithm 1, the example scenario that is depicted in Figure 6.13 is used. It consists of a single status quo node and three architectural elements (the components E1-E3) that are deployed on this node. In this example, we assume that the components exhibit services that can be called by external clients and that the components are known to be responsible for E1: 20%, E2: 40%, and E3: 40% of the overall service calls (workload).

These utilization values might have been revealed by conducting a dynamic analysis. Furthermore, we assume that component E_3 was pinned in the previous phase P2. Hence, the following initial values for the variables given in the Lines 1-5 of the Algorithm 1 are considered.

E_{Pinned}	$\{E_3\}$
$E_{AllAffected}$	$\{E_3\}$
$R(\widehat{=} R_{Sort})$	{The *two most frequently used components* should be deployed to separate VM instances using VM instance types similarly capable as their previously utilized status quo node (priority 2).}
A	{VM images used for starting VM instances should not consist of components that together are responsible for $\geqslant 50\%$ of the previous workload (priority 1).}

Note that E_{Pinned}, $E_{AllAffected}$, R, R_{Sort}, and A are sets. To keep the example simple, just the single rule (say $R_1 \in R$) and the single assertion (say $A_1 \in A$) are used that are stated above. Hence, $R = R_{Sort}$ and $|R| = |R_{Sort}| = |A| = 1$. Furthermore, we also assume that in the previous phase P2 a user configured the priorities 1 and 2 for A_1 and R_1, respectively. As lower numbers correspond to higher priorities, the user prefers remaining compliant with A_1 instead of executing R_1. The loop in Lines 6-26 of the Algorithm 1 is processed only once because there is only a single rule (R_1). The italic phrase of the rule R_1 (see above) highlights the rule's selection criterion that delivers a set of architectural elements E_r as can be seen in Line 7. $E_r = \{E2, E3\}$ as the two most frequently used components in Figure 6.13 are E2 and E3.

The next statement in Line 8 calculates the power set of the set E_r. Hence, $P_r^E = \{\{\varnothing\}, \{E_2\}, \{E_3\}, \{E_2, E_3\}\}$. The next step of the algorithm evaluates the target architectures that result from subsequently executing the rule R_1 for each of P_r^E's elements (Lines 10-12). That means, a single $p_r^E \in P_r^E$ is relocated and the resulting target architecture is then evaluated. Then, the next $p_r^E \in P_r^E$ is relocated (not starting from the status quo deployment,

but rather from the target architecture that resulted from the relocation of the previous p_r^E) and the score for the resulting target architecture is calculated and so on. According to R_1, the target architectures are produced by moving each $p_r^E \in P_r^E$ to a separate VM instance that is similarly capable as the status quo node in Figure 6.13. In this example, let the corresponding VM instance type be *m5.large*. The empty set that is contained in P_r^E lets the deployment of Figure 6.13 unchanged, as no components have to be relocated. Considering the element $E_2 \in P_r^E$, the target architecture depicted in Figure 6.14 that corresponds to migration type SN2I (see Section 6.2) results when executing the rule's action, i.e., when relocating the component E_2 to a VM of instance type *m5.large*. In general, each target architecture is evaluated (Line 11) and the score is stored in the associative array *Score*. In this example, we assume that the four potential target architectures result in the following scores (higher scores are better).

$Score[\varnothing] = 1.5$
$Score[\{E_2\}] = 1.8$
$Score[\{E_3\}] = 2.5$
$Score[\{E_2, E_3\}] = 2$

Score is sorted descending by the scores (Line 13) and stored in $Score_{Sort}$:

$Score_{Sort}[\{E_3\}] = 2.5$
$Score_{Sort}[\{E_2, E_3\}] = 2$
$Score_{Sort}[\{E_2\}] = 1.8$
$Score_{Sort}[\varnothing] = 1.5$

The keys of $Score_{Sort}$ are stored in the list $Score_{Sort}^{Keys}$ (Line 14):

$Score_{Sort}^{Keys} = \{\{E_3\}, \{E_2, E_3\}, \{E_2\}, \varnothing\}$

Then, the algorithm iterates over the entries of list $Score_{Sort}^{Keys}$ (Lines 15-25) to determine the best suited target architecture. However, it is not possible to simply select the winner of the previous evaluation (the first element in $Score_{Sort}^{Keys}$, E3). It has to be ensured that a target architecture does not (1) require the relocation of architectural elements that were pinned, and that it does not (2) violate previously stated assertions which have a higher

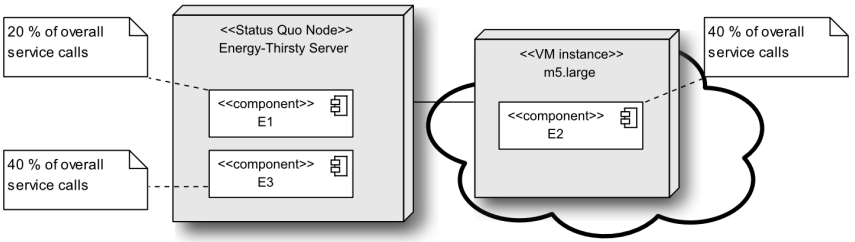


Figure 6.14. Potential target architecture of the example scenario for phase P3 in Figure 6.13

priority. In the following, the subsequent iterations of the for-loop in Lines 15-25 are explained.

1. iteration, $\mathbf{p_r^E} = \{\mathbf{E_3}\}$: Lines 16 and 17 produce the following values.

$E_{FormerlyAffected} = p_r^E \cap E_{AllAffected} = \{E_3\} \cap \{E_3\} = \{E_3\}$
$E_{NeedReassignment} = \{E_3\}$

$E_{NeedReassignment}$ is set to $\{E_3\}$ as E_3 needs reassignment concerning R ($p_r^E = \{E_3\}$) and $E_3 \in E_{FormerlyAffected}$. As $E_{NeedReassignment}$ contains an element, the next if-statement (Lines 18-24) is skipped (the user pinned E_3 in the previous phase P2, so a reassignment of E_3 is not allowed).

2. iteration, $\mathbf{p_r^E} = \{\mathbf{E_2}, \mathbf{E_3}\}$: Lines 16 and 17 produce the following values.

$E_{FormerlyAffected} = p_r^E \cap E_{AllAffected} = \{E_2, E_3\} \cap \{E_3\} = \{E_3\}$
$E_{NeedReassignment} = \{E_3\}$

As $E_{NeedReassignment}$ contains an element, the next if-statement (Lines 18-24) is skipped as in the first iteration.

3. iteration, $\mathbf{p_r^E} = \{\mathbf{E_2}\}$: Lines 16 and 17 produce the following values.

$E_{FormerlyAffected} = p_r^E \cap E_{AllAffected} = \{E_2\} \cap \{E_3\} = \varnothing$
$E_{NeedReassignment} = \varnothing$

As $E_{NeedReassignment}$ does not contain elements, the if-block is entered in Line 18 and all assertions which have a higher priority than the currently processed rule R_1 are assigned to $A_{HigherPrio}$ in Line 19. $A_{HigherPrio} = \{A_1\}$,

as the single present assertion A_1 has priority 1 and the currently processed rule R_1 has priority 2. As can be seen in Figure 6.14, when relocating $p_r^E = \{E_2\}$, the components that are deployed to the new VM instance only account for 40% of the overall service calls. Hence, assertion A_1 is not violated and the if-block can be entered in Line 20. The current rule R_1 is executed in Line 21. That means, (1) $p_r^E = \{E_2\}$ is transferred to a new VM of instance type *m5.large* and (2) this new architecture is set as a starting point for the relocation of the next elements in $Score_{Sort}^{Keys}$ or the processing of the next rule. Afterwards, $E_{AllAffected}$ is set to $\{E_2, E_3\}$ in Line 22, as previously $E_{AllAffected} = E_3$ and $p_r^E = \{E_2\}$ and $E_{AllAffected} = E_{AllAffected} \cup p_r^E$.

4. iteration, $\mathbf{p_r^E} = \varnothing$: Lines 16 and 17 produce the following values.

$E_{FormerlyAffected} = p_r^E \cap E_{AllAffected} = \varnothing \cap \{E_2, E_3\} = \varnothing$
$E_{NeedReassignment} = \varnothing$

As in the third iteration, $E_{NeedReassignment}$ does not contain elements and the if-block is entered in Line 18. Similarly, $A_{HigherPrio}$ is set to $\{A_1\}$ in Line 19. As no further VM instances are produced in the target architecture when no component is relocated because of $p_r^E = \varnothing$ (additionally to the VM instance that was previously produced in the third iteration), A_1 is also not violated in the fourth iteration. Hence, the if-block can be entered in Line 20. But as no components are relocated when applying the rule R_1 in Line 21, $E_{AllAffected}$ remains unchanged.

The heuristic algorithm now ends as all $p_r^E \in P_r^E$ were processed and all rules were executed. The resulting target architecture of Figure 6.14 is ultimately chosen by the heuristic, as it delivers the best score of all target architectures that (1) do not relocate pinned components and that (2) do not violate assertions that have a higher priority than the rules that were used to produce them.

A4: Adaptation

The activity A4 allows the reengineer to manually adjust the target architecture towards case-specific requirements that could not be fulfilled during

generation activity A3. For example, the generation process might not have yielded an expected assignment of a critical component. Furthermore, for leveraging the elasticity of a cloud environment, the reengineer might adapt a reconfiguration strategy.

A5: Evaluation

For being able to judge about the produced target architecture and the configured reconfiguration strategy, A5 evaluates the outcomes of the activities A3 and A4. The evaluation involves static and dynamic analyses of the target architecture. For instance, metrics as LCOM or WMC (e.g., described by Kan [2002]) can be utilized for static analyses. Considering the target architecture's expected runtime behavior, we employ a simulation on the basis of an extended version of the cloud simulator tool CloudSim [Calheiros et al. 2011] called CDOSim [Fittkau et al. 2012a]. As CDOSim itself is also used as a component of the simulation-based evolutionary algorithm for deployment and reconfiguration optimization, we refer to Section 8.2 for an introduction to CDOSim.

A6: Transformation

This activity comprises the actual transformation of the enterprise system from the generated and improved target architecture to the aimed cloud environment. As described before, no further support for actually accomplishing the implementation is provided as CloudMIG targets the cloud migration planning phase.

6.3.3 Roles

There exist different reasons and motivations for utilizing CloudMIG. For example, SaaS providers might want to investigate and compare alternatives for restructuring an existing system and deploying it to specific cloud environment candidates. This is the most obvious reason for incorporating CloudMIG and forms one of the fundamental incentives for institutions interested in a cloud migration.

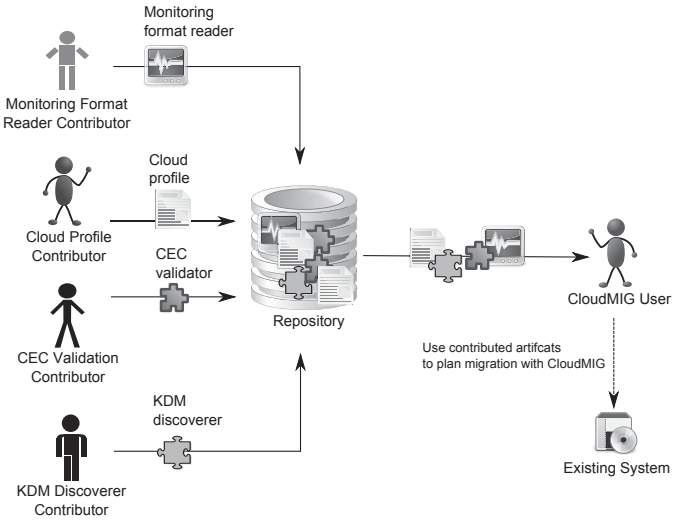


Figure 6.15. The five different roles in CloudMIG (illustrated as stickmans)

However, there exist further motivations for employing and contributing to CloudMIG, such that the available functionality is extended. These motivations can be converted to different roles, whereas a user applying the CloudMIG method can take on multiple roles. There exist five roles that are illustrated as stickmans in Figure 6.15.

The four roles on the left side of Figure 6.15 contribute artifacts to an open, publicly-accessible repository, the role on the right side of Figure 6.15 uses these artifacts while applying CloudMIG. The contributed artifacts can be submitted online. A supporting tool regularly checks the common online repository for new artifacts or new artifact versions and updates its local repository as needed.[6] We describe the five different roles in the following.

[6]We intend to create a publicly-accessible online repository and to integrate a corresponding submission and update procedure in *CloudMIG Xpress* in our future work. Furthermore, an approval mechanism needs to be set up for reasons of quality assurance and to prevent vandalism.

Monitoring Format Reader Contributor A Monitoring Format Reader Contributor (MFRC) provides or updates a component that enables the import of monitoring data of a particular format in a new workload profile (see Chapter 9). As described before, a workload profile constitutes a part of the utilization model. The extracted monitoring data models the actual usage patterns that are used for simulation purposes. There exist various tools that can be used for monitoring a system's actual load. However, most of them use proprietary data formats. Hence, an MFRC contributes a new monitoring format reader that allows integrating further monitoring data sources.

Cloud Profile Contributor A Cloud Profile Contributor (CPC) provides or updates a cloud profile to/in the global repository. Cloud profiles constitute central models in the context of CloudMIG. However, there exists a plethora of cloud providers that also frequently offer several different cloud environment configurations. Furthermore, the contents of cloud profiles need to be kept current as, for example, an outdated pricing or CEC model is of little use and may even lead to suboptimal decisions in the course of a cloud migration. Hence, a central, publicly-accessible repository is extremely beneficial in that regard. For instance, cloud profiles could be contributed by SaaS providers interested in a cloud migration to a specific cloud environment, or by cloud providers themselves that are interested in simplifying the cloud migration to their offered services.

CEC Validation Contributor A CEC Validation Contributor (CVC) provides or updates a plugin for detecting *CEC violations* for systems built with a specific programming language. As described before and explained in Section 7.3.2 in greater detail, providing support for additional programming languages does not imply the need to create a full-fledged component for the detection of *CEC violations* that are specific for that language. However, a lean language-dependent detection component is needed for a small subset of CECs despite processing generic KDM-based models. These lean detection components, so called *CEC validators*, can also be submitted to and obtained from the global repository.

KDM Discoverer Contributor A KDM Discoverer Contributor (KDC) provides or updates a plugin for the extraction of KDM models from the source code developed in a specific programming language. Those plugins are

called *KDM discoverers*. For example, at the time of writing, there exist KDM discoverers for Java and C#. Corresponding models of a software system can then be created during CloudMIG's extraction activity A1 using these KDM discoverers. See Chapter 9 for further details.

CloudMIG User A CloudMIG User is interested in performing a cloud migration and wants to support the corresponding planning phase by utilizing the CloudMIG method. A CloudMIG User can obtain the aforementioned artifacts from the global repository, for example, further cloud profiles to broaden the search for appropriate target cloud environments.

6.3.4 Cloud Environment Model

The Cloud Environment Model (CEM) constitutes the foundation for CloudMIG's capabilities to detect *CEC violations* and to optimize cloud deployment architectures and runtime reconfiguration rules. It is a meta-model for describing PaaS- and IaaS-based cloud environments from the perspective of a cloud user. Hence, detailed information concerning an underlying cloud platform is often unknown and, by the concept of cloud computing, abstracted by a cloud provider. CEM is built as an Ecore model [Steinberg et al. 2009] and can, as describe below, be converted to a pure KDM model with an appropriate transformation. CEM comprises a model for describing CECs.

Instances of CEM represent specific cloud profiles that describe, among others, the provided services, legacy code containers, and virtualized hardware resources of a cloud environment. Cloud profiles have to be modeled only once and can then be reused by other reengineers with the help of the global repository that was introduced in Section 6.3.3. This applies to the CECs that are included in the cloud profiles, too.

Overview The CEM is organized in layered packages as presented in Figure 6.16. As the CEM is fairly comprehensive, this section primarily explains the basic underlying concepts and the elements that are important for describing the conformance checking approach and the deployment and

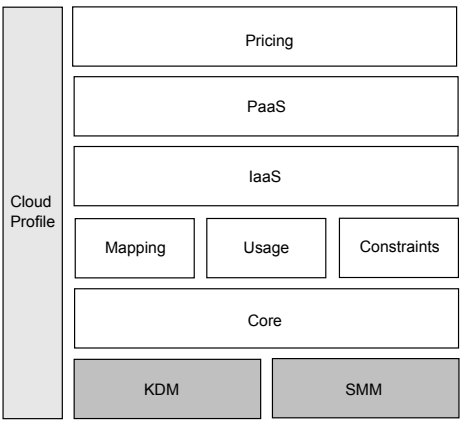


Figure 6.16. The packages of CEM (based on [Frey et al. 2013a])

reconfiguration optimization in later chapters. A more detailed model of CEM can be found in Appendix A.

The *Core* package of CEM includes basic elements like abstract cloud services or partitions. The latter allows to model both Amazon EC2's availability zones and regions (cf. Section 2.2.2), for instance. The further packages build upon the *Core* package. The *Mapping* package comprises model elements that enable the integration of legacy system parts into a cloud environment. "Mapping" therefore means the assignment of an SUA's elements to entities available in the cloud domain. Potential incompatibilities are handled by means of adapters that have to be created manually in the subsequent transformation step (CloudMIG activity A6), for instance. Those adapters should not be confused with the gateways that were introduced in Section 3.2.2 in the context of the *Chicken Little* migration approach. *Chicken Little*'s gateways are applied only during cut-over to enable a concurrent operation of an old and a new system, i.e., to facilitate a phased interoperability or a parallel operability cut-over strategy. In contrast, the adapters of CEM are part of the envisioned target architectures for incorporating cloud services or converting data formats, for instance.

The *Usage* package contributes model elements for describing and extracting the utilization model used by CloudMIG. In doing so, it incorporates measures and measurements modeled with SMM. The *Constraints* package models the CECs and is covered in Chapter 7 in greater detail. The *IaaS* package and *PaaS* package comprise elements for the corresponding cloud service models. The first follows the structural elements of the cross platform cloud API Deltacloud[7] to some degree. The last is designed more generic as PaaS clouds exhibit an even broader bandwidth.

The *Pricing* package contains elements for building a pricing model. Such a model describes pricing information regarding provided cloud services. It is possible to include various price functions and billing modes. For example, specific VM instance types can be billed by usage time or be discounted when reserving a larger contingent. Further possibilities for modeling prices include the specification of a step function—e.g., for determining different prices in relation to the size of transmitted data chunks—or the definition of an arbitrary linear function.

The *Cloud Profile* package forms the entry point for modeling specific cloud environments (cloud profiles). It is designed to be orthogonal to the other packages and conceptually can access all of their elements. An excerpt is presented in Figure 6.17. A `CloudEnvironment` can contain several `CloudEnvironmentConfigurations`. Taking Google App Engine (GAE) as an example, there exists, among others, a `CloudEnvironmentConfiguration` for *Google App Engine for Java* and one for *Google App Engine for Python*, for instance. Furthermore, Figure 6.17 shows elements for referencing and including an IaaS' `HardwareConfiguration` (e.g., an Amazon EC2 "High-Memory Extra Large Instance"), an `EnvironmentConstraintConfiguration` from the *Constraints* package for incorporating CECs, and the abstract classes `AbstractCloudService` and `AbstractCloudAppDataContainer` for providing convenient generic extension points for cases where the concrete instances in other layers may not be sufficient. The alignment with KDM is described in the following.

[7] `http://deltacloud.apache.org/`

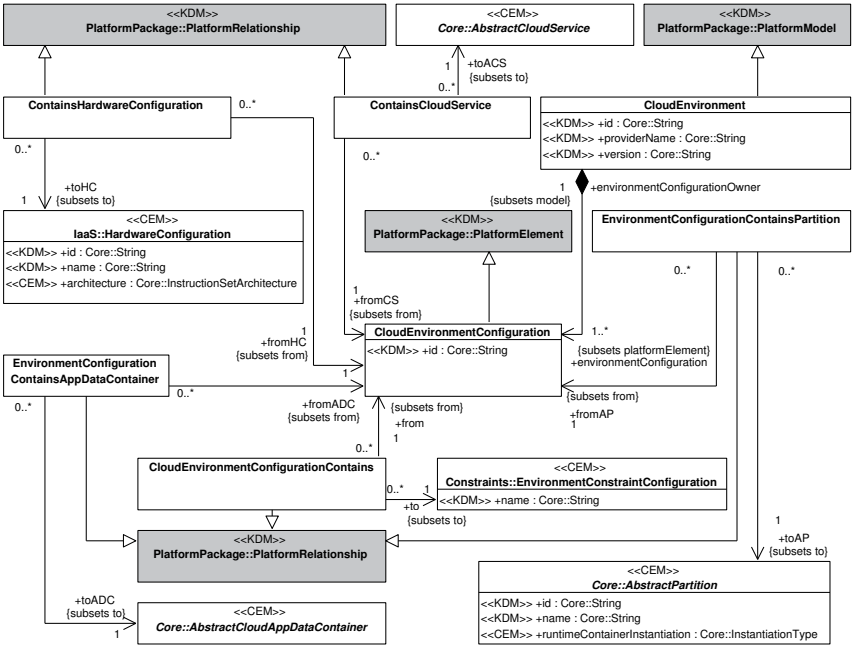


Figure 6.17. The *Cloud Profile* package of CEM (excerpt) (cf. [Frey et al. 2013a])

Alignment with KDM Besides the domain-specific elements of the *Cloud Profile* package, one can recognize the alignment of CEM with KDM in Figure 6.17. Generally, CEM builds upon KDM's platform and structure packages following the piggyback pattern [Spinellis 2001] for the realization of domain specific languages (DSLs). We provide a transformation from the domain model implemented as an Ecore model to a KDM-compatible version. Future KDM-conform modernization tools may therefore be able to process our model. The compatibility is achieved by avoiding to introduce new meta-model elements. The CEM classes that inherit from KDM classes (gray) in Figure 6.17 are rather transformed to KDM `Stereotypes`. Their attributes become KDM `TagDefinitions`, and the CEM packages are realized as KDM `ExtensionFamilies`, to cover just the major modeling elements.

```
 1 <extension name="cloudprofile">
 2   <stereotype name="CloudEnvironment" type="PlatformModel">
 3     <tag tag="id" type="String"/>
 4     <tag tag="providerName" type="String"/>
 5     <tag tag="version" type="String"/>
 6   </stereotype>
 7   <stereotype name="CloudEnvironmentConfiguration" type="PlatformElement">
 8     <tag tag="id" type="String"/>
 9   </stereotype>
10 </extension>
11
12 <model xsi:type="platform:PlatformModel" stereotype="//@extension.0/
       @stereotype.0">
13   <taggedValue xsi:type="kdm:TaggedValue"
14     tag="//@extension.0/@stereotype.0/@tag.0"
15     value="cloudmig.cloudprofiles.gae"/>
16   <platformElement xsi:type="platform:PlatformElement" stereotype="//@extension
         .0/@stereotype.1">
17     <taggedValue xsi:type="kdm:TaggedValue" tag="//@extension.0/@stereotype.1/
           @tag.0" value="cloudmig.cloudprofiles.gae.java"/>
18     <ownedRelation xsi:type="platform:PlatformRelationship" to="//@model.0/
           @platformElement.1" from="//@model.0/@platformElement.0"/>
19   </platformElement>
20 </model>
```

Listing 6.1. Google App Engine for Java cloud profile (KDM excerpt) (cf. [Frey et al. 2013a])

This approach uses the light-weight extension mechanism of KDM. The mentioned elements constitute new so called virtual meta-model elements. They have to consider given restrictions. For example, the `CloudEnvironmentConfiguration` inherits from the KDM type `PlatformElement` and therefore cannot incorporate the `EnvironmentConstraintConfiguration` class (a `PlatformElement` as well) via a composition, as this association does not exist for two `PlatformElements` in the KDM meta-model. It rather has to utilize an instance of KDM's `PlatformRelationship` (the class `CloudEnvironmentConfigurationContains`) to link both elements. Listing 6.1 illustrates this concept by means of a KDM Google App Engine for Java cloud profile extract in XML Metadata Interchange (XMI) notation. For example, the `PlatformModel` in Line 12 represents CEM's `CloudEnvironment`, as its `stereotype` attribute refers to

the according KDM `Stereotype` in Line 2. The virtual meta-model element `CloudEnvironment` is in turn contained in a KDM `ExtensionFamily`[8] that models CEM's *Cloud Profile* package (Line 1). Further details regarding the GAE cloud profile can be found in Appendix B.

6.4 Summary

This chapter described the CloudMIG method that supports SaaS providers to migrate enterprise software to the cloud. Enterprise software systems often exhibit a fluctuating load [Ranganathan and Jouppi 2005] and are therefore well-suited candidates for exploiting the cloud's elasticity. CloudMIG supports the planning phase of a cloud migration, i.e., the focus lays on model-based (1) conformance checking and (2) optimization of cloud deployment architectures and reconfiguration rules instead of on the actual migration execution phase. These aspects (1) and (2) also form major contributions of this thesis. Hence, through describing the general CloudMIG method, this chapter gave the context for describing these contributions in the next chapters in detail.

Further noteworthy characteristics of CloudMIG are given by its strong reliance on OMG's ADM standards KDM and SMM that form ideally suited foundations for building meta-models in the context of software modernization scenarios. Furthermore, CloudMIG targets the migration of enterprise software systems to IaaS- and PaaS-based cloud environments. That means, it supports SaaS providers in migrating software systems to the cloud and in offering existing services in the form of cloud services, whereas simply comparing existing SaaS-based solutions or replacing local applications with web-based alternatives is out of CloudMIG's scope.

[8]The KDM Ecore model used from the tool MoDisco names the "extensionFamily" role of the according containment relationship in the "KDMFramework" super class element "extension"

Chapter 7

Conformance Checking

Not every cloud environment is similarly suited for running enterprise software, as the cloud environments—particularly PaaS-based cloud environments—impose varying constraints on the applications they host. For example, the ability to access the underlying file system may be permitted in a non-uniform way, or the usage of particular network protocols might be restricted. Hence, for each potential target cloud environment being under consideration in the course of a cloud migration, a costly and error-prone analysis has to be performed to validate the specific constraints and to judge the suitability of the competing cloud environment candidates.

CloudMIG provides the meta-model CEM to specify the diverse characteristics of cloud environments. CEM includes the *Constraints* package for defining CECs in a generic fashion. Thus, the specific *CECs* for a cloud provider can be documented in a reusable manner with the help of a cloud profile and serve as a validation input for arbitrary legacy systems. Besides means for describing CECs in a cloud provider-agnostic way, CloudMIG provides appropriate mechanisms for automating the detection of a software system's violations regarding those constraints. A system's conformance can be examined with the assistance of constraint validators. This *conformance checking* process operates on KDM-based system models that were extracted by CloudMIG's activity A1 and can, among others, apply metrics modeled with SMM through our MAMBA Execution Engine (MEE, cf. Section 10.2). Each constraint validator can check an existing or reconstructed model of the software system for code artifacts that would lead to *CEC violations* when being deployed unmodified. Additional constraint validators can be plugged into the conformance checking process as needed. Figure 7.1 shows

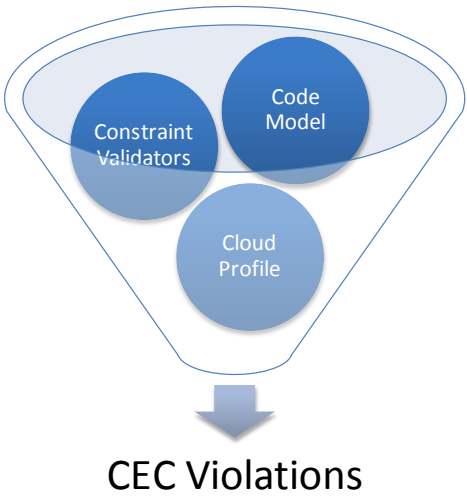


Figure 7.1. Conformance checking overview

a high-level illustration of CloudMIG's conformance checking approach and its mentioned basic elements. The detection of *CEC violations* is part of CloudMIG's activity A3, as the generation of a target architecture involves the mapping of a system's code to resources and services that are provided by a cloud environment. This mapping procedure would eventually result in the deployment of a system component to the cloud during the migration execution phase and therefore constitute the actual starting point for provoking a *CEC violation*.

This chapter utilizes and builds upon the following previously published work:

1. *Frey, Sören and Hasselbring, Wilhelm, "The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications," in International Journal on Advances in Software, ISSN 1942-2628, Vol. 4, Nr. 3 and 4, pp. 342-353, 2011.*

2. *Frey, Sören and Hasselbring, Wilhelm and Schnoor, Benjamin, "Automatic Conformance Checking for Migrating Software Systems to Cloud Infrastructures*

and Platforms," in Journal of Software: Evolution and Process, Vol. 25, Nr. 10, pp. 1089–1115, 2013.

The chapter is organized as follows. Section 7.1 provides an overview of CloudMIG's conformance checking approach. The mechanisms for modeling CECs in a cloud provider- and language-independent fashion are described in Section 7.2. The conformance checking process for automatically detecting *CEC violations* is then detailed in Section 7.3. Section 7.4 describes potential consequences of revealing *CEC violations* during this process. More specifically, it analyzes the impact of detected *CEC violations* on the general suitability of a cloud environment in the context of a cloud migration project. Furthermore, it derives an approach for prioritizing the correction of different classes of *CEC violations* for planning the actual migration execution. Finally, Section 7.5 sums up this chapter.

7.1 Overview

Migrating software to cloud-based platforms involves the necessity to assess and compare target cloud environment candidates. Several functional as well as non-functional properties of cloud environments are relevant to eventually make a choice, such as the offered services, software stacks, SLAs, prices, data center security certificates, and geographical availability. A further important property of potential target cloud environments is also given by posed technical constraints. Most often, such constraints are mentioned informally in the context of user support documents or web pages. For instance, consider the exemplary formulation stated in the free text description of Google App Engine for Java's servlet environment.[1]

> *"However, you must use one of the methods on* `ThreadManager` *to create your threads. You cannot invoke* `new Thread()` *yourself or use the default thread factory."*

[1] `https://developers.google.com/appengine/docs/java/runtime/`

Despite not stating concrete consequences if systems try to spawn threads with the help of the mentioned constructor of `java.lang.Thread` or by using the default thread factory via the `java.util.concurrent.Executors.defaultThreadFactory()` method for this purpose, the corresponding actions are discouraged by the cloud provider and are announced to be inoperative.[2] However, the stated restriction quoted above indicates at the same time a possible precautionary measure for circumventing those issues by requiring systems to use `com.google.appengine.api.ThreadManager` instead. Hence, as we will see in Section 7.2.1, besides modeling given constraints of a cloud environment as a part of a comprehensive cloud profile, the CEM allows to include such possible solutions that can provide hints for users of the CloudMIG method in the case corresponding *CEC violations* are detected. Users might neither be aware of the general existence of those kinds of restrictions, nor of the elements in their software systems that violate these restrictions or of potential solutions. In contrast, CECs, *CEC violations*, and potential solutions are made explicit in CloudMIG.

Regarding the roles that were described in Section 6.3.3, three roles are relevant in the context of the conformance checking approach. (1) Cloud Profile Contributors (CPCs) model cloud profiles with the help of the CEM. The cloud profiles describe specific cloud environments in a reusable manner and also contain specifications of the CECs which a cloud environment establishes. Furthermore, CloudMIG provides means to automatically detect *CEC violations* of an application and points a reengineer to elements of the software system that cause the violations. (2) CEC Validation Contributors (CVCs) provide constraint validators that allow a (3) CloudMIG user to perform the detection mechanism and search for *CEC violations*. The detected *CEC violations* can be utilized by a CloudMIG user to obtain a quick overview of problematic system parts which need special attention and as a basis for comparing competing cloud environment offers.

As cloud profiles as well as the constraint validators can be reused by CloudMIG users, the general cost-benefit ratio of using the automatic conformance checking process turns out to be advantageously. Considering

[2] When creating a thread via a constructor of `java.lang.Thread` in an application that is running in Google App Engine for Java, a `java.security.AccessControlException` is raised.

the scenario that a software engineer may have to find and compare cloud environments that are suitable as migration targets, the advantages of using the presented approach are clear. There exists a great number of cloud environment candidates that exhibit a plethora of diverse CECs and the cloud provider landscape as well as their offered services are changing quickly. For any potential cloud environment, the *CECs* would otherwise have to be identified manually. Even doing so for a single cloud solution, as for example Google App Engine for Java, is not a trivial task as the elicitation of the involved constraints is hampered by incomplete information, various scattered relevant data sources, and non-standardized formats, for instance. Then, a trial-and-error approach might be followed or the software system's source code would have to be inspected manually to detect the related *CEC violations* that are specific for each cloud environment candidate. Here, the according documentation or developer knowledge is often missing or incomplete.

Additionally, considering the large numbers of detected *CEC violations* and that they are furthermore often scattered all over the systems (see the evaluation scenarios in Chapter 11), this constitutes a tedious and error-prone task even for a single cloud environment. Moreover, some cloud providers do not report the *CEC violations* during the deployment or start-up process of the guest applications. For example, as in the case of Google App Engine for Java, some restricted method calls may lead to thrown exceptions late after deployment when they are called the first time, as is the case with the restrictions regarding the creation of threads that was mentioned above.

This section is structured as follows. Section 7.1.1 describes basic concepts that are relevant in the context of the conformance checking approach. Section 7.1.2 gives several examples of CECs and *CEC violations*. Section 7.1.3 reasons about the general detectability of *CEC violations*, before Section 7.1.4 presents the characteristics of CECs that can be detected by CloudMIG.

7.1.1 Basic Concepts

To further clarify the used constraint-related terminology and to provide a definite semantics for notions that were already often used informally before, this section provides definitions for several key concepts of CloudMIG's conformance checking approach.[3] We begin with CECs.

Definition: Cloud Environment Constraint (CEC)

A restriction imposed by a cloud environment regarding a (potential) guest application. The restriction can either relate to the implementation, deployment, or a non-functional property of the guest application.

Before providing a definition for *CEC violations*, the incorporated notions of *faults*, *errors*, and *failures* are briefly introduced. These terms are often used in the systems engineering domain in the context of the dependability [Avizienis et al. 2004] and trustworthiness [Becker et al. 2006] concept:

Fault: A defect in a software system, e.g., a bug in the source code of a software, an outdated entry in a configuration file, or invalid characters in a data file that are not allowed according to a specific file format.

Error: An invalid internal state of a software system during runtime because of the activation of a fault. That means, the actual internal behavior deviates from the intended behavior.

Failure: An invalid external state of a software system during runtime that results from an error. That means, the actually delivered service or externally observable behavior deviates from its specification.

Furthermore, to define *CEC violations*, the notions of a *part of a software system* and *part activation*, as used in this thesis, have to be defined first.

[3] Note that the definitions of CECs and *CEC violations* were revised compared to previous versions [Frey and Hasselbring 2011a; Frey et al. 2013a] to improve comprehensibility.

Definition: Part of a Software System (Part)

A part of a software system (*part* for short where unambiguous) is a subset of the software system's artifacts, a specific detail of one or more of its artifacts, or a subset of its programming language statements. A part can be manifested in several distinct forms. For example, a programming language statement can be manifested in the source code as well as in an intermediate representation or in executable machine code.

In the context of a cloud migration, a part of a software system can also be *activated*.

Definition: Part Activation

When a part of a software system is *activated*, it means that it is tried to deploy, execute, or process the part.

Nevertheless, where necessary and not clear from the context, the notion of a part activation will be supplemented by its more detailed version, for example, by stating that a configuration file is deployed but not processed in a concrete situation. Using the aforementioned definitions, a *CEC violation* can now be defined as follows.

Definition: Cloud Environment Constraint Violation (CEC Violation)

A *CEC violation* is a fault of a software system regarding a specific CEC. Further, a CEC violation is manifested in one or more parts of the software system. A part can manifest multiple CEC violations. Where parts are present in several distinct forms, all of those forms relate to the same CEC violation(s) or to none at all.

Consider a directly upcoming activation of a part. The part manifests a CEC violation if one of the following two conditions holds: (1) The activation results in an error and possibly in a failure. (2) The directly upcoming activation does not result in an error, but it is possible that another activation of the same part might result in an error and possibly in a failure.

In the following, we give brief examples for the two conditions mentioned in the *CEC violation* definition above that indicate that a part raises a *CEC violation*.

1. A software system tries to open a network socket. An exception is thrown when the cloud environment does not allow a guest application to open network sockets. Hence, an error occurs that is possibly followed by a failure.
2. A software system processes tasks of different sizes in worker threads and the cloud environment terminates threads after 10 seconds. Hence, the processing of a small task may be finished in time, but the processing of a bigger task may be interrupted.

More detailed descriptions regarding real world examples of CECs and *CEC violations* can be found in Section 7.1.2. In general, *CEC violations* can be regarded as differently serious, for example, fixing a *CEC violation* can be a matter of minutes to months (and beyond). To address this issue, we introduce the concept of *violation severity* that is defined as follows.

Definition: Violation Severity

The severity of a CEC violation indicates the likely effort to fix the CEC violation during a migration process. As this can vary widely depending on specifics of the different legacy applications, we apply the *violation severity* rather pessimistic and propose three simple concrete severities: *Breaking*, *Critical*, and *Warning* associated with high, medium, and low effort, respectively.

The *violation severity* is specified by CPCs in cloud profiles along with the definition of the corresponding CECs. It should be noted that the *violation severity* is biased according to the experience and subjective appraisal of a person modeling a cloud environment. Therefore, it should be seen as a hint for the reengineer. It is important to detect a *CEC violation* at all and to

make the reengineer be aware of it. Nonetheless, beyond that the *violation severities* of detected *CEC violations* can be used as an influencing factor when prioritizing the handling of *CEC violations* (see Section 7.4.2). In general, there exist two reasons when existing *CEC violations* cannot be uncovered (*false negative* prediction, cf. Section 11.1.2). Either, the existing constraint validators falsely disregard them, or there exist *CEC violations* that are not addressed by any constraint validator.

After introducing the essential terms regarding the automatic detection of CECs, all necessary elements for approaching the concept of *conformance checking* have been presented. Simply put, a CloudMIG user employs the conformance checking approach to identify as many as possible of the software's parts that violate the CECs defined in the cloud profiles that are of interest to the user. Therefore, the concept of conformance checking can be defined as follows.

Definition: Conformance Checking

A software system S that is supposed to be migrated to the cloud (SUA) consists of a set of parts P, $\bigcup_{p \in P} p = S$. A CloudMIG user is interested in a set of cloud environments that are represented via the corresponding cloud profiles. These cloud profiles of interest C_{Int} are a subset of all existing cloud profiles C_{All}, $C_{Int} \subseteq C_{All}$. All CECs that are defined in C_{Int} are termed $\oplus(C_{Int})$. To indicate that a specific part p violates a specific CEC γ, we say $p \blacktriangleright \gamma$ and denote the set of all parts that violate the CEC γ by χ_γ.

Basically, the conformance checking approach aims to find all parts of S that violate any CEC that is defined in any of the cloud profiles of interest $\bigcup_{\gamma \in \oplus(C_{Int})} \chi_\gamma$. Besides the specific parts that violate a CEC, the conformance checking approach delivers the proposed solutions if those are defined. The set of proposed solutions for violations regarding a CEC γ is denoted by SOL_γ. A triple $(\gamma, \bigcup_{p \in P} p \mid p \blacktriangleright \gamma, SOL_\gamma)$ denotes a CEC γ, all parts of the system that violate γ, and names also the specified proposed solutions (SOL_γ).

Ultimately, the conformance checking approach aims to deliver $\bigcup_{\gamma \in \oplus(C_{Int})} (\gamma, \chi_\gamma, SOL_\gamma)$, where $|\chi_\gamma| > 0$. However, as sometimes not all existing *CEC violations* can be detected, the conformance checking approach actually delivers a subset of those triples, as denoted with $\bigcup_{\gamma \in \oplus(C_{Int})} (\gamma, \bigcup_{p \in P} p \mid p \rhd \gamma, SOL_\gamma)$, where $p \rhd \gamma$ describes a part p that was actually found by the conformance checking approach to violate the CEC γ, with $(\bigcup_{p \in P} p \mid p \rhd \gamma) \subseteq \chi_\gamma$.

As stated in the conformance checking definition above, for each CEC for which CEC violations were detected, the following three information types are reported to a CloudMIG user: (1) The CEC that is actually violated, i.e., the kind of detected constraint violation. (2) The parts of the software system that violate the CEC. (3) Proposed solutions for correcting the *CEC violation* (in the case those proposed solutions exist).

7.1.2 Exemplary Constraints and Constraint Violations

The definition of *CECs* is non-standardized and each cloud provider may specify the constraints in an arbitrary format or formulation. Even worse, there may exist *CECs* that are entirely undocumented or the documentation may be distributed over several corporate web pages, help files, and user community weblogs. Hence, describing a cloud environment's constraint descriptions in a structured form in a single cloud profile enables to consolidate those information, allows to exchange it with the help of a public repository (see Section 6.3.3), and provides a basis for CloudMIG's automatic capabilities to detect constraint violations.

An example for a constraint formulated in free text is given by the cloud environment Microsoft Windows Azure, where the documentation regarding network port requirements includes the following statement.[4]

> *"Windows Azure blocks UDP traffic. However, you can enable UDP communication between Windows Azure role instances and on-premises computers by using Windows Azure Connect."*

An application that needs to utilize network communication over the UDP protocol has to use Microsoft's specific *Windows Azure Connect* service, instead of merely establishing a UDP-based socket and starting to transmit datagrams over a corresponding port. Hence, it is rather simple to allow applications to further use UDP-based connections, no pervasive modifications should be necessary. Nevertheless, it is important to be aware of this constraint in the case the application that is supposed to be migrated to the cloud utilizes UDP. Thus, the corresponding definition of this constraint should be specified with the lowest-level violation severity, i.e., *Warning*.

A further example of a CEC that is rather simple to tackle in the case corresponding violations exist, is stated by the cloud environment Heroku in the subsequent documentation excerpt regarding caching strategies for the web application framework Ruby on Rails.[5]

[4] http://msdn.microsoft.com/en-us/library/windowsazure/jj136814.aspx

[5] https://devcenter.heroku.com/articles/caching-strategies

> *"Rails's built-in page-caching works by creating a file on the file system. Heroku has an ephemeral file store, so while page caching may appear to work, it won't work as intended. You should instead use action or fragment caching, or alternatively use Rack::Cache as a reverse proxy to avoid requests to your apps at all."*

No pervasive architectural modifications are required to ensure that requests for generated pages can be further served via a caching mechanism. The cloud provider rather states potential lightweight solutions, such as using Ruby on Rails' similar action caching mechanism instead of the page-caching. Hence, the corresponding CEC definition should employ the *Warning* violation severity as well.

In the following, we give three further examples of *CECs* (E1-E3), their associated *violation severities*, and properties of software systems that would lead to corresponding *CEC violations*. In these concise examples, we utilize constraints from the cloud environments Google App Engine for Java and the Amazon Elastic Compute Cloud (Amazon EC2).

E1:	*CEC*:	Using Google App Engine for Java, the total number of files is limited to 3,000 per default.
	CEC violation:	An application that exceeds this limit.
	Violation severity:	Warning (assuming that the creation of new container structures is a rather simple problem).
E2:	*CEC*:	Using Amazon EC2, the local storage of VM instances is transient. For persistent storing one of Amazon's services like the Elastic Block Store (EBS) or the Relational Database Service (RDS) has to be used.
	CEC violation:	An application that writes to the local file system in one of its methods.
	Violation severity:	Critical.

As CloudMIG does not provide support for migrations that incorporate the transformation between programming languages, the last example E3 has the highest possible violation severity assigned (*Breaking*). Considering

E3:	*CEC*:	Using Google App Engine for Java, only languages that are compatible with the Java Virtual Machine (JVM languages) can be used as guest applications.
	CEC violation:	A C++ application.
	Violation severity:	Breaking.

the example E2, Amazon EC2 poses restrictions using the local storage to store data persistently. The cloud environment Google App Engine for Java exhibits a constraint that similarly prevents writing data to the local filesystem. The corresponding documentation states the following.[6]

> *"An App Engine application cannot [...] write to the filesystem. Applications must use the App Engine datastore for storing persistent data. Reading from the filesystem is allowed, and all application files uploaded with the application are available."*

Figure 7.2 shows a Java servlet [Hunter and Crawford 2001] that provokes a corresponding constraint violation, as the type `java.io.FileWriter` is instantiated and used to write text to a file. Furthermore, Figure 7.2 depicts the Google plugin for Eclipse that is provided by the cloud provider to support the development and deployment of applications to Google App Engine. This plugin performs an own basic validation of applications and marks, as can be seen in Figure 7.2, source code statements that use types which are not supported. However, the validation functionality of this plugin does not remedy all problems related to *CECs*. Regarding the above-mentioned constraint of restricting the data storage on a local filesystem, Figure 7.3 shows that the Google plugin for Eclipse does not regard the creation of a temporary file with the `java.io.File.createTempFile()` method as problematic. Nevertheless, when deploying this source code to Google App Engine for Java and running the application, an exception is thrown because of the usage of `java.io.File.createTempFile()` and the error message shown in Figure 7.4 is presented to a user of the web-based

[6] `https://developers.google.com/appengine/docs/java/runtime/`

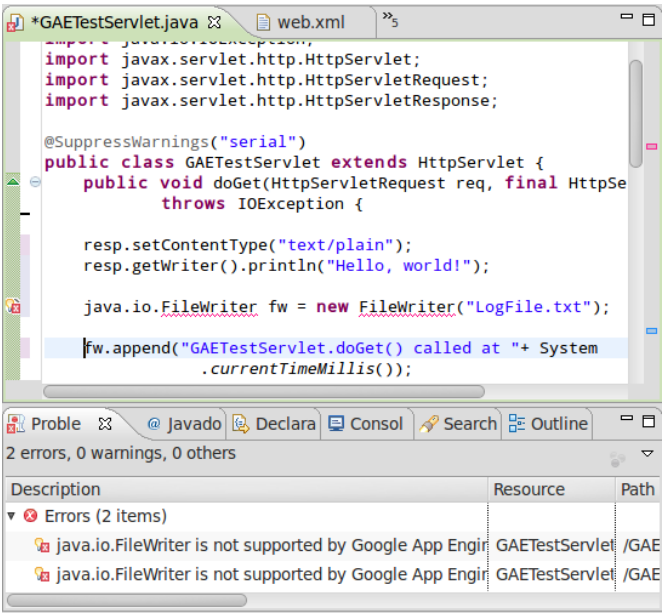


Figure 7.2. Google plugin for Eclipse indicating that the used type `java.io.FileWriter` is not supported by the cloud environment Google App Engine for Java

application. It should be noted that the fault is only detectable at runtime when the corresponding statement is executed the first time, as the deployment process also does not indicate potential problems. In contrast, the CloudMIG method detects the *CEC violation* and points the CloudMIG user to the `java.io.File.createTempFile()` method call.

7.1.3 Detectability of Constraint Violations

This section examines the general detectability of *CEC violations*, presents corresponding limitations, and lists preconditions that have to be met for detecting *CEC violations* with CloudMIG. Section 7.1.2 presented, among others, the example regarding the runtime error that occurs when calling

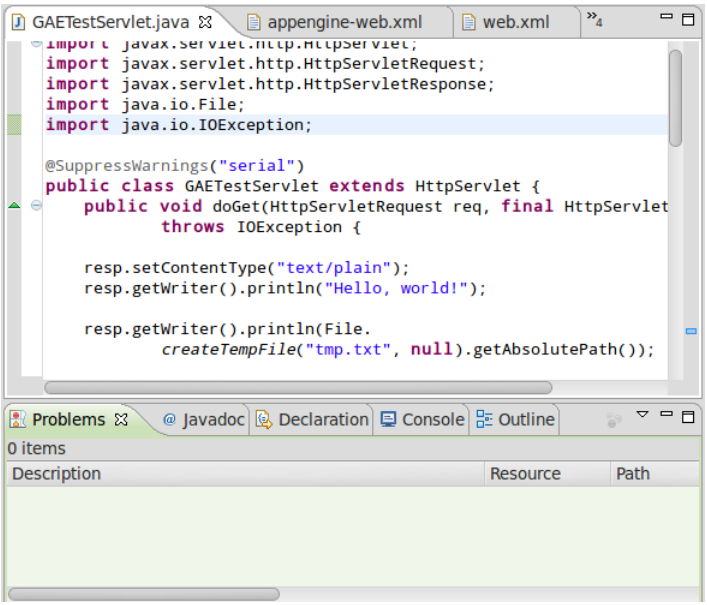


Figure 7.3. Google Plugin for Eclipse does not indicate the constraint violation resulting from the usage of `java.io.File.createTempFile()` with Google App Engine for Java

the `java.io.File.createTempFile()` method in an application that runs in the cloud environment Google App Engine for Java. The CEC definition that can be derived from the cloud provider's documentation forbids to store data using the local storage in general. As CloudMIG detects that a method call to `java.io.File.createTempFile()` would write data to the file system, it can infer that this method call violates the defined CEC (see Section 7.3 for details). Hence, the *CEC violation* can be detected by analyzing solely the KDM model that describes the source code.

Considering the Listing 7.1, a call to `java.io.File.createTempFile(String, String)`[7] is, however, not included in the source code. Though, the method

[7]The method signature details are omitted in the other mentions of this method for reasons of readability.

Figure 7.4. Google App Engine displays an error message when running an application that calls the method `java.io.File.createTempFile()`

could still be called and produce an error during runtime. The Java code in Listing 7.1 uses reflection to call arbitrary static methods. Assuming the user would be asked to enter the contents of the variables in the Lines 8-13 at runtime, `java.io.File.createTempFile()` could be called anyway. After entering the corresponding class name, method name, parameter types, and parameter values used for the actual method call as indicated in the comments of Listing 7.1, `java.io.File.createTempFile()` would be called (Line 22).

Therefore, a static source code analysis is not sufficient for detecting the *CEC violation*. A warning could, of course, be displayed when a dynamic method call with the help of Java's Reflection API is detected. But to avoid many false alarms (false positives), it had to be ensured that the dynamic method call does not restrict the potential targets to specific types and methods, such that a generic, dynamic call remains possible. But such a check would at most decrease the number of false positives, but could not prevent them entirely. The actual problem in the source code of Listing 7.1 is given by the dynamic dispatch in combination with the processed user input, as it cannot be decided what method will be called until it is eventually called.

```
1  import java.lang.reflect.Method; [...]
2  public class DynamicMethodCallExample { [...]
3    /**
4     * Calls a prohibited method via reflection.
5     */
6    protected void callProhibitedMethod() {
7       try {
8         String className = // Read in class name ('java.io.File')
9         String methodName = // Read in method name ('createTempFile')
10        Class methodSignParams[] = // Read in parameters for def. method's signature
11                                   // ('java.lang.String', 'java.lang.String')
12        Object methodCallParams[] = // Read in parameters for calling the method
13                                    // ('tmp.txt',null)
14
15        Class<?> clazz = Class.forName(className);
16        Class<?>[] methodSignatureParamTypes = new Class[methodSignParams.length];
17
18        for(int i = 0; i<methodSignParams.length;++i)
19          methodSignatureParamTypes[i] = Class.forName(methodSignParams[i]);
20
21        Method method = clazz.getMethod(methodName, methodSignatureParamTypes);
22        method.invoke(null, methodCallParams); // Call of prohibited method
23      } catch([...]) {
24            [...]
25      }
26   }
27 }
```

Listing 7.1. Call of a prohibited static method using reflection

Similarly, there exist *CEC violations* that are detectable during runtime, but that defy any possibility of being identified with the help of a static source code analysis at all. For example, the execution of stored procedures in a database may be terminated after two seconds. To be of any use, the detection of those kinds of *CEC violations* has to incorporate information observed during runtime.

Detectability Categories Based on the two examples mentioned above, two so-called *detectability categories* (DCs) of *CEC violations* can be distinguished so far. *CEC violations* of both categories cannot be detected solely with the help of static source code analyses. However, the categories can be distinguished as there exist *CEC violations* that might be detectable relying on static analyses when accepting a range of false positives. These categories (DC1 and DC2) form the first two of overall five detectability categories. The detectability categories are briefly described in the following.

- **DC1:** *CEC violations* can only be detected using data obtained at runtime. Static source code analyses are not suited for reasonably detecting or suggesting potential *CEC violations*.
- **DC2:** *CEC violations* can only be detected using data obtained at runtime. However, static source code analyses can reasonably suggest potential *CEC violations*, when accepting a range of false positives. In contrast to DC1, those false positives can be checked by manual source code inspections.
- **DC3:** *CEC violations* can be detected using data obtained at runtime or data from a static source code analysis. Regarding the latter, given literals do not have to be taken into account.
- **DC4:** *CEC violations* can be detected using data obtained at runtime or data from a static source code analysis. Regarding the latter, given literals have to be taken into account.
- **DC5:** *CEC violations* can be detected using data obtained at runtime or data from a static source code analysis. Regarding the latter, at least one of the following three additional activities has to be applied: Expressions might have to be evaluated (1), a dataflow analysis might have to be conducted (2), or arbitrary system artifacts, such as configuration files or SQL scripts, might have to be taken into account (3).

It should be noted that the detectability categories do not refer to CECs but to specific *CEC violations*. For example, for the case that a cloud environment

prohibits calling a specific method, the examples presented before described a possibility to detect corresponding method calls via analyzing the source code (DC3), i.e., it is searched if an actual method call can be located in the source code. Otherwise, if the method is called by means of reflection, as for example demonstrated in Listing 7.1, a static source code analysis can only detect statements that might eventually lead to a call of the prohibited method, but not reliably detect a *CEC violation* (DC2). Hence, in this example, one CEC might be responsible for *CEC violations* of two different detectability categories.

For using static source code analyses to detect *CEC violations* that come under the detectability category DC4, it is necessary to also consider literals that are defined in the source code. For example, consider a cloud environment that permits to open network ports only within a limited range. For being checkable under DC4, the port number had to be stated as, for example, an integer or string literal, e.g., in the constructor of an appropriate class. If in addition, for instance, string resources have to be taken into account (e.g., when strings are kept in separate files for facilitating internationalization), corresponding *CEC violations* fall under the detectability category DC5. The example regarding the restricted port range stated above also turns to DC5, if a corresponding *CEC violation* could only be detected when, for example, an expression had to be additionally evaluated for calculating a network port number from literals and operators.

The detectability categories are revisited in Section 7.1.4 when describing characteristics of the constraints that are addressed by CloudMIG. As CloudMIG focuses on *CEC violations* that can be detected with the help of KDM models that are extracted from a system's source code, four preconditions have to be met in this regard to be able to execute the automatic conformance checking process:

Preconditions and Effort The source code has to be available (1). A parser for the incorporated programming languages has to be present (2). It must be possible to extract a KDM model, either directly with the help of the parser or with a subsequent model to model transformation (3). The existing *CECs* have to be documented by the cloud provider or by external

sources (4). If all preconditions are met and the *CECs* described in (4) were already converted to a corresponding cloud profile, the effort to produce a constraint validation report is negligible as it can be generated directly by *CloudMIG Xpress*. Otherwise, building a cloud profile from documented *CECs* is also a rather simple task due to the given structure and documentation of the *CEM*.

However, if the preconditions are not met, it may imply considerable effort to first of all build an appropriate parser and transformation to KDM, for instance. On the other hand, this also saves a reengineer from manually inspecting the source code regarding the specific *CECs* for every potential cloud environment candidate. As the conformance checking approach includes generic constraint description and constraint validation mechanisms, there exists no preference regarding specific system types, for example, regarding systems for batch processing or systems that utilize a SOA. This is in contrast to the overall approach CloudMIG that is constructed to support the migration of enterprise applications as these are primary candidates that can benefit from smoothly scaling up and down in the cloud due to varying workload patterns.

Dependence on Programming Language Regarding static analyses on the basis of KDM models, the detectability of *CEC violations* can also be differentiated according to the internally used processing mechanism, as is described in more detail in the context of the architecture description of the *CloudMIG Xpress* tool (cf. Chapter 9). *CEC violations* can either be detected by incorporating or by omitting knowledge that is specific for a programming language. Hence, *language-dependent* and *language-agnostic* detection mechanisms are distinguished. In the following, two Java-based source code examples and their corresponding KDM models are presented to demonstrate the limits of language-agnostic detection mechanisms, and to explain the occasional need for language-dependent detection mechanisms. Listing 7.2 shows Java source code that includes the two classes `Client` and `ObsoleteClass`, where `Client` instantiates `ObsoleteClass`.

```
1 public class ObsoleteClass {}
2 public class Client {
3     static ObsoleteClass oc = new ObsoleteClass();
4 }
```

Listing 7.2. The violation of a CEC in Java that forbids the instantiation of the class `ObsoleteClass` can be detected with a *language-agnostic* mechanism

We assume the definition of a CEC that prohibits the instantiation of the class `ObsoleteClass`. Figure 7.5 shows a UML object diagram that describes the basic elements of a KDM instance that models the corresponding source code of Listing 7.2. This KDM model originates from CloudMIG's extraction activity A1, follows the structure that is produced by the reverse-engineering framework MoDisco [Bruneliere et al. 2010], and forms the basis for detecting the *CEC violation* that results from `Client`'s instantiation of `ObsoleteClass`. The KDM model includes the necessary elements for describing the full semantics of the Java code. The prohibited instantiation is modeled with the help of the `StorableUnit` element that represents the static variable `oc`. It is initialized via the `HasValue` KDM element that references an `ActionElement`. This `ActionElement` in turn provides the connection to calling `ObsoleteClass`' constructor via the interposed `Calls` element. This would already suffice for detecting the *CEC violation*. However, the additional `Creates` element also indicates that `Client`'s attribute `oc` is initialized with a newly created instance of `ObsoleteClass`. Hence, as the relevant instantiation of `ObsoleteClass` is explicitly contained in the KDM model, no further information, especially no language-dependent information, is needed for detecting the *CEC violation*. Thus, it can be revealed with language-agnostic detection mechanisms.

In contrast, the Listing 7.3 and the corresponding KDM model in Figure 7.6 show a *CEC violation* that cannot be identified with language-agnostic detection mechanisms. The source code includes a `java.nio.channels.FileChannel` instance that is used in Line 20 to write to a file. Assuming a CEC specification that prohibits writing data to the filesystem, for allowing the detection of the *CEC violation*, KDM elements would have to be present that allow inferring the semantics of writing data to a file.

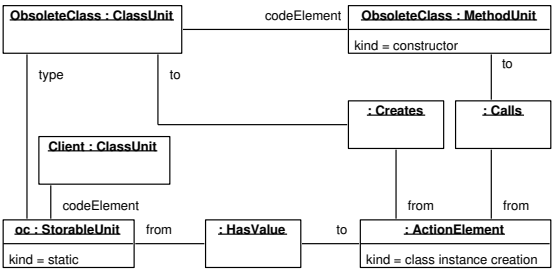


Figure 7.5. KDM instance that models the source code of Listing 7.2. Language-agnostic validation of a *type instantiation constraint* is possible, as the instantiation of class `ObsoleteClass` is explicitly represented in the KDM model.

However, examining the KDM model in Figure 7.6 shows that no such elements exist. As in the example before, all necessary elements to describe the full semantics of the Java source code are included in the KDM model. Though, all method calls are modeled with ordinary `Calls` elements that refer to generic `MethodUnits`, as is included in the KDM excerpt in Figure 7.6, where a call to `java.nio.ByteBuffer.clear()` is shown that corresponds to `buf.clear()`, for instance. Similarly, the call to the `write` method provides no further information that could reveal the writing to the filesystem.[8]

CloudMIG does not utilize techniques from the area of pattern recognition [Jain et al. 2000] to retrieve relevant information from mining identifiers or comments, for instance. Hence, the corresponding knowledge has to be additionally provided to the *CEC violation* detection mechanism. That means, for detecting that data is written to the filesystem, in this example, the detection has to be aware of the following two facts: The KDM model results from Java source code (1) and the `java.nio.channels.FileChannel.write()` method (a Java method) writes data to the filesystem (2). Thus, the detection mechanism is language-dependent. Chapter 9 describes the corresponding architectural details.

[8]The `fc.write(buf)` statement itself is not included in the KDM excerpt for reasons of brevity, as the model becomes unwieldy because of the method's parameter.

```
1  import java.io.RandomAccessFile;
2  import java.nio.ByteBuffer;
3  import java.nio.channels.FileChannel;
4   [...]
5
6  public class FilesystemWriteExample {
7
8    public static void main(String[] args) {
9          [...]
10         RandomAccessFile raFile = new RandomAccessFile(
11                   "OutputFile.txt", "rw");
12         FileChannel fc = raFile.getChannel();
13
14         ByteBuffer buf = ByteBuffer.allocate(40);
15         buf.clear();
16         buf.put("Output".getBytes());
17         buf.flip();
18
19         while (buf.hasRemaining()) {
20            fc.write(buf);
21         }
22
23         fc.force(true);
24         fc.close();
25          [...]
26   }
27 }
```

Listing 7.3. The *CEC violation* that results from the writing to the filesystem in Line 20 cannot be revealed with *language-agnostic* detection mechanisms

It should be noted that despite of the definition of language-dependent and language-agnostic detection mechanisms, no additional detectability category is formed. This is due to the fact that the *CEC violations* that fall under the detectability categories D2-D5[9] are in general not biased towards language-dependent or language-agnostic detection mechanisms. As a new category would therefore widely interfere with D2-D5, the distinction according to the dependence on language-specific knowledge can be rather seen as an orthogonal, cross-cutting dimension.

[9] D1 does not address *CEC violations* that can be detected with static analyses.

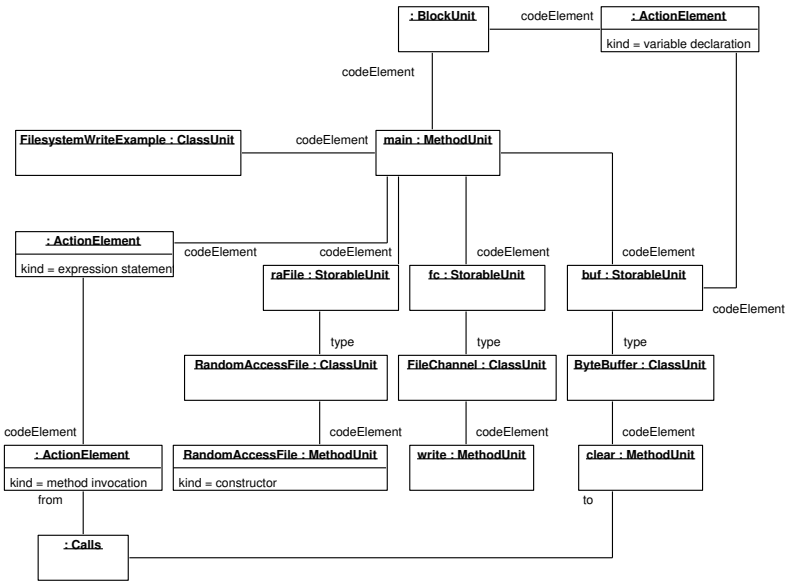


Figure 7.6. KDM instance excerpt that models the source code of Listing 7.3. The `write MethodUnit` does not indicate that data is written to the filesystem.

Dynamic Analysis for CEC Violation Detection The CloudMIG method focuses on *CEC violations* that can be validated using a static source code analysis. However, as described in the previous paragraphs, several *CEC violations* can only be reliably detected when incorporating information from a dynamic analysis. Such information can either originate from a system running in the status quo deployment or from the migrated system that already runs in a cloud environment. The latter case can make sense as some cloud environments do not prevent systems from being deployed despite of exhibiting several *CEC violations*. Those *CEC violations* may only lead to errors when the corresponding statements get called the first time. In general, to decide if a *CEC violation* is raised by a system, appropriate runtime data can be obtained and processed corresponding to three different approaches that are described in the following. Each approach can be applied to the status

quo deployment or to the cloud deployment. Furthermore, each approach can be implemented as a constraint validator (CloudMIG uses so-called constraint validators to check for *CEC violations*, cf. Section 7.3.2).

1. **Offline Analysis:** Runtime data is collected over time, for example, service calls, monitoring log data, firewall message logs, and response times. The historic data is mined in an offline analysis to search for records that indicate *CEC violations* regarding specific CECs, for instance, long lasting method calls that would exceed a cloud-specific threshold or the opening of a network port that would be prevented by a cloud environment.
2. **Online Analysis - System Monitoring:** This approach performs an online analysis and may incorporate data at runtime from various sources similarly as described above. It monitors the real operation of the production system. As the corresponding monitored information is immediately available, the analysis can be applied online.
3. **Online Analysis - Constraint Validator Drivers:** A corresponding constraint validator could be designed to function as a kind of driver that tests the system at runtime regarding specific CECs. That means, it would invoke a functionality—or more generic, trigger system activities according to a specific protocol—and record the system's responses. As the corresponding information regarding a single check is immediately available, and the test procedure might include various checks and last a considerable amount of time, the analysis could be applied online as well. For examining the system's responses, the constraint validators that are employed in the second approach can be reused. However, the driver components have to be tailored for each system that is supposed to be migrated to the cloud.

7.1.4 Addressed Constraints

As described in the previous Section 7.1.3, the CloudMIG method allows to consider a range of CECs and potential *CEC violations* that can be checked

with the help of diverse detection mechanisms. These mechanisms cover both static and dynamic analyses and differ in the examined system artifacts, dependence on specific programming languages, and processing of literals, for instance. CloudMIG allows to integrate detection mechanisms from all of the detectability categories DC1-DC5, as well as static and dynamic analyses. Though, the focus of CloudMIG's tool architecture lays on statically analyzing KDM models that are extracted from a system's source code (cf. Chapter 9). These analyses involve a wide range of elements and statement types, e.g., the corresponding KDM elements for packages, classes, import statements, member declarations and initializations, loop statements, method calls, passed parameters, and conditional statements. Therefore, CloudMIG provides detection mechanisms for CECs that induce *CEC violations* that are members of the DC2 and DC3 categories. Each CEC can be instantiated by a cloud profile contributor (cf. Section 6.3.3) to model the specifics of the CEC regarding a particular cloud environment. For example, if a cloud environment forbids calling a specific method *M*, the fully qualified name of method *M* would be specified by the cloud profile contributor as an attribute of a `MethodCallConstraint` instance. Further details regarding the modeling of addressed CECs are described in the following Section 7.2.

7.2 Constraint Modeling

This section describes the means for modeling CECs with the CloudMIG method. The corresponding meta-model allows to address CECs of arbitrary cloud environments and is provided by CEM's *Constraints* package. Hence, a cloud profile contributor (cf. Section 6.3.3) can model the specific CECs that were identified for the described cloud environment by including particular elements of the *Constraints* package in the provided cloud profile. Besides enabling the modeling of arbitrary cloud environments and corresponding CECs, CloudMIG also allows to describe these CECs in a way that is agnostic regarding particular programming languages. Indeed, as described in Section 7.1.3, the *CEC violation* detection approach may require language-dependent detection mechanisms in some cases. However, the

actual specification of CECs does not demand cloud profile contributors to define multiple language-dependent descriptions of a single CEC.

The remainder of this section is organized as follows. Section 7.2.1 describes the CEM's *Constraints* package. Then, Section 7.2.2 describes how OMG's SMM can be utilized for generic constraint modeling.

7.2.1 The Constraints Package of CEM

CEM's *Constraints* package provides the elements to define the CECs of arbitrary cloud environments. As can be seen in Figure 6.16, the *Constraints* package only relies on elements of CEM's *Core* package. CECs can be specified for IaaS as well as for PaaS-based cloud environments. Table 7.1 gives an overview on the CECs that are defined in the *Constraints* package and that are described below. Besides the CECs themselves, the table states for each CEC whether *CloudMIG Xpress* provides corresponding static detection mechanisms (i.e., constraint validators, cf. Section 7.3.2).[10] Furthermore, the table shows if corresponding *CEC violations* can be detected by means of static and/or dynamic analyses. For static analyses, it additionally states if the detection mechanisms have to follow a language-dependent or language-agnostic approach. Each of the stated CECs corresponds to a class with the same name in the *Constraints* package of CEM. The package can also be transformed to KDM with the mechanism described in Section 6.3.4. For example, the tool *CloudMIG Xpress* uses this KDM representation to process corresponding cloud profiles. An excerpt of the *Constraints* package that also demonstrates the mapping to KDM is shown in Figure 7.7.

In the following, we describe the CECs and the further elements that are included in the *Constraints* package. The `EnvironmentConstraintConfiguration` forms a container structure that enables to include a set of CECs in a cloud profile. Concrete CECs are modeled by inheriting from the class `AbstractConstraint`, which provides an attribute for a name, a description, and a `ViolationSeverity`, for instance.

[10] As of this writing, the latest version of *CloudMIG Xpress* is V. 0.5 Beta.

Table 7.1. The CECs of CEM's *Constraints* package

Constraint Name	Addressed in *Cloud-MIG Xpress* V. 0.5 Beta	Detectable with Static Analysis	Detectable with Dynamic Analysis	Language-agnostic Static Validation
BasicOCLConstraint	✓	✓	-	✓
DBConnectionTimeoutConstraint	-	✓	✓	-
FileSystemAccessConstraint	✓	✓	✓	-
FirewallPortRangeConstraint	-	✓	✓	-
LanguageConstraint	✓	✓	✓	✓
LibraryConstraint	-	✓	✓	✓
LocalTransientStorageConstraint	✓	✓	✓	-
MaxTotalNrOfFilesConstraint	✓	✓	✓	✓
MethodCallConstraint	✓	✓	✓	✓
NICConstraint	-	✓	✓	-
OSConstraint	-	✓	✓	-
ReflectionConstraint	✓	✓	✓	-
RuntimeContainerLifetimeConstraint	-	✓	✓	-
SocketOpeningConstraint	✓	✓	✓	-
SMMConstraint	✓	✓	-	✓
SpecificIPAddressConstraint	-	✓	✓	-
TypesInstantiationConstraint	✓	✓	✓	✓
TypesWhitelistConstraint	✓	✓	✓	✓

BasicOCLConstraint is a CEC that can be defined by specifying an OCL expression. It is the most basic constraint as it enables to specify restrictions in a generic form. The `BasicOCLConstraint` can be utilized for use cases that do not match with any of the existing specialized CECs. It is checked via statically analyzing KDM models that were extracted from a system's source code and does not depend on any language-specific knowledge. The OCL expression has to start with the `context` keyword that specifies the context for the expression. Then, it has to define an invariant:

context *ctx* **inv:** *invariant*

OCL keywords are marked bold. The OCL expression has to incorporate only KDM elements. The context *ctx* has to correspond to a KDM class, for instance. As an example, we assume we had to rebuild the constraint

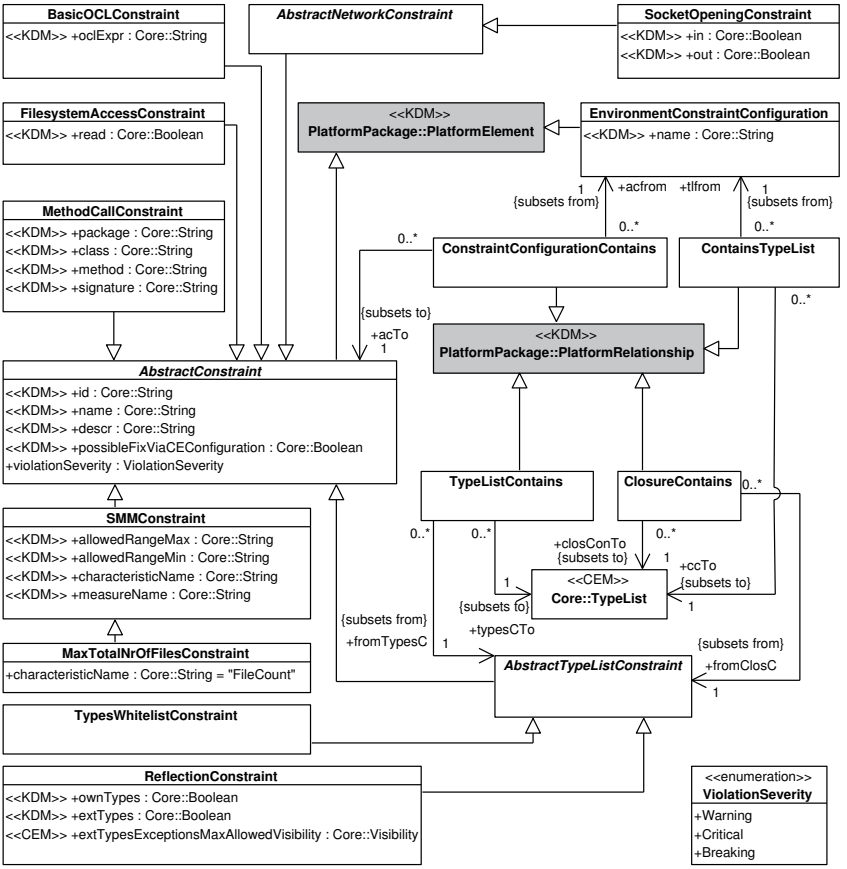


Figure 7.7. The *Constraints* package of CEM (excerpt) including the mapping to KDM elements (cf. [Frey et al. 2013a])

`TypesInstantiationConstraint`, that models the prohibition of instantiating a specific type (this CEC is described in more detail below), with a `BasicOCLConstraint`. We revisit the example of Listing 7.2 and Figure 7.5 that described a corresponding *CEC violation* as the class `ObsoleteClass` was instantiated by the class `Client`. When rebuilding the CEC with a

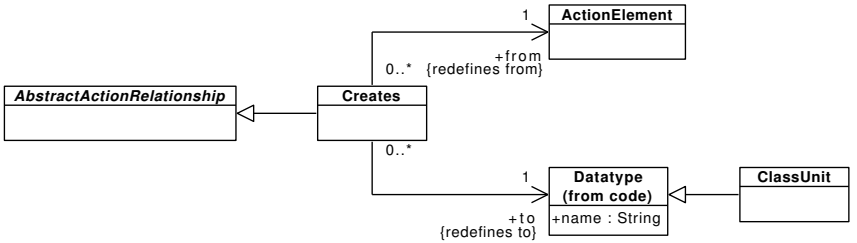


Figure 7.8. The simplified meta-model class structure of KDM for modeling the instantiation of a class (from [Object Management Group 2011a])

`BasicOCLConstraint` to forbid the instantiation of `ObsoleteClass`, the meta-model elements of KDM have to be considered that represent the instantiation. The KDM class `Creates` models the instantiation of a type. The relevant KDM elements for building an appropriate OCL expression are shown in Figure 7.8. `Creates` is a specific `AbstractActionRelationship` from KDM's *Action* package that connects the element that is instantiated with the element that triggers the instantiation. The latter is an `ActionElement` that is reached via the `from` reference. The instantiated element is modeled by the `Datatype` class from KDM's *Code* package and is reached via the `to` reference. A class is a specific `Datatype`. Therefore, Figure 7.8 shows the class `ClassUnit` that inherits from `Datatype`. Given these elements, the OCL expression for building a `BasicOCLConstraint` that raises a violation if the class `ObsoleteClass` is instantiated, can be specified as follows:

context Creates **inv: self**.to.name < > 'ObsoleteClass'

The OCL expression sets the class `Creates` as the *context*. When a `Creates` instance is found in a KDM code model, the `self` keyword refers to this specific instance of `Creates`. The invariant states that the name of the element that can be reached though the `to` reference (the instantiated element) is not allowed to equal "ObsoleteClass." For detecting corresponding *CEC violations*, the OCL expression is evaluated for each `Creates` element that exists in a KDM model that was extracted from the source code of an enterprise system. A *CEC violation* is detected if the invariant does not hold,

i.e., if the name of the element that can be reached via the `to` reference equals "ObsoleteClass."

The other CECs allow cloud profile contributors to define restrictions in a more convenient way, as the CECs are tailored for use cases that are more specific. They reduce the complexity of defining the CECs, as the cloud profile contributors rather need to configure the properties of CECs via specifying literals instead of full-flavored expressions. The `SMMConstraint` constitutes an exception as it requires cloud profile contributors to specify a sophisticated and powerful SMM model. The `SMMConstraint` is therefore described separately in Section 7.2.2.

DBConnectionTimeoutConstraint is used for constraints that define a timeout for connections to a database service that is provided by a cloud environment. That means, if a connection is not used for a specific time period, an application will face a connection timeout error when trying to use this connection again after the specified time passed. This CEC may result in *CEC violations* that fall into any of the detectability categories DC1-DC5. For example, a database connection timeout could be set in the source code via calling a corresponding method. Hence, detection would be possible using static, language-dependent analyses. However, the call to such a method and the used parameter would not allow to definitely determine if a *CEC violation* would occur for a specific database service of a cloud environment and for all read and write operations from and to the database. Therefore, corresponding *CEC violations* would be considered to fall under DC2. Detecting *CEC violations* by means of dynamic analyses are more reliable for this CEC in any case. If this would also constitute the only possibility to detect *CEC violations* for a specific setting, they would be members of the detectability category DC1.

FileSystemAccessConstraint allows to model a forbidden file system access. As shown in the example of Listing 7.3, corresponding *CEC violations* cannot be detected with a static analysis and language-agnostic detection mechanisms. However, intercepting and monitoring operating system calls at runtime would also allow to build a constraint validator that applies a dynamic analysis.

FirewallPortRangeConstraint can be used to model restrictions regarding the opening of network ports. It allows to specify a valid port range together with allowed network protocols for both incoming and outgoing traffic (cf. Figure A.3 in Appendix A). Regarding the detection of *CEC violations* apply similar conditions as described for the `DBConnectionTimeoutConstraint`. Both can exhibit *CEC violations* that cover the whole range of detectability categories DC1-DC5 and are generally better suited for detection mechanisms that include a dynamic analysis.

LanguageConstraint is a CEC that restricts the usage of programming languages for applications that can be deployed and executed in a cloud environment. The KDM models that are extracted from a software system include the information regarding the programming language of the underlying source code. Hence, corresponding *CEC violations* of the *LanguageConstraint* can, despite the name suggests the opposite, be detected with a static, language-agnostic analysis. Nevertheless, building a constraint validator that examines log files or memory signatures from running applications would also allow to use dynamic analyses. However, those kinds of detection mechanisms would depend on the specific trails of applications that are built with a particular programming language.

LibraryConstraint can be used to prohibit the usage of a specific library or to allow only the use of a particular version of a library (cf. Figure A.3 in Appendix A). Similarly to the *LanguageConstraint*, corresponding information that reveals the usage of a specific library is included in a KDM instance. However, also mechanisms for detecting the *CEC violations* with the help of runtime data can be built.

LocalTransientStorageConstraint models a restriction regarding the persistency of data that is stored locally by applications running in a cloud environment. For example, several IaaS-based cloud environments offer VM instances that lose all of the data that an application writes to the local hard drive when shutting down the VM instance. Considering Table 7.1, the entries for this CEC correspond to the ones of *FileSystemAccessConstraint* and also to their explanations.

MaxTotalNrOfFilesConstraint allows to define a maximum number of files that can be used. It is implemented employing the according SMM model that is outlined in the later Section 7.2.2. Furthermore, the system artifacts that form an actually deployed application are also modeled in extracted KDM instances. For example, instances of KDM's `BinaryFile`, `Image`, or `ExecutableFile` class can be included in the KDM models. Hence, the static analysis does not depend on any knowledge that is specific for a particular programming language. Moreover, building a constraint validator that gathers those types of information at runtime is also possible.

MethodCallConstraint can be used to prohibit the calling of a specific method. Method calls are represented in KDM models identically, irrespective of the programming language that underlies the KDM model. Hence, corresponding *CEC violations* can be detected using language-agnostic detection mechanisms. However, the example in Listing 7.1 demonstrated that a method call might be only detectable by using language-specific detection mechanisms if it is not explicitly contained in a KDM model, for example, when employing means of reflection.

NICConstraint allows to model a restriction regarding the number of Network Interface Cards (NICs) that can be used by an application running in the cloud environment that is modeled by the associated cloud profile. Depending on the used programming languages, libraries, configuration files, functions, and parameters, corresponding *CEC violations* may fall under each of the detectability categories DC1-DC5 as already described in the context of the `DBConnectionTimeoutConstraint`. The same applies to the possibility to employ dynamic analyses and language-dependent detection mechanisms.

OSConstraint models a restriction regarding the operating systems that can be used to run applications in a cloud environment. This may either refer to operating systems that have to be deployed by the cloud user in the form of VM images, or as part of the platform that is offered by PaaS-based cloud environments. As existing KDM extraction mechanisms could be easily extended to include the used operating system in a KDM instance, the language-agnostic detection would be possible using a (trivial) static analy-

sis. However, considering platform-independent programming languages such as Java, the usage of a specific operating system in the context of the status quo deployment does not necessarily imply that the system could not be run using another operating system. Hence, determining if a *CEC violation* actually is raised requires using language-dependent knowledge, as (at least) a list of compatible operating systems has to be provided for each programming language. Furthermore, the same applies for dynamic analyses. Indeed a constraint validator can be easily built that checks the used operating system at runtime, but inferring compatibility with other operating systems cannot be computed without further language-dependent information.

ReflectionConstraint is a CEC that forbids the usage of reflection mechanisms. The usage of those mechanisms differs among the programming languages that offer reflection. As this kind of information is not included in a KDM instance (e.g., see the description of Listing 7.1) language-dependent knowledge is needed to statically detect corresponding *CEC violations*. Furthermore, analyzing applications at runtime that use reflection mechanisms is also possible. For example, a simple approach would use suitable instrumentation mechanisms such as AspectJ [Kiczales et al. 2001] or PostSharp[11] to intercept with the system's control flow at a fine-grained level and try to detect the usage of reflection mechanisms.

RuntimeContainerLifetimeConstraint enables modeling a maximum lifetime of a "runtime container." The term relates to instances of the class `AbstractCloudRuntimeContainer` from CEM's *Core* package. For example, `AbstractWorker` from the *PaaS* package inherits from `AbstractCloudRuntimeContainer` and represents worker elements of a PaaS-based cloud environment, for example, processes and threads. Employing a dynamic analysis should be preferred when building a corresponding detection mechanism. However, when accepting several false positive results, a static analysis can be used as well for indicating problematic parts (DC2). For example, the simulation tool *CDOSim* provides a mode that only relies on a static analysis for estimating the instruction count of an application. This instruction count

[11] `http://www.sharpcrafters.com/`

is then used together with language-specific information to simulate the deployment to a specific cloud environment (cf. Section 8.2).

SocketOpeningConstraint models a restriction to open network sockets for incoming or outgoing network traffic. Statically detecting related *CEC violations* requires language-specific knowledge. For example, it has to be known which methods of a programming language open a network socket. Building a constraint validator that employs a dynamic analysis is also possible, e.g., through examining log files of a firewall.

SMMConstraint allows to define a CEC through specifying an SMM model. The CEC is then checked though evaluating the SMM model against an extracted KDM instance (cf. Section 7.2.2). Similarly to a `BasicOCLConstraint`, an `SMMConstraint` can only be used in the context of a static analysis.

SpecificIPAddressConstraint is a constraint that limits the usage of IP addresses to a specific address range. Similarly to the `DBConnectionTimeoutConstraint`, potential *CEC violations* regarding the `SpecificIPAddressConstraint` might be detectable with a static analysis by means that span DC2-DC5. In those cases, language-specific knowledge has to be included, e.g., whether a specific method call with a particularly formed string literal parameter specifies the request of an IP address. However, there might exist settings that require information that was obtained during runtime and therefore fall under DC1.

TypesInstantiationConstraint restricts the instantiation of a specific type. As explained in the description of Figure 7.8, the necessary elements for detecting the instantiation of a type with the help of a static analysis are included in the KDM models that are extracted from a system's source code. Furthermore, this mechanism is agnostic to specific programming languages. Detecting the instantiation of specific types by means of a dynamic analysis might be possible, but would require fine-grained instrumentation or detailed log data.

TypesWhitelistConstraint is a CEC for defining a set of white-listed types that are allowed to be used. A *CEC violation* is raised when a type is used that is not part of this list. KDM models are sufficient for detecting corre-

sponding *CEC violations* with the help of a static analysis. Similarly to the `TypesInstantiationConstraint`, using a dynamic analysis may be possible but would require deep inspection.

When specifying any of the CECs that were described before, it is possible to supplement the definition in the cloud profile with any number of proposed solutions, i.e., instances of the class `ProposedSolution` (cf. Figure A.3). Such instances are specified by cloud profile contributors (cf. Section 6.3.3) to support CloudMIG users in the case that specific *CEC violations* are detected. For example, if a *CEC violation* regarding a `LocalTransientStorageConstraint` is detected, the CloudMIG user could be presented a proposed solution that suggests to use one of the cloud environment's specific services for implementing persistency.

7.2.2 Generic Constraint Modeling with SMM

The Structured Metrics Metamodel (SMM) of the OMG enables the specification of measurement processes and defines accompanying concepts, such as measures, measurements, and observations (cf. Section 3.3.3). SMM models can be used as a powerful means for defining CECs by specifying an `SMMConstraint`. Similarly to the `BasicOCLConstraint`, the `SMMConstraint` can be detected using static analyses and is constructed to be more generically than the other specialized CECs. Moreover, utilizing an `SMMConstraint` enables to benefit from the smooth alignment of KDM and SMM models. For checking the constraint, the SMM model is evaluated against the extracted KDM instance with the help of our MAMBA Execution Engine (MEE).[12] MEE applies metrics that are modeled with SMM to KDM instances. In this section, we sketch *MEE*'s measurement process to demonstrate an example for building a simple SMM instance that can be used to specify an `SMMConstraint` that mimics the functionality of the `MaxTotalNrOfFilesConstraint`. That means, the corresponding SMM model counts the number of file elements that are present in a KDM instance. This result can then be used to check if

[12] In our previous work [Frey and Hasselbring 2011a; Frey et al. 2013a], the acronym MEE stood for *Metrics Execution Engine*. It was renamed to MAMBA Execution Engine in the course of the integration into the MAMBA framework [Frey et al. 2011; 2012].

the measured number of files is below the threshold that is defined by the `SMMConstraint`.

Basic Involved SMM Elements The general structure of a valid SMM instance consists of a `Characteristic` which defines a common trait of the referenced measures. There must exist at least one measure which relates to the characteristic. Figure 7.9 shows the included elements of the SMM instance that is used for counting the number of files. Figure 7.9 also shows three different SMM measure types. The `BinaryMeasure` applies the defined functor operation to the results given by its both base measures. Furthermore, an `AdditiveMeasure` is defined which accumulates the results given by its single base measure. There is also a `Counting` measure defined which applies a stated operation directly to KDM elements. Each measure relates to one `Scope` that defines the class type to which the measure can be applied. Finally, an SMM instance can contain measurements. Those elements are the results of metric computations and will be attached automatically by *MEE* during the measurement process.

The Measurement Process The measurement process of *MEE* applies SMM measures to KDM-based models. It can be divided into the following three phases.

1. Initialization
2. Recursive measure application
3. Assignment

During the initialization phase, *MEE* has to discover several elements as a start. First, it determines the entry point of the overall defined metric. Therefore, it has to find the measure which is not a base measure for any other measure defined in the current context. The modeler of an SMM instance has to ensure that there exists exactly one measure which fulfills this condition. In the following, this measure is called the *main measure*. After

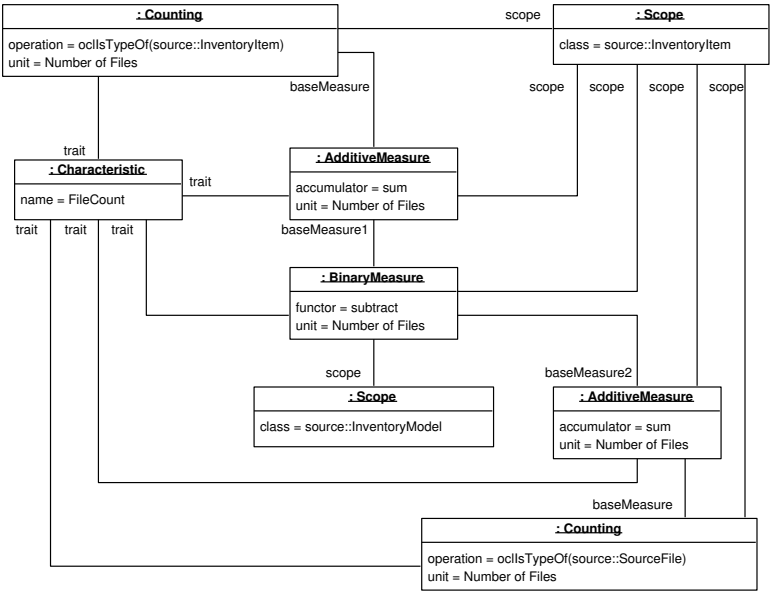


Figure 7.9. A UML object diagram showing an SMM file count measure variant (an SMM instance) to compute the number of files omitting source code files (cf. [Frey et al. 2013a])

the engine has found the *main measure*, it discovers the KDM instances for all types that match the *main measure*'s scope because this measure has to be applied to all of them separately. Afterwards, the second phase starts and the *main measure* along with its associated measures will be applied to the KDM instances discovered in the preceding phase. Further proceeding depends on the measures' types. For example, a single utilization of `Counting` applied directly on a KDM instance element produces a single measurement, whereas the `AdditiveMeasure` results in computing multiple measurements and the according summing up. As mentioned above, the measurement results will be attached to the SMM instance. This is accomplished in the last phase. The integration of MEE in the measurement framework MAMBA is described in Section 10.2.

File Count Measure The Figure 7.9 that was introduced above shows an SMM instance which can be used to count the number of files. Hence, the characteristic is named "FileCount." As it is utilized in a constraint validator that considers only files that are being deployed to production systems, this SMM instance omits source files. Generally, the KDM instances contain, among other models, an `InventoryModel`. All files related to the software system are described in such a model. The `BinaryMeasure` will be applied by *MEE* as a *main measure* because it is not a base measure for any other measure. The engine searches for an `InventoryModel`. If it discovers one, it will first apply the `baseMeasure1` to all `InventoryItems` contained in the model. The result will be the number of all files included in the software system model. Afterwards, the engine applies the `baseMeasure2` to all `InventoryItems`. The resulting measurement represents the number of all source files contained in the KDM model. Finally, the engine subtracts the number of source files from the number of all files. Thus, the result $r \in \mathbb{N}_0$ will be the number of all files except the source files. Each `SMMConstraint` defines a range of values $[min, max]$ it accepts as valid results (cf. Figure A.3 in Appendix A). For example, if a cloud environment allows deploying a maximum of 500 files, this range would be set to $[0, 500]$ by a cloud profile contributor. If r is not included in this range ($r \notin [0, 500]$), a *CEC violation* is raised.

7.3 Automatic Constraint Validation

CloudMIG employs so-called *constraint validators* for detecting *CEC violations*. A constraint validator is a software component that is provided by *CEC validation contributors* (cf. Section 6.3.3). Hence, CloudMIG users not only can draw on provided cloud profiles and CECs that are specified therein, but also can reuse existing means for automatically checking the conformance of an SUA regarding particular cloud environments. The next Section 7.3.1 describes CloudMIG's constraint validation process for detecting *CEC violations*. Then, Section 7.3.2 presents basic concepts of the involved constraint validator components.

7.3.1 Constraint Validation Process

CloudMIG's constraint validation process for the detection of *CEC violations* is formed by a set of interrelated conformance checking activities. These activities and their relations are shown in Figure 7.10.

Activities and their Relations The constraint validation process includes 11 activities. Not all activities have to be executed in any case. For example, if no constraint validator exists for a defined CEC, the process may terminate earlier as no automatic detection can be applied for this CEC. The activities and their relations are described in the following, the numbers correspond to the activity numbers as depicted in Figure 7.10.

1. **Load KDM Model** The constraint validation process examines the KDM model that is extracted from a software system that is supposed to be migrated to a cloud environment (SUA). This activity loads the corresponding KDM model so it can be processed in subsequent activities.
2. **Detect Programming Language** As some CECs can only be detected by language-dependent detection mechanisms (cf. Section 7.1.3), it is necessary to consider the programming language of the SUA. This information can be easily recovered from the extracted KDM model. In the case language-agnostic constraint validators suffice for analyzing defined CECs, the detected programming language is disregarded.
3. **Load Constraint Validators** Initially, all constraint validators have to be loaded. As constraint validators can be dynamically added by CEC validation contributors, they have to be bound to the CEC violation detection platform to, for example, query their capabilities in subsequent activities.
4. **Select next Cloud Environment Configuration** The CloudMIG user may have selected multiple cloud environment configuration (cf. Section 6.3.4) candidates. They are processed subsequently. This activity fetches the next unprocessed cloud environment configuration.

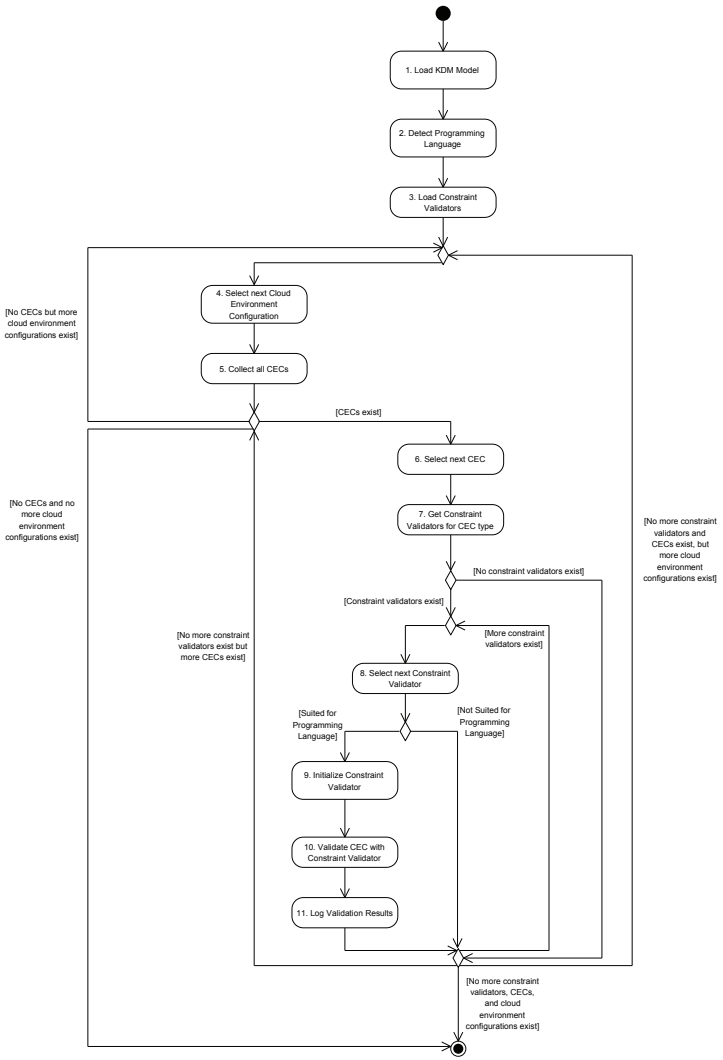


Figure 7.10. Constraint validation process

5. **Collect all CECs** A cloud environment configuration contains an arbitrary number of CECs. This activity retrieves the CECs of a cloud environment configuration from the corresponding cloud profile.

6. **Select next CEC** Similarly as cloud environment configurations, the CECs of each cloud environment configuration are processed subsequently. This activity fetches the next unprocessed CEC.

7. **Get Constraint Validators for CEC type** In general, each CEC corresponds to a specific type that inherits from the `AbstractConstraint` class of CEM's Constraints package (cf. Figure 7.7). Each constraint validator is capable to process just a single CEC type (cf. Section 7.3.2). However, there may exist multiple constraint validators that can detect *CEC violations* for a single CEC type. Hence, this activity selects the relevant constraint validator subset from all constraint validators.

8. **Select next Constraint Validator** The constraint validators are processed subsequently. Thus, this activity fetches the next unprocessed constraint validator that is suited for the used programming language.

9. **Initialize Constraint Validator** The constraint validator has to be initialized. For example, the extracted KDM model is passed or all registered SMM models may be delivered to the validator.

10. **Validate CEC with Constraint Validator** The constraint validator processes the passed KDM model in this activity. In general, each constraint validator traverses KDM's graph structure and searches for patterns that are specific for violations of a particular CEC type. Hence, the validation procedure of each constraint validator is determined by two characteristics $C1$ and $C2$:

    - **C1:** An arbitrary number $n \in \mathbb{N}^+$ of those patterns
    - **C2:** The method for traversing the graph of interconnected KDM elements

    Section 10.3 provides an example for $C1$ and $C2$.

11. **Log Validation Results** After the detection of *CEC violations* finished for the currently processed CEC, a report of the corresponding results is stored. The report includes details regarding each KDM element that causes a *CEC violation*.

Conformance Checking States The constraint validation process integrates the detection of *CEC violations* for all cloud environment configurations that are of interest to the CloudMIG user. This consideration of multiple cloud environment configurations is mainly represented by the constraint validation process' activity 4 that subsequently fetches those configurations. Regarding a single cloud environment configuration that is supposed to be used as a target for the cloud migration, CloudMIG distinguishes several states of the corresponding legacy software system according to the validation of its CECs, so-called Conformance Checking States (CCSs). The CCSs are shown in Figure 7.11.

Before the initial validation of the *CECs*, the software system is marked as "unchecked." Afterwards, further proceeding depends on the existence of *CEC violations* and the *violation severities* of detected *CEC violations*. If *Breaking* violations exist, the SUA's CCS is considered to be "incompatible." For example, CloudMIG does not provide support for migrations that would imply a transformation from one programming language into another. Therefore, the workflow ends in the case that *Breaking* violations exist. Otherwise, the software system is either regarded as "compatible" (no *CEC violations* exist) or "pending" (*Critical* or *Warning* violations exist). The distinction is made as for "pending" legacy systems feedback for the reengineer and tracing to the sources of the *CEC violations* has to be provided, for example. CloudMIG's activities A3–A5 are executed subsequently in both cases. The detected *CEC violations* can vary over time. For example, considering *CEC violations* that are raised because of an incompatibility with a VM instance's operating system that is planned to be used. When modifying the deployment plan during CloudMIG's activity A4, another operating system could be selected. Consequently, the corresponding CEC violations would not be raised when re-running the constraint validation process. Therefore both Conformance Checking States "Aligned (Clean)" (no *CEC violations* exist) and "Aligned

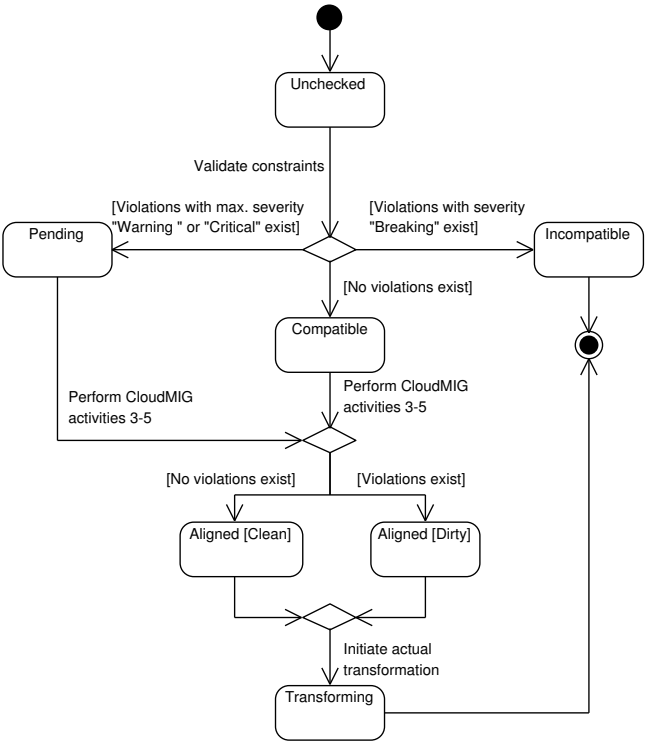


Figure 7.11. The conformance checking states of a software system for a specific target cloud environment configuration in CloudMIG (cf. [Frey et al. 2013a])

(Dirty)" (*Critical* or *Warning* violations exist) can be reached along both paths after final constraint validation. Accomplishing the actual transformation (CCS "transforming", cf. CloudMIG's activity A6 in Section 6.3.2) ends the migration from CloudMIG's perspective.

7.3.2 Constraint Validators

The CECs are analyzed with so-called constraint validators. As with components for most of the approach's activities, they can be plugged into the architecture, too (cf. Chapter 9). A constraint validator checks if a software system complies with a particular CEC type. Furthermore, constraint validators can be suited for searching *CEC violations* that can be found with language-agnostic as well as with language-specific detection mechanisms (cf. Table 7.1). CloudMIG's architecture and the accompanying tool *CloudMIG Xpress* provide a set of predefined constraint validators. In addition, CEC validation contributors can add further constraint validators. Creating additional constraint validators can be reasonable in several cases, for example:

- For adding support for CECs that are not already covered by the predefined constraint validators (cf. Table 7.1)
- For expanding the detection capabilities regarding a CEC type that is already addressed, e.g., for covering corresponding *CEC violations* of the same CEC type but of additional detectability categories
- For supporting a further programming language so that language-specific CECs can also be checked for a new language

Classes playing a central role for checking CECs are shown in Figure 7.12. The stereotypes «CEM» and «KDM» indicate if the methods or attributes use types from the CEM or from KDM, respectively. The classes in Figure 7.12 form a logical interface between the CEM (dark gray) and the meta-model for specifying constraint validators (light gray). This is of particular importance as the instances of these meta-models are created by distinct roles and have to rely on the common logical interface. Constraint validators are provided by CEC validation contributors. In contrast, cloud profiles that contain specifications for CECs are instances of CEM and are provided by cloud profile contributors. Constraint validators have to provide a subclass of `AbstractConstraintValidator`. For reasons of convenience

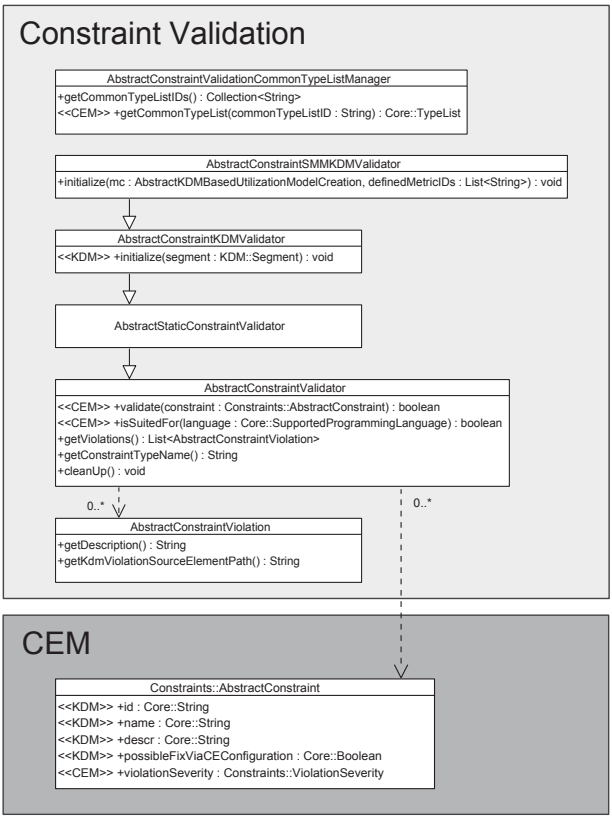


Figure 7.12. Important classes for checking the conformance of CECs (based on [Frey et al. 2013a])

there exist three further abstract subclasses, namely variants for static validation (`AbstractStaticConstraintValidator`), for static validation with respect to KDM source code models (`AbstractConstraintKDMValidator`), and for incorporating metrics that are modeled with SMM and are applicable to KDM-based models (`AbstractConstraintSMMKDMValidator`). Constraint validators refer to a specific constraint type (sub classes of `Abstract-`

Constraint from CEM's *Constraints* package), for a given *CEC* there may exist various constraint validators. An AbstractConstraintValidationCommonTypeListManager handles predefined lists of types. Those type lists are either provided by CloudMIG to the constraint validators, or can be contributed by constraint validators themselves. An example regarding the utilization of type lists is given below. This example also demonstrates the principle functioning of a single constraint validator for checking the conformance of a software system regarding a specific CEC. The example uses the CEC SocketOpeningConstraint and the corresponding constraint validator SocketOpeningConstraintValidator. Figure 7.13 shows the validation of the SocketOpeningConstraint. The component coordinating the validation is called CECValidationController in this example. The method calls and the responses are described in the following, the numbers correspond to the method call numbers in Figure 7.13.

1. The call to getConstraintTypename() returns the name of the CEC that can be processed by the constraint validator. The method call is part of activity 7 ("Get Constraint Validators for CEC type") of the constraint validation process in Figure 7.10. The SocketOpeningConstraintValidator addresses the SocketOpeningConstraint. Therefore, "SocketOpeningConstraint" is returned. This example assumes that the processed cloud environment configuration defines a SocketOpeningConstraint that is currently processed. Hence, SocketOpeningConstraintValidator is part of the subset of all constraint validators that is assembled in the constraint validation process' activity 7 and the validation does not end at this point.

2. The call to the isSuitedFor method of SocketOpeningConstraintValidator is part of the constraint validation process' activity 8 ("Select next Constraint Validator"). The CECValidationController queries the constraint validator whether the programming language of the source code that underlies the KDM instance is supported. The example assumes that Java was used, so isSuitedFor is called with CEM's enumeration value SupportedProgrammingLanguage.Java. Furthermore, the example assumes that SocketOpeningConstraintValidator supports Java and thus returns true. It should be noted that in the case of language-agnostic validation

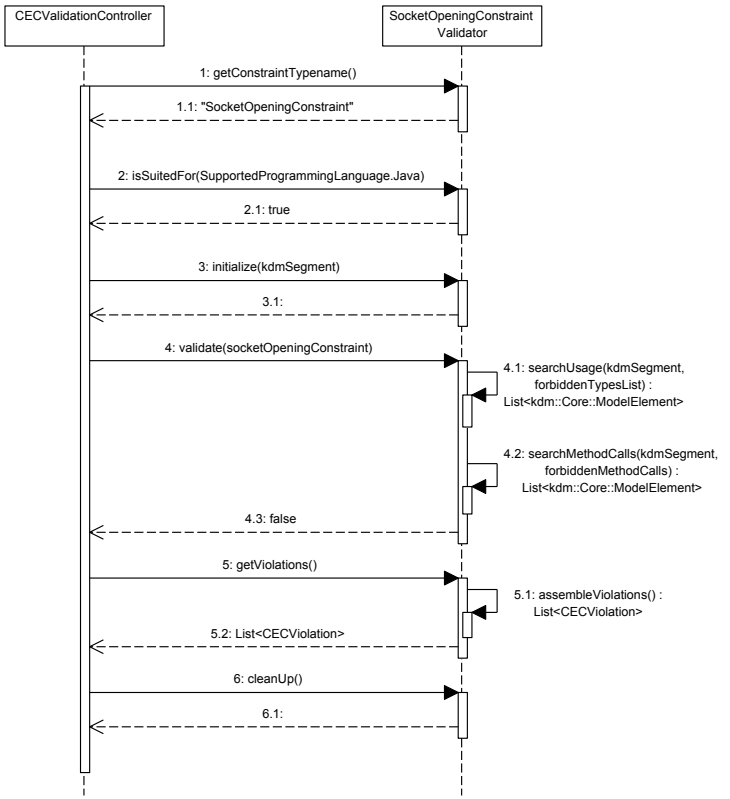


Figure 7.13. Validation of a `SocketOpeningConstraint` with a `SocketOpeningConstraint-Validator`

plugins the `isSuitedFor` method always returns `true`. However, as stated in Table 7.1, `SocketOpeningConstraint` cannot be statically analyzed with language-agnostic detection mechanisms.

3. The call to the `initialize` method of `SocketOpeningConstraintValidator` represents the constraint validation process' activity 9 ("Initialize Constraint Validator"). As `SocketOpeningConstraintValidator` inherits from

`AbstractConstraintKDMValidator`, the extracted KDM model is passed to the `initialize` method.[13]

4. The call to the `validate` method of `SocketOpeningConstraintValidator` represents the constraint validation process' activity 10 ("Validate CEC with Constraint Validator"). The `CECValidationController` passes the CEC that is specified in the cloud profile to the `SocketOpeningConstraint-Validator`. The constraint validator can therefore access all properties of the CEC that were specified by a cloud profile contributor. The predefined `SocketOpeningConstraintValidator` uses a list of forbidden types and of forbidden methods. That means, those Java types and methods included in the lists are known to open a network socket.

   As `SocketOpeningConstraintValidator`'s Java variant is used, the type list includes specifications, e.g., for the JRE types `javax.net.ssl.SSLSocket` and `java.net.DatagramSocket`, whereas the method list contains a specification for the method `javax.rmi.ssl.SslRMIClientSocketFactory.createSocket()`, for instance. The KDM model is searched taking account of both lists. That means, it is searched for elements that instantiate forbidden types or call forbidden methods. This example assumes that those elements are found and *CEC violations* are therefore detected. Hence, the `validate` method returns `false` to indicate that the validation did not succeed.

5. The call to the `getViolations()` method of `SocketOpeningConstraintValidator` is part of the constraint validation process' activity 11 ("Log Validation Results"). The `CECValidationController` fetches all detected constraint violations and logs them (omitted in Figure 7.13 for brevity).

6. The `cleanUp` method of `SocketOpeningConstraintValidator` is called to release allocated resources after the constraint validation process finished.

[13] A KDM `Segment` element that contains the complete KDM model is actually passed to the `initialize` method

7.4 Impact and Handling of Constraint Violations

CloudMIG targets the planning phase of a cloud migration project. The conformance checking approach reveals *CEC violations*, but does not cover the actual migration to a cloud environment or the correction of detected *CEC violations*. The number and severity of the detected *CEC violations* already form important criteria when comparing diverse cloud environment candidates. However, this section shows that uncovered *CEC violations* can actually be utilized to infer further information that can play an important role for guiding an overall cloud migration project.

First, the *CEC violations* are used to define the so-called Cloud Suitability and Alignment (CSA) hierarchy that enables a classification of existing enterprise software systems (SUAs) regarding their suitability for specific cloud environments and their level of alignment after initial migration steps. Second, an approach for determining an adequate ordering regarding the correction of *CEC violations* is presented. It aims at effectively and early mitigating risk and uncertainty that arise from a specific *CEC violation* portfolio.

The remainder of this section is organized as follows. Section 7.4.1 introduces the CSA hierarchy, before CloudMIG's approach for prioritizing *CEC violations* is described in Section 7.4.2.

7.4.1 CSA Hierarchy

To reason about the challenges emerging when migrating a specific system to a cloud environment and restructuring its architecture to facilitate a smooth integration into the cloud's service landscape, the system's suitability has to be judged upfront as well as the level of alignment with the cloud environment once the first steps are accomplished. To enable an evaluation and classification of software systems in this respect, we introduce the coarse grained Cloud Suitability and Alignment hierarchy (CSA hierarchy).

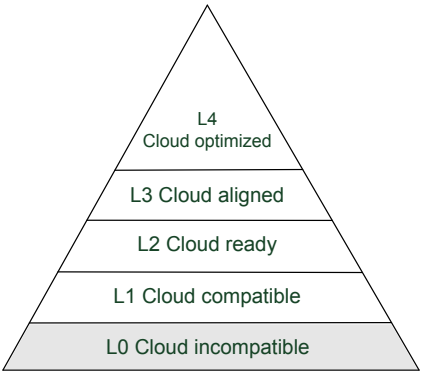


Figure 7.14. The CSA hierarchy (cf. Frey and Hasselbring 2011b)

As illustrated in Figure 7.14, it comprises the five levels *cloud incompatible*, *cloud compatible*, *cloud ready*, *cloud aligned*, and *cloud optimized*. The levels are defined employing the notions of CECs, CEC violations, and associated CEC violation severities and build upon and also formalize the conformance checking states that were defined in Section 7.3.1. The five CSA hierarchy levels are described in the following.

- ***L0* Cloud incompatible:** At least one CEC violation with severity *Breaking* exists.
- ***L1* Cloud compatible:** No CEC violations with severity *Breaking* exist.
- ***L2* Cloud ready:** No CEC violations exist.
- ***L3* Cloud aligned:** The execution context, utilized cloud services, or the migrated software system itself were configured to achieve an improved resource consumption (measurable in decreased costs that are to this effect charged by the cloud provider) or scalability without pervasively modifying the software system.
- ***L4* Cloud optimized:** The migrated software system was pervasively modified to enable automated exploitation of the cloud's elasticity. For

example, it's architecture was restructured to increase the level of parallelization. An evaluation was conducted to identify system parts which would experience an overall benefit from substitution or supplement with offered cloud services. These substitutions and supplements were performed.

The CSA hierarchy is *constructive* in its levels L1-L4. For example, for classifying a software system as *cloud aligned* it has to be *cloud compatible* and *cloud ready* as well. It should be noted that the CSA hierarchy solely considers technical concerns related to a migration to the cloud. In particular, it follows the basic direction of the CloudMIG approach and does not take organizational or economic restrictions into account, for example regarding governance issues, security policies, or a company's business model.

The utilization of the CSA hierarchy can be demonstrated by taking the example scenario of the initial experiment in Section 6.1.1 into account. There exist no *CEC violations* that would impede proper execution after Apache OFBiz's database is transfered to Eucalyptus' persistent block storage, for instance. Concerning Eucalyptus, this activity is sufficient to lift Apache OFBiz 9.04 from *cloud compatible* to *cloud ready*. However, only through selecting the Memory.M VM instance type the application would be *cloud aligned* (cf. Figure 6.5), as this VM instance type enables to minimize resource consumption as is expressed through the lowest possible costs.

The CSA hierarchy defines the relationship of a specific configuration of a software system (e.g., regarding the version of the system's software architecture) and a specific version of a cloud environment. A system being *cloud ready* concerning a specific cloud environment might be *cloud incompatible* regarding another one. Moreover, even for the same cloud environment this could change over time due to modifications of the incorporated cloud services offered by the cloud environment.

Hence, the classification of a software system S regarding the CSA hierarchy depends on its configuration Θ and the cloud services Λ offered by a cloud environment. More specifically, the cloud environment provides n cloud services. A cloud service k is present in a particular version v: $\lambda_k^v \in \Lambda$. The

classification of S regarding the CSA hierarchy level is then called Γ. A *CSA tuple* is defined as follows:

$$\left(S, \Theta, \bigcup_{k}^{n} \lambda_k^v, \Gamma \right) \tag{7.4.1}$$

CSA tuples can be utilized to compare cloud environment alternatives or competing software architectures when considering reengineering activities, for instance.

7.4.2 Prioritization of Constraint Violations

When deciding for a target cloud candidate after validating the *CEC violations* for an existing software system (SUA), the detected *CEC violations* have to be fixed manually in CloudMIG's activity A6 (cf. Section 6.3.2). All of these *CEC violations* have to be addressed to enable proper operation. Nonetheless, reasoning about a suitable order has the potential to improve the actual transformation towards the intended target architecture. The observed *violation severities* can be used as a first indicator to partition the *CEC violations*. Here, the *CEC violations* with higher *violation severities* should be handled primarily because by tendency they imply more uncertainties regarding the overall development effort.

However, the *violation severities* are defined generically in a cloud profile, i.e., in a model describing the target cloud environment itself. Therefore, an existing system's characteristics cannot be taken into account when cloud profile contributors specify violation severities. Referring to this, other important influencing factors that are unique to an SUA can be seen in the type and extent of consequences for other system parts once tackling a particular *CEC violation*, as well as the specific complexity of the element exhibiting the *CEC violation* caused by numerous dependencies.

These factors can be covered through incorporating a type-level coupling metric [Kan 2002]. Furthermore, the *CEC violations* should be prioritized and

grouped reflecting manageable existing boundaries to facilitate appropriate assignment to developers, for instance. Hence, the prioritization approach partitions the *CEC violations* along classes and considers their coupling values as well as the involved *violation severities* as mentioned before.

A system model of an SUA contains a set of classes that we call $\tilde{c}$. We denote the coupling of a class $c \in \tilde{c}$ as `coup(c)` and define $\alpha(c)$ as the number of the class' *CEC violations* to the total number of the system's *CEC violations* ratio. CloudMIG solely provides support for migrating software systems to cloud environments where no *Breaking* violations are detected (cf. Section 7.3.1). Therefore, we further distinguish merely *Critical* from *Warning* violations and define $\gamma(c)$ as the number of the class' *Critical* violations to the total number of the system's *Critical* violations ratio.

$\alpha(c)$ and $\gamma(c)$ have to be considered in combination when reasoning about a prioritization of *CEC violations*. For example, a class exhibiting a large number of *CEC violations* where all violations are just *Warning* violations can have a similar impact to a class that reveals just a few *CEC violations* which are all of the *Critical* severity. Thus, both are included when prioritizing the *CEC violations* and, as they are regarded as equally important, the **class violation weight (cvw)** of class $c \in \tilde{c}$ is defined as follows:

$$cvw(c) = \alpha(c) + \gamma(c) \tag{7.4.2}$$

The final prioritization procedure also incorporates the coupling influence factor and produces three partitions matching the priority levels `I-III`. Priority level `I` corresponds to the most and priority level `III` to the least urgent categories for classes raising *CEC violations*, respectively. Denoting $mean(coup(\tilde{C}))$ as the mean of the coupling values of all classes in the system and $mean(cvw(\tilde{C}))$ as the mean of the class violation weights of all classes in the system, the **prioritization function `prio`** is defined that assigns a priority level to a class $c \in \tilde{c}$ in the following way:

$$prio(c) = \begin{cases} I & , \; coup(c) > mean(coup(\tilde{C})) \; \wedge \\ & \quad cvw(c) \; > mean(cvw(\tilde{C})) \\ II & , \; else \\ III & , \; coup(c) \leqslant mean(coup(\tilde{C})) \; \wedge \\ & \quad cvw(c) \; \leqslant mean(cvw(\tilde{C})) \end{cases} \tag{7.4.3}$$

The prioritization function `prio` is defined for a class $c \in \tilde{C}$ that raises at least one *Critical* or *Warning* violation. As described before, classes included in constellations (a software system evaluated with respect to a specific cloud environment) exhibiting *Breaking* violations or classes without any *CEC violation* need not to be prioritized for rework. Furthermore, once the most problematic classes have been isolated, dependent classes can be identified by directly querying the underlying KDM models or by using MEE to calculate dependency metrics, for instance. An example regarding `prio` and the involved priority levels `I-III` is included in the evaluation of the conformance checking approach in Chapter 11.

7.5 Summary

When migrating enterprise software systems to the cloud, (future) SaaS providers have to consider restrictions that are posed by potential target cloud environments. Those restrictions, such as the limitation to access the filesystem or to open network sockets, are called Cloud Environment Constraints (CECs) in the context of the CloudMIG method. If those restricted operations get activated anyway, an error occures that might be eventually leading to a failure. Those kinds of operations—more generically, those system parts—that might be responsible for a corresponding error are called *CEC violations*.

There exist three *violation severities*: *Warning*, *Critical*, and *Breaking*. The severity increases from *Warning* to *Breaking*. The violation severity of a CEC is defined in a cloud profile by cloud profile contributors. The *Con-*

straints package of CEM provides a meta-model for defining arbitrary CECs (cf. research question Q3.1 in Chapter 5). These CECs do not depend on specific characteristics of particular cloud environments (cf. research question Q3.2). There exist 18 predefined CECs that enable to fluently specify concise CEC definitions. Moreover, OCL and SMM may be used to define more sophisticated and specific CECs.

This chapter reasoned about the general detectability of CECs and distinguished between language-agnostic and language-dependent detection mechanisms. Furthermore, differences regarding the analyzability with static and dynamic analyses were incorporated in the definition of five detectability categories DC1-DC5. The detectability categories also answer the research question Q3.3 (cf. Chapter 5). CloudMIG mainly focuses on the detection of *CEC violations* that can be uncovered by means of the detectability categories DC2 and DC3 by using static analyses that examine extracted KDM models.

CEC violations can be detected with the help of *constraint validators*. A constraint validation process defines the ordered activities that have to be accomplished for detecting *CEC violations* with assistance of the constraint validators. The revealed *CEC violations* can either be used to compare different cloud environment candidates, as a kind of to-do-list for manual correction purposes, or to classify a software system regarding the introduced *Cloud Suitability and Alignment (CSA) hierarchy*. This hierarchy enables a classification of software systems regarding their suitability for specific cloud environments and their level of alignment after initial migration steps. Moreover, this chapter used characteristics of detected *CEC violations* and their ratio in certain parts of a system to define the prioritization function `prio`. It aims to mitigate risks and uncertainties that occur from detected *CEC violations* early and efficiently by prioritizing the manual correction of the *CEC violations*.

Chapter 8

Deployment and Reconfiguration Optimization

CloudMIG's activity A3 covers the generation of an optimized target architecture (cf. Section 6.3.2). Regarding an existing enterprise software system (SUA), the software architecture may have to be restructured to better align with the service landscape of a particular cloud environment. A cloud deployment architecture is then created by mapping the restructured software architecture to runtime containers of the selected cloud environment, for example, specific virtual machine instances. In general, the activity A3 aims to optimize the target architecture regarding resource efficiency and scalability (cf. challenges C3 and C4 in Section 6.1.1). A negative example from Islam et al. [2012] that demonstrates an inefficient resource usage, as is tackled by CloudMIG, is shown in Figure 8.1. The example considers a CPU resource that has to cope with demand that follows a sine wave. As a constant capacity is added to the resource per time unit, the scenario results in periods of under-provisioning as well as over-provisioning. Hence, resources should be rather added or removed dynamically in response to a varying resource demand. The so-called dynamic resource scaling can be controlled by reconfiguration operations. Thus, besides optimizing a cloud deployment architecture, runtime reconfiguration rules have to be streamlined with given usage patterns. The set of all design decisions that together form a potential target architecture is called a *Cloud Deployment Option (CDO)*.

This chapter describes in detail the approach G1 for generating CDOs that are improved regarding the challenges resource efficiency (C3) and scalability

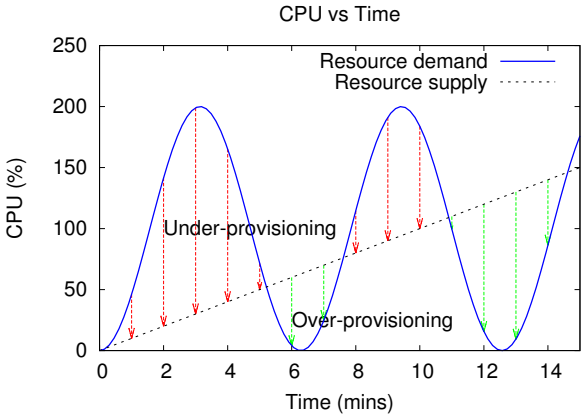


Figure 8.1. Under- and over-provisioning of a CPU resource (cf. [Islam et al. 2012]). The percentage values indicate the fraction of an IaaS-based cloud environment's standard VM instance type capability that could be used by the application.

(C4) (cf. Section 6.3.2). The approach employs techniques of the search-based software engineering field. It uses a genetic algorithm for iteratively optimizing CDOs and searching well-suited candidates. To evaluate the *fitness* of potential solutions, simulation runs are used that, among others, replay workload profiles. Therefore, real usage data of a software system may be utilized to assess a specific software architecture, deployment architecture, and a reconfiguration rule set that controls the dynamic scaling of resources that are provided by a particular cloud environment. The genetic algorithm that uses simulation runs to explore the search space of all possible CDOs is called *CDOXplorer*.

This chapter utilizes and builds upon the following previously published work:

1. *Frey, Sören and Fittkau, Florian and Hasselbring, Wilhelm, "Search-Based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud," in Proceedings of the 35th International Conference on Software Engineering (ICSE 2013), pp. 512-521, 2013.*

2. *Fittkau, Florian and Frey, Sören and Hasselbring, Wilhelm, "Cloud User-Centric Enhancements of the Simulator CloudSim to Improve Cloud Deployment Option Analysis," in Proceedings of the European Conference on Service-Oriented and Cloud Computing (ESOCC 2012), pp. 200-207, 2012.*
3. *Fittkau, Florian and Frey, Sören and Hasselbring, Wilhelm, "CDOSim: Simulating Cloud Deployment Options for Software Migration Support," in Proceedings of the IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA 2012), pp. 37-46, 2012.*

The remainder of this chapter is organized as follows. Section 8.1 provides an overview of the genetic algorithm CDOXplorer. CDOXplorer uses our tool CDOSim [Fittkau et al. 2012a] to simulate CDOs and to obtain corresponding fitness values. CDOSim is briefly described in Section 8.2. Section 8.3 presents the CDO model that forms the basis of CDOXplorer's optimization procedure. The effectiveness of genetic algorithms is determined to a great extent by the specific operations that control the reproduction. The corresponding tailored crossover and mutation operators of CDOXplorer are explained in Section 8.4. CDOXplorer also exhibits adaptive and hybrid characteristics that are described in Section 8.5. Section 8.6 analyzes properties and size of the search space that CDOXplorer has to examine, before Section 8.7 sums up this chapter.

8.1 Overview

Migrating and deploying enterprise software to the cloud entails a wealth of challenges and potential pitfalls. Besides the need to detect and manually correct CEC violations (cf. Chapter 7), it is tedious to, for example, select an adequate cloud environment and the best-suited virtual machine (VM) instance types—with regard to inevitable trade-offs between costs and performance—from the plethora of available cloud offerings [Prodan and Ostermann 2009]. Then, the application and deployment architecture have to be reworked to conform with the chosen cloud and to enable compliance with defined service level agreements (SLAs) and the included Quality of

Service (QoS) stipulations [Kuperberg et al. 2011]. Furthermore, to exploit the cloud's elasticity and the usually employed pay-per-use model, it is necessary to implement and calibrate rules for cost-efficient dynamic resource scaling according to observed usage patterns [Galán et al. 2009]. In summary, all of those design decisions form a multitude of Cloud Deployment Options (CDOs) that need to be explored for well-suited candidates. Unfortunately, techniques for automatically evaluating all CDOs do not exist and a comprehensive manual analysis is most often inapt due to time and budget constraints [Grundy et al. 2012]. Furthermore, as CloudMIG is also intended to support distributed enterprise systems, the QoS-aware composition of software components that run on one node is considered as a single service that is provided to an arbitrary number of components on other nodes. Such deployment optimization problems are intractable as they are known to be NP-hard [Canfora et al. 2005].

As a building block for approaching this problem, our simulation tool CDOSim [Fittkau et al. 2012a; b] facilitates the simulation of CDOs for determining their respective response times, costs, and SLA violations. CDOSim allows to manually configure and simulate CDOs on the basis of a reverse-engineered architectural system model with monitored or synthetic workload. However, the design space that spans for all possible CDOs is huge, the elements of a single CDO exhibit complex non-linear interdependencies, and CDO simulation runs are very time-consuming and can take from a few minutes to several hours. Hence, simulating a great number of CDOs is most often still not a viable option and it is therefore very likely that a suboptimal solution is chosen.

Moreover, there usually exists no single CDO that causes the lowest costs along with the lowest average response times and the lowest number of SLA violations. Thus, a potential cloud user is interested in automatically finding the most adequate trade-off solutions among which the CDO candidate can be selected that best suites the user's specific needs. The set of these most adequate trade-off solutions constitutes a pareto optimum. The CDOs included in such a pareto optimum cannot be improved concerning one objective without deteriorating another objective, for example, considering a trade-off between costs and response times.

The genetic algorithm CDOXplorer explores the CDO search space on the basis of automatically extracted architectural models resulting from CloudMIG's activity A3 (cf. Section 6.3.2) and approximates the corresponding pareto optimum. Similar problems are addressed by methods of the search-based software engineering field, where genetic algorithms are widely used [Harman 2011]. In general, genetic algorithms group the candidates—so-called individuals—in populations and use a fitness function to assess the candidates. Then, the best-suited individuals are selected. They reproduce through so-called mutation and crossover operations and after several generations, the individuals that inherited superior properties become dominant (cf. Section 4.2). To assess the fitness of CDOs, CDOXplorer uses simulation runs of CDOSim to restrict the search space and to steer the exploration towards promising CDOs. Thus, CDOSim is no longer used only for analyses, but for design purposes as well.

CDOXplorer supports IaaS-based cloud environments, where the most often used building blocks are VMs. PaaS-based cloud environments are not supported by CDOXplorer, as CloudSim only allows to simulate CDOs on the basis of virtual machines. By incorporating CDOSim, CDOXplorer is a member of the simulation-based optimization class (cf. Section 4.3). Here, the evaluation of the used fitness function is, in contrast to most genetic algorithms, very expensive and requires strict limitations regarding the population size and number of included generations. CDOXplorer not only optimizes the allocation of software components to VMs, but also searches for reconfiguration rules that are aligned with the cloud's elasticity and the specific performance and pricing models of the available cloud environments. A common challenge in the design of genetic algorithms becomes apparent as they do not guarantee to converge to a global optimum, especially, if a low number of generations and low population sizes are used.

The rest of this section explains the fundamentals of the genetic algorithm CDOXplorer. Section 8.1.1 describes the basic concepts. Central underlying assumptions are explained in Section 8.1.2, before Section 8.1.3 briefly presents CEM's packages and the most important elements that play an essential role for the construction of CDOXplorer.

8.1.1 Basic Concepts

The notion of a Cloud Deployment Option (CDO) is of particular importance in the context of cloud deployment and reconfiguration optimization. As the term implies, there usually exist various options for deploying a software system to the cloud and specifying reconfiguration rules that facilitate exploiting the cloud's elasticity. The term was occasionally used informally before. In the context of this thesis, a CDO is defined as follows.

Definition: Cloud Deployment Option (CDO) (based on Fittkau [2012]; Fittkau et al. [2012a])

A *Cloud Deployment Option (CDO)* is a combination of the following four aspects regarding the restructuring and deployment of a software system to the cloud: (1) The target cloud environment configuration. (2) The mapping of existing components to a number of virtual machine instances. (3) The instance types of the virtual machines. (4) 0..n reconfiguration rules for exploiting the cloud's elasticity.

The target cloud environment configuration stated in the definition above refers to the corresponding element in CEM (cf. Section 6.3.4). Instead of only referring to a target cloud provider, a target cloud environment configuration is included in the definition as a cloud provider may offer several distinct cloud environment configurations.

Input and output of CDOXplorer The genetic algorithm CDOXplorer produces a set of near-optimal CDOs, i.e., the CDOs that are contained in the best known pareto front that results after executing CDOXplorer (PF_{Known}). These near-optimal CDOs therefore constitute the *output of CDOXplorer*. As illustrated in Figure 8.2, the *input of CDOXplorer* consists of the following four different models.

- **Architectural KDM model:** The KDM model that is extracted from the software system that is supposed to be migrated to the cloud (SUA)

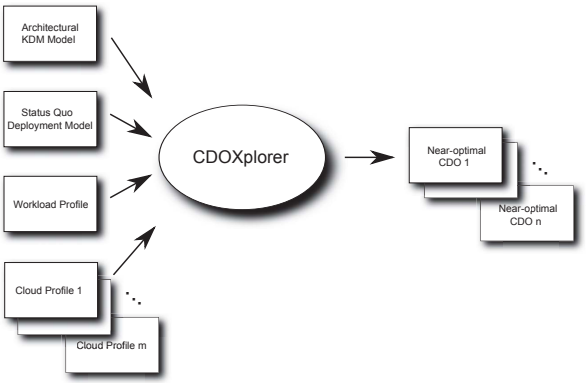


Figure 8.2. Input (left side) and output (right side) of CDOXplorer

- **Status quo deployment model:** Describes the status quo deployment of the software system
- **Workload profile:** A workload profile that includes response time data corresponding to a typical usage pattern and that was measured using the status quo deployment
- **Cloud profiles:** The cloud profiles that are supposed to be included in CDOXplorer's search for near-optimal CDOs

Basic evolutionary characteristics As described in Chapter 4, the reproduction of each generation includes the following four basic steps S1-S4 [Harman 2011]:

S1 Select parents

S2 Recombine parents (crossover)

S3 Mutate offspring

S4 Evaluate offspring's fitness

The first step S1 selects individuals for reproduction. For CDOXplorer, S1 is based on the selection operation of the NSGA-II algorithm [Deb et al.

2002] for selecting appropriate pairs of parents and ensuring the diversity of solutions (cf. Section 10.4 for implementation details). CDOXplorer applies two tournament rounds for choosing among candidates. Hence, a prolific individual has to be fitter than at least two others. CDOXplorer produces two children from two parents via executing a custom crossover (S2) and mutation operator (S3) that are detailed in Section 8.4. CDOSim is used for evaluating the individuals in the fourth step S4. In this step, the values of the objectives that have to be optimized are obtained by simulating CDOs. The current implementation of CDOXplorer (cf. Section 10.4) has to be run on a single computing node in a single thread. As the simulations are very time-consuming, the overall number of simulations therefore has to be strictly limited. The basic parameters of CDOXplorer are configured per default as follows. Populations contain 50 individuals (population size α). 25 individuals are selected from each generation for reproduction (number of parents μ) and each generation spawns 50 children (number of children λ). Furthermore, CDOXplorer processes 60 generations per default (number of generations ρ). Those numbers result from a comparison of diverse configurations regarding the experiments described in Chapter 12. However, the default configuration parameters can be adjusted to project-specific contexts taking into account, among others, the actual sizes of KDM models and workload profiles, number of present cloud profiles, and allowed runtime of CDOXplorer.

Optimization process Figure 8.3 shows the iterative simulation-based optimization process. The illustration details the basic CDO optimization feedback loop that is depicted in Figure 5.4. Let ρ be the number of generations of the genetic algorithm. The optimization process begins with the generation of α random individuals, i.e., α CDOs covering arbitrary IaaS-based cloud environments and random code mappings to arbitrary VM instance types, for instance. These α random individuals have to be instantiated, such that CDOSim can process the CDOs in the next step. As the CDO instantiation is tightly coupled with the model for describing CDOs, it covers the research question *Q4.1* (cf. Section 5.1). A CDO instance can then be used to start the simulation with the tool CDOSim. Each of the α CDO instances is simulated in a single CDOSim run. As CDOSim is uti-

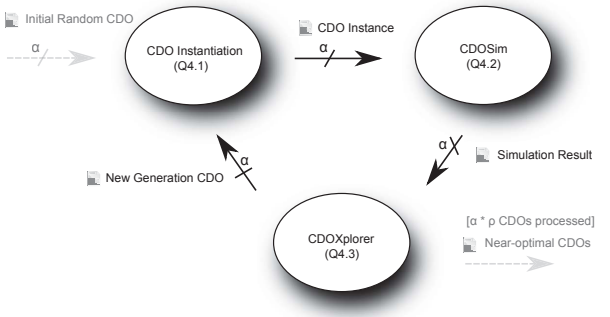


Figure 8.3. Iterative simulation-based CDO optimization with CDOXplorer (detailing Figure 5.4 in Section 5.4). CDOSim: Simulation tool, CDOXplorer: Adaptive and hybrid genetic algorithm, α: Population size, ρ: Nr. of generations.

lized to automatically assess CDOs, it addresses the research question *Q4.2* (cf. Section 5.1). α simulations also produce α simulation results that are processed by CDOXplorer. As the genetic algorithm is used for optimizing CDOs, it addresses the research question *Q4.3* (cf. Section 5.1). CDOXplorer produces offspring individuals with the help of its crossover and mutation operations. That means, it generates α new CDOs that are again instantiated and simulated. Afterwards, CDOXplorer selects with each optimization run the α fittest CDOs for reproduction and validation in the next iteration. Hence, the population size α remains stable over the course of the generations. CDOXplorer ends if all generations are processed and the maximum number of CDOs ($\alpha * \rho$) are simulated. The near-optimal CDOs can then be easily determined by calculating the pareto optimum.

Transforming status quo deployments to CDOs Further basic concepts that are relevant in the context of the genetic algorithm CDOXplorer are described with the help of the example deployment of a software system that is shown on the left side of Figure 8.4. The example assumes that this system is supposed to be moved to the cloud environment Amazon EC2.[1] The system consists of eight software components (each marked with the

[1] `http://aws.amazon.com/ec2/`

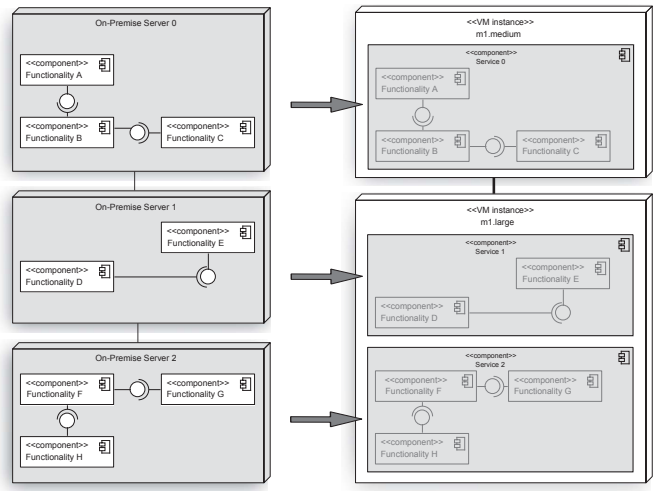


Figure 8.4. Mapping on-premise servers and deployed components (left) to atomic services in a basic CDO example (right)

stereotype «component») that are currently deployed to three interconnected on-premise server machines. As defined in Section 6.2, those machines are also called *status quo nodes* as they constitute elements of the deployment architecture that describes the status quo assignment of components to physical machines. This example also assumes that the system fulfills some basic preconditions, for example, that it can run on one of the operating systems that are supported by Amazon EC2. As it complies with those preconditions, the system is *cloud compatible* (cf. Section 7.4.1) concerning Amazon EC2 and can be deployed to some of its VMs. Furthermore, the notion of a status quo node is also used to define a *service* in the following.

Definition: Service

In the context of deployment and reconfiguration optimization with CDOXplorer, a *service* describes all of the components that are deployed to a single status quo node. A service is an atomic unit concerning the allocation to a VM.

According to the definition above, the components `Functionality D` and `Functionality E` of Figure 8.4 may not be deployed separately to different VMs, for instance. This design decision regarding the atomicity of a service was made to (1) prevent a further explosion in the number of combinations and CDOs that have to be searched, and to (2) render pervasive changes unnecessary that may be required when distributing tightly connected components over different VMs. Thus, the three status quo nodes and the deployed components shown in the left part of Figure 8.4 result in the three services *Service 0* to *Service 2* in the right part of Figure 8.4 that exhibit a similar assignment of components. In general, when migrating such a service to the cloud, it can be deployed on one or more VMs and can be used by zero or more other services, for example, through integrating additional communication mechanisms.

As can be seen in the lower right part of Figure 8.4, a single VM can host multiple services. In the given example, it was decided to consolidate *Service 1* and *Service 2* into a joint VM that is started with Amazon EC2's VM instance type *m1.large*. Such VM instance types describe the hardware resources that are available to VMs. For example, at the time of writing, Amazon EC2's *m1.large* VM instance type provides 7.5 GB memory and two virtual cores that together provide approximately the CPU capacity of four 1.7 GHz Xeon processors from 2006. The remaining *Service 0* in Figure 8.4 is deployed to an own VM that builds upon the *m1.medium* VM instance type. The basic CDO of Figure 8.4 is now given by the number of chosen VM instances, the assignment of components—constrained by the defined services—to these VM instances, and the selection of a VM instance type for each VM instance. Furthermore, up to this point, reconfiguration rules are omitted for the sake of simplicity. Considering the cloud migration types that were introduced in Section 6.2, the example of Figure 8.4 corresponds to type *SC2I*, as the target service model is IaaS, all of the existing components are migrated to the cloud, and no prior utilization of virtualization technology was assumed.

Reasoning about the broader array of all potential CDOs for, e.g., just the single cloud environment Amazon EC2, 12 of its VM instance types, and no reconfiguration rules, already reveals the general complexity of CDO

analysis. When assuming up to three VM images that contain combinations of the three services *Service 0* to *Service 2* and that up to two VMs can be started from a VM image, these restrictive settings already yield 4,741,632 CDO candidates (a detailed search space analysis is given in Section 8.6.1). Without using potent heuristics, all CDOs would have to be simulated for reliably finding competitive solutions.

A further essential part of a CDO is the definition of a strategy to exploit the cloud's elasticity by means of reconfiguration rules. As CDOXplorer supports the optimization of CDOs that target the IaaS service model, reconfiguration rules are specified on the basis of virtual machines. In the context of this thesis, basic *reconfiguration rules* are defined as follows.

Definition: Reconfiguration Rule

A *reconfiguration rule* defines a scaling action for a specific combination of a VM instance type and service(s). The scaling action follows a specific scaling type and is triggered and automatically executed if a defined condition is met.

Various scaling types are defined in Section 8.3. An example scaling type is *scale-out*, i.e., starting an additional VM instance. A corresponding condition could be defined, that starts a further VM instance when a running VM instance's CPU utilization is above 70% for over 10 minutes, for instance. It should be noted that reconfiguration rules are defined referring to a combination of VM instance types and one or more services. For example, this facilitates to further distinguish VM instances and to provide different reconfiguration rules for scenarios that use the same VM instance types for deploying different services.

In general, reconfiguration rules are defined during the analysis of potential CDO candidates regarding a specific SUA. Those defined reconfiguration rules are then executed at runtime after the SUA was migrated towards the chosen CDO. The implementation of a CDO has to incorporate an autonomic control loop for managing the rules in a self-adaptive way at runtime. For example, such a control loop is IBM's MAPE-K (Monitor, Analyze, Plan,

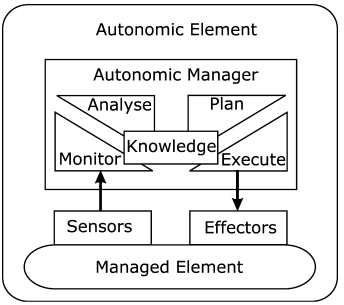


Figure 8.5. IBM's MAPE-K reference model for autonomic control loops (from [Huebscher and McCann 2008])

Execute, Knowledge) [IBM 2003; Huebscher and McCann 2008] that is illustrated in Figure 8.5.

When considering CDOs that were produced by CDOXplorer, the *managed elements* that are controlled by an *autonomic manager* (cf. Figure 8.5) correspond to virtual machines. MAPE-K's elements of the autonomic manager and its *sensors* and *effectors* either might be provided and managed by the chosen cloud environment itself, or a CloudMIG user has to manually incorporate appropriate mechanisms. For example, through including monitoring technologies (sensors) and utilizing a cloud environment's API for dynamic resource scaling (effectors).

Problem statement After introducing the basic structure of the genetic algorithm CDOXplorer, the multi-objective optimization problem that is tackled by CDOXplorer can be described as follows.

Let Φ be the set of all *feasible* CDOs (feasibility of CDOs is explained in Section 8.3.4, *feasibility* in general is described in Chapter 4) for a given set of IaaS-based cloud environments and a given architectural model, status quo deployment model, and workload profile of a software system. The goal is to find a CDO $x \in \Phi$ that minimizes the values of the three objective functions $costs(x)$, $rt(x)$, and $sla(x)$ that calculate the costs, average response time, and number of SLA violations of x, respectively. The costs refer to the total

amount of monetary units owed to a cloud provider because of utilizing provided services. The response times refer to the average response times of the methods that are included in a workload profile. Lastly, the SLA violations indicate the number of method calls with response times that exceed a given threshold.

However, there might not exist such a single CDO x that surpasses all other feasible CDOs regarding each objective function. Hence, CDOXplorer aims to approximate the true pareto-optimal front $PF_{True} \subseteq \Phi$ via a best known pareto-optimal front $PF_{Known} \subseteq \Phi$. To judge the quality of such approximations, a set of performance measures Ξ is used that can evaluate multi-objective optimizers such as CDOXplorer. Without the loss of generality, let each $p \in \Xi$ be minimizing performance measures. Therefore, the problem tackled by CDOXplorer can be formulated as follows.

$$\forall p \in \Xi : \text{CDOXplorer aims to minimize } p(PF_{True}, PF_{Known}) \text{ such that ultimately } PF_{Known} = PF_{True}$$

The actually used performance measures (Ξ) are explained in Section 12.1.2.

8.1.2 Assumptions

This section briefly describes four fundamental assumptions (AS1-AS4) that underlie the creation and optimization of CDOs with CDOXplorer. Furthermore, those assumptions provide the context for actually implementing CDOs for particular software systems and cloud environments.

- **AS1:** It is assumed that specific recurring usage patterns are known or can at least be estimated while planning a migration to the cloud, as the reconfiguration rules—that are executed at runtime when a system is actually migrated—are aligned to these usage patterns. CloudMIG users are presumably interested in two predictive values regarding a specific cloud environment. The first demanded value indicates how the application would perform (and to what costs), if it would be deployed to the cloud environment widely unmodified. Moreover, they are most

likely interested in the possible improvement that could be achieved by implementing the best-suited solution. Thus, reasoning about an optimal deployment and reconfiguration rules during the planning phase of a migration is a worthwhile endeavor. In summary, it is assumed that the CDO optimization is performed offline with the help of known usage patterns, whereas the reconfiguration rules are supposed to be executed online after the actual migration (during operation). However, CDOXplorer can also be used for modifying the reconfiguration rules at runtime in a self-adaptive fashion, for example, based on predictions of a common workload forecasting methodology, such as an Autoregressive Integrated Moving Average (ARIMA) model or exponential smoothing [Cryer and Chan 2010].

- **AS2:** It is assumed that the existing enterprise software system can be adapted towards a scalable architecture with reasonable effort. Alternatively, substantial reworking or the deployment of more coarse grained services may be necessary for monolithic systems.
- **AS3:** It is assumed that a load balancer is available that keeps track of existing VMs and distributes the workload across these VMs. That means, additional components may have to be created and additional cloud services might have to be integrated for managing the pool of used VMs. A corresponding auxiliary architecture might follow the logical structure that is illustrated in Figure 8.6. A so-called *rule engine controller* distributes the reconfiguration rules that were created by CDOXplorer to load balancers. Besides distributing the workload among the VMs, the load balancers are therefore also responsible for starting and stopping VMs because of triggered scaling events. However, the rule engine controller may also be implemented in a way so it can start and stop VM instances on its own. A scaling event is initiated if a reconfiguration rule's defined condition becomes *true*.
- **AS4:** It is assumed that the defined reconfiguration rules can be implemented with the APIs of any IaaS-based cloud environment. Besides using a cloud environment's native APIs, it is also possible to use cloud APIs

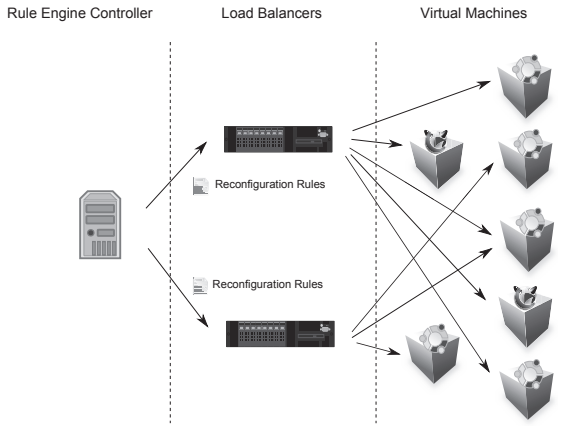


Figure 8.6. Logical structure for dynamic resource scaling with reconfiguration rules

such as Deltacloud[2] or JClouds,[3] or supporting techniques such as programming directives for elastic computing [Dustdar et al. 2012] that are available for various programming languages.

These are reasonable assumptions if a CloudMIG user intends to migrate an existing enterprise software system to the cloud.

8.1.3 Involved CEM Packages

This section outlines the CEM packages that are relevant for the optimization of CDOs and that were not already described in previous sections. CEM was introduced in Section 6.3.4. Its packages are described in detail in Appendix A. CDOXplorer employs elements from CEM's *IaaS*, *Mapping*, and *Pricing* packages. An excerpt of these packages is presented in Figure 8.7. The packages are briefly described below.

[2] http://deltacloud.apache.org/

[3] http://www.jclouds.org/

(a) *IaaS* package excerpt

(b) *Mapping* package excerpt

(c) *Pricing* package excerpt

Figure 8.7. CEM packages relevant for the optimization of CDOs

IaaS package The IaaS package provides elements for modeling IaaS-based cloud environments, that are also addressed by CDOXplorer. Figure 8.7a shows an excerpt of the IaaS package. Basic building blocks of IaaS-based cloud environments are virtual machines. A VM instance is represented by the `AbstractVMInstance` class. `VMImages` contain the software that can be run in an arbitrary number of VM instances. The IaaS package uses the `HardwareConfiguration` element to model the computing capacities of VM instances. For example, it describes the number and type of available CPUs, memory, and network bandwidth.

The availability of those `HardwareConfigurations` can be restricted to certain `Locations` (data centers or geographical regions). CDOXplorer has to consider those restrictions when creating CDOs. Furthermore, testing different `HardwareConfigurations` to form CDOs or using a varying number of VMs, enables CDOXplorer to quickly overcome larger distances of the search space, as corresponding variations commonly result in widely differing results.

Mapping package The *Mapping* package includes elements that describe the mapping of extracted KDM source code elements to cloud code con-

tainers, such as VM images. `AbstractCloudAppContainer` is the base class for those containers. The *Mapping* package constitutes the connecting link between entities existing in the cloud domain and a KDM model that describes the software system that is supposed to be migrated to the cloud. A system's source code is mapped via so-called `CloudCodeModels` that themselves contain instances of `AbstractCloudModule`. A specific `AbstractCloudModule` is `LegacyModule` that references KDM elements of the legacy system. As described before, a system that is deployed on a status quo node results in an atomic service. Varying the mapping of present services to `LegacyModules` is a further possibility for CDOXplorer to explore novel CDOs.

Pricing package The *Pricing* package includes elements for modeling a cloud environment's price list. For example, it enables to model the prices for transferring data between VMs. An objective that is optimized by CDOXplorer determines a CDO's costs. Hence, elements of the *Pricing* package are incorporated by CDOXplorer to calculate the corresponding objective's values. All prices inherit from `AbstractPrice`. The `VMInstancePrice` is a specific price and describes the price for using a certain VM instance type. Prices are determined by a particular price function. Those mathematical functions are represented by the `AbstractPriceFunction` element. For example, a cloud service's recurring base fee, that does not further depend on the extent the service is used, can be modeled by using a `ConstantPriceFunction` element.

All `AbstractPriceFunctions` reference an `AbstractPriceFunctionUnit` that specifies the unit of the price function. For specifying that a price is billed, for example, per time unit (e.g., \$1.5/hour usage of VM instance type *small*), a `TimePriceFunctionUnit` can be used. In contrast, using a `UnitOfInformationPriceFunctionUnit` enables to define prices that depend on the amount of processed data (e.g., \$2/GB stored data). Those prices that can be expressed by a linear function are modeled with a `LinearPriceFunction` element. That means, the costs that arise because of those prices increase linearly according to a consumed amount of a unit, such as a price for using a VM instance that corresponds to a specific VM instance type that is billed per hour.

8.2 The Simulator CDOSim

CDOXplorer employs the simulator CDOSim [Fittkau 2012; Fittkau et al. 2012a; b] to validate the fitness of CDOs. CDOSim builds on the cloud simulator CloudSim [Calheiros et al. 2009; 2011]. However, CloudSim focuses primarily on the perspective of a cloud provider, as it relies on corresponding model entities such as hosts and VM scheduling policies. Those implementation details of an underlying cloud platform are usually intransparent to a cloud user and are therefore not included in CloudMIG's cloud profiles. In contrast, CDOSim emphasizes the cloud user perspective and adds corresponding important features, such as the possibility to start and stop VM instances on demand and the simulation of interactions (e.g., method calls) between VM instances [Fittkau et al. 2012b]. CDOSim was built to (1) enable the transformation of cloud profiles to cloud environment models that can be simulated with CloudSim, whereas the restricted cloud user perspective is taken into account. Furthermore, CDOSim (2) adds capabilities that enable to transform extracted KDM models to CloudSim application models, transform monitored workload traces to CloudSim's workload model, and that enable the simulation of CDO models along with the included reconfiguration rules as produced by CDOXplorer.

The rest of this section overviews the simulation tool CDOSim and is structured as follows. Section 8.2.1 describes the fundamental concepts of CDOSim. Section 8.2.2 summarizes findings regarding CDOSim's accuracy.

8.2.1 Fundamental Concepts

An essential characteristic of datacenter simulators is the type of workload that can be simulated [Aksanli et al. 2012]. Two essential workload types that have to be processed by datacenters are distinguished by Barroso and Hölzle [2009]: *Online services* and *batch (offline) processing systems*. An online service processes its users' requests and has to take into account the latency that is perceived by the users. CDOSim addresses the simulation of those online services and is therefore in line with the CloudMIG method that targets enterprise software systems.

For including computational capabilities of computing nodes in the simulation model, CDOSim uses the metric *Mega Integer Plus Instructions per Second* (MIPIPS). An *integer plus instruction* stands for an instruction in a specific programming language that sums up two integer values.

Definition: MIPIPS (based on Fittkau et al. [2012a])

The number of integer plus instructions in millions that can be processed by a computing node per second. The metric is used to model the computational capabilities of status quo nodes and of cloud environments' VM instance types.

For describing the computational capabilities of a computing node for calculating other instruction types than integer plus instructions, e.g., multiplying double values, CDOSim employs weights.

Weighting of instructions The weights refer to a MIPIPS value that is used as a basis. For example, values of 1,000 MIPIPS and 500 Mega Double Multiply Instructions per Second (MDMIPS) might have been determined for a specific VM instance type. That means the VM instance type can process twice as much integer plus instructions as double multiply instructions per time period. The weight would be set to $1,000/500 = 2$ for the double multiply instructions. The MIPIPS values and weights of VM instance types are also part of cloud profiles that describe IaaS-based cloud environments. CDOSim is accompanied by a benchmark that can measure the MIPIPS values and weights [Fittkau 2012].

Instruction counting approaches The MIPIPS values and weights that are measured on status quo nodes and on VM instance types can be used to estimate how long a method call will last, if a corresponding application is deployed to a cloud environment following a particular CDO. For that reason, the number of instructions of an application that is supposed to be migrated to the cloud has to be counted. CDOSim provides three different instruction counting approaches that are briefly described below.

- **Static approach** The static instruction counting approach requires only the source code of an application that is supposed to be migrated to the cloud and an appropriate KDM discoverer for the programming language that was used to build the application. The static approach traverses the extracted KDM models and counts the instructions according to the visited KDM elements.
- **Dynamic approach** Additionally to the KDM model of an application, the dynamic instruction counting approach requires workload trace information that includes response times regarding the application's methods. This kind of information can be retrieved from monitoring log data, for instance. The dynamic approach is generally more accurate than the static approach and should be preferred if appropriate runtime data is present [Fittkau 2012].
- **Hybrid approach** The hybrid instruction counting approach combines the static and the dynamic approach to mitigate the weaknesses of both approaches. The static approach requires only KDM models but is less accurate, whereas the dynamic approach is more accurate, but requires additional data from a dynamic analysis which might not exist. The hybrid instruction counting approach uses the response time data for those methods that are included in corresponding workload trace information. As those information might not be complete, the hybrid approach calculates the instructions statically for those methods that are not included in the workload trace information.

Integration with CloudMIG Xpress The CloudMIG method incorporates the simulation results of CDOSim in the context of the genetic algorithm CDOXplorer to validate the fitness of CDOs. Figure 8.8 illustrates the integration of CDOSim in *CloudMIG Xpress* that provides tool support for CloudMIG. At the same time, Figure 8.8 helps to clarify the integration of CDOSim's simulation results and the underlying benchmark results in the CloudMIG method in general. As described above, the MIPIPS and weights benchmark is executed on the status quo nodes as well as on the VM instance types of IaaS-based clouds that should be considered by CloudMIG as potential target cloud environments. The MIPIPS and weights values of a

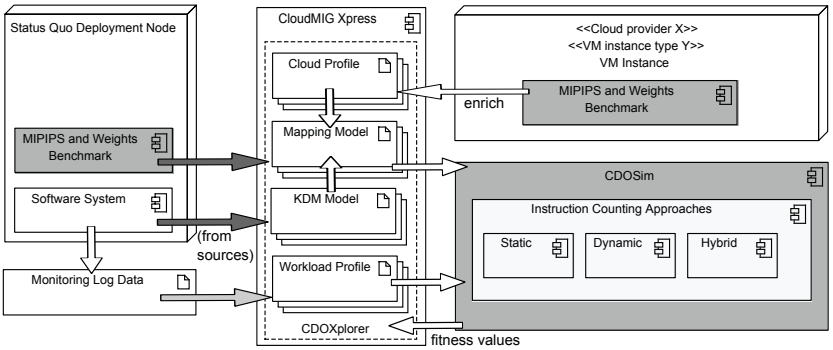


Figure 8.8. Integration of CDOSim with CloudMIG Xpress. The CDOSim and benchmark components are colored gray. Dark gray arrows indicate basic data the user needs to provide for simulating CDOs. Data marked with light gray arrows is only needed for dynamic and hybrid instruction counting. CDOXplorer uses several artifacts and fitness values provided by CDOSim (based on [Fittkau et al. 2012a]).

cloud environment's VM instance types are included in the corresponding cloud profile by cloud profile contributors. Furthermore, monitoring log data is converted into workload profiles and has to be available for using the dynamic and hybrid instruction counting approaches.

8.2.2 Accuracy of CDOSim

The accuracy of CDOSim is evaluated by Fittkau [2012] with the help of the web-based pet shop application iBatis JPetStore 5.0.[4]

Methodology Building on an evaluation of the concept and measurement techniques for deriving MIPIPS, Fittkau [2012] utilizes two different cloud environments to validate the accuracy of CDOSim. The web store application is deployed on a research cluster running the private cloud software Eucalyptus [Nurmi et al. 2009]. The used Eucalyptus server has two AMD

[4] http://sf.net/projects/ibatisjpetstore/

Opteron 2384 processors that provide eight CPU cores in total. The server is equipped with 24 GB DDR2-667 RAM and features a 1 Gigabit/s network connection. Additionally, various instance types of the cloud environment Amazon EC2 are used. The experiments employ the workload that is shown in Figure 8.9 to drive the simulation as well as the Eucalyptus and Amazon EC2 deployments. An additional VM instance is started if the CPU utilization of all allocated VM instances is higher than 70% for more than one minute. Similarly, a VM instance is stopped if the CPU utilization of all allocated VM instances is lower than 30% for more than one minute. The accuracy of CDOSim is measured for several scenarios with the help of the *relative error* metric. It is defined using the following notation.

T:	Set of all minutes in the measurement duration
$m(t)$:	Measured value at timestamp $t \in T$
$s(t)$:	Simulated value at timestamp $t \in T$

The *relative error for a timestamp t* is then defined as stated in the Formula (8.2.1).

$$re(t) = \frac{|m(t) - s(t)|}{m(t)},\ m(t) \neq 0,\ t \in T \tag{8.2.1}$$

In general, the simulated results get compared with the actually measured values. The simulated and measured values are sampled per minute and for each sample pair at timestamp t with $m(t) \neq 0$, the relative error $re(t)$ is calculated. The *relative error for the whole simulation run (RE)* is determined by calculating the mean value as given in the Formula (8.2.2). The following four different REs are calculated.

RE_{CPU}:	Relative error of the CPU utilization
RE_{IC}:	Relative error of the VM instance count
RE_{Costs}:	Relative error of the costs output
RE_{RT}:	Relative error of the response times

$$RE = \frac{\Sigma_t\ re(t)}{|T|} \tag{8.2.2}$$

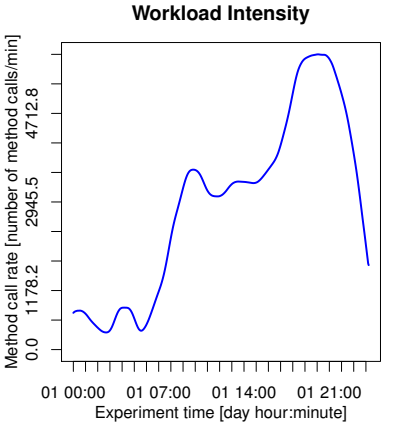
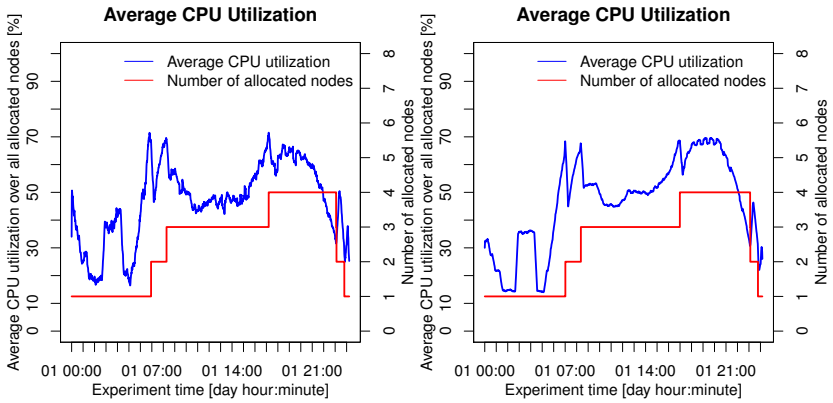


Figure 8.9. The day-night-cycle workload intensity used for evaluating CDOSim's accuracy (cf.[Fittkau et al. 2012a])

A simulation run can also be assessed using a single metric that incorporates the four REs. This metric is the *overall error (OverallRE)* that is defined in the Formula 8.2.3.

$$OverallRE = \frac{RE_{CPU} + RE_{IC} + RE_{Costs} + RE_{RT}}{4} \tag{8.2.3}$$

Results Figure 8.10 illustrates the results for the exemplary scenario *SingleCore.1*. This scenario employs the dynamic approach and the Eucalyptus cluster with a VM instance type `m1.small` that is configured to use a single CPU core and 1 GB RAM. The price for this VM instance type is set to \$0.095 per started hour. Figure 8.10a shows the measured CPU utilization and the number of allocated VM instances as determined on the Eucalyptus cluster. Figure 8.10b shows the same information but determined for a simulation run with CDOSim. The scenario *SingleCore.1* exhibits the following relative errors for the CPU utilization, VM instance count, and costs: $RE_{CPU} = 29.18\%$, $RE_{IC} = 0.64\%$, and $RE_{Costs} = 6.34\%$.



(a) Measured CPU utilization **(b)** Simulated CPU utilization

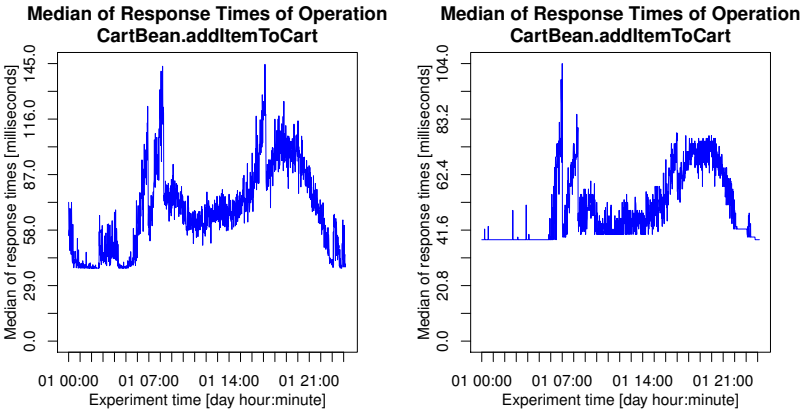
Figure 8.10. Average CPU utilization of allocated nodes in SingleCore.1 (cf.[Fittkau et al. 2012a])

The median response times per minute for the scenario *SingleCore.1* are shown in Figure 8.11, where Figure 8.11a depicts the measured response times and Figure 8.11b shows the response times resulting from the CDO simulation. This scenario exhibits a relative error regarding the response times of $RE_{RT} = 24.85\%$. When combining the four REs for the scenario *SingleCore.1*, the overall relative error is $OverallRE = 15.25\%$.

In summary, the evaluation of the simulator CDOSim shows that it can provide sufficiently accurate results that are suited for estimating major properties of CDOs [Fittkau 2012; Fittkau et al. 2012a; b].

8.3 CDO Model

The CDO model has to cover the characteristics that are stated in the CDO definition (cf. Section 8.1.1). Among those are the reconfiguration rules



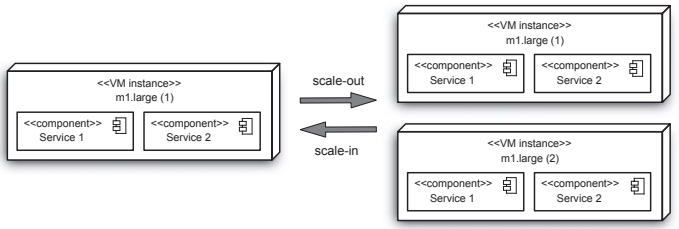
(a) Measured response times **(b)** Simulated response times

Figure 8.11. Median of response times in SingleCore.1 (cf.[Fittkau et al. 2012a])

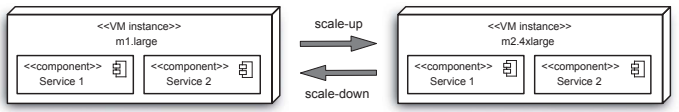
that are used to exploit the cloud's elasticity. They employ different scaling types that are described in Section 8.3.1. The complete *phenotype model* (see Section 4.1.2) of CDOs is presented in Section 8.3.2, whereas Section 8.3.3 describes the corresponding *genotype model*. Finally, Section 8.3.4 defines the feasibility (see Section 4.1.3) of CDOs.

8.3.1 Scaling Types

The CDOs found by CDOXplorer specify a cloud deployment model and a set of reconfiguration rule models. In IaaS-based cloud environments, resources can be dynamically acquired and released by executing *reconfiguration actions* to counteract under- and over-provisioning, benefit from the pay-per-use model, and to ensure the compliance with specified SLAs. Two reconfiguration actions together define one of the scaling types *horizontal scaling* or *vertical scaling* that are shown in the examples of Figure 8.12a and 8.12b, respectively. These scaling types are described in the following.



(a) *Horizontal scaling* with the reconfiguration actions *scale-out* and *scale-in*



(b) *Vertical scaling* with the reconfiguration actions *scale-up* and *scale-down*

Figure 8.12. Examples of the different scaling types

Horizontal scaling Horizontal scaling employs VMs that use the same VM instance type. Furthermore, VM instances are added (*scale-out*) or shut down (*scale-in*). Figure 8.12a shows an example where VMs are used that all correspond to the *m1.large* VM instance type.

Vertical scaling In contrast to horizontal scaling, the reconfiguration actions *scale-up* and *scale-down* are available for vertical scaling (Figure 8.12b). Scaling up adds more resources to a VM, such as a further CPU or more memory, whereas scaling down removes resources. However, to the best of our knowledge, almost no cloud environment currently provides support for arbitrarily exchanging the VM instance type of a VM during operation. Though, the described semantics can be emulated through starting a new VM from a VM instance type whose MIPIPS value is higher (scale-up) or lower (scale-down) than that of the old VM instance. When the new VM instance finishes the startup procedure, the previously used VM is shut down. In the example of Figure 8.12b, the MIPIPS value of *m2.4xlarge* is higher than that of *m1.large*. To illustrate the emulated semantics of vertical scaling, Figure 8.13 shows an example of scaling up a VM instance of the type *m1.large* to a VM instance of the type *m2.4xlarge* that, in this

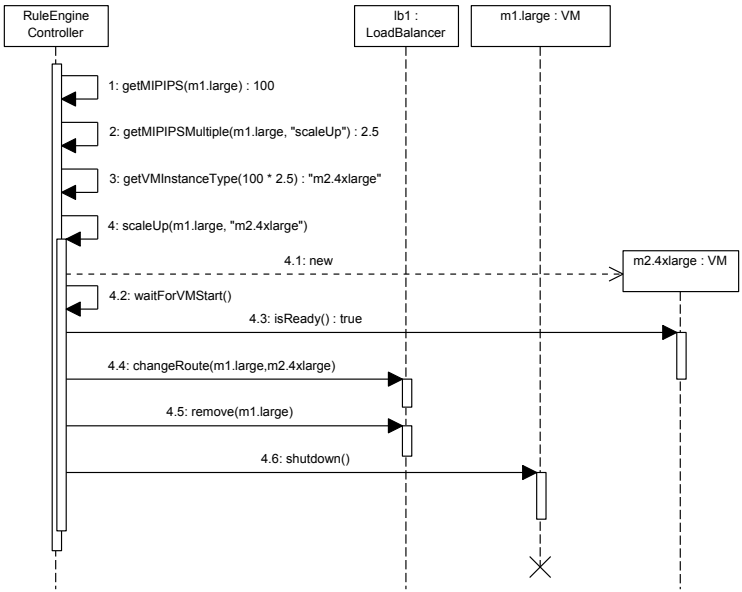


Figure 8.13. Emulated scaling up

example, has a higher MIPIPS value. This procedure is described below. The enumeration values correspond to the message call numbers in Figure 8.13.

1. The rule engine controller (cf. Section 8.1.2) retrieves the MIPIPS value of *m1.large* that is set to *100* in this example
2. The rule engine controller retrieves the *mipipsMultiple* value for scaling up a VM of the instance type *m1.large*. The *mipipsMultiple* value is determined by a corresponding reconfiguration rule and defines the multiple that has to be utilized for calculating the MIPIPS value of the VM instance type that should be used. The example assumes that *mipipsMultiple* is set to *2.5*. Hence, a new VM instance should be started that's type has $250 = 100 \cdot 2.5$ MIPIPS.
3. The VM instance type which has the MIPIPS value nearest to 250 is retrieved (*m2.4xlarge* in this example).

4. The scale-up procedure for the *m1.large* VM instance is started. The second parameter of the method call specifies the VM instance type of the new VM that should be used (*m2.4xlarge*).
   - 4.1 A new VM instance of type *m2.4xlarge* is started.
   - 4.2 The rule engine controller waits for the new VM instance to start
   - 4.3 The rule engine controller checks if the new VM instance is ready. In this example, the new *m2.4xlarge* VM instance finished the start up procedure and can be used.
   - 4.4 The rule engine controller informs the load balancer that the traffic that was routed to the VM instance *m1.large* before should now be forwarded to *m2.4xlarge*.
   - 4.5 The *m1.large* VM instance is removed from the load balancer's list of available VM instances.
   - 4.6 The *m1.large* VM instance is shut down. This ends the logical scale-up reconfiguration action.

It should be noted that, despite of the differences between vertical and horizontal scaling, the service composition is retained on a new VM instance for both of the scaling types (cf. Figure 8.12). Considering the different types of reconfiguration changes from Hofmeister [1994], vertical scaling corresponds to the *geometry* type, as the logical application structure remains unchanged, but components are relocated to a VM of another VM instance type. In contrast, horizontal scaling corresponds to the *structure* type from Hofmeister [1994], as an application's logical structure is changed and components and VM instances are added or removed.

8.3.2 Phenotype Model

This section describes the *phenotype model* (cf. Section 4.1.2) of CDOs that employs the different scaling types that were introduced in Section 8.3.1. The basic structure of CDOs is shown in Figure 8.14. A *Cloud Deployment Option* refers to a single *Cloud Environment* and contains so-called *Node*

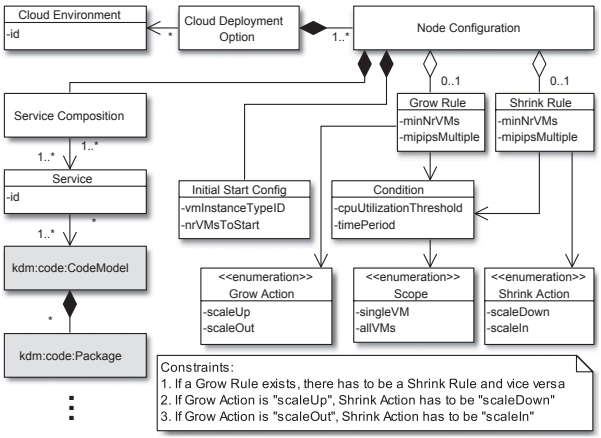


Figure 8.14. Basic structure of CDOs

*Configuration*s. A *Node Configuration* describes specifics of a VM instance. For example, an included *Service Composition* container refers to the *Service*s that are deployed on this VM. To link *Service*s with the represented source code, they reference parts of the extracted KDM elements.

Furthermore, an *Initial Start Config* specifies the VM instance type that should be used for a VM and also the number of VM instances that have to be started initially with this configuration. Moreover, a *Node Configuration* may contain a *Grow Rule* together with a *Shrink Rule*. They represent basic parts of a reconfiguration rule and, from a high-level view, determine how and when computing power is added (*Grow Rule*) or removed (*Shrink Rule*). These rules also specify a minimum number of VM instance types that have to be present and refer to the elements *Grow Action* and *Shrink Action* that define the reconfiguration actions that have to be used for scaling. The reconfiguration actions have to comply with the defined scaling types (see Section 8.3.1). Hence, a *scale-up* action can only be used in combination with a *scale-down* action. The same is true for *scale-out* and *scale-in* actions.

When applying vertical scaling, the *mipipsMultiple* attribute of *Grow Rule*s and *Shrink Rule*s becomes relevant for determining the VM instance type of

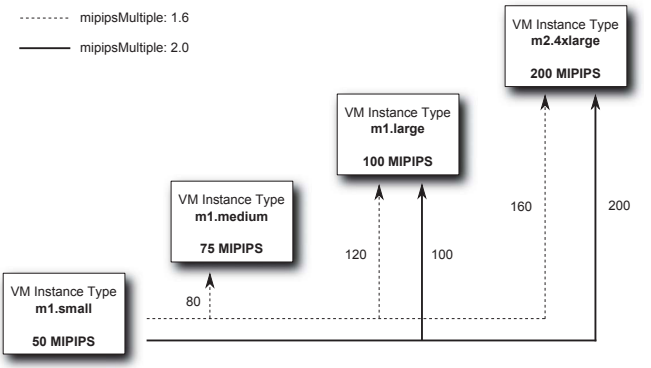
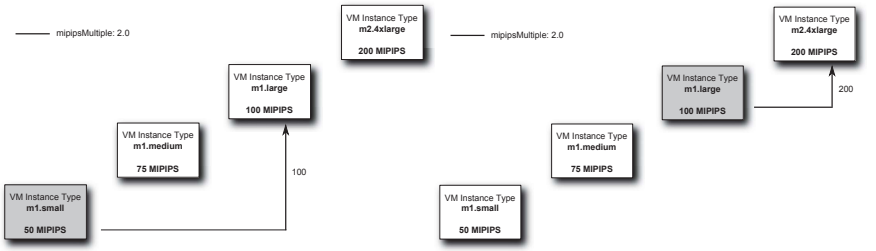


Figure 8.15. Effect of using different *mipipsMultiple* values for scaling up. The values beside the arrows indicate the MIPIPS values that result when multiplying a given *mipipsMultiple* with the MIPIPS value of a particular RPVMI's VM instance type.

the new VM that has to be started. This VM instance type is given by multiplying the MIPIPS value of the current VM instance's type with *mipipsMultiple* and rounding the result to the nearest MIPIPS value of any (the intended) VM instance type. The VM instance that is used as a reference point at a particular point in time is called the *Reference Point VM Instance (RPVMI)*. An example that demonstrates the effect of using different *mipipsMultiple* values for scaling up RPVMIs is shown in Figure 8.15. Starting from a VM instance of type *m1.small* that, in this example, has 50 MIPIPS, the subsequently selected VM instance types differ according to the used *mipipsMultiple* value when scaling up. Setting the *mipipsMultiple* value to 1.6, VM instances of the types *m1.medium*, *m1.large*, and *m2.4xlarge* are started in this order when executing corresponding scale-up reconfiguration actions. For example, the *m1.medium* VM instance type has 75 MIPIPS. Setting a corresponding VM instance as the RPVMI and scaling up the VM instance with *mipipsMultiple* set to 1.6 results in a request to start a VM instance that's MIPIPS value is nearest to $120 = 75 \cdot 1.6$. As the *m1.large* VM instance type exhibits the nearest MIPIPS value (100), a VM instance of the corresponding type is used for scaling up.

In contrast, starting from a VM instance of the VM instance type *m1.small* using a *mipipsMultiple* value of 2.0 results in starting a VM instance of the



(a) VM instance type of RPVMI is *m1.small* **(b)** VM instance type of RPVMI is *m1.large*

Figure 8.16. Shift of RPVMI's VM instance type regarding the scenario of Figure 8.15 when starting from VM instance type *m1.small* and using 2.0 for *mipipsMultiple*. The VM instance type of RPVMI (colored gray) shifts from *m1.small* (a) to *m1.large* (b).

type *m1.large* and then of the type *m2.4xlarge*. Hence, a VM instance of the *m1.medium* type is bypassed. This scenario that employs a *mipipsMultiple* value of 2.0 is also shown in Figure 8.16, where additionally the shift of RPVMI's VM instance type is illustrated. After starting with RPVMI's VM instance type *m1.small* in Figure 8.16a, the RPVMI VM instance type shifts to *m1.large* in Figure 8.16b. Furthermore, a *Condition* (see Figure 8.14) constitutes a trigger for executing the reconfiguration action with the help of a CPU utilization threshold and a time period. For example, concerning a scaling out action, an additional VM instance could be started when the CPU utilization lies above 80% for at least 20 minutes. In this context, the *Scope* element (see Figure 8.14) would define whether the 80% refer to the specific VM or to the average of all VM instances that were started from the corresponding *Node Configuration*.

The CDO example in Figure 8.17 shows an extended version of the CDO from Figure 8.4. A reconfiguration rule that uses vertical scaling was added to the VM that contains the *Service*s *1* and *2*, but not to the other VM that hosts *Service 0*. The exemplary reconfiguration grow rule starts a new VM instance if the CPU utilization stays above 80% for at least 20 minutes, for instance.

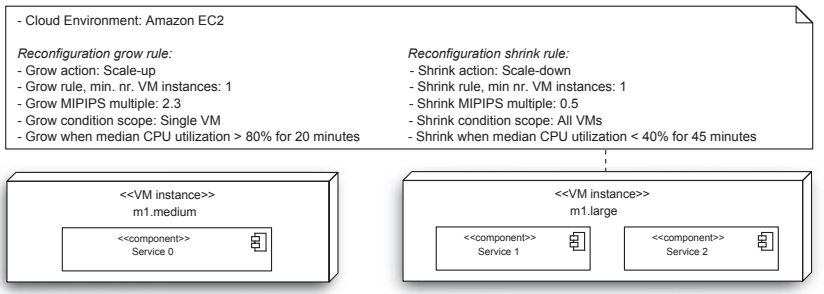


Figure 8.17. CDO example of Figure 8.4 with added reconfiguration rule

8.3.3 Genotype Model

The basic elements of genetic algorithms are specified by *genes*. Considering the classes in Figure 8.14, the ID of a service represents a single gene, for instance. All genes together constitute the so-called *genome* that contains the complete genetic information of all possible CDOs. Genes can be grouped in larger structures that are called *chromosomes* (cf. Section 4.1.2). Figure 8.18 illustrates the basic chromosomes and genes that are processed by CDOXplorer. The chromosomes correspond to the class structure of Figure 8.14 and map to one or more genes that together form a gene sequence. Such a single gene sequence encodes a specific CDO and is called a *genotype*.

The *Node Configuration* chromosome constitutes a container for further chromosomes that correspond to classes shown in Figure 8.14. As there can exist one ore more node configurations each having zero or one pair of a *Grow Rule* and *Shrink Rule* chromosome, the genotypes exhibit variable lengths. The *crossover point*s in Figure 8.18 are detailed in Section 8.4. The abbreviations and range of values that are used for the single genes are listed in Table 8.1. These genes correspond to the attributes of classes from Figure 8.14. Their values are limited to a narrow range and discrete spaces for avoiding a further growth of the search space. However, the ranges of some genes can be arbitrarily adapted if necessary, e.g., considering the condition time period of a shrink rule (S5).

Table 8.1. Design of the used genes

Gene	Range	Description	Chromosome
CE	ℕ	Cloud environment id	Cloud Env.
SE	ℕ	Service id	Service Comp.
IT	ℕ	VM Instance type id	Initial Start C.
NI	ℕ	Nr. of VM instances to start initially	Initial Start C.
GA	0,1	Grow action; 0: scale-up, 1: scale-out	Grow Rule
G1	ℕ	Minimum nr. of VM instances	Grow Rule
G2	1.1-3.0	MIPIPS multiple in steps of 0.1	Grow Rule
G3	0,1	Condition scope; 0: single VM, 1: all VMs	Grow Rule
G4	0.05-1.0	Condition median utilization in steps of 0.05	Grow Rule
G5	5-60	Condition time period in steps of 5 minutes	Grow Rule
SA	0,1	Shrink action; 0: scale-down, 1: scale-in	Shrink Rule
S1	ℕ	Minimum nr. of VM instances	Shrink Rule
S2	0.1-0.9	MIPIPS multiple in steps of 0.1	Shrink Rule
S3	0,1	Condition scope; 0: single VM, 1: all VMs	Shrink Rule
S4	0.0-0.95	Condition median utilization in steps of 0.05	Shrink Rule
S5	5-60	Condition time period in steps of 5 minutes	Shrink Rule

Figure 8.19 shows three examples of CDOs that are encoded as genotypes. Here, the third example `CD03` corresponds to the CDO that is depicted in Figure 8.17. Thus, it contains two node configurations from which only the second exhibits assigned grow and shrink rules, as can be seen by taking into account the general structure of genotypes in Figure 8.18. Hence, the cloud environment Amazon EC2 is encoded by the number 7 in this example (gene `CE`) and the first node configuration comprises only the genes 2-4. The deployed service 0 can be identified by the first gene `SE`. Furthermore, the second node configuration that includes a reconfiguration rule is represented by the genes 5 (`SE`) - 20 (`S5`). The other example `CD01` shows a genotype using the cloud environment Microsoft Windows Azure and only one node configuration, whereas the example `CD02` includes two node configurations and uses Amazon EC2. These genotypes are reused in Section 8.4 as examples.

8.3.4 Feasibility of CDOs

For reasoning about the feasibility of CDOs, the set of all node configurations in a CDO x is denoted as N and the set of all services in x is denoted as S. x is **feasible** if it (1) complies to the structure of CDOs (see Figure 8.18),

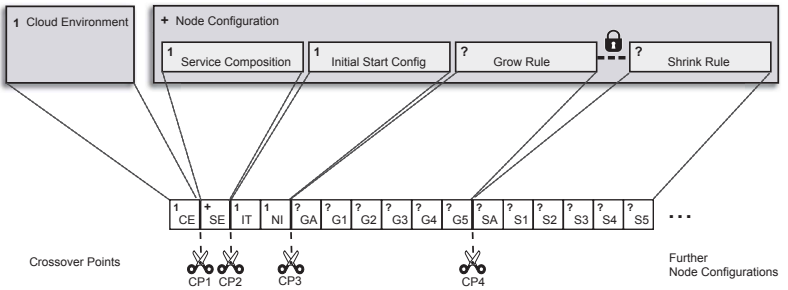


Figure 8.18. Compound chromosome overview. Gray boxes: chromosomes, white boxes: genes (listed in Table 8.1). `1`,`+`,`?` in the boxes' upper left corner indicate that the elements occur exactly once, at least once, and at most once, respectively.

(2) complies to the value ranges defined in Table 8.1, and (3) complies to the constraints that are described below. Furthermore, let Φ be the set of all feasible CDOs and gr name a grow rule and sr name a shrink rule. The following notation is used to define the constraints, where y denotes a gene or chromosome and z denotes a chromosome or CDO.

$y \prec z$: y is contained in z
$\triangle(y,z)$: Number of y in z
T_c: Set of VM instance types of cloud environment c

x has to comply with the following Constraints 8.3.1 - 8.3.7.

$$\forall s \in S, x \in \Phi : \triangle(s,x) \geqslant 1 \tag{8.3.1}$$

$$\forall s \in S, n \in N : \triangle(s,n) \leqslant 1 \tag{8.3.2}$$

$$\forall n \in N : \exists s \in S, s \prec n \tag{8.3.3}$$

$$\forall IT \prec x, CE \prec x, x \in \Phi : IT \in T_{CE} \tag{8.3.4}$$

$$\forall n \in N : \triangle(gr,n) = \triangle(sr,n) \leqslant 1 \tag{8.3.5}$$

$$\forall gr \prec n, sr \prec n, n \in N : gr \succ GA = SA \prec sr \tag{8.3.6}$$

$$\forall gr \prec n, sr \prec n, n \in N : gr \succ G4 > S4 \prec sr \tag{8.3.7}$$

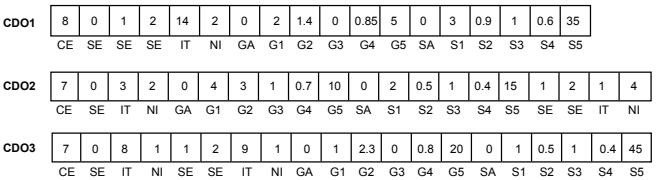


Figure 8.19. CDO examples encoded as genotypes

Constraint 8.3.1 describes that each service has to be present at least once in some node configuration of an individual. Furthermore, duplicated services are not allowed in a single node configuration (Constraint 8.3.2). The Constraint 8.3.3 states that at least one service has to be present in each node configuration.

A specific VM instance type (gene IT) also has to conform with a stated cloud environment (gene CE, see Constraint 8.3.4). Thus, VM instance types of Amazon EC2 cannot be used in conjunction with Microsoft Windows Azure, for instance. This constraint does not preclude that two VM instance types are different, but nevertheless both have to be VM instance types of the same cloud environment CE ($T_{CE} \ni IT_1 \neq IT_2 \in T_{CE}$). For instance, considering the example in Figure 8.20 that illustrates two CDOs as radar charts, $CE = 3$ for both CDOs and $2 = IT_1 \neq IT_2 = 5$ for the CDOs' single node configurations. However, 2 and 5 have to be VM instance types of CE, i.e., CDOXplorer has to ensure that $2 \in T_{CE}$ and $5 \in T_{CE}$.

Furthermore, Constraint 8.3.5 phrases the following limitation: If a grow rule exists in a node configuration, a shrink rule also has to be present in this node configuration and vice versa. Considering grow rules and shrink rules, the grow actions and shrink actions have to match (Constraint 8.3.6), i.e., a scale-out rule has to be accompanied by a scale-in rule and a scale-up rule has to be associated with a scale-down rule. The CPU utilization thresholds of grow rules and shrink rules indicate trigger points when to start or shut down a VM instance. Here, the CPU utilization threshold of a grow rule has to exceed the CPU utilization threshold of a shrink rule (Constraint 8.3.7).

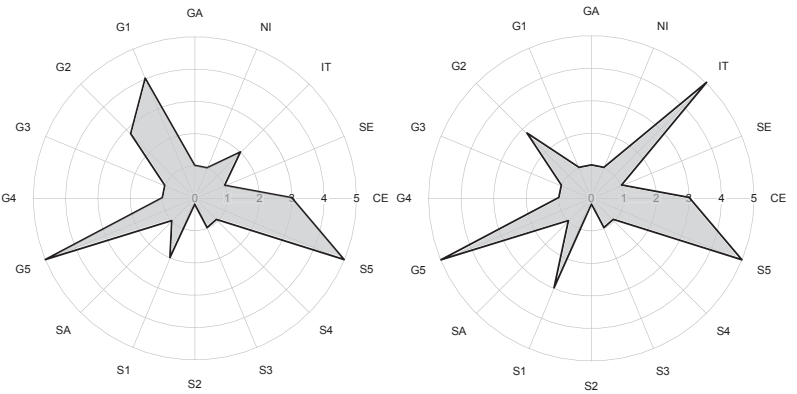


Figure 8.20. Feasible CDO examples with identical cloud environments (CE) but different VM instance types $IT_1 = 2$ (left) and $IT_2 = 5$ (right). Both CDOs include a single node configuration. The utilized VM instance types have to comply with CE ($IT_1 \in T_{CE}$ and $IT_2 \in T_{CE}$).

8.4 Crossover and Mutation Operators

Genetic algorithms typically use crossover and mutation operators to mimic evolutionary reproduction mechanisms for the advancement of a population over several generations (cf. Section 4.2). In the case of CDOXplorer, the individuals of a population that reproduce from generation to generation are CDOs. As the simulation of CDOs is very expensive, CDOXplorer's crossover and mutation operators are designed to only produce feasible CDOs (see Section 8.3.4). Hence, it is guaranteed that the operators are restricted to only spawn new CDOs that comply with the constraints defined in Section 8.3.4, for instance.

This section is structured as follows. Section 8.4.1 describes the crossover operator. Then, Section 8.4.2 describes the mutation operator.

8.4.1 Crossover Operator

The reproduction procedure involves, in the first place, the application of the crossover operator for producing two children from two parent individuals by mixing their genetic information. This technique follows the biological analogy for passing properties of the parents to their offspring. As both reproduction operators are intended to produce only feasible candidates in CDOXplorer, an arbitrary interleaving of genes is not allowed. Therefore, as a first measure, the mixing of genes is restricted to dedicated positions in the genotype that are called *crossover points*. Four crossover points `CP1-CP4` are defined that are also shown in Figure 8.18. They get selected by chance, are aligned to the boundaries of the chromosomes, and specify corresponding gene sequences that can be swapped.

Figure 8.21 shows two examples for applying the crossover operator with the help of the previously introduced CDOs of Figure 8.19. In Figure 8.21a, `CP4` was selected for mixing `CD01` and `CD02`. As `CD01` includes only one node configuration and the second node configuration of `CD02` contains no reconfiguration rules, only the first two shrink rules have to be swapped. An example that considers two CDOs where each contains two node configurations is shown in Figure 8.21b. As `CP2` is selected, both initial start configurations of `CD02` and `CD03` are swapped.

However, not every crossover point can be used with every combination of CDOs. For example, shrink rules can only be swapped when both parents have at least one shrink rule. Furthermore, exchanging the gene `CE` makes only sense if different cloud environments are used. Otherwise, the crossover operation would produce two new CDOs that are genetically identical to their parents. This would (1) not only introduce no new genetic characteristics to the common gene pool, but would also (2) waste resources as the identical CDOs might be simulated several times yielding the same values of the objective functions. The ranges of values described in Table 8.1 and the constraints specified in Section 8.3.4 have to be taken into account. Hence, swapping the gene `CE` implicates also the swapping of the VM instance types (gene `IT`) for retaining consistency, for instance.

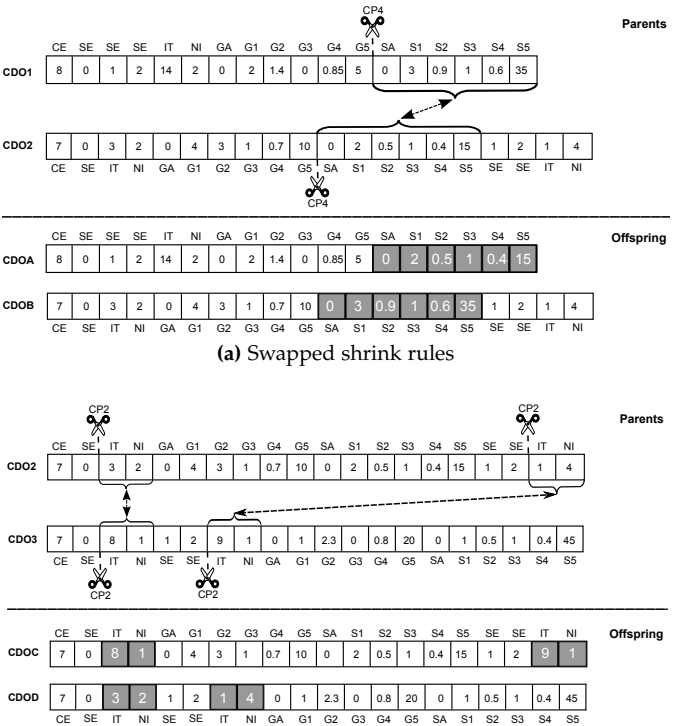


Figure 8.21. Crossover operator examples

8.4.2 Mutation Operator

After two parent individuals have initially produced two children with the help of the crossover operator, the mutation operator is applied to each child. In general, genetic algorithms randomly mutate single genes or gene sequences. This imitates sudden leaps and modifications to the global gene pool that occasionally appear during the evolution process. Considering the influence on the overall optimization procedure, the mutation operator

fosters retaining the diversity of the individuals and helps to avoid convergence to a local optimum. Just as the crossover operator, the mutation operator is aligned to the chromosome boundaries that can be seen in Figure 8.18. As a mutation also has to maintain the inner structure of a chromosome, the mutation operator is divided in five sub operators that are described in the following.

M-CE Mutates the cloud environment id (gene CE), i.e., a different cloud environment is used. The IT gene of each node configuration has to be modified as well as the formerly used VM instance types, as they are not available for the new cloud environment.

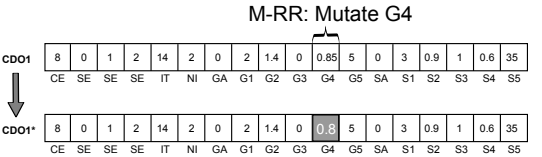
M-NN Mutates the number of node configurations and relocates the services. When a node configuration is added, a service (gene SE) is moved from another node configuration to the new one. When a node configuration is removed, all services are relocated to other node configurations.

M-IS Mutates the initial start configuration of a single node configuration, i.e., another VM instance type (gene IT) is selected or the number of VM instances that are initially started with regard to this node configuration (gene NI) is increased or decreased.

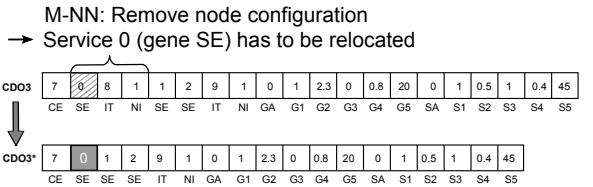
M-SC Mutates the service composition of a single node configuration. A service (gene SE) can be added or removed.

M-RR Mutates a reconfiguration rule, i.e., at least one of the genes GA, G1-G5, SA, S1-S5 is modified. When altering a grow rule, changes may also be necessary for the shrink rule to satisfy the constraints and vice versa.

Figure 8.22 shows two examples that utilize the mutation operator. In Figure 8.22a, the sub operator `M-RR` is used. In this example, the median utilization threshold of the grow rule (gene G4) is lowered by 5 %. The example in Figure 8.22b applies the mutation sub operator `M-NN` to CDO3 from Figure 8.19 that contains two node configurations. In this example, `M-NN` removes the first node configuration. As the service 0 is deployed only there, it has to be relocated to the second node configuration to satisfy the Constraint 8.3.1.



(a) The median utilization threshold of a grow rule condition is lowered from 85% to 80%



(b) The first node configuration is removed during this mutation. As the Service 0 is not present on any other node configuration, it has to be relocated to satisfy the constraints

Figure 8.22. Mutation operator examples

8.5 Adaptivity and Hybridity Characteristics

CDOXplorer uses adaptive mutation and crossover rates to increase the convergence speed of the genetic algorithm. Adaptive genetic algorithms have already been investigated for a long time and the potential to outperform pure variants was demonstrated in numerous applications (e.g., see [Srinivas and Patnaik 1994a; Yun and Gen 2003; Law and Szeto 2007]). The main goals of adapting reproduction mechanisms over time are to maintain the diversity of the populations and to retain convergence capabilities. CDOXplorer uses *parameter control* to implement the adaptivity. Parameter control adapts important parameters of a genetic algorithm, such as the mutation and crossover rate, during algorithm execution. In contrast, *parameter tuning* derives static parameters from several precedent test runs.

CDOXplorer also employs a combination with a local search technique to further improve its search results. Hence, it is also a hybrid genetic algorithm

(cf. Section 4.2). Hybrid genetic algorithms incorporate other optimization techniques such as further evolutionary heuristics or gradient-based search (cf. Section 4.1.1). CDOXplorer combines the population-based search of the genetic algorithm with a local search. Muhlenbein and Mahnig [2002] present an overview regarding the general differences of population-based search and local search, for instance. Similarly to the integration of adaptive mechanisms, combining genetic algorithms with other search techniques has been shown to be beneficial in a large number of application areas (e.g., see [Buckley 1996; Nie and Deng 2008; Man et al. 2008; Sha and Xu 2011]), for example, to further increase the convergence speed. The combination of genetic algorithms with the particular search technique local search is also known to deliver efficient optimization mechanisms (e.g., see [Ishibuchi and Murata 1999; Ishibuchi et al. 2003; Bhuvana and Aravindan 2011]).

The rest of this section describes CDOXplorer's adaptivity and hybridity characteristics in Section 8.5.1 and Section 8.5.2, respectively.

8.5.1 Adaptivity

CDOXplorer adapts the reproduction mechanism between subsequent generations to guide the search and to overcome local plateaus of the search space. The reproduction of two parent individuals is actually not performed in any case, but only with a certain probability. More specifically, the execution of the crossover and mutation operations are accomplished according to a separate crossover rate (cr) and mutation rate (mr). Instead of using fixed rates, CDOXplorer utilizes dynamically changing crossover and mutation rates. These rates adapt to the evolution of the generations' total fitness. Therefore, it is fair to classify CDOXplorer as adaptive.

Basically, CDOXplorer compares successive generations and determines the fraction of new elements in the new generation's current (temporary) pareto optimum. After the reproduction procedure of a generation finished, the fraction of new elements in the corresponding current pareto optimum in relation to the pareto optimum of the direct predecessor generation is denoted γ_c. In each generation, CDOXplorer removes the 50 individuals

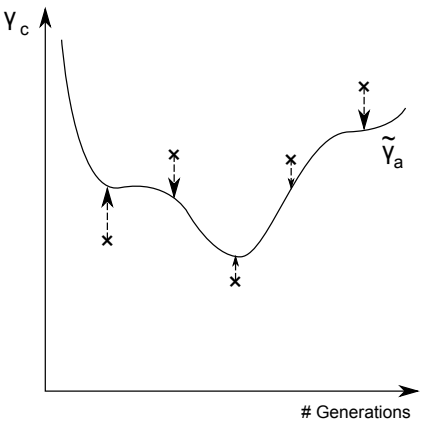


Figure 8.23. Tracking the progress of pareto optimum quality improvement over generations to adapt the crossover and mutation rates. Given two subsequent generations g_n and g_{n+1}, regarding g_{n+1}, γ_c denotes the fraction of new elements in g_{n+1}'s pareto optimum regarding g_n. $\tilde{\gamma}_a$ is the median of all γ_c per generation.

with the worst fitness values. Hence, the quality of the pareto optimum becomes better or at least remains stable with every generation and therefore, higher values of γ_c correspond to a higher improvement. Additionally, for each but the first generation, CDOXplorer calculates the median over all previous fractions of new elements (γ_c) that is called $\tilde{\gamma}_a$. The calculation procedures for actually computing the crossover and mutation rates are chosen by comparing γ_c with $\tilde{\gamma}_a$. Two cases C1 and C2 are distinguished, where C1 corresponds to $\gamma_c <= \tilde{\gamma}_a$ and C2 corresponds to $\gamma_c > \tilde{\gamma}_a$. Figure 8.23 illustrates an example that shows the progress of the pareto optimal quality improvement by comparing, for each generation, γ_c and $\tilde{\gamma}_a$.

C1) $\gamma_c <= \tilde{\gamma}_a$: In this case, the new generation only achieved a lower improvement than the other generations (on average) before. *cr* and *mr* are therefore set as follows: $cr = 1 - \gamma_c$ and $mr = \gamma_c$. Please note that in general crossover operations have a more disruptive impact than mutation operations. Hence, a lower improvement leads to an increased probability of applying the crossover operator and exploring areas of the search space that

are further away ("bigger leaps"). The intention is to investigate essentially new CDO structures that may exhibit fresh potential.

The case C2 is described in the next Section 8.5.2, as it represents at the same time CDOXplorer's hybridity characteristic.

8.5.2 Hybridity

As described in the previous section 8.5.1, CDOXplorer compares γ_c and $\tilde{\gamma}_a$ after the new individuals of each generation got evaluated, i.e., their fitness is determined by simulating the objective values with CDOSim. The comparison of γ_c and $\tilde{\gamma}_a$ distinguishes two cases C1 and C2. C1 was explained in Section 8.5.1, C2 is described below, as it also constitutes CDOXplorer's capability to incorporate a further search technique.

C2) $\gamma_c > \tilde{\gamma}_a$: In this case, the current pareto-optimal set changed more than the median of all previous changes. Therefore, the currently explored search space area appears to be promising and is now examined carefully and with a more fine-grained resolution. Thus, CDOXplorer avoids performing the disruptive crossover operation and restricts the mutation operation to `M-RR` (see Section 8.4.2), that is in turn executed each time ($mr = 1$) with small changes to the reconfiguration rules. The changes to the reconfiguration rules therefore affect the genes `GA`, `G1-G5`, `SA`, and `S1-S5`. Those changes usually have a lower impact on the overall results than, for example, using a different cloud provider or relocating services to new VM images. A modification that shuts down a VM instance after 25 instead of 20 minutes of lower CPU utilization generally influences the objective values to a lower extent, for instance. The modified reconfiguration rules therefore constitute the direct neighborhood of a specific CDO, as they represent solutions that are located in its direct surrounding of the search space. The thorough and structured investigation of this direct surrounding constitutes a local search. The combination with this meta-heuristic optimization method classifies CDOXplorer as *hybrid*.

8.6 Search Space Analysis

Table 8.1 shows that CDOXplorer employs a discrete instead of a continuous search space. Nevertheless, the size of the search space is remarkable. Because of the expensive simulation-based evaluation function, the scale of the search space is of peculiar interest for analyzing the complexity of the search problem that is tackled by CDOXplorer, as the simulation considerably limits the speed for processing diverse areas. This section analyzes the characteristics of the search space in Section 8.6.1. Section 8.6.2 examines the search space for the particular scenario of Figure 8.4 that results in 4,741,632 different CDOs for a status quo deployment that contains three services.

8.6.1 Search Space Characteristics

The surface of the search space is rugged, as already small changes of a gene's value (cf. Table 8.1) may lead to considerably different results. For example, launching new VM instances slightly too late may cause, under adverse conditions, overflows in queues that buffer requests and that cannot be recovered subsequently. Hence, a corresponding small change may result in a significant increase in SLA violations.

The size of the search space that has to be analyzed by CDOXplorer depends on various factors of specific cloud migration scenarios. For example, considering more cloud environments as potential target candidates for a cloud migration scenario significantly expands the search space. To support reasoning about the size of the search space, the already introduced notion concerning the set of all feasible CDOs Φ and the set of all distinct services S is utilized. Hence, given the cardinality $|S|$ of S and the Constraint 8.3.3, there exist $2^{|S|} - 1$ different combinations of services that can be deployed on each node configuration.

Furthermore, let c be the number of available IaaS-based cloud profiles and v the maximum number of VM instances that are allowed to start initially in a simulation of CDOSim. Regarding a cloud profile i, let a_i be the number

of allowed node configurations for i and t_i the number of VM instance types defined in i. Then, the number of all feasible CDOs $|\Phi|$ is given by Formula (8.6.1).

$$\begin{aligned} |\Phi| &= \sum_{i=1}^{c}((2^{|s|}-1)\cdot t_i v \cdot (1+2\cdot \\ &\quad \overbrace{3\cdot 20\cdot 2\cdot 12}^{G1-G3;G5}\cdot \overbrace{3\cdot 9\cdot 2\cdot 12}^{S1-S3;S5}\cdot \overbrace{\sum_{j=1}^{20} j}^{G4;S4}))^{a_i} \\ &= \sum_{i=1}^{c}((2^{|s|}-1)\cdot 391,910,401\cdot t_i v)^{a_i} \end{aligned} \tag{8.6.1}$$

Each summand of the outer sum represents the number of feasible CDOs for a specific cloud profile i. $2^{|S|}-1$ different combinations of services can be deployed on t_i different VM instance types and up to v VM instances can be started initially for each of those configurations. Furthermore, a node configuration does not need to have reconfiguration rules. This case is represented by "1" in the inner braces. Otherwise, when the grow and shrink rules are used, either a *scale-out/scale-in* or a *scale-up/scale-down* combination can be applied (two possibilities). CDOXplorer is configured in a way that allows the genes `G1` and `S1` to take values from 1 to 3. Further on, the value ranges in Table 8.1 show that `G2` contributes 20 and `S2` 9 possibilities, as well as the number of the possibilities for `G3`, `S3`, `G5`, and `S5`. `G4` and `S4` account for the last inner sum, as additionally to their value ranges, Constraint 8.3.7 has to be considered.

8.6.2 Example Scenario

The scenario of Figure 8.4 was already used in Section 8.1.1 to point out the considerable magnitude of the search space. This section describes the scenario in more detail and demonstrates the calculation of the search space size with the help of the Formula 8.6.1. The scenario includes a status quo deployment with three status quo nodes that result in the three services *Service 0* to *Service 2*. The following assumptions were made:

Cloud environment(s):	Amazon EC2 ($c = 1$)
VM instance type(s):	12 VM instance types ($t_i = 12$) of Amazon EC2 ($i = 1$)
Reconfiguration rule(s):	0 (ignore parts for `GA,G1-G5,SA,S1-S5`) in Formula 8.6.1
VM image(s):	3 ($a_i = 3$)
Max. VM instance(s) that can be started initially from VM image:	2 ($v = 2$)

Hence, the number of possible CDOs is given by Formula 8.6.2.

$$|\Phi| = \sum_{i=1}^{1}((2^3 - 1) \cdot 12 \cdot 2 \cdot 1)^3 = 4,741,632 \tag{8.6.2}$$

8.7 Summary

This chapter describes CloudMIG's mechanisms to optimize the deployment of an SUA to the cloud and to also improve the general resource utilization to exploit the cloud's elasticity. Many decisions have to be considered by SaaS providers when planning the move to the cloud, for example, the best-suited cloud environment, deployment architecture, VM instance types, and *reconfiguration rules*—to allow for an efficient, dynamic resource scaling—have to be chosen. Those decisions form a *Cloud Deployment Option (CDO)*. Finding the best-suited CDO constitutes a multi-objective optimization problem. The CloudMIG method defines a meta-model for specifying CDOs and mechanisms for creating CDOs automatically (cf. research question Q4.1 in Section 5.4).

CloudMIG utilizes a genetic algorithm named *CDOXplorer* to explore the huge search space of all possible CDOs. It aims to optimize CDOs that employ IaaS-based cloud environments. For automatically evaluating the fitness of CDOs (cf. research question Q4.2 in Section 5.4), CDOXplorer employs the simulation tool CDOSim that allows to simulate specific CDOs using, among others, monitored usage data. As CDOSim can assess a CDO's costs, response

times, and SLA violations, CDOXplorer also optimizes these three important objectives. CDOXplorer is therefore a simulation-based genetic algorithm (cf. research question Q4.3 in Section 5.4). As the fitness function is very expensive, it is important to ensure the convergence of the optimization procedure, as only a limited number of generations and individuals per population can be used (cf. Chapter 12).

CDOXplorer uses a KDM-based model of the SUA, a status quo deployment model, a workload profile, and one or more cloud profiles as its *input*. The cloud profiles constitute cloud environment candidates that may be utilized for building CDOs. As there usually does not exist a best solution that outperforms all other individuals regarding each objective, multi-objective optimization procedures rather return a pareto-optimal set of individuals. Hence, CDOXplorer also returns a set of pareto optimal CDOs as its *output*. A CloudMIG user can compare these individuals and manually select the best-suited candidate.

CDOXplorer maps each status quo deployment node to an atomic *service*. One or more services can be deployed to VM instances. CDOXplorer uses CEM's *Mapping* package to map KDM elements of the SUA to cloud resources. CEM's *Pricing* and IaaS packages are also utilized for optimizing the CDOs. Furthermore, an appropriate domain-specific design of a genetic algorithm's reproduction operators can significantly improve the algorithm's overall performance. For example, CDOXplorer's mutation operator delegates the mutation to one of five sub operators to obtain feasible candidates and to ensure diversity. To select individuals for reproduction, CDOXplorer reuses NSGA-II's selection operation applying two tournament rounds.

The genetic algorithm CDOXplorer is also *adaptive* as it modifies the mutation and crossover rates according to the advancement of the generations' fitness. Furthermore, CDOXplorer is also *hybrid* as it employs a local search for thoroughly investigating the neighborhood of a set of promising CDOs. Nevertheless, the size of the search space that has to be searched for well-suited CDOs is still huge. Hence, this chapter also presented a search space analysis for reasoning about characteristics of the search space.

Chapter 9

Tool Architecture

To provide tool support for the CloudMIG method, a capable software architecture is essential for incorporating all of CloudMIG's activities, artifacts, models, and roles. An extensible, modular software architecture is required to allow distinct roles (cf. Section 6.3.3) to add artifacts and components to the tool *CloudMIG Xpress*. *CloudMIG Xpress* itself provides an implementation of this architecture and is described in Chapter 10.

This chapter details the underlying software architecture that facilitates the contribution of additional cloud profiles, monitoring format readers, CEC validators, and KDM discoverers by the according roles. Hence, a focus is on an adequate modularization of CloudMIG's elements to allow for a smooth integration of those role-specific contributions (cf. research question *Q5.1* in Section 5.5). Moreover, this chapter investigates further quality characteristics that are relevant for the software tool and that have to be explicitly assembled into the architecture (cf. research question *Q5.2*). Quality properties, such as efficiency, maintainability, and adaptability, can be addressed on the architectural level by using architectural patterns and styles [Bass et al. 2003; Taylor et al. 2009]. Thus, this chapter also examines appropriate architectural patterns and styles and shows how they are integrated for attaining the defined quality goals (cf. research question *Q5.3*).

Systematically designing software architectures on the basis of given functional requirements and quality attributes is, for example, facilitated by the Attribute-Driven Design (ADD) method [Bass et al. 2002; 2003]. The method iteratively decomposes system components based on *architectural drivers* and

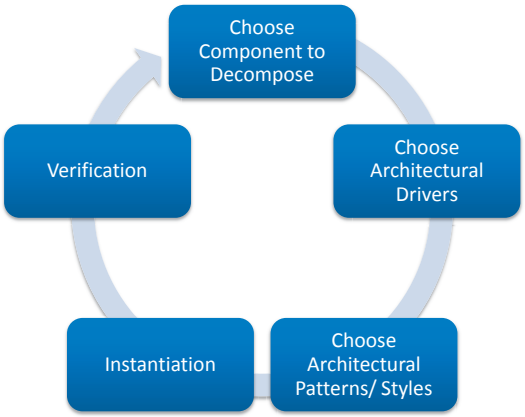


Figure 9.1. The ADD-based iterative architecture design process (based on [Bass et al. 2003])

attribute primitives. Architectural drivers are functional or non-functional requirements that are architecturally significant, whereas attribute primitives denote architectural patterns that can be used to achieve certain quality attributes [Bass et al. 2002]. Further important concepts in this context are defined by the notions of *tactics* and *architectural strategies*. A tactic is a design decision that affects the control of a quality attribute, whereas an architectural strategy is a collection of tactics [Bass et al. 2003]. For further details regarding the ADD method, we refer to Bass et al. [2003].

The tool architecture is constructed using the iterative architecture design process that is illustrated in Figure 9.1. The process builds upon the ADD method and uses a structured approach to derive an appropriate architecture with the help of an architectural strategy. The architectural strategy's tactics incorporate architectural patterns and styles to embed the architectural drivers in the tool architecture.

Central components (selected by the phase *Choose Component to Decompose*) are iteratively decomposed. The phase *Choose Architectural Drivers* selects the corresponding architectural drivers for a component that is to be decomposed. Then, the phase *Choose Architectural Patterns/Styles* selects

architectural patterns and styles that enable to implement the architectural drivers in the phase *Instantiation*. The *Verification* phase checks the consistency of the new architectural elements with all other architectural drivers. The process iteratively decomposes the components starting from a coarse-grained system view (a single component) that covers all requirements.

The remainder of this chapter is structured as follows. Section 9.1 describes the architectural strategy used for building the tool architecture, i.e., the overall architectural drivers and the corresponding chosen architectural patterns and styles as depicted in Figure 9.1. An overview of the architecture that eventually results from the ADD-based process is then presented in Section 9.2. Finally, Section 9.3 summarizes this chapter.

9.1 Architectural Strategy

The ADD-based iterative architecture design process of Figure 9.1 employs architectural patterns and styles that exhibit known quality properties to address architecture-relevant functional and non-functional requirements, i.e., architectural drivers. These design decisions form an architectural strategy [Bass et al. 2003] that is presented in this section.

The corresponding architectural drivers are described in Section 9.1.1. Section 9.1.2 identifies architectural patterns and styles that match with the architectural drivers.

9.1.1 Architectural Drivers

Architectural drivers incorporate central functional and non-functional requirements, as those major goals inevitably have to be considered in the design of the tool architecture. The central functional and non-functional requirements are described in the following.

Functional Requirements Functional requirements are derived from CloudMIG's activities A1-A6 (cf. Section 6.3.1). The requirement descriptions state the corresponding activities to enable a high-level traceability.

F-01	*KDM model extraction*
Activities:	A1
Description:	KDM models are used by CloudMIG to statically analyze an SUA. Hence, instances of KDM's *Source*, *Code*, and *Action* packages have to be extracted from the SUA's source code. The extraction is applied by so-called *KDM discoverers*.

F-02	*Addition of new KDM discoverers*
Activities:	A1
Description:	KDM discoverers can extract instances of KDM's *Source*, *Code*, and *Action* packages from source code of a particular programming language. KDM serves as an intermediate model format. To support SUAs that are built with arbitrary object-oriented programming languages, it has to be possible to create and integrate further KDM discoverers by the corresponding KDM discoverer contributor role.

F-03	*Modeling of status quo deployment*
Activities:	A1
Description:	Additionally to the KDM models that are extracted from an SUA, the status quo deployment of the system has to be taken into account, for example, for automatically generating potential target architectures. Hence, it is necessary to provide modeling capabilities for manually enriching the extracted source code model with information describing it's deployment.

F-04	*Workload profile creation*
Activities:	A1
Description:	Essential parts of CloudMIG's utilization model are workload profiles that describe the SUA's usage patterns. Workload profiles can be created by importing monitoring log data. However, CloudMIG should also support scenarios where no monitoring log data is available. Hence, modeling capabilities have to be provided that enable specifying arbitrary workload profiles.

F-05	*Addition of new monitoring format readers*
Activities:	A1
Description:	Creating workload profiles from monitoring log data requires corresponding reader components that enable to import specific monitoring log formats. The tool has to provide means for integrating further monitoring format readers by the corresponding monitoring format reader contributor role.

F-06	*Application of SMM measures*
Activities:	A1,A5
Description:	Besides workload profiles, CloudMIG's utilization model can incorporate SMM measurements according to specific measures. For this purpose, it should be possible to apply arbitrary SMM-based measures on the extracted KDM models. Furthermore, the application of SMM-based measures is also relevant for rating specific target architectures.

F-07	*Modeling of cloud profiles*
Activities:	A2
Description:	As the number of cloud providers and their offered services change at a rapid pace, modeling capabilities have to be provided that enable to create or modify cloud profiles by the corresponding cloud profile contributor role. An important part of a cloud profile is the set of CECs that are imposed by a cloud environment.

F-08	*CEC violation detection framework*
Activities:	A3,A4
Description:	For investigating the suitability of an SUA regarding a specific cloud environment, the number and characteristics of CEC violations concerning this cloud environment play an important role. A framework is needed that controls the validation of CECs through incorporating CEC validators that can be added by CEC validation contributors.

F-09	*Addition of new CEC validators*
Activities:	A3,A4
Description:	The tool has to provide means that allow adding new CEC validators by the corresponding CEC validation contributor role. This might be necessary if further programming languages should be supported and language-specific CECs exist, for instance.

F-10	*Visualization of detected CEC violations*
Activities:	A3,A4
Description:	To explore detected CEC violations, a visualization has to be provided that augments the regarding source code. Furthermore, the visualization should show a source code model that enables to show and hide specific parts of the SUA. For example, a filter may hide all parts that do not raise CEC violations.

F-11	*Generation of CDOs*
Activities:	A3,A4
Description:	CloudMIG allows to automatically map a status quo deployment to a generated target architecture. Hence, a corresponding component is required that generates CDOs according to, among others, chosen cloud profiles. The component also has to implement CDOXplorer for optimizing CDOs.

F-12	*Modeling of CDOs*
Activities:	A4
Description:	Generated CDOs may not meet specific requirements of a SaaS provider. Hence, modeling capabilities are needed that allow a modification of CDOs or allow to manually build a complete CDO.

F-13	*Simulation of CDOs*
Activities:	A5
Description:	CDOs have to be rated for enabling a comparison and the selection of the best-suited candidate. Hence, a simulation tool is needed that estimates future costs, response times, and SLA violations of a CDO according to a specific workload profile, for instance.

Non-Functional Requirements The non-functional requirements are also derived from CloudMIG's activities A1-A6. The requirements are mapped to the quality characteristics and subcharacteristics of the ISO/IEC 9126-1:2001 standard [ISO/IEC 2001] (abbr. ISO/IEC 9126 in the following) to provide a baseline for connecting the non-functional requirements with known characteristics of architectural patterns and styles in Section 9.1.2.

The ISO/IEC 9126 standard defines the *time behavior* and *resource behavior* as relevant subcharacteristics of *efficiency*. The non-functional requirements *NF-04* and *NF-05* (described below) address the efficiency of some of the tool's software components. However, monitoring format readers may also be contributed by external developers and interested researchers can replace the algorithms for CEC validation and CDOs optimization for assessing alternative solutions. Hence, *NF-04* and *NF-05* refer to components that are provided by the current version of *CloudMIG Xpress* 0.5 Beta.

NF-01	*Simple addition of software component contributions*
Activities:	A1,A3,A4,A5
ISO/IEC 9126:	Changeability
Description:	In contrast to the cloud profile contributor, the monitoring format reader contributor, CEC validation contributor, and KDM discoverer contributor roles can add corresponding software components to the tool. To support and simplify the addition of those components, their coupling should be as low as possible. Hence, it should be possible to plug the components into the tool without having the roles to consider a large number of dependencies. Low coupling is a classic quality characteristic that is known to improve changeability [Li and Henry 1993] (a subcharacteristic of maintainability).

NF-02	*Simple addition of CEC violation views*
Activities:	A3,A4
ISO/IEC 9126:	Changeability
Description:	CEC violations may be visualized in a variety of ways, for example, considering histograms, graphs, and UML diagrams. To enable a simple addition of further views, the corresponding architecture should follow an approach that ensures the independence of the views while at the same time building on a common data model.

NF-03	*Exchangeable CDO optimization and CEC validation elements*
Activities:	A3,A4
ISO/IEC 9126:	Changeability
Description:	The algorithms for CDO optimization and CEC validation constitute important parts of the tool as they directly affect the core contributions of this thesis (cf. Section 1.3). A simplified replacement of the corresponding elements is worthwhile because it facilitates comparing the algorithms with other state-of-the-art approaches. Thus, interested researchers can easily assess and compare alternative solutions.

NF-04	*Efficient CDO optimization and CEC validation*
Activities:	A3,A4
ISO/IEC 9126:	Efficiency
Description:	The optimization of CDOs and detection of CEC violations are computationally expensive. Especially the optimization of CDOs is inherently costly regarding hardware resources and runtime as it employs several simulation runs. Besides those obstacles, the corresponding architectural elements and subsystems should be laid out as efficient as possible to mitigate potential performance problems.

NF-05	*Efficient import of monitoring log data*
Activities:	A1,A5
ISO/IEC 9126:	Efficiency
Description:	Log files that can be used as sources for extracting workload profiles may be considerably large. Log files include information regarding separate service calls and corresponding response times. They have to be parsed and the data is transformed to an SMM instance. The architectural elements covering this import and transformation of monitoring log data should be designed to work efficiently.

9.1.2 Architectural Patterns and Styles

This section builds upon the identified architectural drivers (cf. Section 9.1.1) and the mapping of the included non-functional requirements to characteristics of ISO/IEC 9126. The architectural drivers and the mappings are used as a basis for selecting appropriate architectural patterns and styles that are known to address the specific characteristics.

Most complex systems do not comply to a single style, but rather combine two or more styles that are then called *heterogeneous styles* [Zhu 2005]. As the boundary between architectural patterns and styles is blurred [Taylor et al. 2009], we do not distinguish both concepts in the following. The tool architecture is constructed using four basic architectural styles. These

styles and relevant architectural elements are briefly described below. For a detailed description of these styles we refer to Taylor et al. [2009].

Implicit Invocation The software components that may be contributed by the monitoring format reader contributor, CEC validation contributor, and KDM discoverer contributor roles can be integrated as plugins. They have to inherit from defined abstract classes and are utilized by the tool by using an implicit invocation mechanism. Hence, the plugins register at the platform and their functionality is utilized not directly, but by incorporating a level of indirection. The corresponding publish-subscribe mechanism decouples the plugins from the platform as addressed by *NF-01*. As the plugins only have to inherit from a single class and implement few abstract methods, integrating, for example, a further monitoring data reader in the tool is uncomplicated. The plugins will be revisited in the overview of the architecture that is described in Section 9.2.

Model View Controller (MVC) MVC targets scenarios that provide several different views showing data originating from a common model. Hence, MVC is ideally suited for addressing *NF-02* and providing a basis for integrating visualizations of CEC violations.

Layer To enable a simple replacement of the architectural elements covering the CDO optimization and CEC validation (cf. *NF-03*), the dependencies of these elements should be reduced. This can be achieved by building a layered architecture. The elements and dependencies are structured such that elements that are placed in higher layers only depend on elements of lower layers. Shaping the architecture in a way that considers the CDO optimization and CEC validation as layers therefore makes it easier to replace them by alternative approaches. The same benefits apply to subsystems that deliver models that are required as input. For example, the CDO optimization requires, among others, a KDM model, a workload profile, and a component that simulates CDOs. It should be noted that the described components that are created with the help of layers form the framework parts that can integrate related plugin components. For example, the framework that controls the CEC validation process would itself be realized as a layer

that manages and utilizes corresponding CEC validation plugins via means of implicit invocation as described above.

Batch-sequential For executing a large number of jobs and processing a vast amount of data without manual intervention, systems can be built using batch processing. Regarding the non-functional requirements that were defined before, *NF-04* and *NF-05* can be tackled by constructing the actual CDO optimization and CEC validation layers (cf. the layered architectural style above) and the import of monitoring log data as components that employ batch processing. Once these tasks are started, no manual intervention is needed.

9.2 Architecture Overview

This section describes the conceptual architecture that was created with the help of the ADD-based architecture design process (cf. Figure 9.1). The high-level architecture is described by taking a logical view. This logical view includes (1) the major components that correspond to the functional requirements (cf. Section 9.1.1). Furthermore, it shows (2) the composition and integration of those components—via appropriate connectors—that result from the chosen architectural strategy (cf. Section 9.1).

An overview of the conceptual tool architecture is shown in Figure 9.2. The figure shows the major logical components that abstract from more than 800 classes that are included in *CloudMIG Xpress*' fine-grained design. In the previous Section 9.1.2, architectural layers were introduced as a tactic for facilitating changeability regarding specific architectural elements. The components of the resulting layers are colored gray in Figure 9.2. However, the layering is not strict, i.e., components of higher layers are allowed to access lower layers by bypassing intermediate layers.

Some interfaces of *CloudMIG Xpress* are provided to external components. Those components actually constitute plugins that can be incorporated in *CloudMIG Xpress*. They implement an abstract type that is defined by

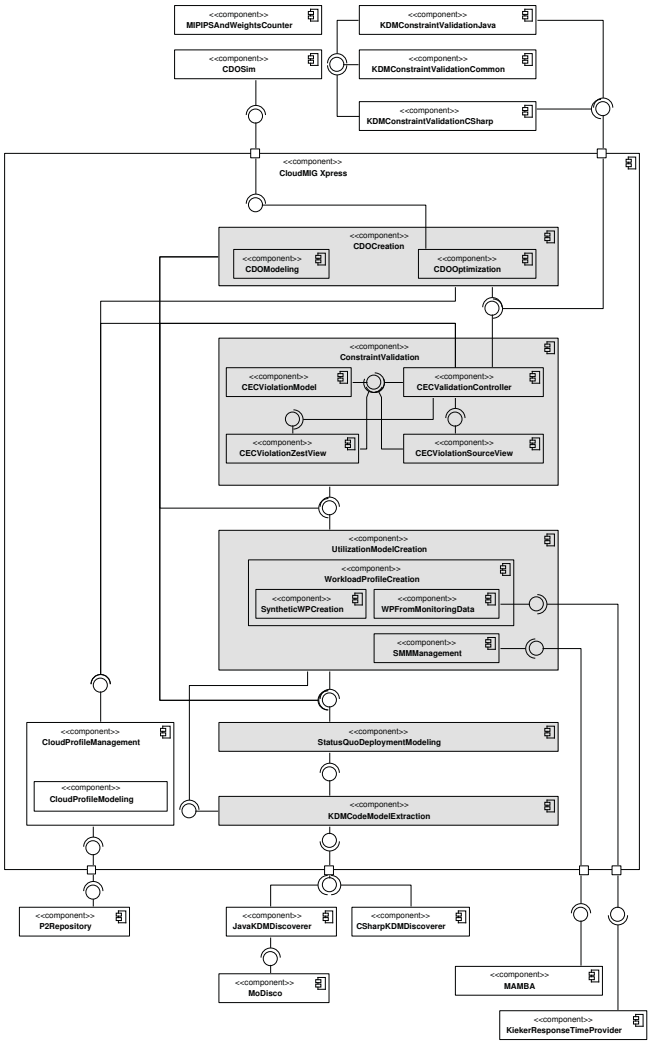


Figure 9.2. CloudMIG Xpress conceptual architecture overview. The components colored gray correspond to the layered architecture that is, among others, implied by NF-03.

some internal component. This allows them to be used according to *CloudMIG Xpress*' protocols. For example, in contrast to the component `MAMBA`, the component `KiekerResponseTimeProvider` is such a plugin.

In the following, the components shown in Figure 9.2 are briefly described. For reasons of traceability, the corresponding architectural drivers (central requirements that impact the architecture) are stated for each component.

CDOCreation *[F-11,F-12,NF-03,NF-04]:* This layer component covers the creation of CDOs both manually and automatically, i.e., via optimization using CDOXplorer.

CDOModeling *[F-12,NF-03]:* This component is a subcomponent of `CDOCreation` and addresses the manual creation of CDOs. CDOs might be manually modified if the automatic creation yields unwanted CDO properties, for instance.

CDOOptimization *[F-11,NF-03,NF-04]:* This component is a subcomponent of `CDOCreation` and addresses the automatic creation of near-optimal CDOs. That means, our genetic algorithm CDOXplorer is implemented that yields pareto-optimal sets of CDOs.

CDOSim *[F-13,NF-04]:* Our simulator builds on CloudSim and provides cloud user-centric enhancements for simulating CDOs [Fittkau et al. 2012a; b]. The simulator determines costs, response times, and number of SLA violations regarding a specific CDO that describes a combination of a specific cloud environment resources, deployment model, and reconfiguration rules (cf. Section 8.1).

CECValidationController *[F-08,F-09,F-10,NF-01,NF-02,NF-03,NF-04]:* This component is a subcomponent of `ConstraintValidation` that itself is implemented according to the MVC pattern. `CECValidationController` constitutes the corresponding MVC controller component. For reasons of performance, the check for CEC violations itself is accomplished in a batch-sequential fashion.

CECViolationModel *[F-08,F-09,F-10,NF-01,NF-02,NF-03,NF-04]:* This component is a subcomponent of `ConstraintValidation` that itself is imple-

mented according to the MVC pattern. `CECViolationModel` constitutes the corresponding MVC model component.

CECViolationSourceView *[F-08,F-09,F-10,NF-01,NF-02,NF-03,NF-04]:* This component is a subcomponent of `ConstraintValidation` that itself is implemented according to the MVC pattern. `CECViolationSourceView` constitutes one of the two corresponding, specific MVC view components. This component adds a source code view to the tool for tracing detected CEC violations to the responsible source code statements.

CECViolationZestView *[F-08,F-09,F-10,NF-01,NF-02,NF-03,NF-04]:* This component is a subcomponent of `ConstraintValidation` that itself is implemented according to the MVC pattern. `CECViolationZestView` constitutes one of the two corresponding, specific MVC view components. This component adds a graph view to the tool that is named according to the underlying visualization toolkit Zest.[1] The view shows a reverse-engineered SUA as a graph. Source code elements are included as nodes, containment relationships are included as edges. Nodes can be augmented with additional information if the corresponding elements are known to raise a CEC violation.

CloudProfileManagement *[F-07,F-08,F-09,F-10,F-11,F-12,F-13]:* Cloud profiles play a central role in the context of CloudMIG. They are needed, for example, for modeling CECs and as a basis for conformance checking and creation and evaluation of CDOs. This component provides functionalities for managing cloud profiles, e.g., for creating, storing, reading, searching, and modifying those cloud profiles.

CloudProfileModeling *[F-07]:* This component is a subcomponent of `CloudProfileManagement`. It covers the manual creation of cloud profiles by the corresponding cloud profile contributor role.

ConstraintValidation *[F-08,F-09,F-10,NF-01,NF-02,NF-03,NF-04]:* This layer component covers the conformance checking process for detecting the CEC violations regarding a combination of SUA and cloud profile. It provides the corresponding framework components that can incorporate further

[1] `http://www.eclipse.org/gef/zest/`

constraint validation plugins. The component is constructed according to the MVC architectural pattern.

CSharpKDMDiscoverer *[F-01,F-02,NF-01]:* This plugin component extracts KDM models from C# source code. In contrast to the `JavaKDMDiscoverer` component that extracts KDM models from Java source code, it does not rely on the `MoDisco` component. The utilized three-phase transformation of C# to KDM is described in Section 10.1.

JavaKDMDiscoverer *[F-01,F-02,NF-01]:* This plugin component extracts KDM models from Java source code. It builds on the `MoDisco` component. However, substantial modifications are required as, for example, `MoDisco` only allows to extract KDM models from Eclipse projects and included resources. `JavaKDMDiscoverer` eliminates this requirement.

KDMCodeModelExtraction *[F-01,F-02,NF-01]:* This layer component constitutes the framework part for extracting KDM models from software systems. It controls the extraction process and also integrates and manages corresponding KDM discoverer plugin components. For example, it defines an abstract `AbstractKDMDiscoverer` class that has to be implemented by KDM discoverer plugins such that they can be used in the course of the KDM extraction process.

KDMConstraintValidationCommon *[F-09,NF-01,NF-04]:* This component provides mechanisms for processing extracted KDM models and detecting CEC violations independently of the used programming language (cf. Section 7.1.3). It also provides fundamental features that can be reused by other CEC validation plugins—that might also cover language-dependent CEC violations—such as essential model querying capabilities.

KDMConstraintValidationCSharp *[F-09,NF-01]:* This plugin component integrates in the conformance checking process and can detect CEC violations that are specific to the programming language C# (cf. Section 7.1.3). It reuses basic features from the `KDMConstraintValidationCommon` component.

KDMConstraintValidationJava *[F-09,NF-01]:* This plugin component integrates in the conformance checking process and can detect CEC violations

that are specific to the programming language Java (cf. Section 7.1.3). It reuses basic features from the `KDMConstraintValidationCommon` component.

KiekerResponseTimeProvider *[F-04,F-05,NF-01,NF-05]:* This plugin component imports monitoring log data from the monitoring tool Kieker [van Hoorn et al. 2009; van Hoorn et al. 2012] for building workload profiles. For reasons of performance, the import process works in a batch-sequential fashion.

MAMBA *[F-04,F-06]:* Our Measurement Architecture for Model-Based Analysis (MAMBA) is used for applying SMM-based measures. MAMBA is described in greater detail in Section 10.2.

MIPIPSAndWeightsCounter *[F-03,F-07,F-13]:* This component constitutes the MIPIPS and weights benchmark as described in Section 8.2. It is not directly referenced from other components as its results have to be entered manually in cloud profiles or status quo deployment models.

MoDisco *[F-01,F-02,NF-01]:* MoDisco [Bruneliere et al. 2010] is a framework for model-driven reverse engineering that aims to facilitate discovering and understanding specific aspects of legacy software systems, such as their architecture, documentation, and revision history. It is used by the `JavaKDMDiscoverer` component to extract KDM models from Java source code.

P2Repository *[F-02,F-05,F-07,F-09]:* This component constitutes the repository for centrally storing cloud profiles [Kund 2013] (not publicly available yet). It is named according to the used Eclipse P2 technology.[2] In future versions, it could also include and distribute plugin components such as additional monitoring format readers and CEC validators (cf. Section 6.3.3).

SMMManagement *[F-04,F-06]:* This component is a subcomponent of `UtilizationModelCreation`. For example, it creates and submits SMM measures to the MAMBA component and provides the measurement results to components that build a workload profile.

StatusQuoDeploymentModeling *[F-03,NF-03]:* This layer component covers the modeling of status quo deployment models (cf. Section 6.2). Hence,

[2] `http://www.eclipse.org/equinox/p2/`

it requires extracted KDM code models that can then be assigned to the on premise server nodes.

SyntheticWPCreation *[F-04,NF-03]:* This component is a subcomponent of `WorkloadProfileCreation` and addresses the creation of synthetic workload profiles. If no monitoring log data is present, synthetic workload profiles can be used for modeling guessed usage patterns with the help of a mathematical definition.

UtilizationModelCreation *[F-04,F-05,F-06,NF-01,NF-03,NF-05]:* This layer component covers the creation of utilization models. For example, it builds on SMM measures to model workload profiles.

WorkloadProfileCreation *[F-04,F-05,NF-01,NF-03,NF-05]:* This subcomponent of `UtilizationModelCreation` addresses the construction of workload profiles. It provides common features for creating synthetic workload profiles and for creating workload profiles from monitoring log data.

WPFromMonitoringData *[F-04,F-05,NF-01,NF-03,NF-05]:* This component is a subcomponent of `WorkloadProfileCreation` and addresses the creation of workload profiles from monitoring log data. It constitutes the framework part that integrates and controls external monitoring data reader plugins.

9.3 Summary

This chapter describes the conceptual software architecture underlying the tool *CloudMIG Xpress* that provides support for the CloudMIG method. The software architecture results from the corresponding research questions *Q5.1-Q5.3* that are described in Section 5.5. A well-suited modularization is addressed by *Q5.1* and pursued by an iterative architecture design process that builds on the Attribute-Driven Design (ADD) method from Bass et al. [2002].

The architecture design process iteratively decomposes important components and identifies architectural drivers, i.e., central functional and non-functional requirements that are architecturally significant. The corre-

sponding quality properties are mapped to ISO/IEC 9126 characteristics and subcharacteristics. They cover *Q5.2* and also form a basis for selecting architectural patterns and styles (*Q5.3*) that are known to foster the attainment of those quality characteristics.

The conceptual architecture is described using a logical view that covers the major components and connectors. It abstracts from more than 800 actual classes and, according to *Q5.3*, includes several overlaying, heterogeneous styles. For example, central features of CloudMIG are structured in a layered architecture to facilitate changeability and maintainability. Features that comprise repeated processing, that does not require manual intervention, follow the batch-sequential style for reasons of performance.

Part III

Evaluation

Chapter 10

Tool Implementation

The application *CloudMIG Xpress* provides tool support for planning the migration of enterprise software systems to the cloud according the CloudMIG method. It is available as open source software[1] under Apache License V2.0. The tool logo is shown in Figure 10.1. *CloudMIG Xpress* follows the software architecture that is described in Chapter 9. It is implemented using the Eclipse Rich Client Platform (RCP) technology [McAffer et al. 2010]. Eclipse RCP emerged from the well-known Eclipse IDE and evolved to a full-fledged application platform that facilitates building cross-platform GUI-based applications. The platform follows a plugin-based approach. It consists of a set of plugins and at the same time also allows additional functionalities to be added as further plugins. As a consequence of this modularization approach, the plugin components of the tool architecture described in Chapter 9 are implemented in the form of Eclipse plugins.

The previous Chapter 9 also described the application of several heterogeneous, overlaying architectural styles. For example, core components, such as the framework parts for checking the conformance of SUAs and optimizing CDOs, are implemented according to a layered architecture for reasons of changeability. Consequently, the general dependency structure that results from the connectors linking the layer components is unidirectional as only components of higher layers are allowed to access components of lower layers. Furthermore, components of higher layers often require data structures from lower layers or depend on activities that were completed in lower layers. Hence, the corresponding core activities of the layer com-

[1] http://www.cloudmig.org

Figure 10.1. Logo of the tool CloudMIG Xpress

ponents are implemented in *CloudMIG Xpress* in the form of a workflow. For example, a CloudMIG user (cf. Section 6.3.3) first has to extract a KDM model (lowest layer). Then, elements of the KDM model can be assigned to on premise server nodes to build a status quo deployment model in the next layer `StatusQuoDeploymentModeling` that is located directly above the `KDMCodeModelExtraction` layer (cf. Figure 9.2). Figure 10.2 shows a screenshot of *CloudMIG Xpress* that depicts its start page and also illustrates the workflow concept. Activities are activated and deactivated based on met preconditions, i.e., whether specific models were already created. The concept is further described with an exemplary pass through that workflow in the user guide that comes with the tool *CloudMIG Xpress*.

This chapter utilizes and builds upon the following previously published work:

1. *Frey, Sören and van Hoorn, André and Jung, Reiner and Hasselbring, Wilhelm and Kiel, Benjamin, "MAMBA: A Measurement Architecture for Model-Based Analysis," Department of Computer Science, Kiel University, Germany, Technical Report, TR-1112, 2011.*
2. *Wulf, Christian and Frey, Sören and Hasselbring, Wilhelm, "A Three-Phase Approach to Efficiently Transform C# into KDM," Department of Computer Science, Kiel University, Germany, Technical Report, TR-1211, 2012.*

The chapter is structured as follows. Section 10.1 shows how a KDM discoverer for a specific programming language can be built, i.e., a plugin component that extracts a KDM model from source code. To illustrate the general approach, the section also describes the implemented KDM discoverers for Java and C#. The next Section 10.2 describes our MAMBA framework that is used for applying SMM measures. The framework and plugin components utilized for constraint validation are described in Section 10.3. Then,

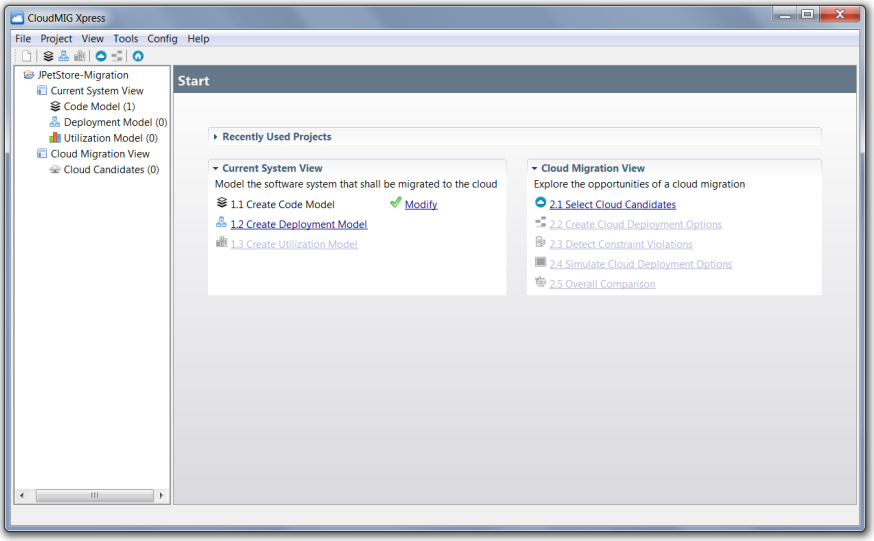


Figure 10.2. CloudMIG Xpress start page. A KDM code model was successfully extracted in this example. As this constitutes a precondition for creating a status quo deployment model according to the defined workflow, such a model can now be created (activated *Create Deployment Model* activity in the *Current System View*). For creating a utilization model, a status quo deployment model also has to be present (deactivated *Create Utilization Model* activity in the *Current System View*).

the implementation of our genetic algorithm CDOXplorer is detailed in Section 10.4, before Section 10.5 sums up this chapter.

10.1 KDM Model Extraction

As described in Section 9.2, the framework part of *CloudMIG Xpress* that controls the KDM extraction process is implemented by the `KDMCodeModelExtraction` component. The *KDM Discoverer Contributor* role can add KDM discoverer plugin components (cf. Section 6.3.3) that are managed by the `KDMCodeModelExtraction` component. The two KDM discoverer plugin com-

ponents JavaKDMDiscoverer and CSharpKDMDiscoverer exist that enable the extraction of KDM models from Java and C# source code (cf. Section 9.2), respectively. Figure 10.3 shows the basic design of these components. A KDM Discoverer Contributor has to create an Eclipse plugin that provides an implementation of the AbstractKDMDiscoverer class.

The AbstractKDMDiscoverer class defines several methods that enable the KDMCodeModelExtraction component to integrate the plugins into the overall KDM extraction process. For example, KDM discoverer plugins are initialized with the help of the initialize method that provides basic information regarding the KDM extraction process, such as the filesystem directories that contain source code artifacts. The AbstractKDMDiscoverer class inherits from the AbstractLanguageRelatedPlugin class. The latter is a base class of *CloudMIG Xpress* for all plugin components that depend on specific programming languages. The design of the JavaKDMDiscoverer and CSharpKDMDiscoverer components is briefly described in the following.

KDM Discoverer for Java-based Systems The class JavaKDMDiscoverer inherits from AbstractKDMDiscoverer and therefore enables the KDMCodeModelExtraction component to incorporate the plugin component in the KDM extraction process. The KDM discoverer builds on a component from Tietjens [2010] and uses the reverse engineering framework MoDisco 0.9. As described in Section 9.2, MoDisco can extract KDM models only from Eclipse projects. The class JavaProjectFactory enables the component to also extract KDM models from source code artifacts that are structured in simple filesystem directories. It creates a temporary Eclipse Java project and integrates a copy of the source code into that project. The class CloudMIGXpressDiscoverKDMCodeModelFromJavaProject inherits from the MoDisco class DiscoverKDMModelFromJavaProject. It enables the JavaKDMDiscoverer class to extract a KDM model from the temporary Eclipse Java project. When the extraction process finishes, the project is removed.

However, the KDM models produced by MoDisco lack some implementation details. Furthermore, the models extracted from Java bytecode—e.g., when no Java source code is present—exhibit limitations. In this context, Prinz [2012] addressed the extraction of extended KDM models from Java bytecode.

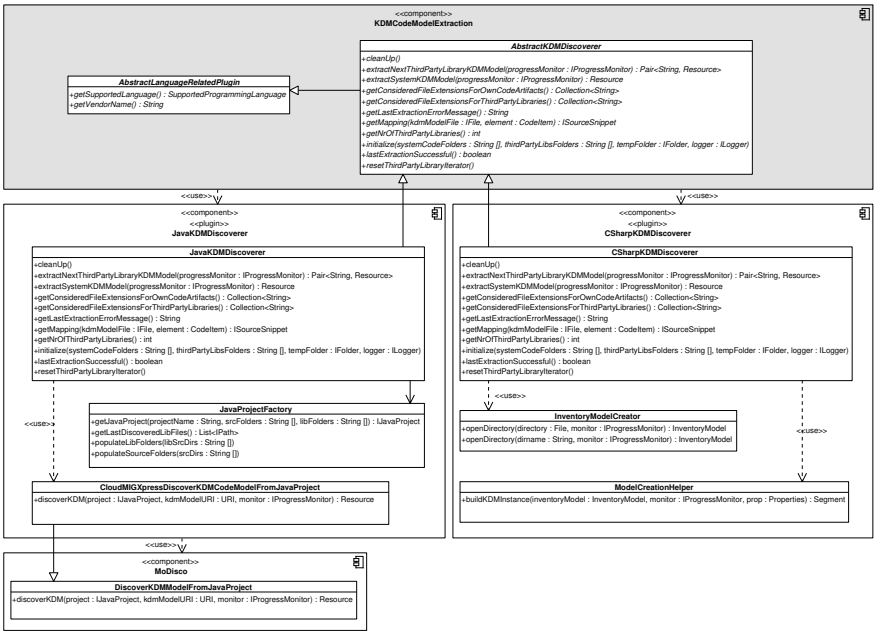


Figure 10.3. Basic design of KDM model extraction. The component colored gray provides the class `AbstractKDMDiscoverer` that has to be used by KDM discoverer plugins.

However, the corresponding component is not yet included in the publicly-available version of *CloudMIG Xpress*.

KDM Discoverer for C#-based Systems The `CSharpKDMDiscoverer` plugin component originates from Wulf [2012]. The contained `CSharpKDMDiscoverer` class inherits from `AbstractKDMDiscoverer` and therefore allows the plugin to be used by the `KDMCodeModelExtraction` component. Figure 10.3 shows basic classes. For example, the `ModelCreationHelper` class creates the KDM model with the help of its `buildKDMInstance` method. For parsing a system's C# source code files and to build the corresponding AST we use the parser generator ANother Tool for Language Recognition (ANTLR) [Parr and Quong 1995]. We developed a lexer and parser grammar on the basis of

Figure 10.4. The three phases used to transform C# to KDM (from [Wulf et al. 2012])

an already existing C# grammar [Wulf et al. 2012]. The KDM discoverer uses a Java-based transformation to map the nodes of the resulting AST to appropriate KDM elements. Thereby, we use the Eclipse Modeling Framework [Steinberg et al. 2009] (EMF)-based KDM specification to create KDM elements in Java.[2] For resolving external libraries within the transformation process, we use either C# decompilers or a separate C# program that utilizes the .NET Reflection API.

The actual creation of a KDM model from C# source code follows a three-phase transformation of C# to KDM [Wulf et al. 2012]. The corresponding three phases are depicted in Figure 10.4. In the first transformation phase *P1*, our transformation component parses all C# source code files of a given software system. It utilizes the AST that is produced while parsing to only transform namespaces and type definitions (e.g., classes, interfaces, and structs) with their corresponding modifiers and names. For instance, it intentionally does not transform any inheritance relations since this would require a complex and time-consuming look-up mechanism.

The second transformation phase *P2* is responsible for transforming member declarations and method definitions but without any member initializers and method bodies. The final third transformation phase *P3* is responsible for mapping C# statements, i.e., especially member initializers and method bodies are transformed. For further details regarding our three-phase approach to efficiently transform C# into KDM, we refer to Wulf et al. [2012].

[2]The KDM model provided by MoDisco is used.

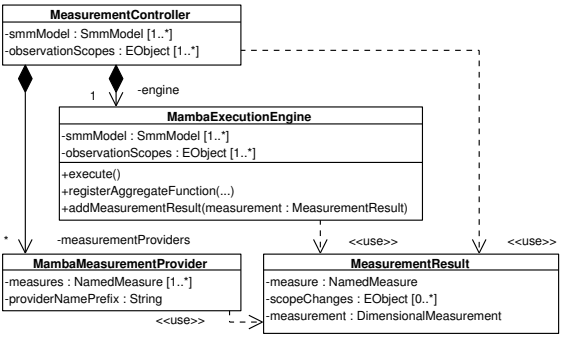


Figure 10.5. Design of the Mamba Execution Engine (MEE) (from [Frey et al. 2011])

10.2 MAMBA

As described in Section 3.3.3, the Structured Metrics Metamodel (SMM) can be used to model measures, observations, and measurement results, for instance. In the context of CloudMIG, it is used for modeling and validating generic constraints and to build workload profiles (see Sections 7.2.2 and 9.2, respectively). To actually apply measures that are defined with SMM—e.g., to count the modules of a software system with the SMM instance shown in Figure 3.12—we developed the Measurement Architecture for Model-Based Analysis (MAMBA) [Frey et al. 2011; 2012]. MAMBA builds upon the Metrics Execution Engine (MEE) we introduced in Frey et al. [2013a] and later renamed to MAMBA Execution Engine (MEE) [Frey et al. 2011]. In the following, the design of MEE and its means for executing open SMM models, as defined below, are described.

Design of the Mamba Execution Engine The core classes in the MEE design are a `MeasurementController`, the actual `MambaExecutionEngine`, as well as a (potentially empty) set of `MambaMeasurementProviders`. The UML class diagram in Figure 10.5 depicts these classes and their relationships. From given resource URIs, the `MeasurementController` loads SMM instances along with the Ecore [Steinberg et al. 2009] models which constitute the `Oberservation`'s `ObservationScope` (see Section 3.3.3)—e.g., KDM models

to be analyzed—, and inspects the given set of requested Measures. For each measurement run, the MeasurementController creates an Oberservation which is passed to an instance of the MambaExecutionEngine responsible for the execution of the SMM model, i.e., computing the Measurements for the Observation with respect to the requested Measures. SMM measures that play a specific role in the context of MAMBA are NamedMeasures. A NamedMeasure is a familiar measure that can be described unambiguously by solely stating its name.

We distinguish between the execution of *closed* and *open* SMM models. A closed SMM model contains no NamedMeasures and MEE can execute these models directly, without requiring any additional input. Details on the execution of closed models with MEE have been presented in Frey et al. [2013a]. For *open* SMM models, the Measurements corresponding to NamedMeasures are provided by so-called MambaMeasurementProviders. On instantiation—given an *open* SMM model—the MeasurementController looks up appropriate MambaMeasurementProviders by name prefixes matching among NamedMeasures (fully qualified *name* attribute) and the MambaMeasurementProvider's *providerNamePrefix*.

Execution of Open SMM Models MambaMeasurementProviders integrate with external tools for static and/or dynamic analysis by importing raw measurement data for the supported NamedMeasurements from the tools' output, and transforming this raw data into MeasurementResult objects which are delivered to the MeasurementController. Each of these MeasurementResult objects (see Figure 10.5) contains information on the observed NamedMeasure and a DimensionalMeasurement (cf. Section 3.3.3). The MeasurementController delegates these MeasurementResult objects to the MambaExecutionEngine which appropriately incorporates the measurement into the SMM model.

In continuous scenarios, an ObservationScope may evolve during the measurement process, e.g., adding components when discovering software architectures from incoming monitoring data. MambaMeasurementProviders indicate such changes by setting the *scopeChanges* in a MeasurementResult object. In this case, the MambaExecutionEngine needs to reprocess the SMM model before incorporating the new measurement.

For further details regarding the Measurement Architecture for Model-Based Analysis (MAMBA), we refer to Frey et al. [2011, 2012].

10.3 Constraint Validation

CloudMIG utilizes so-called constraint validators for checking the conformance of SUAs regarding specific cloud profiles (cf. Chapter 7). The constraint validators can be added to *CloudMIG Xpress* by CEC validation contributors (cf. Section 6.3.3) in the form of plugin components. For performing the constraint validation process, the tool architecture includes the corresponding component `ConstraintValidation` (cf. Section 9.2). It follows the MVC pattern such that detected CEC violations can easily be displayed in several, joint views. Figure 10.6 shows an example of detected CEC violations that are displayed in a graph-based view and also in a code tree view (right side). The source code elements can be augmented with icons that indicate found CEC violations. A color code regarding the nodes in the graph-based view indicates the severities of the CEC violations. Detailed descriptions of CEC violations are displayed in the view in the lower right corner of Figure 10.6 (*CEC Violations - Detail View*).

As described in Section 9.2, the plugin components `KDMConstraintValidationJava` and `KDMConstraintValidationCSharp` provide validators for CEC violations that require language-specific detection mechanisms. Common base classes for implementing those constraint validation plugins and also classes for the language-independent detection of CEC violations are provided by the component `KDMConstraintValidationCommon`. The basic design of those components is shown in Figure 10.7. Important classes of the component `ConstraintValidation`, such as `AbstractConstraintValidator` and `AbstractConstraintViolation`, were already explained in Section 7.3.2. For each CEC that can be detected with language-independent, static detection mechanisms and that is in general covered by *CloudMIG Xpress*, the `KDMConstraintValidationCommon` component provides corresponding constraint validators. For example, the class `TypesWhitelistConstraintValidator` checks the conformance regarding the `TypesWhitelistConstraint` and

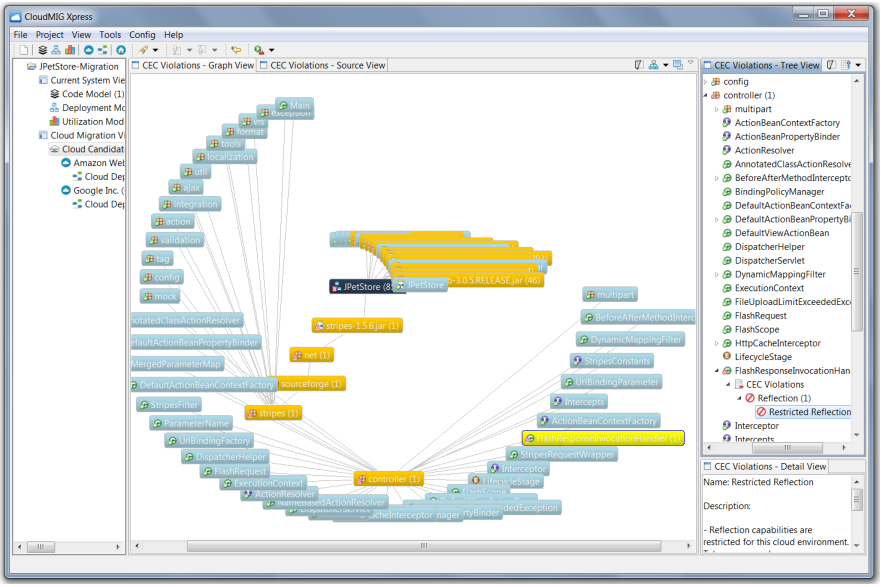


Figure 10.6. CEC violation visualization in CloudMIG Xpress

the `MethodCallConstraintValidator` can detect violations regarding the `MethodCallConstraint` (cf. Section 7.2).

Validators that perform the constraint validation on the basis of KDM models and do not employ SMM measures inherit from the `AbstractConstraintKDMValidatorBase` class. If SMM measures have to be applied, the validators inherit from the `AbstractConstraintSMMKDMValidatorBase` class. Both classes indirectly derive from `AbstractConstraintValidator`. They provide a convenient way for validators to manage the constraint validation process and to collect found constraint violations. For example, the latter is performed with the method `addConstraintViolation`. It stores the references to the KDM elements that raise the violations and allows all validators to subsequently (when the validation process finishes) report the constraint violations in the form of `ConstraintViolation` instances. Constraint validators for CECs that require specific language-dependent detection mecha-

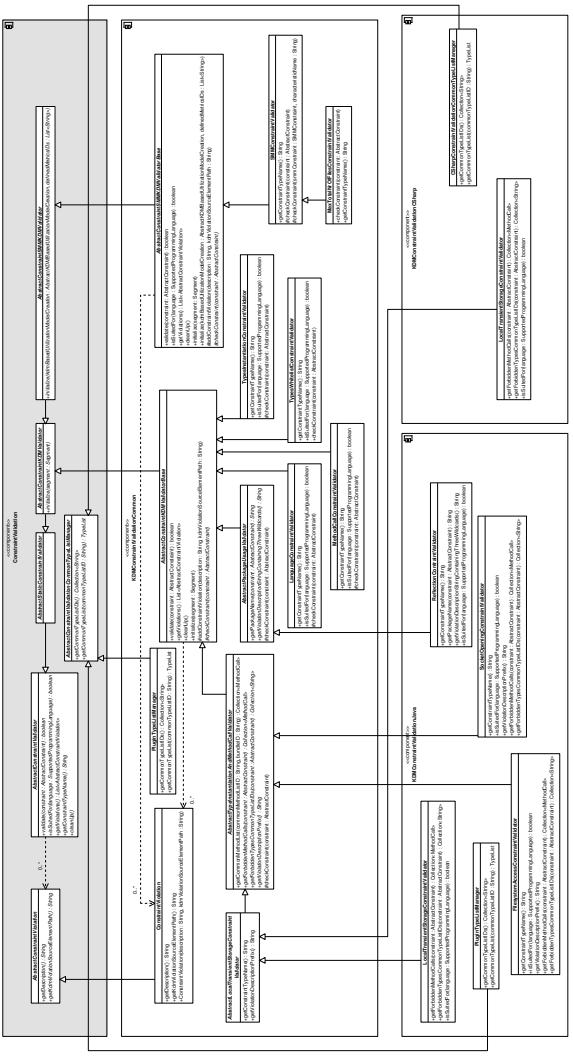


Figure 10.7. CloudMIG Xpress constraint validation design

nisms for Java and C# are included in the KDMConstraintValidationJava and KDMConstraintValidationCSharp components, respectively. These validators make use of the constraint validation base classes provided by KDMConstraintValidationCommon (cf. Figure 10.7).

For example, the class FilesystemAccessConstraintValidator in the KDMConstraintValidationJava component inherits from the AbstractTypeInstantiationAndMethodCallValidator class. The latter class provides mechanisms for validators that validate two general aspects: instantiation of specific types and calls to specific methods. In the case of the class FilesystemAccessConstraintValidator, the validator needs to detect both, the instantiation of types and the call to methods that imply accessing the filesystem. The general constraint validation process is already described in Section 7.3.1, for example, for loading, initializing, and utilizing the constraint validators. The actual CEC validation procedure performed by a constraint validator is determined by two characteristics C1 and C2. C1 describes an arbitrary number of CEC violation patterns that are searched in the KDM model. C2 determines the method for traversing the graph of interconnected KDM elements (cf. Section 7.3.1).

We consider the mentioned class FilesystemAccessConstraintValidator of the KDMConstraintValidationJava plugin as an example. The corresponding constraint validator has to detect elements in a KDM model that read from the filesystem or that write to the filesystem. As corresponding operations can only be detected by incorporating knowledge regarding the used programming language (cf. Section 7.1.3), the validator maintains lists of methods and types that (upon call and instantiation, respectively) access the filesystem. Hence, C1 comprises all fully qualified type names and method names included in these lists. Instead of recursively traversing the KDM code model graph, visiting each node, and checking for a matched pattern, the validator builds upon the AbstractTypeInstantiationAndMethodCallValidator class. This class uses third-party model querying capabilities and therefore delegates the graph traversal process (characteristic C2) to a model query engine. For example, a generic query for finding instantiations of types (e.g., via instances of the KDM class Creates from KDM's *Action* package) is then parameterized with the type names included in C1.

10.4 CDOXplorer

Our simulation-based genetic algorithm CDOXplorer is implemented in the component `CDOOptimization` (cf. Section 9.2). It builds upon the Java-based Opt4J framework for meta-heuristic optimization [Lukasiewycz et al. 2011]. *CloudMIG Xpress* uses Opt4J V. 2.6. Opt4J provides a set of common optimization algorithms, such as evolutionary algorithms and particle swarm optimization, and also aims at simplifying the implementation of arbitrary meta-heuristic optimization algorithms. Figure 10.8 shows the basic classes of CDOXplorer's implementation that allow the integration into the Opt4J optimization framework.

`CloudDeploymentOption` instances include the data structures that are used by CDOSim to simulate CDOs. When the simulation of a CDO finishes, CDOSim delivers objects of classes that implement the `ISimulationRun` and `ISimulationResult` interfaces. These objects are stored in a `ReconfigurationOption` object that is also contained in a `CloudDeploymentOption`. An `ISimulationResult` describes the actual objective values of a CDO resulting from a simulation run, i.e., its costs, response times, and number of SLA violations. *CloudMIG Xpress* uses the transformation described in Section 6.3.4 for transforming Ecore-based cloud profiles into KDM-compatible versions, i.e., into the DSL that is implemented with KDM. These KDM-compatible versions of the cloud profiles are then used as an input for CDOSim. As described in Section 4.1.2 in the context of population-based optimization methods, the data structure of individuals that is used during the breeding procedure is called *genotype*. For implementing the genotype of CDOs that can be employed by Opt4J, the class `CDOGenotype` is used that implements Opt4J's `Genotype` interface (cf. Figure 10.8).

Opt4J uses so-called *creators* to produce individuals. Hence, the class `CDOGenotypeCreator` implements Opt4J's `Creator` interface and creates objects of the `CDOGenotype` class. The data structure of individuals that is used for evaluating individuals, i.e., for applying a fitness function, is called *phenotype* (cf. Section 4.1.2). Hence, for enabling objects of `CDOGenotype` to be evaluated, they have to be transformed into the phenotype represen-

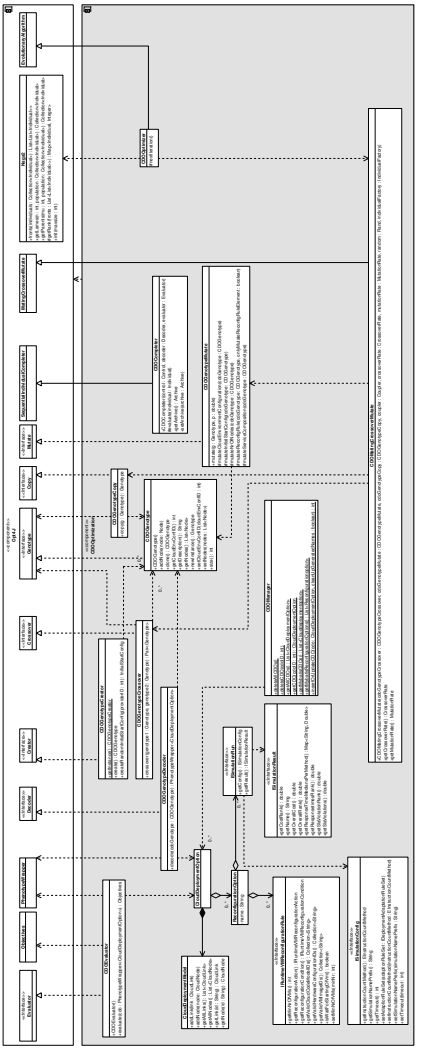


Figure 10.8. Basic design of CDOXplorer using the Opt4J framework

tation. Opt4J uses so-called *decoders* for those transformations and also provides several options for implementing phenotypes. The option utilized by *CloudMIG Xpress* uses Opt4J's generic class `PhenotypeWrapper` that allows to build phenotypes from existing classes. The class `CDOGenotypeDecoder` implements Opt4J's `Decoder` interface for transforming the genotype representation of CDOs (class `CDOGenotype`) into the phenotype representation of CDOs. This phenotype representation uses the generic `PhenotypeWrapper` class in conjunction with `CloudDeploymentOption` as type parameter (`PhenotypeWrapper<CloudDeploymentOption>`). That means, `CloudDeploymentOption` objects are basically used as phenotypes, as a `CloudDeploymentOption` includes the representation of a CDO that is needed for applying the fitness function, i.e., for simulating the CDO with CDOSim.

CDOXplorer's crossover and mutation operators (cf. Section 8.4) are implemented in the classes `CDOGenotypeCrossover` and `CDOGenotypeMutate`, respectively (see Figure 10.8). `CDOGenotypeCrossover` implements Opt4J's `Crossover` interface and `CDOGenotypeMutate` implements Opt4J's `Mutate` interface. The general structure of CDOXplorer follows a typical evolutionary algorithm, for example, by using populations of CDOs and corresponding crossover and mutation operators. Thus, a subclass called `CDOOptimizer` is built that inherits from Opt4J's class `EvolutionaryAlgorithm`. The actual mating process of `CDOOptimizer` that utilizes the crossover and mutation operations and that adapts the crossover and mutation rates (cf. Section 8.5) is controlled by the class `CDOMatingCrossoverMutate`. `CDOMatingCrossoverMutate` inherits from Opt4J's class `MatingCrossoverMutate`.

As described in Section 8.1.1, NSGA-II (cf. Section 4.2) is used for selecting appropriate pairs of parent individuals. Hence, `CDOOptimizer` utilizes Opt4J's class `Nsga2`. It is configured in a way such that the method `getParents` uses two tournament rounds for selecting pairs of parents, i.e., as mentioned in Section 8.1.1, a prolific individual has to be fitter than at least two others. For selecting the individuals that are transferred to a new generation, the NSGA-II procedure uses non-dominated sorting and crowding distance sorting (cf. Section 4.2). CDOXplorer also utilizes this NSGA-II procedure via Opt4J's class `Nsga2`. It is implemented in `Nsga2`'s method `getLames`.

10.5 Summary

This chapter describes implementation details of *CloudMIG Xpress*. *CloudMIG Xpress* provides tool support for the CloudMIG method and implements the tool architecture that is described in the previous Chapter 9. *CloudMIG Xpress* follows a workflow concept to guide the user through a migration planning process. The chapter elaborates on four central areas of the tool implementation and starts with the extraction of KDM models.

In the context of the CloudMIG method, KDM models play an essential role. They are extracted from existing system artifacts and form the basis for checking the conformance of SUAs regarding specific cloud profiles and also for optimizing the mapping of their components to cloud resources. Additional KDM discoverer plugins can be integrated by essentially providing an implementation of a single abstract class (`AbstractKDMDiscoverer`). KDM discoverer components can reuse and extend existing reverse engineering software. For example, our KDM discoverer for Java source code builds on the reverse engineering framework MoDisco.

Furthermore, the chapter describes the core components of our MAMBA framework. It allows to apply SMM-based measures, for example, to extracted KDM instances. The included Mamba Execution Engine (MEE) component computes SMM measures. Raw measurement data can be imported into SMM models with the help of `MambaMeasurementProviders`.

Implementation details regarding the conformance checking process are also described in this chapter. The conformance checking process uses so-called constraint validators to detect specific CEC violations. CEC violations that can be detected with the help of language-independent detection mechanisms (cf. Section 7.1.3) are implemented in the `KDMConstraintValidationCommon` component. For example, this component contains validators for detecting violations regarding `MethodCallConstraints`, `MaxTotalNrOfFilesConstraints`, and `TypesWhitelistConstraints`. Further constraint validators—language-dependent as well as language-independent—can build on several utility functions that are provided by the `KDMConstraintValidationCommon` component.

The chapter then describes the implementation of our simulation-based genetic algorithm CDOXplorer. The implementation utilizes the Java-based Opt4J framework for meta-heuristic optimization. The framework separates the genotype representation of individuals from their phenotype representation (cf. Section 4.1.2). Hence, both representations and a corresponding transformation are implemented. The latter transforms a genotype in a phenotype representation. The class `EvolutionaryAlgorithm` from Opt4J provides basic functionalities regarding arbitrary evolutionary algorithms and also allows to integrate the custom crossover and mutation operators that process CDOs during reproduction (cf. Section 8.4), for instance.

Further details regarding the tool implementation, example scenarios, and extension mechanisms can be found in the user guide that comes with the tool *CloudMIG Xpress*.

Chapter 11

Evaluation of the Conformance Checking Approach

CloudMIG's conformance checking approach aims at uncovering an SUA's CEC violations regarding a potential target cloud environment. This chapter describes the evaluation of the conformance checking approach to (1) show the applicability of modeling CECs and detecting corresponding violations with the help of the CloudMIG method and to (2) analyze its detection performance. *CloudMIG Xpress* provides tool support for the CloudMIG method and is employed in the three case studies CC-1,CC-2, and CC-3. The conformance checking case studies conduct experimental evaluations to address and detail the broadly stated research questions outlined in Section 5.6 that cover the detection of CEC violations.

In a first step, these high-level research questions are systematically refined to analyze the characteristics of CECs in common enterprise software, to examine the approach's applicability to different cloud environments and programming languages, and to quantify its precision and accuracy, for instance. An overview of important components to be used in the evaluation is depicted in Figure 11.1. The Goal Question Metric (GQM) method [Basili and Rombach 1988] is applied to structure the evaluation, to detail its goals, and to systematically derive fine-grained metrics that are used to measure and assess the activities and concepts involved in the conformance checking approach. The conducted experiments of the three case studies incorporate up to five Java-based systems (e.g., jForum and Ace Operator) as well as a C#-based library from our industrial partner, the large German bank HSH Nordbank AG. The experiments use cloud profiles of the public

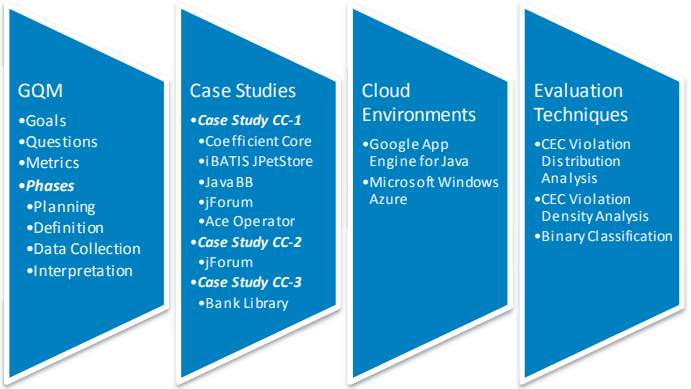


Figure 11.1. Important components of the conformance checking evaluation

cloud environments Google App Engine for Java and Microsoft Windows Azure. CEC violation distribution and density analyses are employed to (1) demonstrate the feasibility of the conformance checking approach and to (2) reveal characteristics of common enterprise software regarding the presence of CEC violations, for example, with respect to severity and clustering properties. Furthermore, binary classification is used to reason about the recognition capabilities of the conformance checking approach.

This chapter utilizes and builds upon the following previously published work:

1. *Frey, Sören and Hasselbring, Wilhelm and Schnoor, Benjamin, "Automatic Conformance Checking for Migrating Software Systems to Cloud Infrastructures and Platforms," in Journal of Software: Evolution and Process, Vol. 25, Nr. 10, pp. 1089–1115, 2013.*
2. *Fenner, Sören, "Migration of Software Systems to Platform as a Service based Cloud Environments," Diploma Thesis, Kiel University, Kiel, Germany, 2011.*
3. *Wulf, Christian, "Automatic Conformance Checking of C#-based Software Systems for Cloud Migration," Master's Thesis, Kiel University, Kiel, Germany, 2012.*

Please note that some of the experiments for evaluating the conformance checking approach are described in the theses 2. (cf. [Fenner 2011]) and 3. (cf. [Wulf 2012]) that are stated above. These experiments are included and highlighted again in the case studies CC-2, and CC-3, respectively.

The chapter is organized as follows. Section 11.1 describes the methodology used to evaluate the conformance checking approach. As a central component, this section overviews the GQM method and its four phases. The first *GQM planning* phase provides the basis for the evaluation in Section 11.2 and describes elementary prerequisites, such as the chosen applications and cloud environments involved in the three case studies. The *GQM definition* phase defines, among others, the goals, questions, metrics, and hypotheses regarding the results of the metrics application. This phase is described in Section 11.3. The measurement activities and the obtained results for assessing the conformance checking approach are described in Section 11.4 that covers the *GQM data collection* phase. Section 11.5 analyzes and interprets the obtained measurement results following the *GQM interpretation* phase. The corresponding analysis and also the overall evaluation approach are subject to certain threats to validity that are addressed in Section 11.6. Finally, Section 11.7 summarizes this chapter.

11.1 Methodology

This section overviews methods and techniques used to evaluate CloudMIG's conformance checking approach. The utilized GQM method is outlined in Section 11.1.1. An important technique that is used for quantifying the conformance checking approach's performance is binary classification. Section 11.1.2 describes binary classification to provide a basic understanding of this technique.

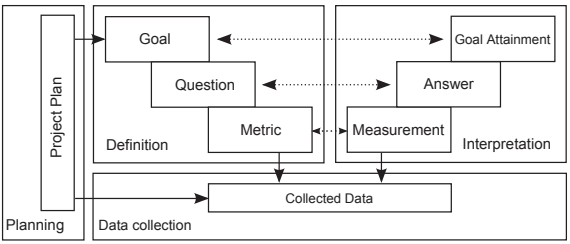


Figure 11.2. The four phases of the GQM method (based on [Solingen 1999])

11.1.1 GQM Method

The GQM method [Basili and Rombach 1988; Basili 1992; Van Latum et al. 1998; Solingen 1999] aims at facilitating software development to follow a goal-oriented paradigm. Various stakeholders pose varying or even conflicting requirements to the development, operation, and maintenance of a software system. Each stakeholder brings in an individual set of high-level, abstract requirements (goals) that have to be detailed and tracked to satisfy the stakeholder's needs. GQM is a structured approach for systematically planning, executing, and reviewing *measurement programs* to evaluate the achievement of those goals.

A central characteristic of the GQM approach is the derivation of questions that examine and detail the specified goals and aim for investigating all potential directions that can influence the goal attainment. These questions are then further detailed to fine-grained software metrics, such as Lack of Cohesion of Methods (LCOM) and Weighted Methods per Class (WMC) [Kan 2002], that are suited for quantifying the level of goal attainment. The GQM method is structured in the four phases *planning*, *definition*, *data collection*, and *interpretation* that are illustrated in Figure 11.2. These phases are briefly described below. For a more detailed introduction to GQM and its included phases, we refer to Solingen [1999].

Planning The planning phase initializes a measurement program, defines its scope, and sets the course for specifying and performing the measure-

ments and analyzing the corresponding measurement results. For example, one of the planning steps establishes the GQM team that leads the measurements and that is independent from a project team that develops an application of interest. Furthermore, an application project is selected and a project team is established. To document issues such as a schedule, management process, and a measurement program's characteristics like the previously identified application project, a project plan is created, too.

Definition The definition phase elicits the high-level goals of the measurement program, i.e., the goals that should be achieved by measuring and analyzing properties of a software system or software development process. GQM goal definition templates [Basili 1992] can be used to support the elicitation process. The collected goals are translated to metrics that are suited to quantify properties of interest with the help of questions that are used to explore and refine the goals. Goals, questions, and metrics have to be consistent and the questions have to be posed in a form that enables judging if a goal is attained in whole or in part. Finding an appropriate level of abstraction for the questions is essential [Solingen 1999]. On the one hand, questions that are too abstract may impede deriving adequate metrics that enable measuring a question's characteristics. On the other hand, questions that are too detailed may hamper deciding if a goal is actually achieved.

Besides the metrics that result from formalizing the questions, the GQM definition phase covers the definition of hypotheses regarding the measurement results that are considered to be likely. The hypotheses support judging the actual outcome of a measurement program. Three plans are produced by the GQM definition phase. The *GQM plan* documents the goals, questions, and metrics. The metrics and the measurement process are formally defined for data collection in the *measurement plan*, and the *analysis plan* defines how the measurements are analyzed and interpreted in the interpretation phase.

Data Collection The metrics are actually applied in the data collection phase. Tools can be used for automatic data collection or if this is not possible, manual data collection forms may be employed. The collected data has to be checked for validity and is stored in a so-called *metrics base*, for

example, a database, spreadsheet, or a simple directory. The data can be processed using three layers. The first *raw data layer* provides the raw data from the metrics base to one or more analysis tools, for example, spreadsheet- or statistical programs. The *processed data layer* enables processing this raw data, e.g., though formulas in a spreadsheet or scripts written for a statistical tool. Then, the *graph and table layer* provides auxiliary means that are suited for presenting the measurement results, such as graphs and tables.

Interpretation The interpretation phase aims for (1) answering the questions that were posed in the definition phase, (2) evaluating whether the measurement results match with the hypotheses, and (3) judging if the defined goals are attained. The interpretation process starts with updating the analysis tools with the latest raw data from the metrics base. The activities of the processed data layer and graph and table layer, that are defined above, are then executed the last time to produce the final measurement results. These results and the analysis plan from the definition phase are then used together for interpretation purposes. The GQM method also covers preparing, organizing, and holding a feedback session with all stakeholders involved in the measurement program to report the measurement results.

11.1.2 Binary Classification

Binary classification [Getoor and Taskar 2007] is a special case of statistical classification. Statistical classification [Michie et al. 1994; Banks et al. 2004] can be described as follows. Consider a set of observations O, where a single observation could be a measurement result, an event, or a part activation (see Section 7.1.1), for instance. For each $o \in O$, select a class c from a set of predefined classes C, $|C| > 1$ and assign o to c, i.e., build tuples (O, C).

As an example for statistical classification, consider the domain of autonomous vehicles and adaptive vehicle dynamics. A camera system may face the task of classifying the road environment according to the classes off-road, major/trunk road, motorway, or urban environment [Tang and Breckon 2011]. Hence, this task corresponds to a four-class road environment classification problem.

Binary classification [Getoor and Taskar 2007] is a special case of statistical classification having two classes, e.g., *yes* and *no*, often called *positives* and *negatives*. Conformance checking also performs such a binary classification. It analyzes part activations (observations) and for each part activation assigns one of the classes *raises CEC violation* (positive) or *raises no CEC violation* (negative).

In general, in the context of binary classification, an actually positive observation is often indicated with $+R$, $-R$ otherwise. If a $+R$ observation is predicted correctly, i.e., that the observed value actually corresponds to the positive class and the observation is classified correctly as being positive, it is called a *true positive* (TP). If it erroneously is predicted to be $-R$, it is called a *false negative* (FN). Likewise, if a $-R$ value is predicted correctly, it is called a *true negative* (TN), otherwise, it is named a *false positive* (FP). If an observation is predicted to be positive, it is denoted $+P$. In contrast, $-P$ is used if a membership of the negative class is predicted.

Figure 11.3 illustrates the possible combinations of an observation's actual and predicted classes in a binary contingency table [Powers 2011]. Cells that correspond to erroneous predictions are colored gray. The number of all real positive observations is denoted rp and is given by $rp = TP + FN$. Likewise, the number of all real negative observations is indicated by rn and is given by $rn = FP + TN$. As illustrated in Figure 11.3, $rp + rn$ gives 100% of the observations. The sum of all observations predicted to be positive (pp) and all observations predicted to be negative (pn) also gives 100% of the observations. In the following, several statistical measures for evaluating the performance of binary classification algorithms are described.

Classification Evaluation Measures The *precision* of a classification algorithm represents the amount of observations that were correctly classified as positives, i.e., the ratio of TP to all observations classified as positives (see Formula 11.1.1). The precision is also called *positive predictive value (PPV)*.

$$Precision = \frac{TP}{TP + FP} \tag{11.1.1}$$

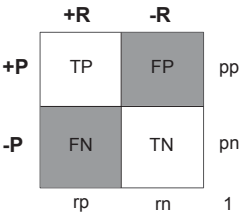


Figure 11.3. Binary contingency table (based on [Powers 2011]). Cells corresponding to erroneous predictions are colored gray.

In contrast, the *negative predictive value (NPV)* is given in Formula 11.1.2. It specifies the amount of the observations that were correctly classified as negatives, i.e., the ratio of TN to all observations that were classified as negatives.

$$NPV = \frac{TN}{TN + FN} \tag{11.1.2}$$

In comparison to precision, *recall* describes the ratio of the observations that were correctly detected as being positive by a classification algorithm to the number of all observations that are actually positive. The recall is also called *true positive rate (TPR)* or *sensitivity*. Formula 11.1.3 shows how recall can be calculated.

$$Recall = \frac{TP}{TP + FN} \tag{11.1.3}$$

As opposed to TPR, the *false positive rate (FPR)* is determined by the measure of Formula 11.1.4 that includes FP in the numerator instead of TP.

$$FPR = \frac{FP}{TP + FN} \tag{11.1.4}$$

The *specificity* (see Formula 11.1.5) is similar to the recall described above, but it rather describes the ratio of the observations that were correctly detected as being negative by a classification algorithm to the number of all

observations that are actually negative. Specificity is also called *true negative rate (TNR)*.

$$Specificity = \frac{TN}{TN + FP} \tag{11.1.5}$$

The *accuracy* determines the amount of all correctly classified observations. This measure is described in the Formula 11.1.6.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{11.1.6}$$

Another common measure is the *F-score* (also called *F-measure*) that integrates both precision and recall. It calculates the harmonic mean of precision and recall as given in Formula 11.1.7.

$$F\text{-}score = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{11.1.7}$$

Receiver Operating Characteristic A further means for reasoning about the performance of classification algorithms is the *receiver operating characteristic (ROC)* [Powers 2011]. In contrast to the measures described before, ROC is a graphical plot that combines TPR and FPR. ROC is usually used to support exploring different parameters of a classification algorithm as ROC illustrates the effects on TPR and FPR. TPR and FPR correspond to the two dimensions of the plot. An example is shown in Figure 11.4.

The marker (1) indicates a perfect classification algorithm that notices all observations that are actually positive and at the same time does not produce any false alarms (false positives). In contrast, classification algorithms producing results that correspond to the red diagonal in Figure 11.4 are useless, as their predictions are equally likely as a random guess. In general, only algorithms that produce classifications that lay above the diagonal, such as the one indicated by marker (1), provide an additional value over a random guess. Those producing classifications below the diagonal are not only useless, but are also misleading as $FPR > TPR$ in their case, as can be seen in the exampled marked with (3) in Figure 11.4.

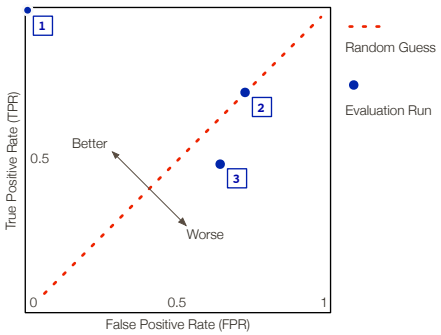


Figure 11.4. Receiver Operating Characteristic (ROC) example (cf. [Bielefeld 2012]). Marker (1) indicates an optimal classification. The classification of marker (2) corresponds to a random guess. Marker (3) shows a classification that is misleading as it produces results worse than random guess.

11.2 GQM Planning Phase

An important objective of GQM's planning phase is to determine the scope of a measurement program and to select an application project (cf. Section 11.1.1). In the context of the evaluation covered in this chapter, the scope is given by CloudMIG's conformance checking approach. As described in Chapter 10, this approach is implemented in the tool *CloudMIG Xpress*. Hence, the experimental evaluation of the conformance checking approach builds upon analyses that examine the tool *CloudMIG Xpress*, that constitutes the object under measurement. As a consequence, the combination of *CloudMIG Xpress* with the further, elementary components used in the case studies CC-1, CC2-2, and CC-3, such as the set of examined open source software systems and specific cloud environments, forms a single, compound application project in the sense of GQM.

Furthermore, the project team corresponds to the developers of *CloudMIG Xpress* (see [CloudMIG Xpress Project 2012] for a list). Besides the project team, GQM involves the GQM team that plans, applies, and analyzes measurements. For the analyses covered in this chapter, three GQM team members successively contribute different parts of the measurements

that are used in the corresponding case studies CC-1 [Frey et al. 2013a], CC-2 [Fenner 2011], and CC-3 [Wulf 2012] (see Section 11.2).

This section describes the above-mentioned application project that is studied in the further phases of GQM for evaluating the conformance checking approach. Section 11.2.1 gives an overview of the conducted case studies. Section 11.2.2 presents the included applications that are analyzed regarding the exhibition of CEC violations. Finally, Section 11.2.3 describes the involved cloud environments that were used to build the cloud profiles along with the contained CEC specifications.

11.2.1 Case Studies Overview

The case studies CC-1, CC-2, and CC-3 all follow the general approach of extracting a KDM representation of one or more software systems and analyzing characteristics of raised CEC violations regarding a particular cloud environment. In general, the case studies all comprise the following common steps.

General Process of Case Studies CC-1, CC-2, and CC-3

1. Extraction of KDM models from s software systems that include instances of KDM's source, code, and action models (cf. Section 3.3.2)

2. Modeling of a cloud profile along with the corresponding CECs (CC-2 reuses the cloud profile from CC-1)

3. Definition of g goals and q questions according to GQM that analyze specific characteristics regarding raised CEC violations

4. Derivation of m metrics from the q questions

5. Execution of *CloudMIG Xpress*' CEC violation detection approach

Table 11.1. Overview of the conformance checking evaluation case studies

Case Study	Applications	Cloud Environment
CC-1	Coefficient Core, iBATIS JPetStore, JavaBB, jForum, Ace Operator	Google App Engine for Java
CC-2	jForum	Google App Engine for Java
CC-3	Bank Library	Microsoft Windows Azure (Virtual Machine Role)

6. Application of the defined m metrics upon the detected CEC violations

7. Analysis of the measurement results to investigate the CEC violations' characteristics of interest and assess attainment of defined g goals

Basic information regarding the different case studies is described in the following with reference to Table 11.1 that lists the examined applications and cloud environments.

Case Study CC-1 CC-1 constitutes the primary case study for evaluating CloudMIG's conformance checking approach. It covers the five Java-based open source applications Coefficient Core, iBATIS JPetStore, JavaBB, jForum, and Ace Operator that are described in greater detail in Section 11.2.2. CC-1 employs the cloud environment Google App Engine for Java (cf. Section 11.2.3) to show the applicability of the conformance checking approach and to explore fundamental conformance properties of common enterprise systems that should be migrated to a PaaS-based cloud. The quantities, types, and severities of detected CEC violations are analyzed to reveal common characteristics, inherent CEC violation distributions, and to investigate the prioritization function `prio` (cf. Section 7.4.2), for instance.

Case Study CC-2 CC-2 reuses the cloud profile created in the context of CC-1, i.e., it analyzes CECs that are imposed by the cloud environment Google App Engine for Java. CC-2 focuses on quantifying the performance of CloudMIG's conformance checking approach by using techniques of binary classification (see Section 11.1.2). As the structured assessment of the detection quality cannot be automated and rather has to be processed

manually in large parts, case study CC-2 restricts the evaluation procedure to the single Java-based application jForum.

Case Study CC-3 CC-3 contributes two further aspects to the evaluation of CloudMIG's conformance checking approach: (1) It examines a closed source software library from our industrial partner HSH Nordbank AG, to demonstrate the applicability of the conformance checking approach in an industrial context. (2) As the bank library is written in C#, CC-3 shows that the software architecture described in Chapter 9 can be smoothly extended with an additional KDM discoverer plugin for C# and a corresponding CEC validator plugin that addresses language-specific detection mechanisms.

11.2.2 Applications

Basic characteristics of the applications analyzed in the context of the conformance checking evaluation case studies are presented in the Tables 11.2 and 11.3. The applications are given the numbers 1-6 (`App 1 - App 6`). The bank library (`App 6`) used in the case study CC-3 is, as described before, the only application that is not available as Open Source Software (OSS). The applications are briefly described in the following.

App 1 - Coefficient Core Coefficient is a Java-based collaboration platform that supports teams to carry out projects, for example, software development projects. It can run in Java EE or web application containers and is built in a modular structure, i.e., it consists of modules. Coefficient's core package (`App 1`), that is analyzed in the context of conformance checking case study CC-1, mainly comprises the *project* module that provides the basic functionality for managing generic projects. For example, it enables specifying a project's structure, managing project versions, and changing the workflow state of project items. Further modules, such as an issue tracker or a file upload module, are excluded from `App 1`.

App 2 - iBATIS JPetStore iBATIS JPetStore is a Java-based web application for shopping pets like fishes, dogs, and cats. It provides several features of a typical web shop, such as a shopping cart, product detail pages, or an

Table 11.2. Applications analyzed in the case studies - basic characteristics I

App	Name	Language (main)	OSS	URL
1	Coefficient Core 0.9.6	Java	Yes	http://coefficient.sourceforge.net/
2	iBATIS JPetStore 4.0.5	Java	Yes	http://ibatisjpetstore.sourceforge.net/
3	JavaBB 0.99	Java	Yes	http://www.javabb.org/
4	jForum 2.1.9	Java	Yes	http://jforum.net/
5	Ace Operator 1.7.0	Java	Yes	http://www.quik-j.com/ace_operator.htm
6	Bank Library	C#	No	Not publicly available

Table 11.3. Applications analyzed in the case studies - basic characteristics II

App	Domain	#Classes (w/o libs)	#Libraries	LOC (w/o libs)
1	Collaboration platform	131	41	11,862
2	Pet store	43	12	2,132
3	Forum software	256	43	12,239
4	Forum software	316	30	29,563
5	Live support	556	26	69,516
6	Financial product assessment	1,043	0	170,656

account management. iBATIS JPetStore traces back to an example pet store web application built by Sun Microsystems Inc. (acquired by the Oracle Corporation in 2010) that was intended to demonstrate the capabilities of the J2EE platform (later renamed to Java EE [Goncalves 2010]). The re-implementation of Sun's original pet store application uses the iBATIS open source persistence framework that was later renamed to mybatis.[1] iBATIS JPetStore was built to demonstrate the efficiency of using the J2EE platform in conjunction with available open source projects in comparison to .NET, a competing software platform developed by the Microsoft Corporation (e.g., see [Begin 2002]). Considering the Lines of Code (LOC) metric, `App 2` is the smallest application used in the conformance checking case studies (see Table 11.3).

App 3 - JavaBB JavaBB is a Java-based server application that utilizes technologies such as Spring [Walls 2011] and Hibernate [Bauer and King 2006] to build an open source forum software for the web. In several areas, JavaBB follows the example of phpBB[2] that provides a similar Internet

[1] http://www.mybatis.org

[2] http://www.phpbb.com

forum software written in PHP. Using JavaBB, forum users can create, cite, and read forum posts, for instance. JavaBB provides several further features typical for a web-based forum. For example, the creation of categories that help to structure the posts, paging, the management of word filters, and user authentication and authorization features.

App 4 - jForum This application comes from the same domain as `App 3` as it also implements a web-based forum software. It also employs Java as its main programming language and provides, of course, several similar features, such as creating messages, previewing posts, adding file attachments to the messages, or locking and unlocking topics as a moderator. However, jForum is nearly 2.5 times larger than JavaBB when considering LOC (see Table 11.3). As an advantage of their software, the authors of jForum state (on the project web page) that their forum was optimized in terms of performance, for example, through employing dedicated caching mechanisms to buffer frequently accessed data.

App 5 - Ace Operator Ace Operator allows web page owners to provide live support to visitors of their web page. Visitors can click a button in their desktop or mobile browser to initiate a chat session with an operator. Ace Operator allows maintaining operator queues, exchanging multimedia content during a chat session, transferring a chat session to another operator, or bringing in additional operators to an ongoing chat session, for instance.

App 6 - Bank Library The bank library is developed by our industrial partner HSH Nordbank AG. The library provides assessment and risk control of financial products. It is implemented in C#, consists of 939 files, and builds on the .NET framework 2.0. The bank library is deployed to online trading and batch processing systems. Users can access the library via a front-end that builds on the Microsoft Excel spreadsheet software.

11.2.3 Cloud Environments

The conformance checking case studies incorporate the cloud environments Google App Engine and Microsoft Windows Azure. They are briefly described below.

Google App Engine Google App Engine[3] (GAE) is a public PaaS-based cloud environment from Google Inc. It does not require to manage VMs, but rather provides a hosted software stack that allows deployed applications to scale transparently. At the time of writing, it offers the following four distinct environments that are intended for applications written in the corresponding programming languages:

- Go runtime environment
- Java runtime environment
- PHP runtime environment
- Python runtime environment

Despite of its name, the Java runtime environment also allows to deploy applications that are not implemented in Java, but use a JVM-compatible language, such as Scala. The case studies CC-1 and CC-2 specifically investigate CECs that are imposed by the Java runtime environment of GAE, as they analyze Java-based applications. Therefore, Google App Engine for Java (GAE for Java) is described in the following.

GAE for Java provides a servlet environment to guest applications. That means, software that should be deployed to GAE can utilize the standard Java servlet interface [Hunter and Crawford 2001] and a Java 7 JVM. Several well-known Java technologies are available and can be used by client applications, for example, JDO, JPA, JavaMail, and JCache. The GAE documentation itself names the offered servlet environment a *sandbox environment* that poses several restrictions, such as preventing access to certain JRE classes. Those documented restrictions form the basis for later specifying CECs as a part of GAE for Java's cloud profile. The documentation justifies those restrictions with security considerations and the enablement of the provided, transparent scalability, for instance.[4]

For storing data persistently, GAE for Java offers several different mechanisms, e.g., the schemaless object datastore can be used that follows a distributed, scalable architecture. It automatically distributes data according

[3] https://developers.google.com/appengine/

[4] https://developers.google.com/appengine/docs/java/overview/

to a varying number of write operations, for instance. Furthermore, Google also provides support for a relational database as well as a BLOB store. There also exist a number of additional services. For example, data access can be accelerated by incorporating the memcache service that provides a distributed in-memory data cache.

Microsoft Windows Azure Windows Azure[5] is a public cloud environment provided by the Microsoft Corporation that offers PaaS as well as IaaS-based services. Windows Azure provides computing resources according to three different roles: Web role, Worker role, and VM role. Despite all roles providing VMs, only the VM role corresponds to an IaaS-based service, whereas the Web and Worker roles correspond to PaaS-based services. The three roles are described in the following.

- **Web role:** Web applications that require Microsoft's web server Internet Information Services (IIS), for example, to deliver web pages that use the ASP.NET technology [MacDonald and Freeman 2010], can utilize the Web role. The provided VM includes an installed IIS, but the Web role relieves developers from maintaining the complete underlying software stack. It allows to simply deploy a web page to the VM and manages, distributes, and scales the resources automatically. The Web role supports programming languages compatible with the .NET framework, for example, C# and F#. However, other programming languages can be used as well, e.g., Java, Ruby, or PHP.
- **Worker role:** Applications that use the .NET framework but do not require the IIS web server, such as batch and background tasks, can employ the Worker role. As with the Web role, Microsoft relieves developers from maintaining the underlying software stack of the VMs, e.g., the operating system and .NET framework installed in the VM are updated automatically. Like the Web role, the Worker role also allows to run programs written in arbitrary programming languages. However, as their runtime environments have to be installed manually, the Worker role (and also the Web role) might not be as compelling for those programs as it is for applications utilizing the .NET platform.

[5] http://www.windowsazure.com

- **VM role:** The VM role allows to install arbitrary operating systems and applications. However, the installed software stack has to be maintained manually by a user of the VM role.

Windows Azure offers several compute instance sizes that can be used for starting VMs. For example, at the time of writing, a large compute instance provides 4x 1.6GHz CPU cores, 7 GB of memory, 850 GB local storage disk space, and 400 Mbps allocated bandwidth. VMs can be started in several data centers, e.g., located in the USA, Singapore, or Ireland. Windows Azure also offers relational databases through their SQL Azure service.

11.3 GQM Definition Phase

Based on the general conformance checking evaluation objectives, case studies, applications, and utilized cloud environments sketched in the GQM planning phase (cf. Section 11.2), this section derives the evaluation goals, questions, and metrics according to GQM's definition phase. The GQM models are specified using GQM goal definition templates [Solingen 1999]. Please note that the research questions denoted in Section 5.6 (WP6 - Evaluation) were used to approach the problem of adequately evaluating this thesis' contributions from a high-level point of view. These research questions are transformed to GQM goals in this section, as both concepts address the evaluation on a similar level of abstraction. Therefore, the underlying research questions from Section 5.6 are stated when describing the corresponding GQM goals in the following.

This section is structured as follows. The GQM goals, questions, and metrics are described separately for each of the three case studies CC-1, CC-2, and CC-3 in the Sections 11.3.1, 11.3.2, and 11.3.3, respectively.

11.3.1 Case Study CC-1

Figure 11.5 shows an overview on CC-1's GQM model, i.e., an outline covering CC-1's goals, questions, and metrics. CC-1 pursues the two goals *G1* and *G2*. In the following, these goals are described with the help of GQM goal definition templates [Solingen 1999].

G1: Show applicability of conformance checking approach

Goal	**G1**	
	Purpose Issue Object (process) Viewpoint	Show applicability of conformance checking approach from a CloudMIG user viewpoint
Question	**Q1.1**	Does the overall amount of CEC violations justify the usage of an automatic detection approach?
Metrics	**M1** **M2**	Amount of the applications' classes that cause CEC violations Densities of CEC violations per application
Question	**Q1.2**	Considering a manual detection approach, could CEC violations be easily spotted in individual classes?
Metric	**M3**	Densities of CEC violations of the applications' classes
Question	**Q1.3**	Does `prio` narrow the focus to particularly critical classes?
Metrics	**M4** **M5**	Distribution of the applications' CEC violation types Yielded partitioning when applying `prio` to detected CEC violations

The goal *G1* aims at validating that CloudMIG's conformance checking approach generally provides adequate means for detecting CEC violations in common enterprise software. This goal addresses the high-level research question Q6.1 from Section 5.6 with regard to the cloud environment GAE.[6] To show that the goal *G1* is reached by the conformance checking approach, three corresponding aspects are examined.

First, evidence is provided that common enterprise software in conjunction with well-known, popular cloud environments can exhibit a great amount

[6]To fully cover the high-level research question Q6.1 from Section 5.6, a further cloud environment is examined in the case study CC-3.

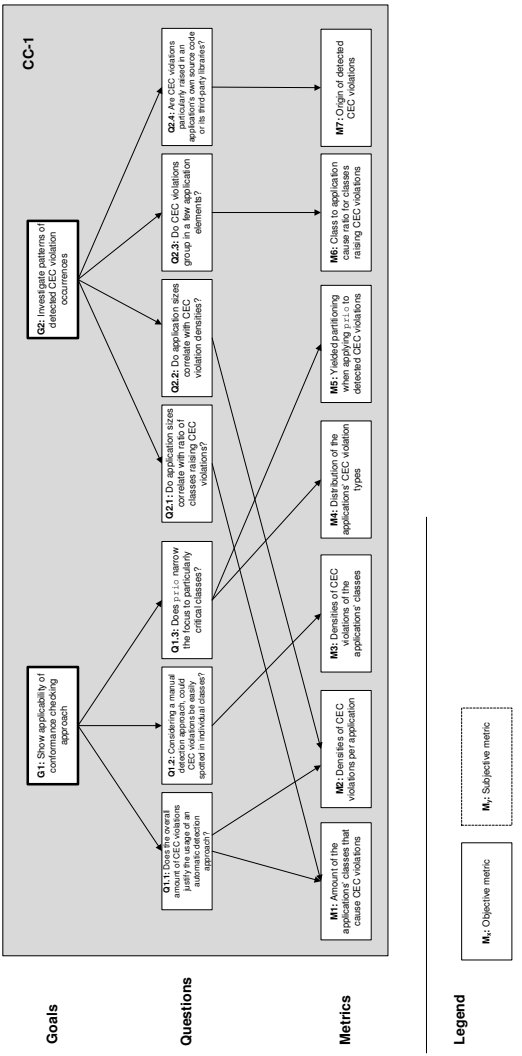


Figure 11.5. Overview on case study CC-1's GQM model

of CEC violations and that an automatic detection approach can simplify the migration to those cloud environments (*Q1.1*). Second, it has to be shown that, in contrast to an automatic detection mechanism, a manual detection approach in turn often cannot easily spot CEC violations. In fact, this is often cumbersome considering even limited chunks of a system, e.g., classes (*Q1.2*). Third, the applicability of `prio` (cf. Section 7.4.2) also has to be demonstrated as it constitutes an important auxiliary means of CloudMIG's conformance checking approach (*Q1.3*).

G2: Investigate patterns of detected CEC violation occurrences

Goal	**G2**	
	Purpose	Investigate
	Issue	patterns of
	Object (process)	detected CEC violation occurrences
	Viewpoint	from a CloudMIG user viewpoint
Question	**Q2.1**	Do application sizes correlate with ratio of classes raising CEC violations?
Metric	**M1**	Amount of the applications' classes that cause CEC violations
Question	**Q2.2**	Do application sizes correlate with CEC violation densities?
Metric	**M2**	Densities of CEC violations per application
Question	**Q2.3**	Do CEC violations group in a few application elements?
Metric	**M6**	Class to application cause ratio for classes raising CEC violations
Question	**Q2.4**	Are CEC violations particularly raised in an application's own source code or its third-party libraries?
Metric	**M7**	Origin of detected CEC violations

To facilitate a systematic detection of CEC violations and support reasoning about general, recurring characteristics of CEC violations in common enterprise software, goal *G2* aims to investigate patterns of detected CEC violation occurrences.[7] The following questions *Q2.1-Q2.4* are examined in this regard.

[7] *G2* addresses the high-level research question Q6.3 from Section 5.6. To fully cover Q6.3, the precision of the conformance checking approach is analyzed in the case study CC-2.

Larger applications, e.g., considering the size metric LOC, might usually likely exhibit more CEC violations than smaller applications. Basically, checking the conformance of a software system S is defined as searching its parts $P, \bigcup_{p \in P} p = S$ for elements that raise CEC violations (cf. Section 7.1.1). Consequently, it is fair to suppose that a higher value of $|P|$ by tendency implies a higher number of CEC violations.[8] On this basis, *Q2.1* investigates whether application sizes not only correlate with the number, but also with the ratio of classes raising CEC violations, i.e., if the ratio changes linearly or disproportionally.

A further related aspect that is analyzed by *G2* concerns the connection between an application's size and the encountered CEC violation densities. A corresponding possible correlation is investigated by *Q2.2*. In comparison, *Q2.3* examines whether CECs tend to group in certain elements of an application. The last question that analyzes general characteristics of CEC violation occurrences is *Q2.4*. It aims to determine whether CEC violations are particularly raised by an application's own source code or by code provided by third-party libraries that are used by the application.

Metrics GQM distinguishes *objective* and *subjective metrics* [Solingen 1999]. In contrast to objective metrics (e.g., LOC), the measurement of subjective metrics involves the assessment by a human (e.g., whether an algorithm is elegant). The case study CC-1 employs merely the objective metrics *M1-M7* (cf. Figure 11.5). These metrics are described in the following.

- **[M1] Amount of the applications' classes that cause CEC violations:** Determines the percentage of the applications' classes that cause at least one CEC violation.
- **[M2] Densities of CEC violations per application:** A CEC violation density determines how many CEC violations an application raises in relation to its size. This enables comparing the amount of raised CEC violations among applications. *M2* measures the number of CEC violations of an application per 1,000 LOC.

[8] We will validate this assumption in the context of *Q2.1*.

- **[M3] Densities of CEC violations of the applications' classes:** All CEC violation densities of all applications per class. To enable comparing these densities among applications and among classes, they are normalized to 100 LOC.[9]
- **[M4] Distribution of the applications' CEC violation types:** Determines the ratio of all CEC violation types per application.
- **[M5] Yielded partitioning when applying `prio` to detected CEC violations:** Calculates $prio(c)$ (see Section 7.4.2) for each class c of each application.
- **[M6] Class to application cause ratio for classes raising CEC violations:** Examines all classes of the applications that raise CEC violations. For each of those classes, *M6* calculates the ratio of the number of CEC violations the class raises regarding the total number of CEC violations raised by the application that includes this class.
- **[M7] Origin of detected CEC violations:** Determines for each application and for each raised CEC violation, whether the CEC violation is caused by the application's own source code or by source code contributed by third-party libraries that are used by the specific application.

11.3.2 Case Study CC-2

Figure 11.6 illustrates CC-2's GQM model. CC-2 addresses the goal *G3* that is described with the help of a GQM goal definition template below.

[9] As single classes usually contribute only a limited number of LOC to an application's overall size, *M3* is normalized to 100 LOC instead to *M2*'s normalization factor (1,000 LOC).

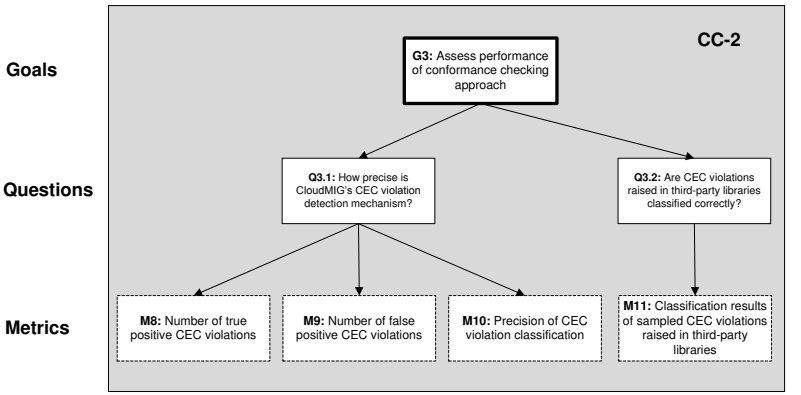


Figure 11.6. Overview on case study CC-2's GQM model

G3: Assess performance of conformance checking approach

Goal	**G3**	
	Purpose	Assess
	Issue	performance of
	Object (process)	conformance checking approach
	Viewpoint	from a CloudMIG user viewpoint
Question	**Q3.1**	How precise is CloudMIG's CEC violation detection mechanism?
Metrics	**M8**	Number of true positive CEC violations
	M9	Number of false positive CEC violations
	M10	Precision of CEC violation classification
Question	**Q3.2**	Are CEC violations raised in third-party libraries classified correctly?
Metric	**M11**	Classification results of sampled CEC violations raised in third-party libraries

G3 examines the quality of CloudMIG's conformance checking approach, i.e., it assesses its performance.[10] The goal *G3* covers two major quality

[10] *G3* addresses the high-level research question Q6.3 from Section 5.6. To fully cover Q6.3, characteristics of CEC violation occurrences in common enterprise software are examined in the case study CC-1.

aspects that are addressed by the GQM questions *Q3.1* and *Q3.2*. First, *Q3.1* analyzes the precision of CloudMIG's CEC violation detection mechanism with the help of binary classification (cf. Section 11.1.2). Second, *G3* examines CloudMIG's detection quality with a specific focus on CEC violations that are raised by third-party libraries (*Q3.2*).

Metrics Binary classification is used to assess the performance of the conformance checking approach. Basically, detected CEC violations have to be inspected manually to distinguish true positives from false positives, i.e., to decide whether a reported CEC violation actually constitutes a correctly identified CEC violation. It is not possible to automate this process and therefore, the metrics *M8-M11* relevant for answering *Q3.1* and *Q3.2* are all subjective metrics (cf. Section 11.3.1). These metrics are described below.

- **[M8] Number of true positive CEC violations:** From a set of reported CEC violations, the number of those CEC violations is determined by manual inspection that actually constitute CEC violations.
- **[M9] Number of false positive CEC violations:** From a set of reported CEC violations, the number of those CEC violations is determined by manual inspection that actually do not constitute CEC violations, i.e., the corresponding reports are false alarms.
- **[M10] Precision of CEC violation classification:** Given the number of true positive and false positive CEC violations from *M8* and *M9*, respectively, *M10* calculates the precision of the conformance checking approach (cf. Formula 11.1.1 in Section 11.1.2).
- **[M11] Classification results of sampled CEC violations raised in third-party libraries:** Sample CEC violations that are raised by an application's used third-party libraries are inspected manually. *M11* checks whether the samples constitute true positives or false positives.

It should be noted that CC-2 does not investigate true negatives and false negatives. As there is no way to automate the assessment of CEC violation discoveries, judging whether CEC violations are erroneously not discovered is not possible, as all parts (cf. Section 7.1.1) of the application would have

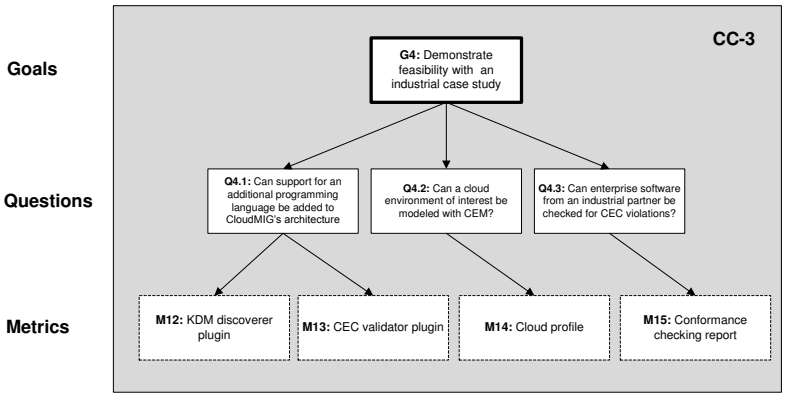


Figure 11.7. Overview on case study CC-3's GQM model

to be inspected and tested manually. That means, for each part, a test case would have to be built that deploys and activates the part in the context of the GAE sandbox environment and checks whether a CEC violation occurs. As the `App4` consists of tens of thousands parts, evaluating true negatives and false negatives cannot be accomplished in the context of CC-2.

11.3.3 Case Study CC-3

Figure 11.7 gives an overview on CC-3's GQM model. CC-3 addresses the goal *G4* that aims at demonstrating the feasibility of the conformance checking approach with the help of an industrial case study. CC-3 employs a library from our industrial partner HSH Nordbank AG that is written in C#, as well as the cloud environment Microsoft Windows Azure (cf. Section 11.2.1). Hence, *G4* is in line with the high-level research questions Q6.1, Q6.2, and Q6.5 from Section 5.6.

The goal *G4* is described with a GQM goal definition template in the following.

G4: Demonstrate feasibility with an industrial case study

Goal	**G4**	
	Purpose Issue Object (process) Viewpoint	Demonstrate feasibility with an industrial case study from a CloudMIG user viewpoint
Question	**Q4.1**	Can support for an additional programming language be added to CloudMIG's architecture
Metrics	**M12** **M13**	KDM discoverer plugin CEC validator plugin
Question	**Q4.2**	Can a cloud environment of interest be modeled with CEM?
Metric	**M14**	Cloud profile
Question	**Q4.3**	Can enterprise software from an industrial partner be checked for CEC violations?
Metric	**M15**	Conformance checking report

To evaluate the feasibility of CloudMIG's conformance checking approach for checking the library from the HSH Nordbank AG for CEC violations regarding the VM role of the cloud environment Microsoft Windows Azure (cf. Section 11.2.3), three main aspects are addressed with the help of the corresponding GQM questions *Q4.1-4.3*.

The bank library is written in C#. Hence, it had to be possible to create corresponding language-dependent artifacts—i.e., a KDM discoverer and a CEC validator plugin—that allow to examine the bank library for CEC violations with the help of CloudMIG (*Q4.1*). Furthermore, CEM has to be used to model a cloud profile for the VM role of Microsoft Windows Azure that includes its basic resources and the definition of CECs, for instance. Therefore, *Q4.2* analyzes the applicability of CEM to model this cloud environment. *Q4.3* subsequently examines CloudMIG's capabilities to use the cloud profile, KDM discoverer, and CEC validator to check the conformance of enterprise software in an industrial context.

Metrics The GQM questions *Q4.1-Q4.3* evaluate CloudMIG's means that allow integrating artifacts into the tool architecture for enabling the approach

to detect CEC violations regarding enterprise software from our industrial partner. Hence, the GQM metrics *M12-M15* denote these artifacts.

- **[M12] KDM discoverer plugin:** The KDM discoverer that extracts a KDM model from C#-based source code.
- **[M13] CEC validator plugin:** The CEC validator that provides language-dependent CEC violation detection mechanisms for C#.
- **[M14] Cloud profile:** The cloud profile that describes the VM role of Microsoft Windows Azure.
- **[M15] Conformance checking report:** Discovered CEC violations of the C#-based bank library regarding Microsoft Windows Azure's VM role.

11.4 GQM Data Collection Phase

This section describes the GQM data collection phase in the context of the conformance checking evaluation. The process and important artifacts and tools utilized to gather the necessary data for applying the previously specified GQM metrics (cf. Section 11.3) are described for CC-1, CC-2, and CC-3 in the following Sections 11.4.1, 11.4.2, and 11.4.3, respectively.

11.4.1 Case Study CC-1

The case study CC-1 models a cloud profile for GAE for Java V. 1.3.6. Despite that not all CECs are documented extensively, the described restrictions form a sufficiently well-suited basis for modeling the related CECs. The information concerning functionality, structure, and sandbox restrictions are distilled from Google's webpage[11] and several further web logs provided helpful information for understanding more facets of the specific CECs. To process the extracted KDM application models of the systems described in Section 11.2.2 and to perform the conformance checking process to detect CEC violations, the tool *CloudMIG Xpress* is utilized. *CloudMIG Xpress* builds

[11] https://developers.google.com/appengine/docs/java/

on the tool MoDisco (cf. Section 10.1) for extracting KDM models from Java source code. For CC-1, *CloudMIG Xpress* employs MoDisco V. 0.8. For each detected CEC violation a data record is reported that comprises the following information:

- ID of the processed application
- The particular KDM model (third-party libraries of an application are contained in separate KDM models)
- The class referring to the source of the CEC violation
- The CEC violation severity
- The CEC type

In the context of CC-1 and *G1*, an important aspect is judging about the feasibility of the conformance checking approach and less exploring its applicability on a wide range of diverging system architectures. Therefore, the type of potential applications is narrowed to Java and web-based systems. The open source systems described in Section 11.2.2 are selected for evaluation purposes. Although all systems are implemented using web technologies, they cover a broader scope in the sense that their business domains are, except for `App3` and `App4`, heterogeneous. The applications allow analyses for varying scales. The sizes of the largest and the smallest application differ by a factor of approximately 33 when considering the Lines of Code (LOC) metric of the application's sources while leaving out used third-party libraries. Several GQM questions incorporate size measures to approach the GQM goals (e.g., *Q2.2*). The number of classes and libraries for each program can be extracted from the KDM models. As a further sufficiently well-suited size measure we employ, as described before, LOC. It it measured with CLOC[12] V. 1.52 omitting comment and blank lines. However, it is only possible to measure LOC for the applications' own sources, as the sources for the third-party libraries are not available. However, third-party libraries are only incorporated for analyzing *Q2.4* and for that GQM question LOC is not relevant.

[12] `http://cloc.sourceforge.net/`

Table 11.4. CECs modeled in the GAE for Java cloud profile for CC-1

Type	#Variants	Violation severity
MaxTotalNrOfFilesConstraint	1	1 Warning
MethodCallConstraint	12	3 Critical, 9 Warning
SocketOpeningConstraint	1	1 Critical
FilesystemAccessConstraint	2	1 Critical, 1 Warning
ReflectionConstraint	1	1 Critical
TypesInstantiationConstraint	1	1 Critical
TypesWhitelistConstraint	1	1 Critical
LanguageConstraint	1	1 Breaking

Modeled Constraints 29 CECs are modeled and included as a part of GAE for Java's cloud profile. It is possible to detect 20 CECs that can be covered by the provided constraint validators corresponding to 8 covered types of CECs. The lack of the residual CECs is caused by the fact that *CloudMIG Xpress* does not provide support for detecting violations that are only identifiable at runtime and just static analyses are implemented for evaluation purposes (cf. Section 10.3).

Table 11.4 lists the types of CECs that can be detected during the conformance checking process and the number of present variants. For example, several variants of a `MethodCallConstraint` exist due to restrictions on calls to different methods. It should be noted that for different variants of a CEC type there can be dissimilar CEC severities assigned. Furthermore, the CECs that inherit from `AbstractTypeListConstraint` (see the `Constraints` package of CEM in Appendix A) show only a single variant in the table. But looking at a `TypesWhitelistConstraint`, the single constraint translates to a prohibition to access 2,388 of the JRE types, for example.

SMM Model for Computing Prioritization Function `prio` GQM question *Q1.3* aims at evaluating the function `prio` for prioritizing the urgency and the correction of detected CEC violations. `prio` employs a type-level coupling metric (cf. Section 7.4.2). For determining the coupling of types on the basis of extracted KDM models, we use SMM to build an appropriate metric and apply this metric with the help of MEE (cf. Section 7.2.2). Figure 11.8 shows the SMM model of the corresponding simple type-level coupling metric. The SMM instance constitutes a measure in the sense of SMM (cf. Section 3.3.3).

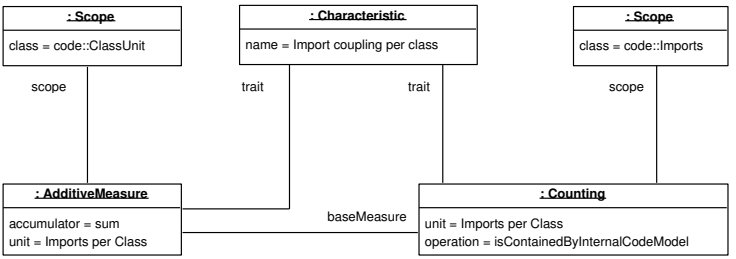


Figure 11.8. SMM import coupling measure (cf. [Frey et al. 2013a])

The measure can be used to determine the number of imports for a class. Though, it will only count the references to internal classes. This measure counts the number of import statements for each class of the software system. The main measure of this SMM instance is the `AdditiveMeasure` that will be applied to `ClassUnit` elements (classes) of KDM instances. If MEE discovers such an element, the operation of the `Counting` measure will be applied to all `Imports` that are contained in the class. The result of the operation that is implemented in a helper class will be *1* if the class referenced by the import statement is defined in the software system. It will return *0* if the class is included in external libraries. Hence, after the `AdditiveMeasure` sums up the according measurements, the result will be the number of referenced internal classes for a single class. Due to the definition in the SMM instance, this metric will be computed for every class in the software system.

11.4.2 Case Study CC-2

The case study CC-2 investigates the performance of the conformance checking approach. The corresponding GQM metrics *M8-M11* involve a manual inspection of detected CEC violations (cf. Section 11.3.2). To reveal false alarms (false positives), a template-based approach is used. Reported CEC violations are analyzed by (1) manually inspecting the source code

Table 11.5. CEC violation inspection template (cf. [Fenner 2011])

CEC Violation ID:	V-#
CEC Type:	...
Violation Severity:	Warning / Critical / Breaking
Source Location:	...
Violation Context:	...
Validation Correctness:	Correct / False
Justification/ Appraisal:	...

elements that are signaled as being the root cause for the CEC violations by *CloudMIG Xpress* and by (2) consulting and cross-checking the GAE documentation. Each CEC violation and inspection result is then described with the help of the template shown in Table 11.5 (cf. [Fenner 2011]).

In the first three rows, the template includes several basic information, i.e., the ID of the CEC violation, the type of the corresponding CEC, and the violation severity as defined in the GAE cloud profile. The fourth row (*Source Location*) contains the class of `App4` that causes the CEC violation, while the fifth row (*Violation Context*) describes the functionality of the corresponding source code for clarifying the details and further circumstances of the CEC violation. *Validation Correctness* states whether the detected CEC violation actually constitutes a CEC violation. Hence, the corresponding row describes whether the CEC violation is a true positive or false positive. The last row (*Justification/Appraisal*) gives an explanation for this assessment.

4,716 CEC violations are detected in the case study CC-2 by *CloudMIG Xpress*. 250 of those CEC violations that are raised by `App4`'s own source code elements are inspected manually. Furthermore, several samples of the CEC violations that are caused by used third-party libraries are inspected (cf. [Fenner 2011]).

11.4.3 Case Study CC-3

The case study CC-3 analyzes a C#-based library from the HSH Nordbank AG (cf. [Wulf 2012]). To extract a KDM model from the bank library's source

code, the three-phase transformation of C# to KDM is used (cf. Section 10.1). For integrating this transformation in *CloudMIG Xpress*, a corresponding plugin is built (*M12*).

To process the extracted KDM model, a cloud profile for Microsoft Windows Azure is created (*M14*) that describes, among others, the basic hardware resources, prices, and data centers. As an exemplary CEC, the cloud profile includes a specification for a `LocalTransientStorageConstraint`, as the VM role of Microsoft Windows Azure does not persist data that is saved to the local storage by a client application. An excerpt of Microsoft Windows Azure's cloud profile that also shows the definition of this CEC is presented in Appendix B. Furthermore, a CEC validator is created (*M13*) that is able to detect CEC violations regarding the `LocalTransientStorageConstraint` in the extracted KDM model. The CEC validator builds upon the `AbstractTypeInstantiationAndMethodCallValidator` class (cf. Section 10.3). Hence, the CEC validator provides lists of C# types and methods that cause CEC violations with regard to the `LocalTransientStorageConstraint`. For example, the CEC validator includes the methods shown in Listing 11.1 that are defined in the Framework Class Library (FCL) of the Microsoft .NET framework (cf. [Wulf 2012]).

```
1 System.IO.Directory.CreateDirectory
2 System.IO.Directory.Delete
3 System.IO.DirectoryInfo.Create
4 System.IO.DirectoryInfo.CreateSubdirectory
5 System.IO.File.AppendAllLines
6 System.IO.FileInfo.AppendText
```

Listing 11.1. Methods of FCL types listed in a CEC valdidator that checks `LocalTransientStorageConstraints` (excerpt, cf. [Wulf 2012])

CC-3 uses the tool NDepend[13] to examine the CEC violations found in the bank library (*M15*). NDepend provides the Code Query Language (CQL) for querying source code elements of any .NET application. Hence, CQL is utilized to select the elements that use types and methods that are included in the CEC specification. Listing 11.2 shows an excerpt of the CQL query used to find these elements (cf. [Wulf 2012]).

[13] http://www.ndepend.com

```
1 SELECT METHODS WHERE ! IsSpecialName AND
2 (
3   IsDirectlyUsing "System.IO.Directory.CreateDirectory (String)" OR
4   IsDirectlyUsing "System.IO.Directory.Delete(String , Boolean)"
5 )
```

Listing 11.2. CQL query for selecting forbidden FCL method calls from the bank library's source code (excerpt, cf. [Wulf 2012])

11.5 GQM Interpretation Phase

The gathered raw data from the GQM data collection phase (cf. Section 11.4) is employed in this section to compile the measurement results for the previously defined GQM metrics and to answer the posed GQM questions (cf. Section 11.3) according to GQM's interpretation phase.

11.5.1 Case Study CC-1

The case study CC-1 comprises the GQM goals *G1* and *G2*. The GQM interpretation procedure is also structured according to these goals.

G1: Show applicability of conformance checking approach

Three GQM questions *Q1.1-Q1.3* are derived from *G1*. These questions are investigated in the following with the help of the GQM metrics *M1-M5* and the corresponding measurements gathered in the GQM data collection phase.

Q1.1: *Does the overall amount of CEC violations justify the usage of an automatic detection approach?*

The absolute numbers of detected CEC violations differ widely. For `App1-App5`, we detected (87/4,386), (3/8), (98/932), (273/1,428), (7,795/612) CEC violations (own sources/ third-party libraries). Taken by itself, these absolute numbers already appear to be quite high and suggest that common enterprise software may benefit from an automatic detection approach when

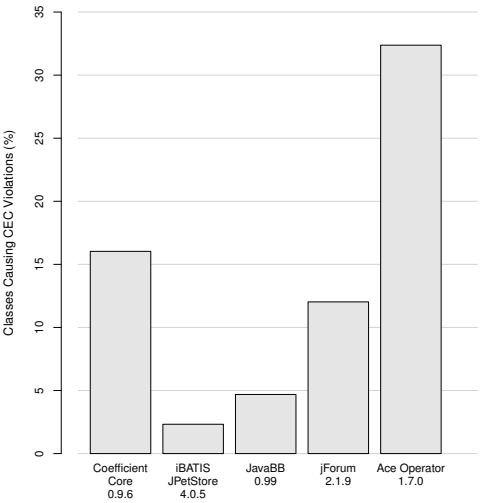


Figure 11.9. Percentage of classes that cause constraint violations (w/o third-party libs, cf. [Frey et al. 2013a])

considering a migration to the cloud. However, to enable comparing the number of detected CEC violations among applications, the question *Q1.1* utilizes the metrics *M1* and *M2*. Figure 11.9 shows the percentage of classes that cause CEC violations with regard to GAE for Java (*M1*). The range spans from approx. 2.5% to 33% for the smallest (`App2`) and the largest application (`App5`), respectively. Generally, with application size comes a steady growth in the share of classes raising CEC violations, but without a uniform growth rate or pattern.

Furthermore, *Q1.1* investigates the densities of CEC violations per application (*M2*). This metric *M2* normalizes the number of detected CEC violations to 1,000 LOC to facilitate comparability among applications. Figure 11.10 shows the measurement results for metric *M2*. The detected constraint violation densities per application are similar for `App1`, `App3`, and `App4` considering *Critical* CEC violation severities, despite `App4` being roughly 3 times bigger than the others. The smallest application `App2`'s density is slightly lower,

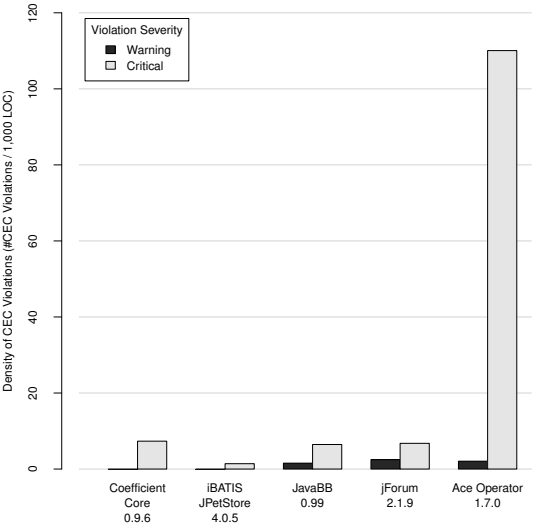


Figure 11.10. The detected constraint violation densities per application (w/o third-party libs, cf. [Frey et al. 2013a])

but the biggest application App5's density exceeds the others in orders of magnitude. Generally, we conclude that the size can also be an indicator for higher densities of CEC violations, too. But for CEC violations with *Warning* as CEC violation severity this statement is weaker.

However, in the light of (1) the absolute numbers of detected CEC violations, (2) the ratio of classes that raise CEC violations, and (3) the generally increasing CEC violation densities for larger applications, it is valid to state the usefulness of our automatic CEC violation detection approach.

Q1.2: *Considering a manual detection approach, could CEC violations be easily spotted in individual classes?*

The investigation of detected CEC violations in the previous question *Q1.1* already revealed a rather high amount of CEC violations regarding the different applications. For example, *M1* determines the classes that are

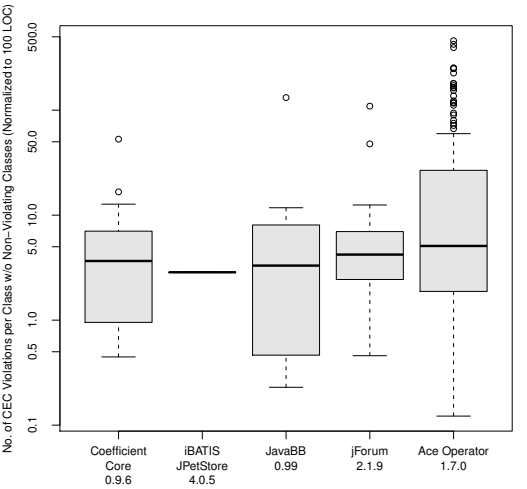


Figure 11.11. The detected constraint violation densities per class (w/o third-party libs, cf. [Frey et al. 2013a])

responsible for one ore more CEC violations. The question *Q1.2* together with *M3* analyzes the CEC violation densities on the class level to evaluate the general complexity of manually identifying CEC violations within a recognized, faulty class. Figure 11.11 shows the detected constraint violation densities per class. The largest application `App5` exhibits the largest jitter. It is remarkable that the medians of all classes in all applications lie in a rather narrow band of approx. 3-5 (normalized to 100 LOC for each class).

Corresponding with the already high number of overall detected CEC violations per application (except for `App2`), (1) the average CEC violation densities per class are also rather high for each of the applications. Furthermore, due to the common, considerable deviation from the medians and the occasionally observed jitter, it is (2) hard to find patterns in this regard. Hence, manually spotting CEC violations in individual classes is not a straightforward task, that might also be highly influenced by further

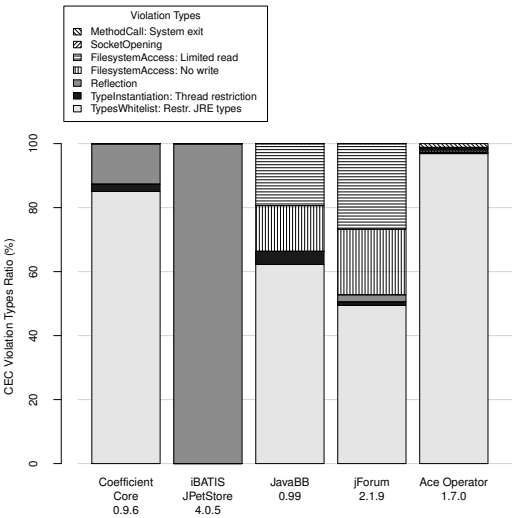


Figure 11.12. Distribution of detected violation types (w/o third-party libs, cf. [Frey et al. 2013a])

characteristics, such as the source code complexity, quality of available cloud environment documentation, and the types of CECs defined for a cloud environment.

Q1.3: *Does* `prio` *narrow the focus to particularly critical classes?*

To evaluate `prio`, the question *Q1.3* employs the metrics *M4* and *M5*. The formula of `prio` involves the *class violation weight* (cvw) for each class (cf. Section 7.4.2). Hence, the detected CEC violations have to be subdivided according to the types of underlying CECs (*M4*). Figure 11.12 presents the corresponding distribution of detected CEC violation types. It is no surprise that the `TypesWhitelistConstraint` is dominant, as this CEC can match for a plethora of cases as stated in Section 11.4.1. `APP2` does not coincide, but this is likely due to the low overall number of CEC violations found for this application. CEC violations related to restricted file system access are in addition to it quite frequently observed.

After breaking down the types of CEC violations, the metric *M5* delivers the partitioning of classes that raise CEC violations into priority levels I-III through applying the prioritization function `prio`. Figure 11.13 presents the partitioning that results from applying the prioritization function `prio` in combination with the import coupling measure described in Section 11.4.1. It should be noted that the application `App2` is omitted in Figure 11.13 due to legibility and moreover the CEC violations of `App2` (considering solely its own sources) stem from a single class. Therefore, a prioritization at the class level would be rather pointless for `App2`. Analyzing the results for the other applications, we observe that priority level `I` is chosen in a range of 0% to 14% of the classes (`App3` and `App1`). 26% to 44% of the classes (`App5` and `App4`) are assigned priority level `II` and 42% to 66% of the classes (`App1` and `App5`) receive priority level `III`. A pattern that we register for our test subjects is that the size of an application is correlated inversely to the share of classes that are assigned the priority level `I`. However, the correlation does not exhibit a constant factor and additionally `App3` constitutes an exception lacking priority level `I` classes at all. Taking a closer look at the other applications' priority level `I` classes, we notice that the vast majority of them is bigger in terms of LOC than the average of the classes raising CEC violations.

G2: Investigate patterns of detected CEC violation occurrences

Four GQM questions *Q2.1-Q2.4* are derived from *G2*. These questions are investigated in the following with the help of the GQM metrics *M1*, *M2*, *M6*, and *M7* and the corresponding measurements gathered in the GQM data collection phase.

Q2.1: *Do application sizes correlate with ratio of classes raising CEC violations?*

The ratio of the applications' classes that raise CEC violations are shown in Figure 11.9. This ratio corresponds to *M1* that is defined in Section 11.3.1 as a GQM metric for *Q2.1*. For employing a regression analysis regarding the correlation of application sizes and the ratio of classes raising CEC violations, the test set, that comprises five applications in the context of CC-1, is too small. However, *Q1.1* already described that with bigger application sizes

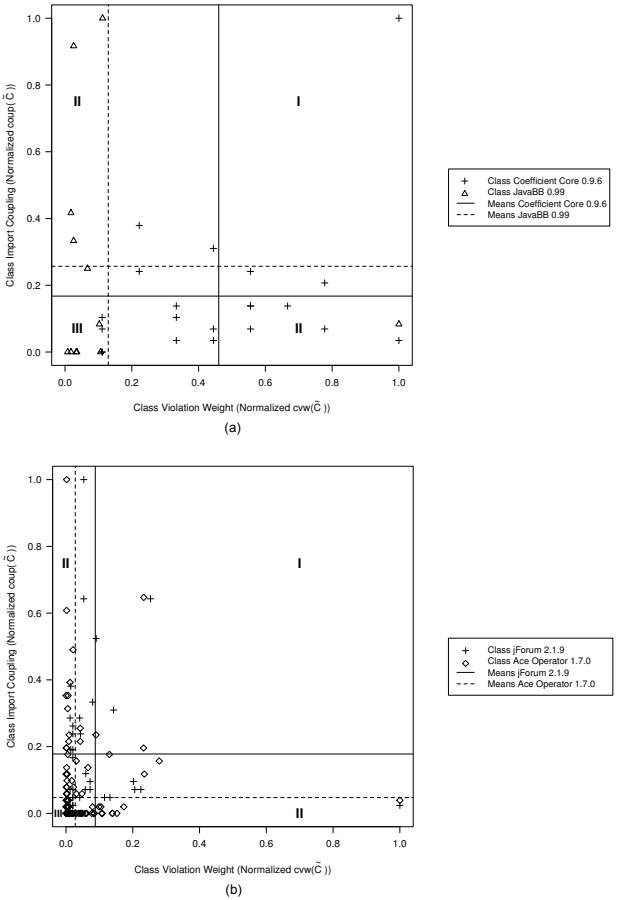


Figure 11.13. Partitioning of classes that raise constraint violations into priority levels `I-III` through applying the prioritization function `prio`. Split into two diagrams (a) and (b) due to legibility (cf. [Frey et al. 2013a]).

the share of classes raising CEC violations increases, but without a uniform growth rate or pattern. Hence, the measurements resulting from *M1* suggest that *Q2.1* may be approved.

Q2.2: *Do application sizes correlate with CEC violation densities?*

The detected CEC violation densities are shown per application in Figure 11.10. The applications' CEC violation densities are defined in Section 11.3.1 as a GQM metric for *Q2.2*. As with *Q2.1*, employing a regression analysis regarding the correlation of application sizes and CEC violation densities is impeded by the small test set that covers just five applications in the context of CC-1. Nevertheless, *Q1.1* already described that an application's size can also be an indicator for higher densities of CEC violations.

Q2.3: *Do CEC violations group in a few application elements?*

The question *Q2.3* addresses the distribution of CEC violations among the applications' elements. Using *M6*, it investigates whether CEC violations group in clusters. Figure 11.14 shows a scatterplot for classes raising CEC violations and the relation (1) to the relative number of CEC violations these classes raise and (2) to their size. The predominant number of those classes are responsible for 5% or less of the overall number of CEC violations raised by their applications. The root causes are wide spread over the systems. But there exist some outliers, the likely most spectacular one refers to `App3` and is responsible for approx. 60% of CEC violations. All CEC violations from the small `App2` are located in one class.

Q2.4: *Are CEC violations particularly raised in an application's own source code or its third-party libraries?*

The previously addressed questions only considered the applications' own source code and deliberately disregarded the used third-party libraries (cf. Section 11.4.1). In contrast, *Q2.4* examines whether CEC violations are particularly raised in an application's own source code or in its third-party libraries with the help of *M7*. In Figure 11.15, the third-party libraries are incorporated, it presents the origin of CEC violations. Already the total numbers of detected CEC violations stated above showed vast differences. Once more, `App5` is a special case, as it is the only system that produces substantially more CEC violations in its sources than its third-party libraries do. Furthermore, it can be seen in Figure 11.15 a)-c) and partially in d) that

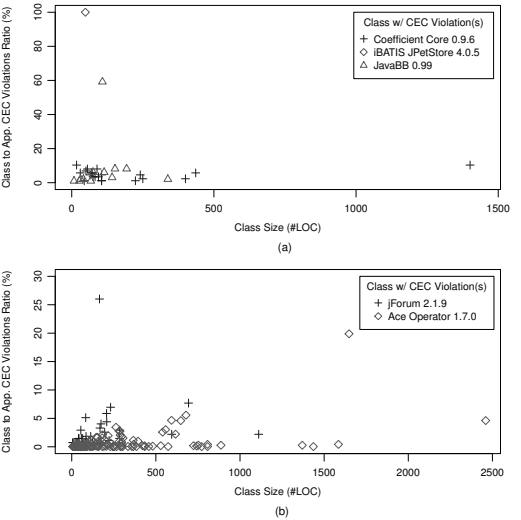


Figure 11.14. Classes being responsible for an amount of constraint violations and the relation to their size (w/o third-party libs). Split into two diagrams (a) and (b) due to legibility (cf. [Frey et al. 2013a]).

few libraries are responsible for most of the CEC violations. Identifying and handling those primarily seems to be a worthwhile approach.

11.5.2 Case Study CC-2

The case study CC-2 comprises the GQM goal *G3*. This section describes the corresponding GQM interpretation phase.

G3: Assess performance of conformance checking approach

The two GQM questions *Q3.1* and *Q3.2* are derived from *G3*. These questions are investigated in the following with the help of the GQM metrics *M8-M11*

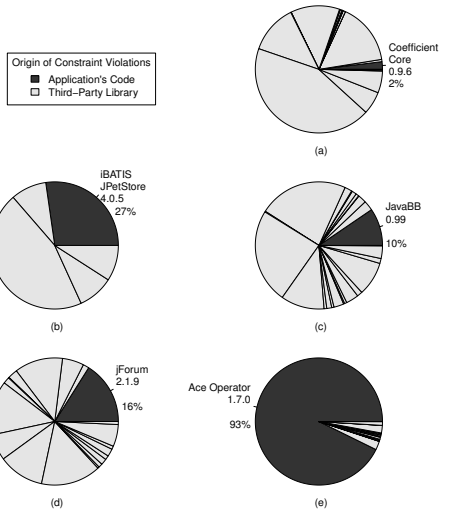


Figure 11.15. Distribution of the constraint violations' origin. The origin of a violation is located in the application itself, if the source code of the application directly causes the violation (dark gray slice). Each light gray slice represents a third-party library that is used by the application. The origin of a violation is located in such a third-party library, if the source code of the library causes the violation (cf. [Frey et al. 2013a]).

and the corresponding measurements and raw data gathered in the GQM data collection phase.

Q3.1: *How precise is CloudMIG's CEC violation detection mechanism?*

As described in Section 11.4.2, CC-2 performs a manual inspection of CEC violations that are reported by *CloudMIG Xpress* (cf. [Fenner 2011]). By using an inspection template, each CEC violation included in the test set is analyzed in a structured way. Based on these resulting inspection templates, *M8-M10* are used to determine the number of true positives, false positives, and the precision of CloudMIG's CEC validation classification, respectively. In the following, two exemplary CEC violations (V-102 and V-117) and the corresponding completed inspection templates from the

Table 11.6. Inspection template of CEC violation V-102 (cf. [Fenner 2011])

CEC Violation ID:	V-102
CEC Type:	TypesInstantiationConstraint
Violation Severity:	Critical
Source Location:	net.JForum.view.admin.AdminAction
Violation Context:	The functionality to listen for POP emails starts a new thread that runs in the background.
Validation Correctness:	Correct
Justification/ Appraisal:	Manual code review (see Listing 11.3) verifies the instantiation of a `java.lang.Thread`. Instantiating a new thread is not allowed in the App Engine.

diploma thesis of Fenner [2011] are presented to illustrate the overall manual inspection approach that leads to the measurement results for *M8-M10*. Listing 11.3 shows the CEC violation V-102 (raised in Line 4) regarding a `TypesInstantiationConstraint`. This CEC prohibits the creation of threads (class `java.lang.Thread`). Table 11.6 shows the corresponding CEC violation inspection template.

```
1  package net.jforum.view.admin; [...]
2  public class AdminAction extends Command { [...]
3    public void fetchMail() throws Exception {
4      new Thread (new Runnable() {
5        public void run() {
6          try {
7            new POPListener().execute(null);
8          }
9          catch (Exception e) { e.printStackTrace(); }
10     }}).start();
11     this.main();
12   }
13 }
```

Listing 11.3. Source code excerpt of CEC violation V-102 (cf. [Fenner 2011])

A further inspected CEC violation example (V-117) is shown in Listing 11.4. The source code raises a CEC violation regarding a `TypesWhiteListConstraint` (Line 15), as GAE for Java does not permit accessing the JRE type `java.awt.Color`. Regarding GAE for Java's cloud profile, this type is not

Table 11.7. Inspection template of CEC violation V-117 (cf. [Fenner 2011])

CEC Violation ID:	V-117
CEC Type:	TypesWhiteListConstraint
Violation Severity:	Critical
Source Location:	net.JForum.util.Captcha
Violation Context:	The `Captcha` module uses constants of the `java.awt.Color` type to generate graphical elements in Captchas.
Validation Correctness:	Correct
Justification/ Appraisal:	Manual code review (see Listing 11.4) verified the usage of `java.awt.Color`. This type is not available in the Java App Engine environment.

included in the cloud profile's type list that describes the white list (cf. Appendix B). Hence, the responsible `TypesWhitelistConstraintValidator` (cf. Section 10.3) reports the corresponding CEC violation.

```
1  package net.jforum.util; [...]
2  public class Captcha extends ListImageCaptchaEngine { [...]
3    protected void buildInitialFactories() {
4      this.initializeChars();
5      this.backgroundGeneratorList = new ArrayList();
6      this.textPasterList = new ArrayList();
7      this.fontGeneratorList = new ArrayList();
8      int width = SystemGlobals.getIntValue(ConfigKeys.CAPTCHA_WIDTH);
9      int height = SystemGlobals.getIntValue(ConfigKeys.CAPTCHA_HEIGHT);
10     int minWords = SystemGlobals.getIntValue(ConfigKeys.CAPTCHA_MIN_WORDS);
11     int maxWords = SystemGlobals.getIntValue(ConfigKeys.CAPTCHA_MAX_WORDS);
12     int minFontSize = SystemGlobals.getIntValue(ConfigKeys.CAPTCHA_MIN_FONT_SIZE);
13     int maxFontSize = SystemGlobals.getIntValue(ConfigKeys.CAPTCHA_MAX_FONT_SIZE);
14     this.backgroundGeneratorList.add(new GradientBackgroundGenerator(
15       new Integer(width), new Integer(height), Color.PINK, Color.LIGHT_GRAY));
16     [...]
17   }
18 }
```

Listing 11.4. Source code excerpt of CEC violation V-117 (cf. [Fenner 2011])

With the help of the completed inspection templates, such as those shown in the Tables 11.6 and 11.7, CC-2 comprises assessments for 250 reports of CEC violations that are raised by `App4`'s own source code (cf. Section 11.4.2).

With the exception of one CEC violation, all are classified as true positives (*M8*), i.e., they actually constitute CEC violations of `App4` regarding GAE for Java. The single case that is noticed as incorrectly reporting a CEC violation rather constitutes a misinterpretation of the GAE documentation than an actual weakness of the conformance checking approach.

The regarding CEC violation refers to a `ReflectionConstraint` defined in the GAE for Java cloud profile. The documentation of GAE for Java states that reflection is only allowed for "own" classes.[14] Due to the configuration of the `ReflectionConstraint`, the usage of `java.lang.reflect.Method.invoke()` is erroneously reported as a CEC violation. Fenner [2011] decides to not count this CEC violation as a false positive, as this issue is not caused by the detection mechanism itself and an altered configuration had corrected this false alarm. Hence, the number of true positives is 250 (*M8*) and false positives is 0 (*M9*). Consequently, for the metric *M10*, the conformance checking approach accomplishes a 100% precision (cf. [Fenner 2011]).

Q3.2: *Are CEC violations raised in third-party libraries classified correctly?*

Besides `App4`'s own source code that is analyzed in CC-2 in the context of the previous GQM question *Q3.1*, *Q3.2* examines samples of reported CEC violations originating in used third-party libraries. In the following, source code for an exemplary CEC violation (Listing 11.5) that is raised by a used third-party library, and the corresponding completed inspection template (Table 11.8) are shown.

[14] `https://developers.google.com/appengine/docs/java/runtime?#The_Sandbox`

Table 11.8. Inspection template of CEC violation V-2081 (cf. [Fenner 2011])

CEC Violation ID:	V-2081
CEC Type:	SocketOpeningConstraint
Violation Severity:	Critical
Source Location:	com.mysql.jdbc.NamedPipeSocketFactory.NamedPipeSocket
Violation Context:	In the Java MySQL Connector Library, a connection to the MySQL Server is initialized by instantiating a `Socket` directly to the server.
Validation Correctness:	Correct
Justification/ Appraisal:	Manual code review (see Listing 11.5) verifies the instantiation of a `java.net.Socket`. Instantiating a socket is not allowed in the App Engine.

```
1  package com.mysql.jdbc; [...]
2  public class NamedPipeSocketFactory implements SocketFactory { [...]
3    private Socket namedPipeSocket;
4    public Socket connect(String host, int portNumber /* ignored */,
5        Properties props) throws SocketException, IOException {
6      String namedPipePath = props.getProperty(NAMED_PIPE_PROP_NAME);
7      if (namedPipePath == null) {
8        namedPipePath = "\\\\.\\pipe\\MySQL"; //$NON-NLS-1$
9      } else if (namedPipePath.length() == 0) {
10       throw new SocketException(Messages
11           .getString("NamedPipeSocketFactory.2") //$NON-NLS-1$
12           + NAMED_PIPE_PROP_NAME
13           + Messages.
14               getString("NamedPipeSocketFactory.3")); //$NON-NLS-1$
15     }
16     this.namedPipeSocket = new NamedPipeSocket(namedPipePath);
17     return this.namedPipeSocket;
18   }
19 }
```

Listing 11.5. Source code excerpt of CEC violation V-2081 (cf. [Fenner 2011])

The KDM models extracted from Java libraries include less details than those resulting from source code files. This is due to the fact that the KDM discoverer for Java builds upon the MoDisco library discoverer that's capabilities are, at the time of writing, limited (cf. Section 10.1). Nonetheless, the CEC violation regarding a defined `SocketOpeningConstraint` that is raised by the

source code of Listing 11.5 in Line 16 through creating a `java.net.Socket`, is detected by *CloudMIG Xpress*. Likewise, all of the inspected sample CEC violation reports (*M11*) exhibit a correct classification.

11.5.3 Case Study CC-3

The case study CC-3 employs the GQM goal *G4* to evaluate the conformance checking approach with the help of an industrial partner (cf. [Wulf 2012]). This section describes the corresponding GQM interpretation phase.

G4: Demonstrate feasibility with an industrial case study
The three GQM questions *Q4.1-Q4.3* are derived from *G4*. These questions are investigated in the following with the help of the GQM metrics *M12-M15*.

Q4.1: *Can support for an additional programming language be added to CloudMIG's architecture?*

To detect CEC violations of the C#-based bank library regarding a Microsoft Windows Azure (VM role) cloud profile, a KDM discoverer plugin and a CEC validator plugin for *CloudMIG Xpress* have to be available. These plugins are covered by *M12* and *M13*, respectively.

M12: The KDM discoverer plugin builds upon the three-phase transformation of C# to KDM (cf. Section 10.1). The transformation is wrapped in a plugin that uses CloudMIG's API for contributing a KDM discoverer, i.e., it provides a subclass of `AbstractKDMDiscoverer` (cf. Section 9.2). This subclass of `AbstractKDMDiscoverer` then delegates to the transformation component and translates between *CloudMIG Xpress*' and the transformation component's mechanisms for controlling the KDM extraction process. Hence, the plugin also serves as an adapter to integrate the C# to KDM transformation component in *CloudMIG Xpress*' architecture.

M13: The CEC validator plugin covers language-dependent detection mechanisms regarding C# (cf. Section 7.1.3). As described in Section 11.4.3, the plugin provides an exemplary constraint validator for the CEC `LocalTransient-`

`StorageConstraint`. It builds upon the `AbstractTypeInstantiationAndMethodCallValidator` class and therefore defines lists of C# types and methods that provoke a corresponding CEC violation when used in an SUA's source code.

Q4.2: *Can a cloud environment of interest be modeled with CEM?*

An excerpt of Microsoft Windows Azure's cloud profile is presented in Appendix B (*M14*). The VM role and the two other roles of Microsoft Windows Azure (cf. Section 11.2.3) are each modeled with `CloudEnvironmentConfiguration` elements. Corresponding to the CEC that is checked by the CEC validator described in the context of *Q4.1*, the VM role's `CloudEnvironmentConfiguration` element includes a `LocalTransientStorageConstraint` specification that is contained in an `EnvironmentConstraintConfiguration` element. Furthermore, Microsoft Windows Azure's VM role is modeled with the help of CEM's `IaaS` package for providing VM resources.

Q4.3: *Can enterprise software from an industrial partner be checked for CEC violations?*

With the help of the Microsoft Windows Azure cloud profile and the KDM discoverer and CEC validator for C#, the bank library (`App6`) can be checked for CEC violations. As described in Section 11.4.3, CC-3 uses the tool NDepend and CQL to evaluate the integration of these artifacts with CloudMIG's general conformance checking approach and tool architecture.

The defined CQL query delivers 12 forbidden method calls, whereas *CloudMIG Xpress* only detects 7 of them (~58% detection rate). However, the missed method calls do not result from weaknesses of the conformance checking approach. Instead, the corresponding methods are not contained in the KDM model that is extracted by the three-phase transformation of C# to KDM. This is due to the fact that the transformation, at the time of writing, omits complete method bodies as soon as one of the included statements is not supported (cf. [Wulf 2012]).

An exemplary CEC violation regarding a `LocalTransientStorageConstraint` that is detected by *CloudMIG Xpress* and NDepend is shown in Listing 11.6.

```
1 internal string GetOutputPath (string PathName)
2 {
3   string pathUp = Path.GetDirectoryName(PathName);
4   string outputPath = pathUp + '\\' + "Results";
5   DirectoryInfo op = new DirectoryInfo(outputPath);
6   op.Create();
7   return outputPath ;
8 }
```

Listing 11.6. A CEC violation in `App6` detected with *M12-M14* (cf. [Wulf 2012])

The CEC violation is raised by calling the `System.IO.DirectoryInfo.Create()` method (Line 6), as this method is included in the CEC validator's list of prohibited methods (cf. Listing 11.1). All of the CEC violations that can be detected based on the restricted KDM models extracted from C# source code are found (*M15*). Hence, it is fair to approve *Q4.3* within the limited frame of CC-3. However, there are threats to validity, such as the low number of CEC specifications. These threats to validity (and also the threats to validity regarding CC-1 and CC-2) are described in the next Section 11.6.

11.6 Threats to Validity

The case studies CC-1, CC-2, and CC-3 are subject to threats to validity. These threats to validity are described in the following Sections 11.6.1, 11.6.2, and 11.6.3 for CC-1, CC-2, and CC-3, respectively.

11.6.1 Case Study CC-1

CC-1's threats to validity come from the chosen *experiment setup* and the specific aspects of the investigated *service models*. Further areas of the case study that exhibit threats to validity are the general *applicability* of the approach as well as the *prioritization of* CEC *violations with* `prio`.

Experiment Setup The case study CC-1 employs CEC validators that can detect CEC violations by statically analyzing extracted KDM models. Applying dynamic analyses seems to be useful to provide additional validators that can increase the coverage for some constraint types as well. For example, detecting violations of the `SocketOpeningConstraint` at runtime or violations concerning the `MethodCallConstraint` that are caused by dynamic dispatch promise to be beneficial. Furthermore, identifying and removing orphaned classes, components, or even subsystems is a typical task in reengineering projects [Demeyer et al. 2004]. Static and dynamic analyses could be utilized for identification purposes. Considering only system parts that are actually being used might also lead to a significant reduction of CEC violations. However, the actual usage of a software system depends on specific operational scenarios and therefore cannot be generalized in CC-1.

Regarding the applications investigated in the context of CC-1, only system types are analyzed that are relatively well-suited. Google App Engine for Java supports JVM-based web software systems. Only representatives of this system type are studied and the number of probands is also rather small. However, within the scope of *G1* and *G2*, the experiments provided valuable insights and showed that our approach can be successfully utilized. All detected CEC violations have to be either addressed in reengineering measures or at least be considered in the decision-making process while evaluating target cloud environment candidates.

Service Models CC-1 analyzes CEC violations regarding GAE for Java that is a PaaS-based cloud environment. In general, compared to current IaaS environments, PaaS offerings are more heterogeneous as they add diverse abstraction layers or specific preconfigured software stacks to the basic infrastructure building blocks. From a cloud user perspective, these additional boundaries constitute further CECs that need to be taken into account. Considering other PaaS environments, the results from the evaluation will primarily shift because of two factors. First, PaaS environments that do not support the Java runtime environment will yield incomparable results. As stated in Section 7.3, the approach CloudMIG does not aim to migrate software systems through applying a programming language transformation. It terminates its workflow if CEC violations with the assigned *violation*

severity Breaking are detected. The `LanguageConstraint` listed in Table 11.4 is a *Breaking* constraint. Second, the majority of detected CEC violations are `TypesWhitelistConstraints`. Hence, concordance of results for other PaaS environments that offer a JRE compatibility will particularly be sensitive to type restrictions that diverge from those defined in Google App Engine for Java's sandbox environment.

Regarding IaaS environments, it can be expected that the number of detected violations would be considerably lower. This is due to the fact that IaaS environments by definition lack narrow restrictions concerning the underlying software stack that in turn could translate into additional modeled CECs and violations of those. Nevertheless, the same CEM elements can be used to model the constraints for the IaaS and the PaaS service model. This is due to the fact that the constraints are designed in a generic way. The regarding *Constraints package* does not depend on packages that are located higher in the layered CEM architecture (cf. Figure 6.16).

Applicability *G1* investigates the applicability of the conformance checking approach. Corresponding threats to validity are described in the following. We start with examining the general cost-benefit ratio of using our automatic conformance checking process. The cloud profiles as well as the constraint validators can be reused by reengineers. These artifacts can be shared with the public repository presented in Section 6.3.3. Considering the scenario that a reengineer may have to find and compare cloud environments that are suitable as migration targets, the advantages of using the presented approach are clear. There exists a great number of cloud environment candidates and the cloud provider landscape as well as their offered services are changing quickly.

For any potential cloud environment, the CECs would otherwise have to be identified manually. Even doing so for a single cloud solution, as for example Google App Engine for Java, is not a trivial task as the elicitation of the involved constraints is hampered by incomplete information, various scattered relevant data sources, and non-standardized formats, for instance. Then, the software system's source code would have to be inspected manually to detect the related CEC violations that are specific for each cloud

environment candidate. Here, the according documentation or developer knowledge is often missing or incomplete. Additionally, considering the large numbers of detected CEC violations and that they are furthermore often scattered all over the systems in our evaluation scenarios, this constitutes a tedious and error-prone task even for a single cloud environment. Moreover, some cloud providers do not report the CEC violations during the deployment or start-up process of the guest applications. For example, as in the case of Google App Engine for Java, some restricted method calls may lead to thrown exceptions late after deployment when they are executed the first time. As a tradeoff, it must be possible to extract a KDM model from an existing system to apply our automatic conformance checking process. This might not be the case for many existing programming languages as they currently lack appropriate tool support. However, the used application MoDisco provides a suitable framework for developing and enhancing discoverers. Moreover, there exist further tools that can extract KDM models. For example, gcckdm[15] is an open source plugin for the popular gcc compiler that produces a KDM model from C and C++.

The corresponding preconditions that have to be met for executing our automatic conformance checking process are also summarized in Section 7.1.3.

Prioritization of CEC Violations with `prio` The partitioning that results from applying the prioritization function `prio` can serve as a guideline when reasoning about the order of reengineering the classes that raise CEC violations. `prio` provides a partitioning of classes that are responsible for CEC violations rather than a strict partial ordering. It addresses two important factors that presumably have effects regarding the reengineering effort, that is to say the coupling [Hall et al. 2005; Chidamber et al. 1998] and CEC violation severities of classes. However, as there may be other factors not covered by `prio`, we solely provide coarse grained partitions indicating the urgency of reworking the classes. Higher priority levels exhibit a greater amount of uncertainty that, from a migration planning perspective, should be eliminated as early as possible. Furthermore, coupling metrics that are rather sophisticated could be employed as well. To integrate characteristics

[15] `https://github.com/KdmAnalytics/gcckdm/`

specific for each application, `prio` does provide a relative partitioning rather than using fixed thresholds.

Moreover, it combines the arithmetic means of the coupling and *violation severity* values as we are explicitly interested in finding outliers and do not aim for uniform distributions among priority levels. This construction can explain the observed continuous decrease in the number of priority level `I` classes that is in most cases associated with increased sizes of the applications. However, it has to be stated that the focus of our evaluation was on analyzing the constraint validation capabilities and evaluating the general feasibility of our approach. Hence, the vast amount of identified CEC violations was not fixed manually for being able to actually conduct the migration. Therefore, generalizability of the presented results regarding the prioritization function `prio` is limited and *G1* primarily demonstrates its application in the context of the selected software systems.

11.6.2 Case Study CC-2

The goal *G3* of the case study CC-2 aims at assessing the performance of the conformance checking approach. Binary classification is used to determine the approach's precision by manually inspecting reported CEC violations and recognizing true positives and false positives. Uncovering true positives and false positives by actually migrating an SUA to a chosen cloud environment is not covered by CC-2. As can be seen by the results of CC-1 (cf. Section 11.5.1), there exists a huge amount of CEC violations. Regarding an arbitrary CEC violation that is to be analyzed, the CEC violations that may impact this specific CEC violation or that hamper deploying the SUA in general would have to be fixed before.

Furthermore, CC-2 does not analyze true negative and false negative CEC violations and therefore, further quality metrics, such as the approach's accuracy (cf. Section 11.1.2), cannot be computed. Finding true or false negatives basically means that each part of a software system has to be analyzed for judging whether it raises a CEC violation despite no corresponding CEC violation is reported by CloudMIG. True negatives and false negatives

could be either determined by actually migrating an SUA to a chosen cloud environment, or by following a manual inspection approach. When migrating an SUA to a chosen cloud environment, a test case for each part would have to be built that verifies its correct functioning. All reported CEC violations would have to be fixed before migrating the SUA to the cloud environment. Only then, the created tests could be executed for spotting potential CEC violations that were not found by CloudMIG, or for confirming that a system part does actually not provoke CEC violations. CC-2 also mainly investigates CEC violations that are raised by `App4`'s own source code, whereas *M11* just examines samples of CEC violations that are caused by used third-party libraries.

11.6.3 Case Study CC-3

The case study CC-3 demonstrates that CloudMIG's conformance checking approach can be used in industrial contexts that may require the construction of additional KDM discoverers, CEC validators, and cloud profiles (*M12-M14*). Considering these artifacts, the construction of an additional KDM discoverer is the most complex. As described before, the built three-phase transformation of C# to KDM (cf. Section 10.1) currently omits several statement types that are not transformed to KDM. Hence, several CEC violations discovered with the tool NDepend cannot be detected with *CloudMIG Xpress*. However, this issue relates to the C# to KDM transformation and not to the conformance checking approach itself. Furthermore, the tool NDepend does not allow to include some of the prohibited method calls, that are contained in the corresponding CEC definition of CC-3, in the CQL statement used for querying `App6`'s source code (cf. [Wulf 2012]). Therefore, the detection rate of *CloudMIG Xpress* stated in Section 11.5.3 may actually be lower.

CC-3 also uses a single exemplary CEC definition for analyzing `App6`. This is due to the fact that (1) Microsoft Windows Azure constitutes an IaaS-based cloud environment and therefore does not exhibit as much CECs as PaaS-based cloud environments, and that (2) the focus is on the construction of the artifacts *M12-M14* (cf. [Wulf 2012]). The methodology used to evaluate

the CEC detection approach exhibits advantages, but also disadvantages. On the one hand, comparing the found CEC violations with matches obtained by the mature tool NDepend reinforces the reliability of the results as an automated approach is used. On the other hand, a structured manual inspection may reveal ambiguous or misinterpreted results.

11.7 Summary

This chapter describes the evaluation of CloudMIG's conformance checking approach for detecting an SUA's CEC violations. The GQM method is used to plan and perform the evaluation, i.e., coarse-grained evaluation goals are transformed to applicable metrics via specifying appropriate questions that review and detail the goals from multiple angles. GQM defines the following phases: planning, definition, data collection, and interpretation. Three case studies CC-1, CC-2, and CC-3 are used to evaluate different properties and viewpoints of CloudMIG's conformance checking approach. For each case study, all of GQM's phases are accomplished and the threats to validity are described. The rationale, setup, and basic outcome of each case study is briefly summarized in the following.

Case Study CC-1: CC-1 aims to show the applicability of the conformance checking approach and to investigate patterns of detected CEC violations. The case study extracts KDM models from five web-based open source Java applications, creates a cloud profile along with related CECs for the PaaS-based cloud environment Google App Engine (GAE) for Java, and accomplishes the CEC violation detection process with the help of *CloudMIG Xpress*. CC-1 demonstrates that the conformance checking approach is feasible and that it can successfully detect a wide range of CEC violations regarding enterprise software and a well-known, popular cloud environment. We observed that with bigger LOC application sizes comes a steady growth in not only the total number of classes that raise CEC violations, but also in the ratio of those responsible classes. Moreover, bigger LOC application sizes often correlate with higher overall violation densities.

When utilizing the prioritization function `prio`, bigger LOC application sizes lead to a decreased ratio of classes that are assigned high urgency priority level I. `prio` intentionally narrows the focus towards outliers. Furthermore, the responsible elements that raise CEC violations, as for example method calls or import statements, are often wide spread across the systems and are seldom condensed in bigger clusters. Additionally, the constraint violations' source can be primarily traced to used third-party libraries instead of their own source code for four of the five applications. Hence, identifying those third-party libraries that cause a great number of violations can be a first step when planning the adaptations that have to be accomplished. For example, in the case of GAE for Java there exists a list of libraries that are known to work with the cloud environment.[16] Therefore, trying to find suitable substitutions where the effort to integrate such a library is lower than reworking the identified violations can be a viable strategy. In addition to it, our prioritization function `prio` can be used as a decision support when planning the necessary refactorings to fix the constraint violations.

Case Study CC-2: CC-2 builds on experiments from Fenner [2011]. The case study aims at assessing the performance of the conformance checking approach. It reuses the cloud profile of GAE for Java and the KDM model from an application utilized in the context of CC-1. Binary classification is used to judge the quality of the conformance checking approach. The CEC violations detected by *CloudMIG Xpress* are inspected manually for revealing false alarms (false positives). The case study shows that the precision of the conformance checking approach is excellent.

Case Study CC-3: CC-3 builds on experiments from Wulf [2012]. The case study aims to demonstrate the feasibility of the conformance checking approach in an industrial case study. It creates artifacts that integrate in CloudMIG's tool architecture and that enable detecting CEC violations of a C#-based application from an industrial partner—a library for the risk assessment of financial products from the large German bank HSH Nordbank AG—regarding the IaaS-based cloud environment Microsoft Windows Azure (VM role). The three-phase transformation of C# to KDM (cf. Section 10.1)

[16] `http://code.google.com/p/googleappengine/wiki/WillItPlayInJava`

is used to extract a KDM model. CC-3 also provides a corresponding CEC validator plugin for *CloudMIG Xpress* that implements language-specific detection mechanisms for C#. Furthermore, a cloud profile for Microsoft Windows Azure is built and used as a basis for detecting CEC violations in the bank library's source code. The detected CEC violations are verified with the mature static analysis tool NDepend.

Chapter 12

Evaluation of the Deployment and Reconfiguration Optimization Approach

This chapter evaluates the simulation-based genetic algorithm CDOXplorer for optimizing the deployment and reconfiguration of software in the cloud. CDOXplorer is a component of the CloudMIG method that enables cloud users to find near-optimal CDOs for existing software systems. The evaluation aims to demonstrate CDOXplorer's applicability, to assess its performance, and to show that it produces competitive results and even can surpass other state-of-the-art optimization and search techniques.

Figure 12.1 illustrates important components of this evaluation. The evaluation employs quantitative experiments and follows the GQM method (cf. Section 11.1.1) to plan, perform, and analyze the experiments. As described in Section 10.4, CDOXplorer is built using the Opt4J framework for meta-heuristic optimization. The experiments examine the ERP system Apache OFBiz 10.04.[1] Basically, a KDM model is extracted from Apache OFBiz and a corresponding status quo deployment model is used as a foundation for searching near-optimal CDOs with the help of CDOXplorer.

Cloud profiles are modeled for the IaaS-based public cloud environments Amazon EC2 and Microsoft Windows Azure, as well as for a private Eucalyptus cloud. These cloud profiles are considered as potential target cloud environments. Hence, CDOXplorer searches appropriate CDOs of Apache

[1] http://ofbiz.apache.org/

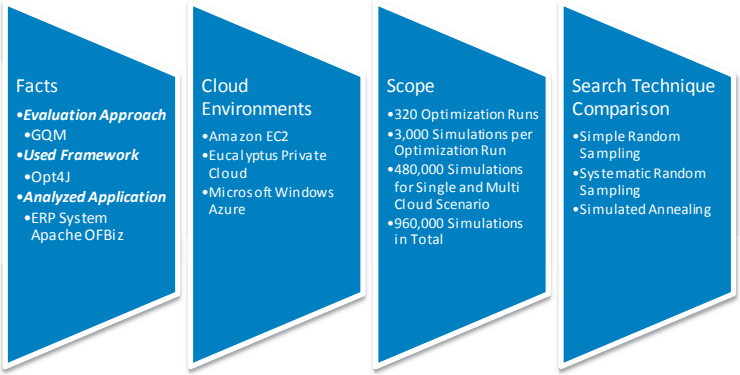


Figure 12.1. Important components of the CDO optimization evaluation

OFBiz' status quo deployment with resources, performance key figures, and pricing models defined in these cloud profiles. The evaluation presented in this chapter relies on extensive experiments that in total involve the execution of 320 optimization runs. Each optimization run of CDOXplorer comprises 3,000 simulation runs (a generation contains 50 individuals, CDOXplorer simulates 60 generations per default, cf. Section 8.1.1). Hence, 960,000 simulations are run in total, where 480,000 simulations are used for a single and a multi cloud scenario each. To assess the performance of CDOXplorer, the produced results are compared with those from the state-of-the-art optimization and search techniques simple random sampling, systematic random sampling, and simulated annealing.

This chapter utilizes and builds upon the following previously published work:

- *Frey, Sören and Fittkau, Florian and Hasselbring, Wilhelm, "Search-Based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud," in Proceedings of the 35th International Conference on Software Engineering (ICSE 2013), pp. 512-521, 2013.*

The remainder of this chapter is structured as follows. Section 12.1 describes the methodology and employed techniques used to evaluate CDOXplorer.

The next sections follow the phases of GQM. Section 12.2 describes GQM's planning phase in the context of the CDO optimization evaluation. Section 12.3 defines the goals, questions, and metrics according to GQM's definition phase. GQM's data collection phase, that performs the experiments and applies the defined metrics, is then described in Section 12.4. The gathered results are subsequently used to reason about CDOXplorer's quality and answer the posed questions according to GQM's interpretation phase in Section 12.5. The threats to validity are then described separately in Section 12.6, before Section 12.7 summarizes this chapter.

12.1 Methodology

The experiments for evaluating CDOXplorer are planned, performed, and analyzed according to the GQM method (cf. Section 11.1.1). Hence, the evaluation is structured along GQM's phases *planning*, *definition*, *data collection*, and *interpretation* in the corresponding Sections 12.2-12.5. This methodology section explains methods and techniques that are employed in the context of the evaluation.

The section is structured as follows. Section 12.1.1 describes other well-known search and optimization techniques that are used to compare and rank CDOXplorer's capabilities. The reasons for choosing these techniques are explained later in this chapter. As the optimization of CDOs constitutes a multi-objective optimization problem (cf. Section 8.1.1), CDOXplorer and all of the techniques chosen for comparison are implemented as multi-objective optimizers. The methods used for assessing the performance of CDOXplorer and of all the other utilized multi-objective optimizers are described in Section 12.1.2.

12.1.1 Search and Optimization Techniques

Several state-of-the-art search and optimization techniques are used for comparison purposes. However, as the objective values cannot be obtained

by solving functions analytically, some popular classes of approaches, such as gradient-based optimization methods (cf. Section 4.1.1), cannot be used. Though, direct search methods are suited for simulation-based optimization [Kolda et al. 2003]. Therefore, we use the simple yet effective stochastic algorithms *simple random sampling (SI-RS)* and *systematic random sampling (SY-RS)*, as well as the nature-inspired meta-heuristic *simulated annealing (SI-AN)*. Please note that the direct search methods simple and systematic random sampling are used for optimization purposes as, for example, described by Kolda et al. [2003]. That means, the specific search techniques are used to find optimal solutions. Similar to CDOXplorer, every run of SI-RS, SY-RS, and SI-AN comprises 3,000 simulation runs. To describe the different techniques, the concept of a *global optimization problem* is employed (cf. also Section 4.1.3 that introduces the notion of a *multi-objective optimization problem*). In the following, this concept is briefly described based on the definition from Schoen [1991].[2]

A **global optimization problem** is given by Formula 12.1.1.

$$f^* = \min_{x \in A} f(x) \tag{12.1.1}$$

Where A is a subset of $\mathbb{R}^N$, with $N > 0, N \in \mathbb{N}$. $f : A \to \mathbb{R}$ is the objective function and f^* is the global optimum of f over A. Furthermore, a general stochastic algorithm for global optimization consists of the following basic steps [Schoen 1991]:

- *Sampling step:* The strategy to draw the next sample $a \in A$. For example, by chance or due to a specific distribution function.
- *Optimization step:* Determines whether a local search is performed from a number of samples.
- *Stopping rule:* Specifies the condition for stopping the optimization. For example, after a fixed number of steps or fulfillment of a convergence criterion.

[2] In the following sections, the presented search and optimization techniques will be tailored to fit the multi-objective CDO optimization problem. As described in Section 8.1, there usually exists no single global optimum and a CloudMIG user is interested in the pareto-optimal set of CDOs for supporting the decision making process of selecting a specific CDO.

Simple random sampling, systematic random sampling, and simulated annealing are briefly explained in the following.

Simple Random Sampling (SI-RS) Simple random sampling (or pure random search) serves as a baseline algorithm. It is shown in Listing 12.1 (cf. [Schoen 1991]).

```
1 let f* := +∞
2 if a stopping criterion is satisfied then stop;
3 otherwise let x~𝕌(A) be a uniform random vector in A
4 let f* = min(f*, f(x))
5 go to 2
```

Listing 12.1. Simple random sampling (pure random search, based on [Schoen 1991])

After initializing the global optimum to $+\infty$ (we assume a minimizing optimizer) in Line 1, the algorithm subsequently draws random samples from A (x, Line 3) and applies f (Line 4) until a stopping criterion is satisfied. f^* is set to a new value in Line 4 if $f(x) < f^*$. After a stopping criterion is met, f^* constitutes the best found solution.

Systematic Random Sampling (SY-RS) Compared to simple random sampling, systematic random sampling (cf. [Iachan 1982]) does not draw each sample by chance, but due to a given pattern. Systematic random sampling assumes the set A (from above) to be a partially ordered set, such that its elements can be put in sequence according to an ordering relation. Listing 12.2 sketches a basic algorithm for systematic random sampling using the notation from [Schoen 1991].

```
1 let f* := +∞
2 let x~𝕌(A) be a uniform random vector in A
3 if a stopping criterion is satisfied then stop;
4 otherwise let f* = min(f*, f(x))
5 let x~𝕌(A) be the next k−th uniform vector in A
6 go to 3
```

Listing 12.2. Systematic random sampling (based on simple random sampling algorithm from [Schoen 1991])

After initializing f^* similarly to the simple random sampling algorithm (Line 1), a random sample $x \in A$ is drawn in Line 2. If no stopping criterion is fulfilled (Line 3), $f(x)$ is calculated for possibly updating f^* in Line 4. Systematic random sampling differs from simple random sampling as it uses a sampling interval k to draw subsequent samples in Line 5 (cf. [Iachan 1982]). If a fixed number of samples s is used as a stopping criterion, the sampling interval is often set to $k = \frac{|A|}{s}$. Hence, the algorithm traverses A in a structured and often circular manner, i.e., it can start from the beginning of the sequence if its end is reached.

Simulated Annealing (SI-AN) Simulated annealing is a nature-inspired meta-heuristic that mimics a cooling process in a physical system. At the beginning of the optimization procedure, it is more likely to overcome local optima but also more likely to unintentionally leave promising areas of the search space. In the course of the optimization process, it becomes more unlikely to move forward from an already found best solution (as the physical system cools down). Basically, the simulated annealing method tries to balance the opportunities associated with covering a wide area of the search space with the chances to perform a deep inspection in the neighborhood of already found, valuable results. Hence, simulated annealing assumes that some sort of locality can be defined for determining neighbors of a solution. Listing 12.3 shows a basic simulated annealing algorithm (cf. [Schoen 1991]).

The algorithm starts with drawing a random sample $x \in A$ (Line 1) and setting its objective value as the current optimum (Line 2). T (Line 3) is a monotonically non-increasing sequence that represents the notion of temperature. According to the process of slowly cooling materials in physical systems, the simulated annealing algorithm can overcome locally stable states. If "equilibrium is reached" (Line 6), the inner loop is left and the next lower temperature value in the sequence T is chosen (Line 12). As long as the equilibrium is not reached, a specific value of T remains unchanged and determines the probability of moving to other areas of the search space even if they might not seem to be promising. Hence, the inner loop starting from Line 5 describes the search for better solutions at a

```
1  let x ∈ A
2  let f* := f(x)
3  let T > 0 be the "initial temperature"
4  if a stopping criterion is satisfied then stop;
5  otherwise do
6     if "equilibrium is reached" then exit this loop;
7     let y be a random neighbor of x
8     let U~𝕌([0,1]) be a uniform random number in [0,1]
9     if exp(−(f(y) − f(x))/T) > U then let x := y
10    let f* = min(f*, f(x))
11    go to 6
12 let T be a new temperature value
13 go to 4
```

Listing 12.3. Simulated annealing (based on [Schoen 1991])

specific temperature. First, a neighbor of x (y) and then a random number from $[0, 1]$ (U) is chosen (Lines 7 and 8, respectively).

Line 9 determines whether y is accepted. If y produces a better result than x ($f(y) < f(x)$), y is accepted by all means. If y produces a worse result than x, y is only accepted with a probability that depends on the current value of T and U. As T is a monotonically non-increasing sequence, this probability decreases over time. If a stopping criterion is satisfied (Line 4), the algorithm terminates and f^* constitutes the best found approximation regarding the global optimum.

12.1.2 Performance Assessment of Multi-Objective Optimizers

In most multi-objective optimization scenarios, a single best solution, that clearly surpasses other solutions, does not exist, as the employed objective functions often conflict (cf. Section 4.1.3). Hence, multi-objective optimizers, such as CDOXplorer, are rather constructed to approximate the pareto optimal solution set. For assessing the performance and enabling a comparison of different multi-objective optimizers, there exists an array of performance measures that can be used to judge the quality of the ap-

proximation sets, e.g., the utility function indicators R1-R3 [Hansen and Jaszkiewicz 1998], spacing [Van Veldhuizen and Lamont 2000], and lines of intersection [Knowles and Corne 2000]. An overview and analysis of different performance measures is contributed by Zitzler et al. [2003]. There exist three classes of performance measures [Zitzler et al. 2003]:

- *Unary quality measures:* Each approximation set is assigned a single number that represents a certain quality aspect.
- *Binary quality measures:* Pairs of approximation sets are assigned a number that represents a certain quality aspect.
- *Attainment function approach:* Estimates the probability of achieving goals in the objective space from multiple approximation sets.

Unary quality measures are most widely utilized to cover a broader range of a multi-objective optimizer's quality aspects, several unary quality measures are most often used in combination [Zitzler et al. 2003]. The evaluation of CDOXplorer and the comparison with the other search and optimization techniques that are described in the previous Section 12.1.1 also use two different unary quality measures: *Inverted Generational Distance (IGD)* and *Hypervolume Indicator (HV)*. Hence, IGD and HV constitute the performance measures Ξ that are referred to in Section 8.1.1. Both measures are described in the following. As IGD is a modified version of the *Final Generational Distance (FGD)* measure, the latter is also briefly described below. We employ the following notation that was already utilized in Chapter 8.

PF_{True}:	The true pareto-optimal front
PF_{Known}:	The best known pareto-optimal front, i.e., the best pareto-optimal front that is known due to the execution of a specific multi-objective optimizer

Final Generational Distance (FGD) FGD measures the distance from PF_{Known} to PF_{True}. Using the notation from above, FGD is defined as follows (cf. [Van Veldhuizen and Lamont 2000]).

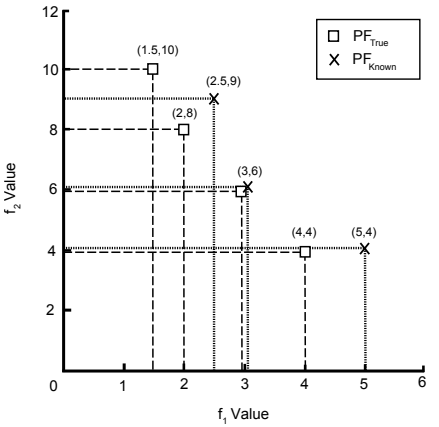


Figure 12.2. Pareto fronts PF_{Known}/PF_{True} example with two objective functions f_1 and f_2 (based on [Van Veldhuizen and Lamont 2000])

$$FGD = \frac{(\sum_{i=1}^{n} d_i^p)^{\frac{1}{p}}}{n} \tag{12.1.2}$$

n is the number of elements in PF_{Known}. For each $i \in PF_{Known}$, d_i denotes the Euclidean distance in objective space between i and the nearest element to i in PF_{True}. Smaller values for FGD are better. The evaluation of CDOXplorer uses the common variant of FGD having $p = 2$. An example from Van Veldhuizen and Lamont [2000] that uses two objective functions f_1 and f_2 is shown in Figure 12.2. For this example, FGD is given as follows.

$$d_1 = \sqrt{(2.5-2)^2 + (9-8)^2} = 1.118$$
$$d_2 = \sqrt{(3-3)^2 + (6-6)^2} = 0$$
$$d_3 = \sqrt{(5-4)^2 + (4-4)^2} = 1$$
$$FGD = \sqrt{1.118^2 + 0^2 + 1^2}/3 = 0.5$$

Inverted Generational Distance (IGD) Similarly to FGD, IGD aims to measure the distance between PF_{Known} and PF_{True}. However, IGD starts from

points in PF_{True} to calculate the Euclidean distance to the nearest points in PF_{Known}. Using the notation introduced above, IGD is defined as follows [Zhang et al. 2008].

$$IGD = \frac{\sum_{v \in PF^*_{True}} d(v, PF_{Known})}{|PF^*_{True}|} \tag{12.1.3}$$

Smaller values for IGD are better. PF^*_{True} is a set of uniformly distributed points in the objective space along PF_{True}. For each $v \in PF^*_{True}$, $d(v, PF_{Known})$ denotes the Euclidean distance between v and the nearest point in PF_{Known}. Revisiting the example from Figure 12.2 and setting $PF^*_{True} = PF_{True}$, IGD is given as follows.

$$d((1.5, 10), PF_{Known}) = \sqrt{(1.5-2.5)^2 + (10-9)^2} = 1.414$$
$$d((2, 8), PF_{Known}) = \sqrt{(2-2.5)^2 + (8-9)^2} = 1.118$$
$$d((3, 6), PF_{Known}) = \sqrt{(3-3)^2 + (6-6)^2} = 0$$
$$d((4, 4), PF_{Known}) = \sqrt{(4-5)^2 + (4-4)^2} = 1$$
$$IGD = (1.414 + 1.118 + 0 + 1)/4 = 0.883$$

Hypervolume Indicator (HV) HV does not measure the distance between PF_{Known} and PF_{True} directly. It rather measures the space that is dominated by a given pareto front (cf. [Zitzler and Thiele 1998]). Using the notation introduced above, HV is defined as follows [Fonseca et al. 2006].

Let $PF_{Known} = \{p^{(1)}, p^{(2)}, ..., p^{(n)}\}$ be the best known pareto front with $|PF_{Known}| = n$. Assuming d objectives and a minimization problem, the performance measure HV calculates the region that is simultaneously dominated by PF_{Known} and that is bounded above by a reference point $r \in \mathbb{R}^d$ with $r \geqslant (max_p(p_1), ..., max_p(p_d))$, where $p = (p_1, ..., p_d) \in PF_{Known}$, $PF_{Known} \subset \mathbb{R}^d$, and the relation $\geqslant$ applies componentwise.

In contrast to FGD and IGD, bigger values are better for HV. Figure 12.3 shows an example from Fonseca et al. [2006] that illustrates HV in the two-objective case. The gray-shaded area constitutes the hypervolume. Calculating the orthogonal polytope in higher dimensions—in the case many objectives are

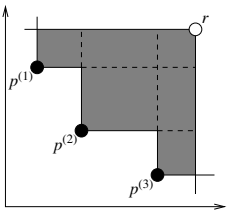


Figure 12.3. The Hypervolume Indicator (HV) in the two-objective case; $p^{(1)}$, $p^{(2)}$, and $p^{(3)}$ are elements of PF_{Known} (from [Fonseca et al. 2006])

used—is very compute-intensive and many corresponding algorithms have been proposed, for example, see [Fleischer 2003] and [Fonseca et al. 2006].

12.2 GQM Planning Phase

This section describes the GQM planning phase (cf. Section 11.1.1) for evaluating the deployment and reconfiguration optimization approach. Section 12.2.1 gives an overview on the general process and important activities for preparing, performing, and analyzing the included experiments. CDOXplorer and the further search and optimization techniques examined in the context of the experiments explore the CDO search space regarding the ERP system Apache OFBiz. Section 12.2.2 briefly describes Apache OFBiz. The IaaS-based cloud environments used within the scope of the experiments are described in Section 12.2.3. Finally, Section 12.2.4 analyzes the resulting search space for these scenarios.

12.2.1 Overview

The experiments investigate the performance of CDOXplorer for exploring the CDO search space and finding near-optimal solutions according to the GQM method. The quality of CDOXplorer is judged and compared against other state-of-the-art search and optimization techniques (cf. Section 12.1.1)

with the help of common performance measures for assessing arbitrary multi-objective optimizers (cf. Section 12.1.2). CDOXplorer is a central component of CloudMIG's tool architecture and implemented with the help of the Opt4J framework for meta-heuristic optimization (cf. Section 10.4). As described in Section 8.1.1, CDOXplorer requires several input models that have to be present for performing the CDO optimization process, such as an extracted KDM model and a status quo deployment model.

The **general process of the experimental evaluation** comprises the following steps.

1. Definition of g goals and q questions according to GQM that analyze specific characteristics regarding the applicability and performance of CDOXplorer
2. Derivation of m metrics from the q questions
3. Extraction of a KDM model from Apache OFBiz (cf. Section 12.2.2) with instances of KDM's code and inventory models (cf. Section 3.3.2)
4. Benchmarking all VM instance types of the cloud environments described in Section 12.2.3 with the help of the MIPIPS and weights benchmark (cf. Section 8.2)
5. Modeling cloud profiles incorporating, for example, specific pricing models and the benchmark results from 4.
6. Benchmarking server machines of a local cluster with the help of the MIPIPS and weights benchmark
7. Deployment of Apache OFBiz to the local cluster machines
8. Creation of a status quo deployment model that represents the deployment of Apache OFBiz (via its KDM model from 3.) to the local cluster (on premise deployment from 7.) and that includes the benchmark results from the local cluster machines (6.)
9. Definition of a varying usage pattern regarding the system Apache OFBiz
10. Applying the defined varying usage pattern to the local deployment of Apache OFBiz (with a load driver) and monitoring the response times

11. Creation of a workload profile (cf. Section 6.3.2) from the monitored log data (10.)
12. Execution of CDOXplorer and the further search and optimization techniques with all of the the prepared models (3., 5., 8., 11.)
13. Application of the defined m metrics upon the resulting approximations of the true pareto-optimal front
14. Analysis of the measurement results to investigate the performance of CDOXplorer and the attainment of defined g goals

12.2.2 Apache OFBiz

The E-Commerce/ERP system Apache OFBiz was already used for the initial experiment that verifies basic challenges which are addressed by CloudMIG (cf. Section 6.1.1). The evaluation of CDOXplorer utilizes Apache OFBiz 10.04. It is a web-based application, is available as open source software, and follows a three-tier architecture. The backend modules are mainly built using Java and Java EE technologies. The presentation tier can employ several different technologies, such as Java Server Pages (JSP). Data is stored in the data layer using a relational database system, for example, Oracle,[3] MySQL,[4] or PostgreSQL.[5]

Apache OFBiz is built of components that use a common data model and implement specific business functionalities. For example, Apache OFBiz provides components for order management, manufacturing management, warehouse management, promotion and pricing management, and customer management.

[3] http://www.oracle.com/us/products/database/overview/index.html

[4] http://www.mysql.com

[5] http://www.postgresql.org

12.2.3 Cloud Environments

Three cloud profiles are used for evaluating CDOXplorer. The cloud profiles model the public cloud environments Amazon EC2 and Microsoft Windows Azure (VM role), as well as a private cluster running the Eucalyptus cloud software.

Single and Multi Cloud Scenarios For analyzing single and multi cloud scenarios, the built cloud profiles are used in the two corresponding scenarios SC_S and SC_M as follows.

SC_S**:** Amazon EC2
SC_M**:** Amazon EC2, Microsoft Windows Azure, Eucalyptus cluster

Excerpts regarding the three cloud profiles can be found in Appendix B. For a description of the VM role of Microsoft Windows Azure we refer to Section 11.2.3, as the corresponding cloud profile is also used for evaluating CloudMIG's conformance checking approach. Amazon EC2 and the Eucalyptus cluster are briefly described below.

Amazon EC2 Amazon.com, Inc. (through their subsidiary Amazon Web Services, Inc.) can be seen as one of the pioneer cloud providers offering computing resources in an elastic, pay-as-you-go, and self-service fashion. Amazon EC2 undoubtedly not only helped to coin the term cloud computing, but also to trigger the emergence of a new research area (cf. Section 2.1.1).

The cloud environment Amazon EC2 (Elastic Compute Cloud)[6] offers a wide range of different VM instance types. At the time of writing, the portfolio includes, for example, VM instance types that are equipped with 613MB-68.4GB RAM, 160GB-3,370GB local instance storage, and 1-88 EC2 compute units.[7] There also exists a VM instance type that includes a ded-

[6] http://aws.amazon.com/ec2/

[7] An EC2 compute unit is a synthetic unit from Amazon Web Services, Inc. for measuring the computing capabilities of VM instance types.

Table 12.1. VM instance types of Eucalyptus for evaluating CDOXplorer (based on [Fittkau 2012])

Name	#CPU cores	RAM (MB)
m1.small	1	1,024
c1.medium	1	2,048
m1.xlarge	2	2,048
m1.large	4	2,048
c1.xlarge	6	2,048

icated General-Purpose Graphics Processing Unit (GPGPU) for addressing High Performance Computing (HPC) scenarios, for instance.

VM instances can be launched in several data centers that are called *regions*. Each region is further subdivided into one or more *availability zones* that constitute logical partitions for enabling fault isolation. Amazon EC2 provides regions in south america, EU, and in several locations in the US and asia. Furthermore, Amazon EC2 offers various services that can be used in conjunction with EC2. For example, *CloudWatch* for monitoring cloud resources, the NoSQL database *DynamoDB*, *Glacier* for archiving data, or *virtual private cloud* for connecting VM instances to an existing network over a VPN connection.

Eucalyptus Cluster Eucalyptus[8] is an open source software for building an IaaS-based cloud environment. It was already used in Section 6.1.1 in the context of the initial experiment for verifying basic challenges that are addressed by CloudMIG. In contrast, the cloud profile used for the evaluation of CDOXplorer originates from Fittkau [2012] and utilizes a different configuration of Eucalyptus. The VM instance types are configured as shown in Table 12.1.

Eucalyptus provides API-compatibility regarding Amazon EC2 and can in general be seen as an operating system for private and public clouds. The used cloud profile describes a private cloud environment of the Software Engineering Group, Kiel University, Germany, that is employed for evaluation purposes.

[8] http://www.eucalyptus.com

12.2.4 Search Space Analysis

The number of all CDOs ($|\Phi|$) that are feasible in the context of this evaluation for SC_S and SC_M can be calculated with Formula 8.6.1. As a detailed example regarding the multi cloud scenario SC_M (cf. Section 12.2.3), Table 12.2 lists the corresponding configuration of CDOXplorer that is used within the scope of this evaluation for SC_M. Table 12.2 also denotes the appropriate values for the variables of Formula 8.6.1. Thus, $|\Phi|$ is given for SC_M as follows (cf. Section 8.6).

$$\begin{aligned}
|\Phi| &= \sum_{i=1}^{c}((2^{|s|}-1)\cdot 391,910,401\cdot t_i v)^{a_i} \\
&= \sum_{i=1}^{c}(391,910,401\cdot t_i v)^{a_i} \\
&= (391,910,401\cdot 12\cdot 2)^3 + (391,910,401\cdot 5\cdot 2)^3 + \\
&\quad (391,910,401\cdot 5\cdot 2)^3 \\
&\approx 9.5253\cdot 10^{29}
\end{aligned} \tag{12.2.1}$$

Hence, examining the complete search space for these three cloud environments in the context of SC_M would imply simulating all of the $9.5253\cdot 10^{29}$ CDOs. Assuming that a simulation of a single CDO may take only one minute (which is a moderate assumption due to the experiences of this evaluation), simulating all potential solutions is not a viable option. As the cloud profile of Amazon EC2 includes many VM instance types, $|\Phi|$ for SC_S still comprises approx. $(391,910,401\cdot 12\cdot 2)^3 \approx 8.3214\cdot 10^{29}$ CDOs.

12.3 GQM Definition Phase

Based on the applications, utilized cloud environments (cf. Section 12.2), and the general objectives for evaluating the deployment and reconfiguration optimization, this section derives the evaluation goals, questions,

Table 12.2. Configuration of CDOXplorer in the context of the experimental evaluation for SC_M

Variable	Value	Comment
c	3	Three cloud environments are investigated (Amazon EC2, Eucalyptus cluster, and Microsoft Windows Azure (VM role)).
S	$\{S_1\}$	There exists one service, as Apache OFBiz is deployed to a single node in the status quo deployment model ($\|S\| = 1$).
v	2	The maximum number of VM instances that are allowed to start initially in a simulation of CDOSim.
t_1	12	The number of VM instance types defined in the cloud profile of Amazon EC2.
t_2	5	The number of VM instance types defined in the cloud profile of the Eucalyptus cluster.
t_3	5	The number of VM instance types defined in the cloud profile of Microsoft Windows Azure (VM role).
$a_1 - a_3$	3	Maximum of three concurrent node configurations for Amazon EC2, Eucalyptus cluster, and Microsoft Windows Azure (VM role).

and metrics according to GQM's definition phase. GQM goal definition templates [Solingen 1999] are used for defining the GQM model. This section is structured as follows. Section 12.3.1 gives basic definitions that constitute prerequisites for specifying the GQM goals, questions, and metrics. The corresponding GQM model is then described in the next Section 12.3.2.

12.3.1 Basic Definitions

As described in Section 12.1.2 for FGD and IGD, measures for assessing the performance of multi-objective optimizers often judge the approximation of PF_{Known} towards the true pareto-optimal front (PF_{True}). In the case of evaluating CDOXplorer, there exist different true pareto-optimal fronts for SC_S and SC_M. However, they can only be obtained by simulating all CDOs that are feasible for SC_S and SC_M. This is not possible in a reasonable amount of time (cf. Section 12.2.4). As stated before, 320 optimization runs are conducted with 960,000 simulations in total. We therefore obtain 480,000 simulations for SC_S and SC_M. In both cases, the overall 480,000 simulations are utilized to compute a pareto-optimal front called OPF_{Best}, that is used as a (PF_{True}) proxy for evaluating the respective scenario. OPF_{Best} is denoted $OPF_{Best}(SC_S)$ for SC_S and $OPF_{Best}(SC_M)$ for SC_M.

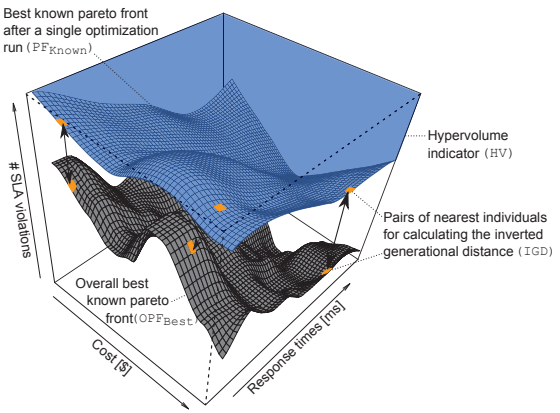


Figure 12.4. The measures Inverted Generational Distance (IGD) and Hypervolume Indicator (HV, colored blue) are used for evaluating the performance of CDOXplorer (cf. [Frey et al. 2013b])

Figure 12.4 illustrates the utilized performance measures IGD and HV in conjunction with OPF_{Best}. The pareto-optimal front PF_{Known}, that is also shown in Figure 12.4, is the result of each execution of CDOXplorer or one of the other examined search and optimization techniques (cf. Section 12.1.1). IGD measures the approximation of PF_{Known} to OPF_{Best}. Starting from the individuals in OPF_{Best}, the nearest individuals in PF_{Known}, in terms of the Euclidean distance, are used for calculating the IGD metric. HV determines the hypervolume covered by PF_{Known} in the three-dimensional objective space that is spanned by CDOXplorer's objectives: costs, response times, and number of SLA violations.

12.3.2 GQM Model

This section describes the GQM model for evaluating CDOXplorer. Figure 12.5 shows an overview on the GQM model. It consists of the two goals *G1* and *G2*. The GQM model is described with the help of GQM goal definition templates [Solingen 1999] in the following.

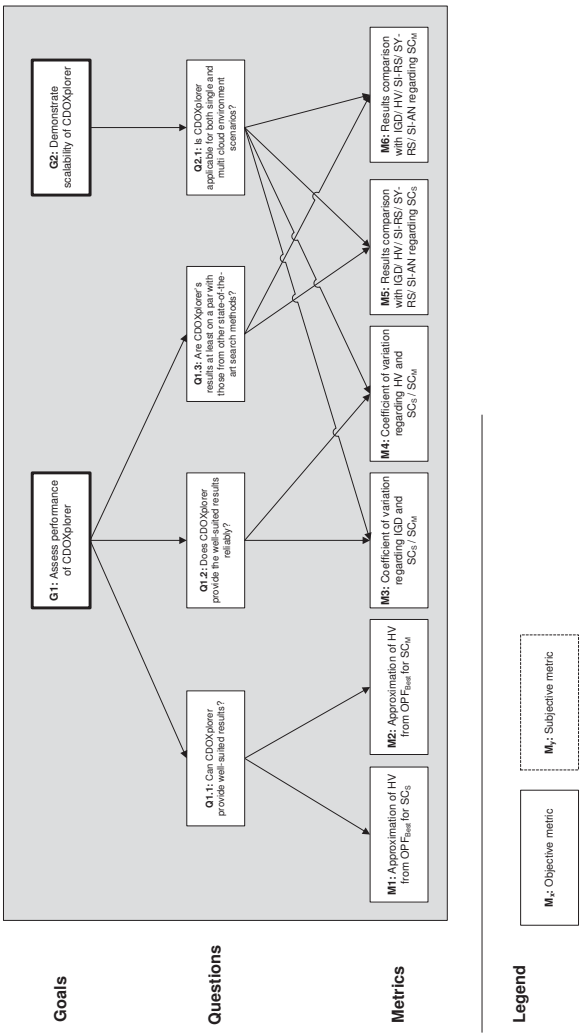


Figure 12.5. Overview on the GQM model for evaluating the deployment and reconfiguration approach

G1: Assess performance of CDOXplorer

Goal	**G1**	
	Purpose	Assess
	Issue	performance of
	Object (process)	CDOXplorer
	Viewpoint	from a CloudMIG user viewpoint
Question	**Q1.1**	Can CDOXplorer provide well-suited results?
Metrics	**M1**	Approximation of HV from OPF_{Best} for SC_S
	M2	Approximation of HV from OPF_{Best} for SC_M
Question	**Q1.2**	Does CDOXplorer provide the well-suited results reliably?
Metrics	**M3**	Coefficient of variation regarding IGD and SC_S / SC_M
	M4	Coefficient of variation regarding HV and SC_S / SC_M
Question	**Q1.3**	Are CDOXplorer's results at least on a par with those from other state-of-the-art search methods?
Metrics	**M5**	Results comparison with IGD/ HV/ SI-RS/ SY-RS/ SI-AN regarding SC_S
	M6	Results comparison with IGD/ HV/ SI-RS/ SY-RS/ SI-AN regarding SC_M

The goal *G1* aims at assessing the performance of CDOXplorer. For judging the quality of the genetic algorithm, three GQM questions *Q1.1-Q1.3* are derived from *G1*. The questions refine and further analyze different aspects that contribute to the overall performance of CDOXplorer.

The first relevant aspect is analyzed by *Q1.1* that investigates CDOXplorer's capabilities for providing well-suited results. As results, CDOXplorer delivers pareto-optimal fronts. Hence, the quality of these pareto-optimal fronts has to be examined. The evaluation of this criterion is of particular importance. The simulations used for our simulation-based algorithm are computationally expensive. Hence, we strictly limited the number of generations and population size. This could affect CDOXplorer's capability for producing well-suited approximations of pareto-optimal fronts.

Q1.2 analyzes the reliability of CDOXplorer for delivering well-suited results. This aspect also substantially contributes to the overall quality of our genetic algorithm, as the non-determinism used in CDOXplorer, for

example, regarding the selection of crossover points (cf. Section 8.4.1), could possibly lead to considerable variations among optimization runs.

The last aspect that is important for judging CDOXplorer's performance is the quality of the produced pareto-optimal fronts in comparison to the results of other state-of-the-art search and optimization techniques. Therefore, the GQM question *Q1.3* investigates the competitiveness of CDOXplorer and compares its results to those of SI-RS, SY-RS, and SI-AN.

G2: Demonstrate scalability of CDOXplorer

Goal	**G2**	
	Purpose	Demonstrate
	Issue	scalability of
	Object (process)	CDOXplorer
	Viewpoint	from a CloudMIG user viewpoint
Question	**Q2.1**	Is CDOXplorer applicable for both single and multi cloud environment scenarios?
Metrics	**M3**	Coefficient of variation regarding IGD and SC_S/SC_M
	M4	Coefficient of variation regarding HV and SC_S/SC_M
	M5	Results comparison with IGD/ HV/ SI-RS/ SY-RS/ SI-AN regarding SC_S
	M6	Results comparison with IGD/ HV/ SI-RS/ SY-RS/ SI-AN regarding SC_M

The goal *G2* aims at demonstrating the scalability of CDOXplorer. That means, if CDOXplorer can retain its performance when an increasing number of cloud profiles is considered or, otherwise, if a potential performance degradation is still acceptable. The scalability is of particular interest as the number of generations and the population size remain stable but the search space size grows linearly with each new cloud profile.

Metrics As all existing performance measures for assessing multi-objective optimizers exhibit different strengths and weaknesses and therefore are often used in combination [Zitzler et al. 2003], the evaluation does not rely on a single measure. Instead, the two popular unary quality measures IGD and HV are used (cf. Section 12.1.2). GQM distinguishes objective and subjective metrics (cf. Section 11.3). As can be seen in Figure 12.5, the evaluation of CDOXplorer only utilizes the objective metrics *M1-M6* that in

turn build on the measures HV and IGD. For defining *M1-M6*, we employ the following notation.

$\triangle(x)$	Median of x
$PF_{(sc,st)}$	The set of pareto-optimal fronts delivered by the experiments for a specific *scenario (sc)* and a specific *search technique (st)*. $sc = \{SC_S, SC_M\}$, $st = \{CDOXplorer, SI-RS, SY-RS, SI-AN\}$. Example: $PF_{(SC_S,SY-RS)}$ is the set of pareto-optimal fronts delivered by the experiments for the single cloud scenario (SC_S) in combination with the SY-RS search technique.
$pm(PF_{(sc,st)})$	The performance measurement results for applying the performance measure $pm = \{HV, IGD\}$ to all elements in $PF_{(sc,st)}$ (defined above). Hence, $pm(PF_{(sc,st)})$ constitutes a set. Example: $HV(PF_{(SC_S,SY-RS)})$ is the set of hypervolumes of all pareto-optimal fronts in $PF_{(SC_S,SY-RS)}$.
$cv_{(sc,pm,st)}$	*Coefficient of variation* for performance measurement results of $pm(PF_{(sc,st)})$ (defined above). Coefficient of variation gives information about the relative dispersion regarding a sample's mean value μ and is defined by $cv = \frac{\sigma}{\mu}$, with σ being the sample's standard deviation. Hence, $cv_{(sc,pm,st)}$ computes the coefficient of variation from all measurement results in $pm(PF_{(sc,st)})$.

Using the introduced notation, the GQM metrics *M1-M6* are described in the following.

- **[M1] Approximation of HV from OPF_{Best} for SC_S:** OPF_{Best} represents the overall best pareto-optimal front and therefore, the measurement result of HV regarding OPF_{Best} and SC_S constitutes the largest covered hypervolume area regarding SC_S. *M1* computes CDOXplorer's approximation of HV from OPF_{Best} for SC_S as follows.

$$\frac{\triangle(HV(PF_{(SC_S,CDOXplorer)}))}{HV(OPF_{Best}(SC_S))} \tag{12.3.1}$$

  M1 calculates the median HV value regarding the pareto-optimal fronts that are produced by CDOXplorer for SC_S and determines the degree this median HV value can approximate the HV reference value of OPF_{Best}.

- **[M2] Approximation of HV from OPF_{Best} for SC_M:** *M2* is similar to *M1*, but is determines CDOXplorer's approximation of HV from OPF_{Best} regarding the multi cloud scenario SC_M instead of the single cloud scenario SC_S. Hence, *M2* is computed as follows.

$$\frac{\triangle(HV(PF_{(SC_M,CDOXplorer)}))}{HV(OPF_{Best}(SC_M))} \tag{12.3.2}$$

- **[M3] Coefficient of variation regarding IGD and SC_S/SC_M:** The third metric `M3` calculates the coefficient of variation (cv) that gives information about the relative dispersion regarding a sample's mean value μ. As the performance measure IGD delivers results in an artificial and incomparable unit, we convert the standard deviation σ for SC_S and SC_M in combination with IGD to relative and therefore comparable values. Using the notation introduced above, these values are given by $cv(SC_S, IGD, CDOXplorer)$ for our single cloud scenario and $cv(SC_M, IGD, CDOXplorer)$ for our multi cloud scenario.
- **[M4] Coefficient of variation regarding HV and SC_S/SC_M:** *M4* is similar to *M3*, but it determines the coefficient of variation for SC_S and SC_M regarding HV instead of IGD. HV delivers results in an artificial and incomparable unit as well. Using the notation introduced above, *M4* computes $cv(SC_S, HV, CDOXplorer)$ for the single cloud scenario and $cv(SC_M, HV, CDOXplorer)$ for the multi cloud scenario.
- **[M5] Results comparison with IGD/ HV/ SI-RS/ SY-RS/ SI-AN regarding SC_S:** *M5* enables to compare CDOXplorer with other search and optimization techniques. Basically, it delivers measurement results regarding IGD and HV for CDOXplorer and all of the other search and optimization techniques regarding the single cloud scenario SC_S. Using the notation introduced above, *M5* calculates the arithmetic mean, standard deviation, median, min, and max values of $pm(PF_{(SC_S,st)})$, with $pm = \{HV, IGD\}$ and $st = \{CDOXplorer, SI-RS, SY-RS, SI-AN\}$.
- **[M6] Results comparison with IGD/ HV/ SI-RS/ SY-RS/ SI-AN regarding SC_M:** *M6* is similar to *M5*, but it uses results from the multi cloud scenario SC_M instead of SC_S to calculate the arithmetic mean, standard deviation, median, min, and max values of $pm(PF_{(SC_M,st)})$, with $pm = \{HV, IGD\}$ and $st = \{CDOXplorer, SI-RS, SY-RS, SI-AN\}$.

12.4 GQM Data Collection Phase

The basic procedure, important artifacts, and tools utilized to gather the necessary data for applying the previously specified GQM metrics (cf. Section 12.3) were already sketched in Section 12.2.1 in the context of the general process of the experimental evaluation. This section details several components of this process that are relevant for applying the defined metrics and describes the corresponding GQM data collection phase.

CDOXplorer, SI-RS, SY-RS, and SI-AN are guaranteed to run the same number of simulations, i.e., there exists no timeout. As explained in Section 12.1.1, the algorithm SI-RS creates 3,000 CDO individuals by chance and serves as a baseline algorithm. SY-RS also produces 3,000 individuals by chance, but works in a different way. It iterates over all f feasible CDO candidates and randomly selects the k-th CDO from the $[0, floor(\frac{f}{3{,}000})]$ interval at the beginning of an optimization run, where $floor(x)$ rounds $x \in \mathbb{R}_+$ down to the next natural number. The next CDO is then given by the $(k + floor(\frac{f}{3{,}000}))$-th candidate and so forth (cf. Section 12.1.1). The third algorithm used for comparison purposes is SI-AN. Basically, it mimics the temperature cooling process of materials. To emulate such a cooling process in our problem context and to tailor it to the CDO optimization domain, we reuse the mutation sub operators introduced in Section 8.4.2 as illustrated in Figure 12.6.

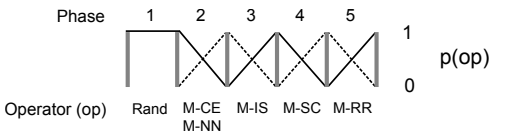


Figure 12.6. Simulated annealing comprises five phases and reuses CDOXplorer's mutation sub operators in phases 2-5. $p(op)$ denotes the probability an operator (op) is used in a specific phase. The *Rand* operator delivers feasible CDOs with random chromosomes (cf. [Frey et al. 2013b]).

The temperature is adapted according to five phases. The first phase delivers feasible CDOs with random chromosomes (cf. Section 8.3). The phases 2-5 utilize mutation sub operators with specific probabilities to reduce

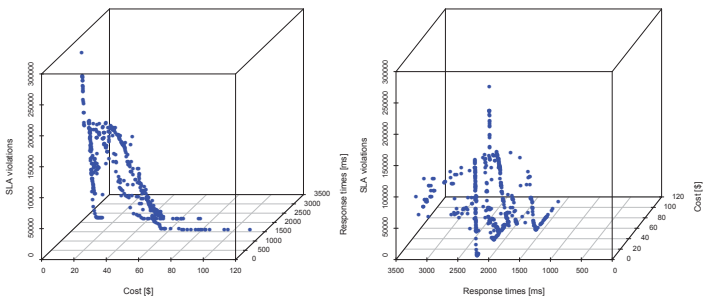


Figure 12.7. The best known pareto front for SC_S: $OPF_{Best}(SC_S)$ (cf. [Frey et al. 2013b])

disruptive modifications over time. For example, triggering the start of an additional VM instance due to exceeding a CPU utilization threshold (M-RR) five minutes later, very likely has a lower impact than using a different cloud environment (M-CE). Hence, SI-AN uses the M-CE operator in an earlier phase than M-RR, for instance. SI-AN also enables a smooth transition between the phases and ensures diversity of the CDO search space by fading in and out the usage of mutation sub operators in subsequent phases instead of simply using a single operator. Hence, SI-AN decreases the probability of utilizing an established operator with the same rate it uses to raise the probability a new operator is used.

CDOXplorer utilizes 60 generations with populations of 50 individuals (cf. Section 8.1.1). Hence, CDOXplorer applies 3,000 simulations in a single run. The runs for CDOXplorer, SI-RS, SY-RS, and SI-AN are each repeated 40 times for single (SC_S) as well as for multi cloud scenarios (SC_M). Thus, the experiments conduct 320 optimization runs with 960,000 simulations in total. We therefore obtain 480,000 simulations for the single and multi cloud scenario that are used to compute two pareto-optimal fronts that are set to OPF_{Best} for evaluating the respective scenarios (cf. Section 12.3.1). The first of these pareto-optimal fronts ($OPF_{Best}(SC_S)$) that results from 480,000 simulations is shown in Figure 12.7.

As described in Section 12.2.1, the open source ERP system Apache OFBiz 10.04 is deployed on a machine of our local cluster. Customers that browse

the webstore and put products in their shopping carts are emulated by producing workload according to a typical day/night usage pattern. More customers visit the webstore in the evening instead in the morning hours and the traffic largely reduces at night. The measured response times and the MIPIPS value of our hardware are then used in a workload profile for generating CDOs and driving the CDOSim simulations. The SLA violation threshold is set to 2s. For Eucalyptus, we define a synthetic cost model where the prices for VMs follow the capabilities of our VM instance types.

12.5 GQM Interpretation Phase

This section answers the previously formulated GQM questions (cf. Section 12.3) with the help of the measurement results of the corresponding GQM metrics. Hence, the section describes the GQM interpretation phase based on the pareto-optimal fronts that are produced by the overall 320 optimization runs of CDOXplorer, SI-RS, SY-RS, and SI-AN.

G1: Assess performance of CDOXplorer

Three GQM questions *Q1.1-Q1.3* are derived from *G1*. These questions are investigated below with the help of the GQM metrics *M1-M6* and the corresponding measurements gathered in the GQM data collection phase.

Q1.1: *Can CDOXplorer provide well-suited results?*

M1 and *M2* analyze the quality of the results produced by CDOXplorer. The metrics investigate the degree the hypervolumes of OPF_{Best} can be approximated by CDOXplorer for SC_S and SC_M, respectively. For SC_S and SC_M, the HV of OPF_{Best} is 0.462 and 0.573, respectively, whereas CDOXplorer achieves 0.448 and 0.565. Thus, the actual quality of the found pareto optima are sufficiently well-suited, as these results turn into 96.96% (*M1*) and 98.56% (*M2*) approximation of OPF_{Best} for the single and multi cloud scenario SC_S and SC_M, respectively.

Table 12.3. Coefficient of variation for analyzed scenarios regarding CDOXplorer

Metric	SC_S	SC_M
IGD	7.77%	15.84%
HV	0.46%	0.32%

Q1.2: *Does CDOXplorer provide the well-suited results reliably?*

To assess whether CDOXplorer can reliably provide well-suited results, the metrics *M3* and *M4* calculate the coefficient of variation for all combinations of SC_S, SC_M, IGD, and HV (cf. Section 12.3). The corresponding results are shown in Table 12.3. The results for SC_S and SC_M vary in a band of 7.77% and 15.84% around the mean value μ for IGD (*M3*), and of 0.46% and 0.32% around μ for HV (*M4*), respectively. Hence, IGD results in the single cloud scenario are up to 3.85% lower or higher than μ, for instance. Thus, the results indicate that CDOXplorer can reliably find well-suited solutions. However, the value for IGD increases for a higher number of cloud profiles, but without further experiments, it is not possible to judge whether this observation constitutes an actual trend.

Q1.3: *Are CDOXplorer's results at least on a par with those from other state-of-the-art search methods?*

This GQM question addresses the competitiveness with other state-of-the-art search approaches by comparing CDOXplorer with SI-RS, SY-RS, and SI-AN. Table 12.4 lists the results for the performance measures IGD and HV for the single cloud scenario SC_S. Table 12.5 shows these results for SC_M. The tables list the arithmetic mean, standard deviation, median, min, and max values of 40 complete, repeated optimization runs for each combination of performance measure, scenario, and search method. Bigger values are better for HV but worse for IGD. All best mean and median values are set in bold. As can be seen, CDOXplorer outperforms all other search methods in SC_S and also in SC_M.

We use the Mann-Whitney non-parametric test to evaluate statistical significance. The null hypothesis H_0 states that the results from CDOXplorer

Table 12.4. Performance of the search methods regarding the single cloud scenario (SC_S)

Metric		Search Method			
		CDOXplorer	SI-RS	SY-RS	SI-AN
IGD	Mean	**2.70E-02**	3.67E-02	4.11E-02	3.28E-02
	SD	2.10E-03	2.13E-03	3.61E-03	2.85E-03
	Median	**2.72E-02**	3.65E-02	4.21E-02	3.20E-02
	Min (best)	2.16E-02	3.34E-02	3.40E-02	2.76E-02
	Max (worst)	3.03E-02	4.07E-02	4.83E-02	3.95E-02
HV	Mean	**4.48E-01**	4.41E-01	4.41E-01	4.44E-01
	SD	2.08E-03	1.96E-03	2.89E-03	2.09E-03
	Median	**4.48E-01**	4.40E-01	4.41E-01	4.44E-01
	Min (worst)	4.44E-01	4.36E-01	4.35E-01	4.40E-01
	Max (best)	4.54E-01	4.46E-01	4.46E-01	4.48E-01

Table 12.5. Performance of the search methods regarding the multi cloud scenario (SC_M)

Metric		Search Method			
		CDOXplorer	SI-RS	SY-RS	SI-AN
IGD	Mean	**3.08E-02**	7.18E-02	8.03E-02	3.37E-02
	SD	4.88E-03	2.78E-03	4.41E-03	4.50E-03
	Median	**3.12E-02**	7.17E-02	7.90E-02	3.38E-02
	Min (best)	2.13E-02	6.58E-02	7.23E-02	2.52E-02
	Max (worst)	4.16E-02	7.76E-02	8.88E-02	4.67E-02
HV	Mean	**5.65E-01**	5.20E-01	5.18E-01	5.63E-01
	SD	1.82E-03	1.95E-03	2.43E-03	1.68E-03
	Median	**5.65E-01**	5.20E-01	5.17E-01	5.63E-01
	Min (worst)	5.61E-01	5.16E-01	5.13E-01	5.61E-01
	Max (best)	5.70E-01	5.25E-01	5.25E-01	5.68E-01

cannot be distinguished from those of SI-RS, SY-RS, and SI-AN. Using the Mann-Whitney test and a Bonferroni correction ($\alpha_1 = 0.016$) for multiple comparisons, H_0 is rejected with significance level (α) 0.05 for all combinations of performance measures with SC_S and SC_M. Thus, we can quantify the degree CDOXplorer performs better and compare the medians of SI-RS, SY-RS, and SI-AN to those of CDOXplorer. Figure 12.8 shows the corresponding fractions with regard to the medians, e.g., for the SC_M scenario, the median for IGD is over 60% lower than that of SY-RS.

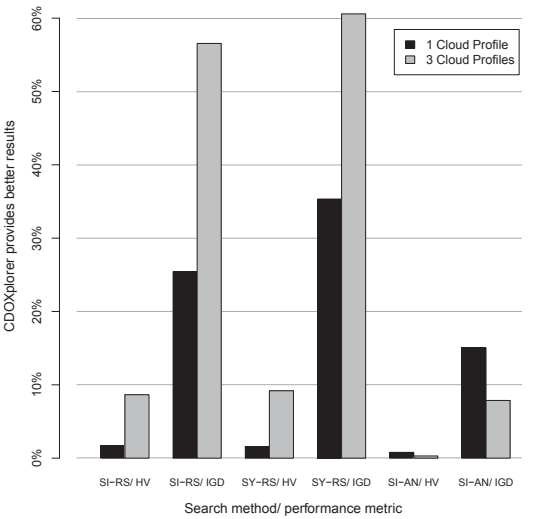


Figure 12.8. CDOXplorer advantage relative to other approaches (cf. [Frey et al. 2013b])

G2: Demonstrate scalability of CDOXplorer

The GQM question *G2* is refined by question *Q2.1* that investigates CDOXplorer's scalability with the help of the single cloud scenario SC_S and the multi cloud scenario SC_M.

Q2.1: *Is CDOXplorer applicable for both single and multi cloud environment scenarios?*

The GQM question *Q2.1* analyzes CDOXplorer's scalability. That means, if CDOXplorer can retain its performance when more cloud profiles are considered for providing CDO candidates or if a potential performance degradation is still acceptable. As described in the context of GQM question *1.2* above, the value for the coefficient of variation (with IGD) and the value for the IGD performance measure itself grows when CDOXplorer processes SC_M. However, the HV value and the coefficient of variation even become better, when more cloud profiles are used.

Furthermore, it is useful to compare these observations with those from the other search methods. CDOXplorer in all cases provides the best results. Considering SI-RS and SY-RS, their values for the IGD metric grow even more and nearly doubled when using SC_M. SI-AN also suffers from a deterioration when transitioning from the SC_S to the SC_M scenario. Though, the IGD values increase less. The values for HV show a similar development. CDOXplorer is better than all other search methods and the scalability is better compared with SI-RS and SY-RS, but SI-AN exhibits a slightly higher improvement.

12.6 Threats to Validity

There are several issues that form a threat to validity. First, we only consider three cloud profiles, as their construction is not trivial and the VM instance types of additional cloud providers need to be benchmarked. This involves potential expenses. Furthermore, the experiments are very time-consuming, our parallelized but multiply repeated optimization runs already took over two months to finish. However, as indicated by the results of GQM question *2.1* (cf. Section 12.5), scalability is a worthwhile area for further analyses.

Restrictions also have to be made concerning the workload profile and the studied enterprise software. In each case, just one sample is used. This is due to the fact that further optimization runs imply even more time-consuming simulations. Therefore, the evaluation strives after using representative instances. Day/night usage patterns with higher and lower demand are frequently found for enterprise systems. Furthermore, Apache OFBiz is very popular and widespread.

There could also exist other optimization methods that provide better results than SI-RS, SY-RS, and SI-AN we used for evaluation. Though, tailoring optimization methods for our context is time-consuming and not straightforward. Quite similarly, there could exist better ways to tailor SI-AN instead of reusing the mutation sub operators. Especially in the light that SI-AN partially comes near to CDOXplorer's results. Further threats to validity

arise from the synthetic cost model for our private Eucalyptus cloud and that SLAs are usually defined in terms of percentile ranges when considering response times. However, aligning the VM prices to the capabilities of the VM instance types is omnipresent with respect to public cloud environments. Altering the absolute threshold into percentiles for defining the SLA objective can be easily done.

12.7 Summary

This chapter describes the evaluation of CloudMIG's deployment and reconfiguration optimization approach, i.e., the simulation-based genetic algorithm CDOXplorer. The evaluation employs the GQM method and is structured along GQM's four phases *planning*, *definition*, *data collection*, and *interpretation*. Extensive experiments are conducted to validate the genetic algorithm's feasibility, scalability, and competitiveness. The open source ERP system Apache OFBiz is used as a test application. In combination with the public cloud environments Amazon EC2 and Microsoft Windows Azure and a private cloud using the Eucalyptus cloud software, the experiments investigate CDOs regarding Apache OFBiz. To examine CDOXplorer's scalability, two different scenarios are defined. The single cloud scenario SC_S only incorporates Amazon EC2, the multi cloud scenario SC_M comprises all three cloud environments.

The evaluation also utilizes three further state-of-the-art search and optimization techniques for comparison purposes. The stochastic algorithms simple random sampling (SI-RS) and systematic random sampling (SY-RS) constitute direct search methods that are tailored for optimizing CDOs. The third algorithm is the nature-inspired meta-heuristic simulated annealing (SI-AN). For assessing the performance of the multi-objective optimizers CDOXplorer, SI-RS, SY-RS, and SI-AN, the popular unary quality measures hypervolume and inverted generational distance are used.

CDOXplorer is implemented using the Opt4J framework for meta-heuristic optimization and, as described in Chapter 8, relies on CDOSim to evaluate

the fitness of CDO candidates via simulations. The simulations of CDOXplorer and of the other search and optimization techniques are driven by a typical day/night usage pattern. It emulates customers that browse the Apache OFBiz webstore and put products in their shopping carts.

The experiments comprise 320 optimization runs having 3,000 simulations each. In total, 960,000 simulations are performed in our test environment lasting, though running several optimization runs in parallel, over two months. The evaluation shows that CDOXplorer can be successfully applied to optimize CDOs. It (1) performs better than all competing search techniques regarding the hypervolume and inverted generational distance measures and (2) scales better than two of the three search techniques. Overall, it can find solutions that are up to 60% better than those of the other search techniques.

Chapter 13

Related Work

Related work comes from three areas that are shown in Figure 13.1. The first area is *cloud migration*, as our approach CloudMIG addresses the migration of existing enterprise applications to cloud infrastructures and platforms. Corresponding to the contributions of this thesis within the context of cloud migration (cf. Section 1.3), the second and third area that contribute related work are given by *conformance checking* and *deployment and reconfiguration optimization* of software in the cloud. To examine and present the related work, each area is processed according to the following steps.

1. Refine Each area that contributes related work is refined to subareas. As can be seen in Figure 13.1, cloud migration is refined to four subareas, for instance.

2. Characterize Devise general characteristics of each subarea. These characteristics can be seen as properties of the subarea suited for comparing corresponding related work.

3. Overview Overview the related work of each subarea in tabular form using the devised characteristics from step 2.

4. Analyze Examine the related work of each subarea and describe details and rationales regarding the overview table from step 3.

This chapter utilizes and builds upon the following previously published work:

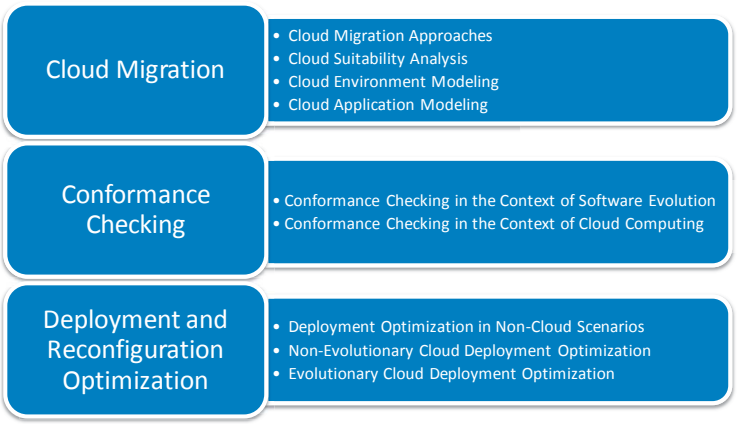


Figure 13.1. Related work overview

- *Frey, Sören and Fittkau, Florian and Hasselbring, Wilhelm, "Search-Based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud," in Proceedings of the 35th International Conference on Software Engineering (ICSE 2013), pp. 512-521, 2013.*

This chapter is structured as follows. Section 13.1 describes the related work concerning the cloud migration area. The related work regarding conformance checking and deployment and reconfiguration optimization is then presented in the Sections 13.2 and 13.3, respectively.

13.1 Cloud Migration

In conjunction with the raising popularity of the cloud computing paradigm, there emerges an increasing need for leveraging cloud technology for existing software systems as well. Migrating software to the cloud that was initially built for a different platform has become a vibrant research area. With regard to CloudMIG, this section describes related work from the following subareas.

Cloud Migration Approaches: Similar to CloudMIG, cloud migration approaches define a process for migrating software systems to IaaS or PaaS-based cloud environments. Likewise, they also cover all basic activities of the horseshoe model of reengineering (cf. Figure 6.10). However, they may also put emphasis on specific activities and merely sketch others, as is done with CloudMIG's activity A6 (transformation, cf. Section 6.3).

Cloud Suitability Analysis: This subarea includes related work that proposes methods and techniques for evaluating the suitability of existing software systems for employing cloud computing technologies. For example, CloudMIG provides the CSA hierarchy for assessing a system's suitability and alignment regarding a specific cloud environment.

Cloud Environment Modeling: CloudMIG's CEM provides means for representing cloud environments and modeling specific characteristics, such as provided hardware resources, services, and pricing models. This subarea describes related work for modeling cloud environments.

Cloud Application Modeling: Besides CEM's elements that enable to model cloud environments, CEM provides means for representing deployed software artifacts and components of an existing system. Hence, CEM also provides elements for modeling cloud-based applications. This subarea describes related work regarding cloud-based application models. However, similarly to CloudMIG, several approaches also provide modeling capabilities for both, cloud environments and cloud-based applications. Those approaches are described in the context of the cloud environment modeling subarea or the cloud application modeling subarea depending on their specific focus.

13.1.1 Cloud Migration Approaches

Related work from further cloud migration approaches is listed in Table 13.1. The related work is examined according to the following six characteristics. The abbreviations in parentheses are used for referencing purposes.

Table 13.1. Related work for cloud migration approaches

Approach	Ind.[1]	Mult.[2]	Conf.[3]	Opt.[4]	Usg.[5]	Exec.[6]	Comments
Bergmayr et al. [2013]	✓	(✓)	-	(✓)	-	(✓)	ARTIST project; Uses MDE techniques; Pre- and post-migration phases
Beserra et al. [2012]	✓	(✓)	(✓?)	-	✓	(✓)	Cloudstep; Also uses the notions of *constraints* and *cloud provider profiles*
Binz et al. [2011]	✓	(✓)	-	(✓)	-	(✓)	CMotion; Supports also migration between clouds
Hajjat et al. [2010]	✓	(✓)	-	(✓)	-	-	Focus on security; Hybrid cloud deployments
Kaisler and Money [2011]	✓	-	-	-	-	-	Abstract guidance; No application and cloud modeling support
Laszewski and Nauduri [2011]	✓	-	(✓)	-	-	✓	Migration to cloud technologies from Oracle
Leymann et al. [2011]	✓	(✓)	-	(✓)	-	✓	MOCCA method; Supports multi-cloud deployments
Menzel and Ranjan [2011]	✓	(✓)	-	(✓)	-	-	CloudGenius; Tool CumulusGenius
Mohagheghi and Sæther [2011]	✓	(✓)	-	(✓)	-	✓	REMICS project; KDM for application modeling
Mohan [2011]	✓	-	(✓?)	-	-	(✓)	High-level seven-step model of migration into the cloud
Shimba [2010]	✓	-	-	-	-	✓	ROCCA and RAF framework
Venugopal et al. [2011]	✓	-	-	-	-	-	Focus on exploitation of multi-core processors in cloud environments
Ward et al. [2010]	✓	(✓)	-	-	✓	✓	Extensions to the Darwin migration framework
Zhang et al. [2009]	✓	-	-	-	-	✓	Relies on OMG's ADM initiative
Zhang and Liu [2011]	✓	-	✓	-	-	✓	CloudMig; Not to be confused with CloudMIG
Zhou et al. [2010a]	✓	-	-	-	-	-	Ontology-based reengineering

Characteristics of Cloud Migration Approaches:

1. (Ind.) Independent from an SUA's technology? Can the migration approach be applied independently of the technology (e.g., programming language, framework) used to build the SUA (e.g., by adapters)?

✓: yes; −: no

2. (Mult.) Supports multiple cloud environments? Can a future SaaS provider choose from more than one cloud environment?

✓: yes, several cloud environment models are available (maybe also extendable with further models); (✓): yes, support for building further cloud environment models is being provided; −: no

3. (Conf.) Checks an SUA's technical conformance? Does the migration approach include checking the conformance of SUAs regarding technical constraints?

✓: yes, automatically; (✓): yes, manually; ($✓^?$): yes, unknown if automatically or manually; −: no

4. (Opt.) Optimization of CDOs? Does the migration approach provide support for optimizing CDOs prior to the actual migration?

✓: yes, all properties as described in the CDO definition (cf. Section 8.1.1) can be optimized; (✓): yes, but only single properties, such as selecting the best suited cloud environment; −: no

5. (Usg.) Incorporates an SUA's usage patterns? Does the migration approach consider the usage patterns of an SUA's status quo deployment (cf. Section 6.2)?

✓: yes; −: no

6. (Exec.) Migration execution support? Does the migration approach provide support for the actual execution of the migration (cf. the migration execution item in a migration plan regarding [ISO/IEC/IEEE 2006] in Section 3.2.3)? Hence, does the migration approach provide support for actually performing a migration to the cloud?

✓: yes, tool support is available; (✓): yes, conceptually; −: no

Cloud Migration Approaches:

The project **ARTIST**[1] aims to develop methods and techniques for supporting the migration of legacy software to the cloud. It is is funded by the EU. Besides a phase that covers the actual migration of a software system, the so-called ARTIST software modernization process sketched by Bergmayr et al. [2013] includes a pre-migration and a post-migration phase. In the pre-migration phase, ARTIST examines the software system to analyze possible consequences of a migration. The ARTIST project uses MDE-based reverse engineering techniques to extract a legacy Platform-Specific Model (PSM), i.e., a model that contains all details regarding the status quo deployment. The PSM model is then transformed to a more abstract Platform-Independent Model (PIM) for applying generic migration or optimization patterns. In a forward engineering step, this PIM is eventually used to generate the source code of the cloud-enabled software.

ARTIST aims to support several technologies of an SUA and also strives for supporting various cloud environments. Due to the early stage of the project (at the time of writing, ARTIST just started), no corresponding models or tools are available. In contrast to CloudMIG, no explicit conformance checking is mentioned by Bergmayr et al. [2013]. However, the authors state that the migrated software will have to comply with a chosen target cloud environment. Hence, manual, semi-automatic, or automatic conformance checking might be included in some form when the project evolves. Furthermore, utilization of an SUA's usage patterns, e.g., for optimizing runtime reconfiguration rules, as is considered by CloudMIG, seems to be out of

[1] http://www.artist-project.eu/

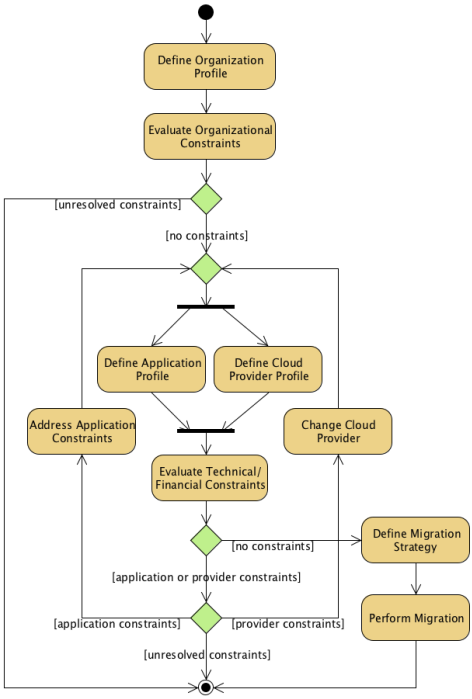


Figure 13.2. Cloudstep process (from [Beserra et al. 2012])

ARTIST's scope. However, opposed to CloudMIG, ARTIST supports an actual migration to the cloud and also verifies a migrated software system through model-based tests after the migration.

A further approach that supports the migration of legacy applications to the cloud is **Cloudstep** [Beserra et al. 2012]. It defines the process depicted in Figure 13.2. Cloudstep also employs the notion of constraints in the context of cloud migration. However, in contrast to CloudMIG, Cloudstep also considers organizational constraints that may hinder a migration to the cloud, for example, the loss of governance or legal restrictions. As shown in Figure 13.2, Cloudstep starts by defining a profile of the organization that

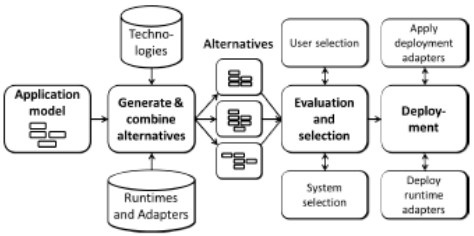


Figure 13.3. CMotion process (from [Binz et al. 2011])

wants to migrate an SUA in the step *define organizational profile*. Such a profile contains the mentioned organizational constraint definitions, for instance. These are evaluated in the second step *evaluate organizational constraints*.

If no severe violations regarding these constraints are determined, the process continues with the two activities *define application profile* and *define cloud provider profile*. The cloud provider profiles correspond to CloudMIG's similarly-named cloud profiles. The application profiles have to be modeled manually, whereas CloudMIG enables to automatically extract KDM-based models of an SUA. However, Cloudstep's application profiles also include modeled high-level usage characteristics, but compared with CloudMIG, automatically extracting usage profiles from monitoring log data seems not to be addressed by Cloudstep. The application profile and cloud profile are then analyzed with regard to the organizational profile in the next *evaluate technical/financial constraints* step. Potential conflicts may then be resolved in the corresponding *address application constraints* or *change cloud provider* steps. If no conflicts regarding the defined constraints occur, the actual migration can be planned and performed in the corresponding *define migration strategy* and *perform migration* steps. In contrast to CloudMIG, Cloudstep does not consider the optimization of CDOs.

An approach that not only considers the migration of software to the cloud, but also the migration of applications between cloud environments for approaching the problem of vendor lock-in, is the Cloud Motion Framework (**CMotion**) [Binz et al. 2011]. An overview of the approach and the general process is shown in Figure 13.3. In contrast to CloudMIG, CMotion uses

high-level application models that merely comprise the components of an application. Hence, no fine-grained conformance checking or CDO performance simulation, as is possible with CloudMIG, can be performed with CMotion. The approach can, however, also generate different deployment options, but its CDO model does not consider elasticity, for instance.

CMotion employs dedicated adapter components in the CDO model for integrating an existing application in specific cloud runtime environments. After generating a set of alternative CDOs (step *generate & combine alternatives* in Figure 13.3), a user can evaluate the CDOs and select the best suited candidate (step *evaluation and selection*). The evaluation bases on a coarse-grained cost model. Component response times, licensing expenditures, and wages for implementing adapters are also seen as costs, for instance. In comparison, CloudMIG uses CDOSim and the simulation-based genetic algorithm CDOXplorer to evaluate and compare CDOs. As described by Binz et al. [2011], CMotion uses an existing deployment framework that employs the application model as a cafe application description [Mietzner and Leymann 2010] for managing the deployment of the selected CDO.

A further migration approach that focuses on hybrid cloud-based deployments of enterprise applications is contributed by Hajjat et al. [2010]. As security and privacy are major concerns of enterprises that evaluate a migration to the cloud [Subashini and Kavitha 2011; Ren et al. 2012; Rosado et al. 2012], the approach enables to explicitly flag those application components that have to remain hosted on premise. CloudMIG also enables to manually compose CDOs that leave delicate components deployed on premise. But in contrast to CloudMIG, Hajjat et al. [2010] propose to build coarse-grained application models that mostly contain components and communication channels between those components. Hence, performing a comprehensive conformance check, as can be done with CloudMIG, is not possible. To find CDOs that are well-suited regarding costs, transaction delays, and compliance with security policies, the approach formulates these aspects as integer programming problems. Hence, similarly to CloudMIG, it also aims at optimizing CDOs, but runtime reconfiguration rules are not considered, for instance. Actually executing planned migrations is also not supported with the approach proposed by Hajjat et al. [2010].

Kaisler and Money [2011] describe challenges and high-level steps for migrating applications to the cloud. The authors propose to build application requirement specifications (ARSs) for specifying the resources an SUA requires to run in a cloud computing environment. However, no further details regarding an application model are provided that would enable to check the conformance of an application. Furthermore, the approach proposes to capture the resources that are offered by a cloud environment in corresponding models. However, in contrast to CloudMIG, no meta-model for describing cloud environments is provided. Major addressed concerns relate to the general problem of selecting the best suited cloud environment and security and privacy issues. However, the provided guidance is rather abstract. No support for actually migrating applications to the cloud is provided. In contrast to CloudMIG, the approach does not incorporate former usage patterns of SUAs in the migration planning process.

A migration approach that focuses on a specific vendor of cloud-based target platform technologies is presented by Laszewski and Nauduri [2011]. The approach describes ways to re-architect or wrap existing applications so cloud technologies from Oracle[2] can be used. For example, Laszewski and Nauduri [2011] propose to modify Java-based applications so that Oracle's WebLogic[3] platform can be employed. This constitutes a first step to subsequently migrate the application to Oracle's Exalogic[4] computer appliance, a combination of pre-configured hardware and software that Oracle offers as building blocks for clouds.

Furthermore, Laszewski and Nauduri [2011] focus on the database migration, i.e., the replacement of an arbitrary relational database with Oracle's database, whereas optimizing existing applications for the proposed target platform is seen as a subsequent step. The authors propose to use existing migration tools were possible, but in contrast to CloudMIG, no guidance for building application models or evaluating the envisioned CDOs is given. For the latter, CloudMIG utilizes the CDO simulation tool CDOSim, for instance.

[2] http://www.oracle.com

[3] http://www.oracle.com/technetwork/middleware/weblogic/overview/

[4] http://www.oracle.com/us/products/middleware/exalogic/overview/

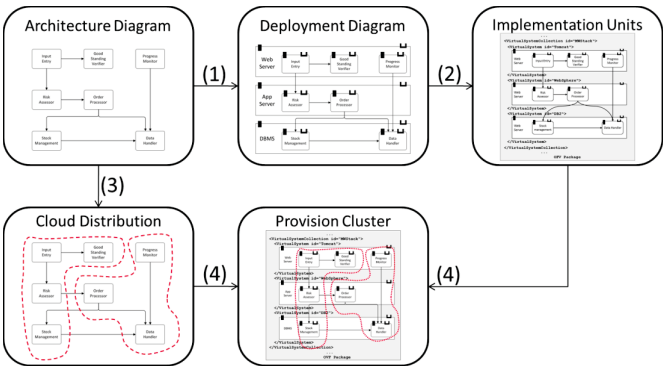


Figure 13.4. MOCCA method (from [Leymann et al. 2011])

The **MOCCA** (MOve to Clouds for Composite Applications) method by Leymann et al. [2011] also provides support for migrating applications to the cloud. An overview and the major artifacts of MOCCA are shown in Figure 13.4. The method assumes that an *architecture diagram* is provided that models the architecture of an SUA. The architectural model may comprise various levels of details, however, it principally includes the components and corresponding relationships of an application. Hence, in contrast to CloudMIG, no fine-grained conformance checking can be performed. The architectural model is enriched with information describing the current deployment of the application in a *deployment diagram*. Hence, this model corresponds to the status quo deployment model employed by CloudMIG. Then, implementation details are added to the architecture and deployment diagram. The corresponding *implementation units* refer to, for example, virtual machine images that include the architectural components and are needed for actually executing these components. Further artifacts that describe the deployment of a set of specific components to a single cloud environment are so-called *cloud distribution*s. The MOCCA method supports utilizing multiple cloud distributions, i.e., to distribute subsets of an application's components to more than one cloud environment. Cloud deployments using more than one cloud environment are not supported by CloudMIG.

MOCCA's cloud distributions can be produced manually or automatically. In the latter case, so-called *labels* have to be added to the architectural model to specify certain characteristics, such as computational units that are needed to process a component. Similarly to CloudMIG, MOCCA also supports to automatically search for near-optimal CDOs (cloud distributions), e.g., regarding costs, by using simulated annealing or hillclimbing optimization methods. However, MOCCA evaluates the fitness of potential solutions by solving mathematical functions, whereas CloudMIG follows a simulation-based optimization approach using the genetic algorithm CDOXplorer. Finally, the artifact *provision cluster* contains all information necessary to actually deploy the created cloud distributions.

Providing support for the migration of multi-component enterprise applications to the cloud is addressed by the approach **CloudGenius** [Menzel and Ranjan 2011]. The approach describes a migration process that, similarly to CloudMIG, also enables to optimize CDOs, e.g., regarding QoS properties and best suited VM images. However, most cloud environments enable to build custom VM images and to derive the costs from used VM instance types per hour rather than from the employed VM images.

Hence, the CDO simulation tool CDOSim and the genetic algorithm CDOXplorer, that are utilized by CloudMIG, rather focus on VM instance types and component allocation as variable parts of the CDO model (cf. Section 8.3). Furthermore, elasticity is not considered by CloudGenius. An implementation of the approach's decision-making support regarding the selection of VM images and cloud infrastructure services is given by the tool CumulusGenius.[5] CloudGenius defines a formal model for specifying software components of an application and cloud environments. However, due to a coarse-grained application model no detailed conformance checking is, in contrast to CloudMIG, included in the migration process.

The **REMICS** project[6] [Mohagheghi and Sæther 2011] also aims for developing methods and tools that provide cloud migration support. REMICS is funded by the EU and stands for *REuse and Migration of legacy applications to*

[5] http://code.google.com/p/cumulusgenius/

[6] http://www.remics.eu/

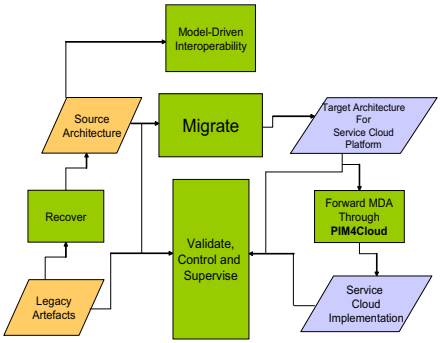


Figure 13.5. REMICS approach (from [Mohagheghi and Sæther 2011])

Interoperable Cloud Services. Its basic approach is illustrated in Figure 13.5. Similarly to CloudMIG, REMICS employs MDE techniques for reverse engineering application models (*recover* activity in Figure 13.5) and creating appropriate CDOs. A further similarity can be seen in the usage of KDM for representing SUAs. However, in contrast to CloudMIG, REMICS does not use the KDM models for checking the conformance of SUAs.

Mohagheghi and Sæther [2011] describe the tight dependence that results from deploying applications to PaaS-based cloud environments and that an SUA has to be suited for a specific platform. However, no further conclusions regarding some sort of conformance checking are drawn. In comparison with CloudMIG, REMICS also considers reverse-engineered models that describe business processes, business rules, and test specifications, for instance. For modeling cloud environments and the corresponding deployments of SUAs, the UML profile PIM4Cloud is developed in the context of REMICS. Hence, PIM4Cloud corresponds to CEM in the case of CloudMIG, however, it solely focuses on the IaaS service model, whereas CEM also allows to model PaaS-based cloud environments.

REMICS also does not seem to provide support for automatically generating and comparing different CDO alternatives, i.e., PIM4Cloud instances. A further difference between CloudMIG and REMICS can be seen in the

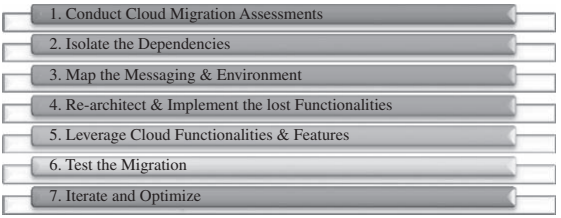


Figure 13.6. Seven-step model of migration into the cloud (from [Mohan 2011])

validate, control and supervise activity in Figure 13.5. This activity of REMICS aims at guaranteeing that certain QoS properties are satisfied by the migrated system. In contrast, CloudMIG does not provide support for actually migrating SUAs or validating the migrated systems.

A high-level cloud migration approach that consists of seven steps is described by Mohan [2011]. It is shown in Figure 13.6. The first step assesses the potential migration of an application to the cloud to investigate issues regarding its code, architecture, or non-functional requirements, for instance. Hence, checking the conformance of an application is, even though abstract, considered by the approach. However, no details regarding the modeling of an SUA are given. The next step examines the status quo deployment and aims at identifying all dependencies of an SUA that have to be considered when mapping an SUA's components to cloud resources. This mapping is performed in the third step. However, the approach does not consider automated techniques for generating optimized mappings before the actual migration. If an application has to be re-architected or modified, these modifications are executed in the next fourth step. The fifth step then leverages a cloud environment's services. That means, regarding our CSA hierarchy (cf. Section 7.4.1), the proposed approach addresses levels L3 and L4 (cloud aligned and cloud optimized, respectively) and subsequently optimizes a chosen cloud deployment. The sixth step tests the already migrated application. In contrast, CloudMIG focuses on the migration planning phase. The seventh step can be seen as iteratively re-running all previous steps and further optimizing the migrated application. Hence, components that

previously might have been stayed on premise might now be considered for a migration based on prior experiences.

Shimba [2010] proposes the approach **ROCCA** for managing a migration to the cloud. ROCCA stands for *Roadmap for Cloud Computing Adoption* and defines the five phases *analysis*, *planning*, *adoption*, *migration*, and *management*. Compared with CloudMIG, ROCCA rather takes an organizational viewpoint, for example, whether organizations are suited for employing cloud technologies. In contrast, CloudMIG focuses on technical challenges of a cloud migration. ROCCA's first phase (analysis) analyzes an organization's suitability and chances of migrating its applications to a cloud environment, e.g., regarding security and compliance concerns.

The second phase (planning) covers the planning of aspects such as the choice of a specific cloud environment and financial planning. Examining the integration of the applications with provided cloud resources and planning SLA policies are covered by ROCCA's third phase (adoption), for instance. Though, building application models, checking their conformance, creating models of a cloud environment, and optimizing CDO models prior to migration are, in contrast to CloudMIG, out of ROCCA's scope. Nevertheless, the approach covers the actual migration execution in its fourth phase (migration). This phase and also ROCCA's fifth phase (management) are not covered by CloudMIG, as the management phase addresses issues such as documenting best practices in a retrospective, for instance. Furthermore, Shimba [2010] also describes the *ROCCA Achievement Framework (RAF)* that is related to CloudMIG's CSA hierarchy (cf. Section 7.4.1). RAF provides means for assessing the degree an organization conforms to the activities proposed by the ROCCA framework. In general, the CSA hierarchy also aims at evaluating the level of attaining the goals of a cloud migration. However, it rather focuses on an application's suitability regarding present CEC violations and the level of alignment with reference to a specific cloud environment.

A migration approach that focuses on the exploitation of multi-core processors' capabilities in cloud environments is presented by Venugopal et al. [2011]. The approach proposes three steps. The first step instruments the

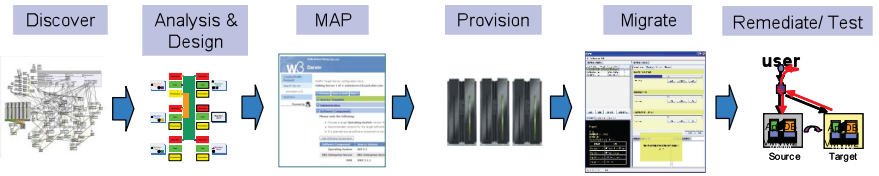


Figure 13.7. Cloud migration approach that uses extensions of the Darwin migration framework (from [Ward et al. 2010])

existing systems and measures system-level process data. The second steps examines system-level parameters that influence the applications' performance, such as parameters determining thread context switches. In the third step, the previously identified parameters have to be adjusted according to a specific target cloud environment's characteristics. However, these steps rather constitute a guideline as they remain abstract. Furthermore, the approach does not cover many important aspects that have to be considered when migrating to a cloud. For example, in contrast to CloudMIG, no means are provided for supporting the selection of a best-suited cloud environment or streamlining CDOs through re-architecting the applications and optimizing the mapping of application components.

A further cloud migration approach that employs **extensions of the Darwin migration framework** is proposed by Ward et al. [2010]. As described by the authors, the Darwin framework was initially developed to support migrations to mainframe computers from IBM. The extensions enable the Darwin framework to be used for migration scenarios that consider clouds as target environments. The proposed migration process includes six phases that are shown in Figure 13.7. The first phase (*discover*) analyzes a set of potential SUAs (the so-called *source environment*) and, similarly to CloudMIG, reverse-engineers models that include information from a static as well as a dynamic analysis. However, in contrast to CloudMIG, a rather coarse-grained model of a status quo deployment is constructed.

This model includes information such as the operating system type or a used host name. However, the actual workload of a status quo deployment

is also captured, for example, CPU utilization or workload characterization. In the case of CloudMIG, the MIPIPS and weights benchmark (cf. Section 8.2) and a workload profile from monitoring data is used to model the actual usage patterns of an SUA. The second migration phase (*analysis & design*) determines SUA candidates that are suited for a migration. However, compared with CloudMIG, no form of conformance checking is mentioned by Ward et al. [2010]. The phase also analyzes potential benefits of migrating the selected SUAs in terms of financial consequences. For estimating characteristics of specific CDOs prior to migration, CloudMIG follows an automated approach and uses the CDO simulation tool CDOSim.

The third migration phase (*map*) produces mappings of the previously selected SUAs to the target cloud environment. However, in contrast to CloudMIG, these mappings have to be created manually and no optimization of potential CDOs is considered. The fourth and fifth migration phases (*provision* and *migrate*, respectively) cover the actual execution of the migration. The provision phase prepares and sets up the target cloud environment resources, whereas the migrate phase actually performs the migration of the mapped SUAs. Those activities are not in the scope of CloudMIG. The sixth migration phase (*remediate/test*) validates the migrated SUAs and remediates detected issues.

A further cloud migration approach is contributed by Zhang et al. [2009]. Compared with the approach described before, it consists of seven (not of six) steps. Similarly to CloudMIG, the approach considers OMG's ADM initiative as a viable foundation that provides appropriate methodologies for representing legacy systems. Furthermore, it also explicitly aligns with the horseshoe model of reengineering (cf. Section 3.1.2). Figure 13.8 shows an overview of the approach. Its first step (*representation*) reverse-engineers the architecture of an SUA. In the second step (*redesign*), the approach modifies the application model in a way that it can be provided as a SaaS-based software. This step identifies application components that can be wrapped and exposed as services.

In contrast, CloudMIG considers a specific cloud environment and potential CDO candidates at this stage. The third step (*MDA* transformation) aims at

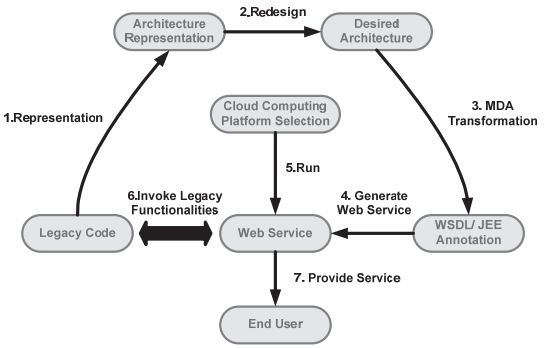


Figure 13.8. Cloud migration approach based on OMG's ADM initiative (from [Zhang et al. 2009])

transforming the target architecture produced in the previous step towards concrete web service technologies that can be used to realize the defined services. This step creates corresponding structural model elements, such as JEE annotations. The web services are then generated from this model in the next fourth step (*generate web service*). An appropriate cloud environment is chosen in the fifth step (*run*). This step also involves the actual execution of the migration, i.e., migrating the transformed system to the cloud. Then, the sixth step (*invoke legacy functionalities*) connects the web services and the legacy code, i.e., the legacy code is invoked from the web services. The final seventh step (*provide service*) provides the migrated system to end users. Besides the similarities between the approach and CloudMIG regarding the utilization of OMG's ADM methodologies, both approaches differ substantially. This is due to the fact that Zhang et al. [2009] do not consider conformance checking and the optimization of CDOs, that constitute central components of CloudMIG.

An approach that supports the migration of applications to the cloud that utilizes a system denoted **CloudMig** is described by Zhang and Liu [2011]. CloudMig also addresses errors that may occur when migrating applications to the cloud. Hence, CloudMig is also related to CloudMIG's conformance checking approach. However, despite the similar names and the common-

alities regarding addressed incidents that may hinder an application from running correctly in a cloud environment, CloudMig should not be confused with our cloud migration approach CloudMIG, both follow different goals. CloudMig focuses on configuration and installation errors that may result from the lack of attention of an operator who migrates an application to the cloud, for instance. For example, such errors may be given by outdated file system paths in configuration files, incompatible versions of an operating system, or insufficient computing resource capacities, such as too little free disk space. However, some of the corresponding properties of a cloud environment that could lead to a system malfunction can actually be seen as CECs, e.g., the versions of available operating systems. But in contrast to CloudMIG, CloudMig does not aim at modeling cloud environments along with CECs.

It rather captures a description of a specific application's status quo deployment in template files. For example, such a template file could contain the path to the database. When migrating the application, the corresponding value for the path has to be set manually by an operator. Hence, the dependencies, such as needed library versions, and complete configuration settings of an application have to be entered manually. Therefore, CloudMig takes the viewpoint of applications instead of generically modeling CECs from a cloud environment's perspective. CloudMig even uses the notion of *constraints* as well. However, in the context of CloudMig, constraints define the dependencies and existing configuration values of an application regarding the status quo deployment. CloudMig also provides a mechanism for automatically checking if the entered configurations are valid during and after a migration. For example, an operator may specify a policy that an application requires at least 1 GB of free disk space during execution. Then, CloudMig can periodically check whether this constraint is met. Hence, the constraint describes a restriction from the application's viewpoint, not from the perspective of a cloud environment. There exist further differences between the approaches. CloudMig does not consider automatically extracting architectural models from an SUA's source code, building workload profiles from monitoring data that capture the usage patterns of a status quo deployment, or optimizing potential CDOs.

A further approach for supporting cloud migration is proposed by Zhou et al. [2010a]. It focuses on identifying components of a legacy application that can be exposed as services in a cloud environment. That means, the services become part of a SaaS platform that is offered by future SaaS providers. The authors utilize an ontology-based approach for supporting the understanding and reengineering process. Ontologies are created for representing the source code and used databases, for instance. A corresponding tool extracts UML class diagrams from the source code. The class diagrams are then transformed to the ontology representation with the help of ATL. The identification of potential service candidates is performed via ontology partitioning [Zhou et al. 2010a]. In contrast, CloudMIG creates diverse (complete) CDO candidates that can be analyzed and compared. Modeling cloud environments along with corresponding CECs and checking the conformance of software systems is also not covered by the proposed approach.

13.1.2 Cloud Suitability Analysis

Related work from cloud suitability analysis is listed in Table 13.2. The related work is examined according to the following four characteristics. The abbreviations in parentheses are used for referencing purposes.

Characteristics of Approaches for Cloud Suitability Analysis:

1. (Tech.) Analyzes technical conformance? Does the approach include checking the conformance of SUAs regarding technical constraints?

✓: yes, automatically; (✓): yes, manually; ($\checkmark^{?}$): yes, unknown if automatically or manually; −: no

2. (Org.) Analyzes organizational conformance? Does the approach include checking the conformance of SUAs regarding organizational constraints? That means, regarding issues that address, for example, compliance, governance, legal situation, data security, or data privacy?

✓: yes; −: no

Table 13.2. Related work for cloud suitability analysis

Approach	Tech.[1]	Org.[2]	Costs[3]	Perf.[4]	Comments
Ebneter et al. [2010]	-	-	-	-	High-level description of cloud adoption best practices with enterprise architecture frameworks
Khajeh-Hosseini et al. [2011]	(✓)	✓	✓	✓	Cloud adoption toolkit; Provides a DSL for modeling usage patterns
Kim et al. [2009]	-	✓	✓	✓	List of potential cloud adoption issues
Loebbecke et al. [2011]	(✓)	✓	-	-	Magic Matrices Method; seven assessment criteria; three cloud readiness categories
Mattoon et al. [2011]	-	-	-	-	Cloud Computing Maturity Model from Oracle
Misra and Mondal [2011]	-	✓	✓	-	Suitability index; Return on investment model
Nasir and Niazi [2011]	✓	✓	-	-	Cloud Computing Adoption Assessment Model (CAAM); Early stage, only outline exists
Tran et al. [2011b]	✓	-	-	-	Cloud Migration Point (CMP) method

3. (Costs) Analyzes costs? Does the approach analyze financial implications of running an SUA in the cloud?

✓: yes; −: no

4. (Perf.) Analyzes performance? Does the approach analyze implications regarding performance characteristics when running an SUA in the cloud?

✓: yes; −: no

Approaches for Cloud Suitability Analysis:

As cloud computing and the adoption of corresponding technologies is a rather young discipline, Ebneter et al. [2010] propose to rely on best practices when considering a migration of SUAs to the cloud. Ebneter et al.

[2010] consider the usage of enterprise architecture frameworks for representing the common knowledge that can be extracted from best practices. The approach therefore proposes guidelines for consolidating corresponding knowledge regarding the adoption of cloud technologies instead of providing concrete methods and techniques for assessing the suitability of specific cloud environments, for instance. In contrast, CloudMIG defines the CSA hierarchy (cf. Section 7.4.1), enables to check the conformance of SUAs regarding potential CEC violations, and allows to analyze and compare different CDO candidates.

A further approach for investigating the suitability of SUAs regarding a migration to the cloud and supporting organizations for adopting cloud technologies is the **cloud adoption toolkit** [Khajeh-Hosseini et al. 2011]. The corresponding conceptual framework is illustrated in Figure 13.9. Khajeh-Hosseini et al. [2011] propose five tools/techniques for evaluating the implications of migrating applications to the cloud. Through performing a *technology suitability analysis* (cf. Figure 13.9), organizations may examine potential technical hurdles such as an SUA's missing support for scalability or insufficient network bandwidth of a cloud environment. Those characteristics are included in a checklist that can be processed and factored in the decision making process.

In contrast, CloudMIG includes information regarding a cloud environment's resources in cloud profiles and allows to automatically check an SUA for CEC violations. However, the toolkit's technology suitability analysis also addresses concerns such as security, data confidentiality, and regulatory requirements that are out of CloudMIG's scope.

Furthermore, the toolkit comprises a tool for *cost modeling*. It allows to use UML deployment diagrams to model the status quo deployment as well as potential CDOs. A DSL allows to model usage patterns of a system. These usage patterns are then utilized to calculate the future costs. Compared with CloudMIG, besides using arbitrary mathematical functions to model usage patterns, a realistic workload profile can also be automatically created from logged monitoring data. This workload profile is then used to replay the workload in combination with a potential CDO candidate during a simu-

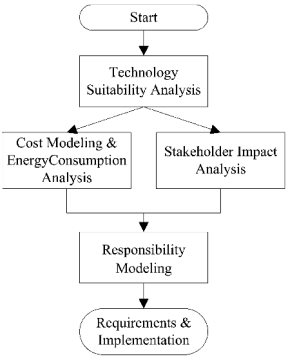


Figure 13.9. Cloud adoption toolkit conceptual framework (based on [Khajeh-Hosseini et al. 2011])

lation with our tool CDOSim. However, the tool from the cloud adoption toolkit also allows to calculate storage costs. This type of cost is not yet considered by CDOSim.

The cloud adoption toolkit also includes support for performing an *energy consumption analysis* regarding the status quo deployment. The consumed and possibly saved energy of the status quo deployment is actually an important aspect when considering a migration to the cloud. Furthermore, the cloud adoption toolkit also allows to perform a *stakeholder impact analysis* as well as a risk analysis through *responsibility modeling*. The former investigates how a migration to the cloud influences involved stakeholders. Responsibility modeling refers to potential risks during the operation of migrated, complex IT systems that may arise if not all involved and responsible parties are identified.

Kim et al. [2009] compile several types of cloud computing adoption issues that should be taken into account when migrating applications to the cloud. The list comprises the issues *outage*, *security*, *performance*, *compliance*, *private clouds*, *integration*, *costs*, and *environment*. For example, the first issue (outage) addresses the availability of cloud services. Kim et al. [2009] consider temporary and permanent outages that may result, for instance,

from cloud platform misconfigurations or provider bankruptcy, respectively. In summary, the authors provide a list of potential issues that can be used as a checklist when examining a migration of applications to the cloud. In contrast, CloudMIG provides tool support for investigating CEC violations, potential CDOs, and future costs. However, several issues addressed by Kim et al. [2009], such as the availability or compliance of cloud environments, are not considered by CloudMIG. Nevertheless, those type of information could be integrated in the CEM and therefore be used to augment cloud profiles, for example, regarding the historic availability of cloud services.

The **Magic Matrices Method** [Loebbecke et al. 2011] is a further approach for assessing the cloud readiness of an organization's applications and IT services. It is employed by the Continental AG, Germany, a global automotive supplier. The method aims to be simple enough, so it can be of practical use, but at the same time aims to allow incorporating the major relevant criteria that have to be considered when adopting cloud technologies. The Magic Matrices Method is structured into the three steps *identification*, *screening*, and *categorization*. The first identification step reveals applications and IT services that are subsequently inspected in the second and third step. This construction shows that the method targets companies that run large software landscapes, as in this context, the identification of relevant application and service candidates can be a complex task on its own. Our approach CloudMIG in contrast focuses on single software systems. Hence, applications chosen with the help of the Magic Matrices Method could be additionally analyzed with CloudMIG regarding CEC violations and potential CDOs.

The second step of the Magic Matrices Method (screening) uses seven defined criteria for evaluating the applications and IT services that were selected in the first identification step. The criteria comprise the network connectivity, compliance, and standardization of applications, for instance. Each criterion is assessed and visualized according to two major parameters in the form of a graphical matrix. Figure 13.10 shows an example regarding the standardization criterion, the contained numbers denote single applications and IT services. Each matrix is constructed so that the upper right corner represents the best suited (cloud ready) solutions. However, each

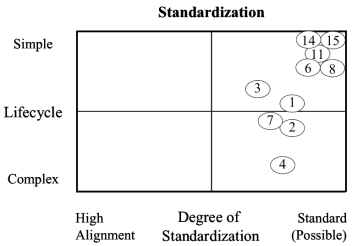


Figure 13.10. Magic Matrices Method, standardization criterion example (from [Loebbecke et al. 2011])

application or IT service has to be inspected manually regarding the defined criteria. In contrast, CloudMIG allows to automatically check CEC violations, for instance.

After aggregating the results obtained in the previous assessment, the third step of the Magic Matrices Method (categorization) assigns the applications and IT services to the following three categories A-C: *likely cloud ready* (A), *not yet cloud ready* (B), and *unlikely to be assessed cloud ready in the next years* (C). Hence, the applications and IT services of category A are first considered for the adoption of cloud technologies. In contrast, CloudMIG's CSA hierarchy comprises five levels and assesses single applications.

Mattoon et al. [2011] describe the **Cloud Computing Maturity Model** in a white paper from the Oracle corporation. The concept is inspired by the CMMI model [Ahern et al. 2008] and provides six levels for measuring an organization's capability regarding the maturity of employing cloud technologies. The six maturity levels range from *none* to *optimized*, whereas CloudMIG's CSA hierarchy uses five levels ranging from *cloud incompatible* to *cloud optimized* (cf. Section 7.4.1). However, Oracle's Cloud Computing Maturity Model rather takes a holistic perspective as it evaluates complete organizations and their IT systems. In contrast, CloudMIG examines single software systems. Oracle's maturity levels determine the maturity of cloud capabilities in enterprises. That means, it assesses the method and degree of organization that is in place for employing cloud technologies. For example,

the second level *ad hoc* indicates that a company is aware of the cloud's potential, but only single divisions start to exploit it and there exists no comprehensive plan on the corporate level.

Additionally to the maturity measure, the Cloud Computing Maturity Model includes a measure for separately assessing the level of cloud adoption. This measure also defines six levels and allows to evaluate the degree of dissemination of cloud technologies. For example, there exists a level that specifies that cloud technologies are used *across units*.

A further approach that is also not restricted to single applications but rather considers comprehensive software portfolios of an organization is contributed by Misra and Mondal [2011]. The approach proposes the following four main characteristics that should be considered when assessing a company's suitability for adopting cloud technologies.

- *Size of IT resources:* For example, possible metrics include the number of used servers and the annual revenue from IT.
- *The utilization pattern of the resources:* The approach distinguishes average usage, peak usage, and the amount of data handling/transaction done. It also defines five workload profiles that can be used for assessing existing applications and services, for example, a constant workload with no variability or a moderately variable workload with occasional surges.
- *Sensitivity of the data they are handling:* Five categories are used for evaluating the data sensitivity ranging from *not sensitive* to *extremely sensitive*.
- *Criticality of work done by the company:* Four categories are used for evaluating the criticality of used applications and services ranging from *standard* to *highly critical*.

Based on an assessment that uses these characteristics, the approach allows to compute a suitability index. The suitability index is a number that is then used to judge a company's suitability for cloud adoption. For example, if the suitability index stays below 3,760, the approach considers the company and its applications as *not suitable for cloud adoption*. Two further categories

exist: *May or may not be suitable for cloud adoption, further investigation required* and *suitable for the adoption of cloud*. Moreover, the approach provides a mathematical model for determining the return on investment of adopting cloud technologies.

Compared with CloudMIG, our approach focuses on single applications and provides means for automatically assessing technical constraints (CECs). The CSA hierarchy builds upon the conformance checking mechanism and does not consider organizational constraints that may impede the adoption of cloud computing. Furthermore, CloudMIG allows to define arbitrary workload profiles as well as realistic profiles from monitoring data and to use these profiles for the evaluation of CDOs. In summary, CloudMIG could be used in the context of a cloud suitability assessment with the approach from Misra and Mondal [2011] to add further technical aspects and raise the level of automation.

The **Cloud Computing Adoption Assessment Model (CAAM)** is proposed by Nasir and Niazi [2011]. However, the current status of CAAM is unclear as the paper merely outlines CAAM and poses five research questions that should be investigated. The research questions may then lead to the actual construction of CAAM. According to these research questions, CAAM covers general challenges of adopting cloud computing and provides support for identifying specific challenges a company faces when considering the use of cloud technologies. A major goal of CAAM is to provide an adoption assessment framework for evaluating a company's organizational readiness for the usage of cloud technologies.

As compared with CloudMIG and its CSA hierarchy, CAAM covers both, technical and organizational challenges of adopting cloud technologies. However, challenges regarding cost and performance estimation are not mentioned by Nasir and Niazi [2011]. Those issues are addressed by CloudMIG. According to CAAM's current state, it does seem to provide a guideline or checklist. In contrast, CloudMIG enables to actually model a status quo deployment and provides means to automatically check an SUA's technical conformance and assess different CDOs.

Tran et al. [2011b] implicitly assess the suitability of software systems for a migration to a cloud environment. The authors describe the **Cloud Migration Point (CMP)** methodology for estimating the size and effort of cloud migration projects. Hence, considering technical challenges of a cloud migration, the better a software system is suited for a specific cloud environment, the less changes and effort are required for performing the migration. For estimating the size and effort for developing arbitrary software systems, utilizing Function Points (FPs) [Albrecht and Gaffney 1983] is a commonly found method. CMP modifies this method and transfers the concept in the cloud migration domain.

Tran et al. [2011b] consider migration tasks from four different categories as cost factors. The categories include *installation and configuration*, *database changes*, *code changes*, and *connection changes*. Our approach CloudMIG covers three of these categories. However, CloudMIG does not yet include detailed models of a database and therefore, potential database changes (and the corresponding category) are not considered. CMP's categories are further divided into several types. For example, migration tasks of the installation and configuration category can be assigned to one of the two types *infrastructure level* or *application level*. The former considers, for example, server machines of the status quo deployment, the latter considers third-level libraries required by the SUA, for instance. For actually determining the effort of migrating a software system to an already selected cloud environment, the CMP method starts with analyzing and listing all migration tasks of all four previously described categories. Each migration task is then assigned to a type that is included in its corresponding category.

Similarly to the FP method, a complexity level (low, average, high) is then assigned to each migration task. The actual CMP value can then be computed with the help of different weights for the types. In contrast to CloudMIG, CMP assumes that the decision for a target cloud environment was already made and also the target architecture and the necessary application modifications are given. Opposed to this, providing support for selecting cloud environments and creating target architectures and complete CDOs are central components of the CloudMIG approach.

13.1.3 Cloud Environment Modeling

There exist several APIs that enable to deploy applications to multiple cloud providers and to manage cloud resources of different cloud environments with a unified programming interface. These so-called multi-cloud APIs include, for instance, Cloud Application Management for Platforms (CAMP),[7] Deltacloud,[8] DMTF Cloud Infrastructure Management Interface (CIMI),[9] jclouds,[10] Libcloud,[11] Simple Cloud,[12] and Unified Cloud Interface (UCI).[13] Those APIs also describe the structure and resources of cloud environments (cf. [Harmer et al. 2009] for an example). CEM's *IaaS* package also reuses some of Deltacloud's concepts. However, please note that these multi-cloud APIs are not covered in this section.

They focus on managing cloud-based applications at runtime through, for example, starting additional VM instances, persisting data to cloud storage, or copying VM images. In contrast, CloudMIG focuses on structural elements of cloud environments for creating the CEM, corresponding cloud profiles, and target architectures. Hence, those APIs are valuable for actually deploying and operating a cloud-based system. They can be utilized when actually executing a migration to the cloud during CloudMIG's activity A6 (cf. Section 6.3.2), but due to their focus on cloud management and operational aspects, they are not included in this section.

Related work from modeling cloud environments is listed in Table 13.3. The related work is examined according to the following six characteristics. The abbreviations in parentheses are used for referencing purposes.

[7] http://www.cloudspecs.org/
[8] http://deltacloud.apache.org/
[9] http://dmtf.org/standards/cloud/
[10] http://www.jclouds.org/
[11] http://libcloud.apache.org/
[12] http://www.simplecloud.org/
[13] http://code.google.com/p/unifiedcloud/

Table 13.3. Related work for cloud environment modeling

Approach	IaaS[1]	PaaS[2]	Pri.[3]	Perf.[4]	SLAs[5]	Compl.[6]	Comments
Bakshi [2011]	(✓)	-	(✓)	-	-	-	Cloud data center framework and infrastructure technology architecture
Hickey and Rahmouni [2010]	(✓)	-	-	-	-	-	Builds upon semantic web technologies such as OWL and RDF
Liu et al. [2011]	(✓)	(✓)	(✓)	-	(✓)	(✓)	NIST Cloud Computing Reference Architecture
Liu et al. [2012]	(✓)	(✓)	(✓)	-	(✓)	(✓)	Cloud Computing Reference Architecture (CCRA)
Moreno-Vozmediano et al. [2012]	(✓)	-	(✓)	-	(✓)	-	Cloud OS; Operating system architecture for IaaS-based clouds
Moscato et al. [2011]	(✓)	(✓)	-	(✓)	(✓)	-	mOSAIC Cloud Ontology
Rochwerger et al. [2009]	(✓)	-	(✓)	(✓)	(✓)	-	Reservoir architecture for federated cloud computing
Tianfield [2011]	(✓)	-	-	-	-	-	Cloud Platform Architecture (CPA)
Tsai et al. [2010]	(✓)	-	(✓)	-	(✓)	-	Service-Oriented Cloud Computing Architecture (SOCCA)
Zhang and Zhou [2009]	(✓)	-	-	-	(✓)	-	Cloud Computing Open Architecture (CCOA)

Characteristics of Approaches for Cloud Environment Modeling:

1. (IaaS) Covers IaaS-specific cloud resources? Does the approach cover IaaS-specific cloud resources, such as VM images, VM instance types, VM instances, and load balancer services?

✓: yes, high level of detail; (✓): yes, low level of detail; −: no

2. (PaaS) Covers PaaS-specific cloud resources? Does the approach cover PaaS-specific cloud resources, such as libraries, runtime environments, and computing structures that form the provided cloud platform?

✓: yes, high level of detail; (✓): yes, low level of detail; −: no

3. (Pri.) Covers pricing model? Does the approach cover the pricing model of cloud environments, i.e., does it allow to specify prices for utilizing offered cloud services?

✓: yes, high level of detail; (✓): yes, low level of detail; −: no

4. (Perf.) Covers performance characteristics? Does the approach cover performance characteristics, such as the maximum number of instructions per seconds of a CPU that is used for a specific VM instance type?

✓: yes, high level of detail; (✓): yes, low level of detail; −: no

5. (SLAs) Covers SLAs? Does the approach cover SLAs that are offered by a cloud environment?

✓: yes, high level of detail; (✓): yes, low level of detail; −: no

6. (Compl.) Covers compliance issues? Does the approach cover concerns related to compliance, for example, followed standards and certifications of internal processes?

✓: yes, high level of detail; (✓): yes, low level of detail; −: no

Approaches for Cloud Environment Modeling:

Bakshi [2011] propose a high-level *cloud data center framework* and a *data center infrastructure technology architecture*. The architecture includes the basic building blocks for describing the infrastructure of an IaaS-based cloud environment. In contrast, CEM also allows to model PaaS-based cloud environments. The architecture from Bakshi [2011] is structured into nine layers. For example, *application software* is deployed to *virtual machines* (first and second layer, respectively). A virtual network enables the VM instances to communicate with each other (*virtual network access* layer), persistent data

is stored in the *storage* layer. However, the layers are described on an abstract level and with the help of specific, exemplary technologies. Compared with CloudMIG, CEM provides a meta-model and corresponding elements for detailing these layers.

The cloud data center framework builds upon the previously described data center infrastructure technology architecture. The framework adds further layers. The *service orchestration* layer provides a service repository and maps the services to resources of the infrastructure technology architecture, for instance. Cloud users manage and instantiate the services through a *service portal* or a *service application programming interface*. Further layers concern security and billing issues, for instance. However, similarly to the previously described data center infrastructure technology architecture, the description of the overall framework is rather abstract. In contrast, CEM provides elements for detailing, for instance, a pricing model.

An approach that is in an early stage and mainly outlines basic building blocks for modeling cloud environments is proposed by Hickey and Rahmouni [2010]. The authors address the adaptive topology found in common cloud environments, for example, through adding new physical hosts to expand capacity or starting new virtual machines. However, in contrast to CloudMIG, Hickey and Rahmouni [2010] do not consider PaaS-based cloud environments. Furthermore, their approach proposes to use semantic web technologies for modeling cloud environments. For example, it builds upon RDF and uses OWL to define topologies. Though, in the current form a detailed topology definition is missing.

The **NIST Cloud Computing Reference Architecture** was already described in Section 2.3. Besides an overview regarding general concepts and roles in the context of cloud computing, the reference architecture describes the basic building blocks of cloud environments. For example, these building blocks comprise model elements for physical resources as well as for considering security and privacy concerns. In contrast, CloudMIG provides a more detailed view on a cloud environment's technical components, as it constitutes a full-fledged meta-model. However, the reference architecture also addresses issues such as contract management and compliance that

are not considered by CloudMIG. For are more detailed description of the NIST cloud computing reference architecture, we refer to the corresponding Section 2.3.

A further reference architecture is proposed by Liu et al. [2012]. Their **Cloud Computing Reference Architecture (CCRA)** follows the SOA Reference Architecture from *The Open Group* [Group 2012] to some degree. Architectural Building Blocks (ABBs) are used in both reference architectures as basic structural elements for composing the architecture. CCRA consists of several layers that contain a set of ABBs. It also defines four principles that should be followed when building actual cloud environment models. The principles imply consideration of specific domain concepts. For example, the principle *virtualization support* advocates for including VM image management capabilities. The other principles cover *service management support*, *on-demand service provisioning and subscription support*, and *interoperability support*.

An overview on CCRA and its included layers is shown in Figure 13.11. The *operational system layer* contains the ABBs for describing the physical compute infrastructure and the utilized vitualization environment, for instance. Compared with CloudMIG, our approach covers several elements for modeling a cloud environment's structure, that are also addressed by CCRA, in more detail. Additionally, CEM allows to specify a cloud environment's CECs and map detailed application code models to cloud resources. However, several ABBs considered by CCRA are out of CEM's scope. For example, CCRA also covers cloud consumer portals in its *cloud customer* layer and a user profile management in its *cloud governance* layer. Corresponding elements are not contained in CEM.

An operating system architecture for IaaS-based cloud environments termed **Cloud OS** is proposed by Moreno-Vozmediano et al. [2012]. The authors describe important components for building such software systems that allow managing hardware infrastructures and enable providing IaaS-based core services, for example, starting VM instances. Furthermore, supporting auxiliary functionalities such as user management and authentication mechanisms is also considered by *Cloud OS*. Figure 13.12 shows an overview on

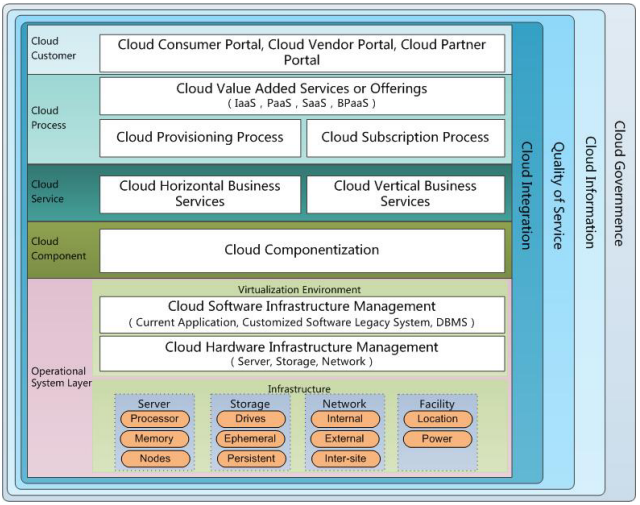


Figure 13.11. Cloud Computing Reference Architecture (CCRA) overview (from [Liu et al. 2012])

the proposed *Cloud OS* architecture. The high-level architecture is structured into the three layers *Drivers*, *Core*, and *Tools*.

The *Drivers* layer enables usage of a present hardware infrastructure. It also provides an abstraction of the employed physical resources, for example, through using hypervisors. The *Cloud OS* also enables to build a federation between clouds for reasons of failover, for instance. The *Drivers* layer also provides the corresponding functionality. The *Core* layer includes basic building blocks of an IaaS-based cloud. For example, the *VM manager* controls the life cycle of virtual machines, the *Image manager* stores, copies, and retrieves VM images, and the *Accounting and auditing* component tracks resource usage for billing and security reasons.

The *Tools* layer provides, on the one hand, interfaces for cloud users and administrators through the components *Cloud interfaces* and *Administrator tools*, respectively. On the other hand, the *Tools* layer comprises the components *Service manager* and *Scheduler*. The former component controls additional

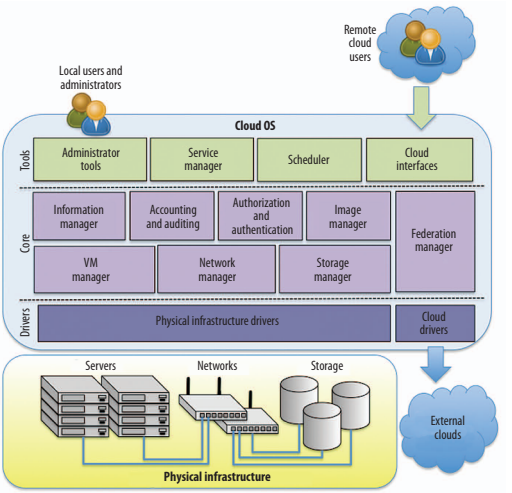


Figure 13.12. Cloud OS overview (from [Moreno-Vozmediano et al. 2012])

cloud services, such as databases, message queues, or BLOB storage services. For example, the *Service manager* deploys those services to VMs and controls dynamic service scaling for handling fluctuating workload. The *Scheduler* component schedules the access of VMs to physical resources, such as CPUs.

Compared with Cloud OS, CloudMIG also supports PaaS-based cloud environments and provides a detailed meta-model (CEM). CEM also considers fine-grained CEC specifications and allows to map application code models. However, as CEM describes cloud environments from a cloud user perspective, it does not consider the mapping of virtual resources to the underlying hardware as those mappings are transparent for cloud users. Furthermore, components for managing cloud resources, for example, through an administrative graphical user interface, are out of CEM's scope.

Moscato et al. [2011] use semantic web technologies such as OWL to describe the **mOSAIC Cloud Ontology**. The ontology aims at providing a unified view on heterogeneous cloud environments. The ontology was created

in the context of the mOSAIC EU project[14] that investigates an API and platform for developing applications that can employ multiple clouds. In contrast to CloudMIG's CEM, the mOSAIC Cloud Ontology also contains elements that are commonly transparent to cloud users, such as hosts and distributed file systems. Nevertheless, the mOSAIC Cloud Ontology also focuses on a cloud user perspective. For example, it includes elements for modeling the employed encryption, accounting, and VM description.

Furthermore, besides IaaS-based cloud environments, the mOSAIC Cloud Ontology allows to model PaaS-based cloud environments, as is also supported by CEM. The mOSAIC Cloud Ontology also addresses non-functional properties such as availability and performance. For the latter, the ontology considers specifications for a CPU's FLOPS, for instance. Those performance-related properties can be modeled with the help of CEM as well.

Methods and tools for federated cloud environments were addressed in the Reservoir EU project.[15] The **Reservoir architecture** [Rochwerger et al. 2009] was developed in the context of this project. It enables cloud providers to cooperate and offer diverse, large-scale infrastructure services to SaaS providers. Hence, cloud-based applications may span across multiple cloud environments and leverage the enhanced service offerings. The Reservoir architecture defines the basic components and their associations that exist in and between the federated data centers (so-called *Reservoir sites*). In each Reservoir site, three layers of abstraction structure the internal architecture. External service providers interact with the *service manager* layer for, e.g., deploying their services and billing compute resource usage.

The second layer within a Reservoir site is the *VEE manager (VEEM)*. In the context of the Reservoir architecture, virtual machines are represented by so-called *virtual execution environments (VEEs)*. Virtualized resources that are managed, for example, through hypervisors and host VEEs, are termed *virtual execution environment hosts (VEEHs)*. Hence, VEEMs control the allocation of VEEs to VEEHs, for instance. The aforementioned layers communicate through the so-called *VEE management interface (VMI)*. Thus,

[14] http://www.mosaic-cloud.eu/

[15] http://www.reservoir-fp7.eu/

the service manager interacts with the VEEM through a VMI instance. To link different Reservoir sites, VMIs are utilized as well. These VMIs connect the individual VEEMs. A Reservoir site's third layer is given by a set of *VEEH*s. This third layer is connected to a VEEM with the help of a so-called *virtual host interface (VHI)*.

In contrast to the Reservoir architecture, CloudMIG's CEM focuses on single cloud environments. However, it provides dedicated support for PaaS-based cloud environments and allows to map fine-grained application models to cloud resources. Furthermore, CEM takes a strict cloud user perspective and therefore omits components of a cloud infrastructure that are most often transparent to the cloud users, such as utilized hypervisors.

Besides taking a cloud user perspective, CEM also clearly distinguishes cloud application and cloud environment concerns. That means, it employs its *Mapping* layer (cf. Section 6.3.4) to assign the cloud application code model elements to resources of the cloud environment. Such a clear distinction between the both domains is also considered by Tianfield [2011]. For describing cloud applications and cloud environments, the author presents the Cloud Application Architecture (CAA) and the **Cloud Platform Architecture (CPA)**, respectively. The Cloud Application Architecture (CAA) is described in Section 13.1.4 in greater detail.

In contrast to CEM, CPA does not consider PaaS-based cloud environments, as it focuses on virtual machines. CPA's basic building blocks are given by elements that virtualize hardware to virtual compute, network, and storage elements. CPA further includes a *cloud hypervisor* that manages *virtual machines*. Cloud users can utilize the cloud resources through exposed *cloud APIs* and *customer portals*. As central goals of CEM are to (1) enable the creation of CDOs based on the structure of code models and cloud environments and to (2) model a cloud environment's CECs, elements such as CPA's customer portal are out of CEM's scope. As opposed to CEM, CPA is also a rather abstract model that omits details such as a cloud environment's pricing model or performance characteristics.

The **Service-Oriented Cloud Computing Architecture (SOCCA)** [Tsai et al. 2010] is an approach that aims to improve the interoperability of cloud

environments though employing SOA techniques. SOCCA comprises four layers that are briefly described below.

- *Individual Cloud Provider Layer:* This layer includes the individual cloud environments. To enable a federation of cloud environments and for building cloud-based applications that span multiple clouds, the cloud resources are exposed as services. For example, storage, computing, and communication resources are encapsulated into separate services and can be integrated, for example, by means of service orchestration.
- *Cloud Ontology Mapping Layer:* To overcome varying terminology and implementation details of the separate cloud environments, this layer provides ontologies that map the individual features to common concepts. For example, SOCCA proposes to build a *storage ontology*, *computing ontology*, and *I/O ontology*.
- *Cloud Broker Layer:* Cloud brokers bring together cloud users and one or more cloud environments. The cloud brokers act as agents and provide different services for matchmaking. For example, cloud environments can publish details regarding their individual resources and their pricing model, or SLAs can be dynamically negotiated.
- *SOA Layer:* This layer is utilized by application developers to register their services in a service registry, for instance. Trough interacting with the cloud broker layer, it is possible to define the deployment of the (composite) services.

As described before, SOCCA provides a comprehensive architecture for building cloud applications on the basis of multiple clouds and through incorporating SOA techniques. However, Tsai et al. [2010] use a rather high level of abstraction for describing SOCCA. In contrast, CEM provides a detailed meta-model that focuses, besides the integration of cloud applications, on the specification of single cloud environments. SOCCA does also not allow to model PaaS-based cloud environments. Though, defining PaaS-based cloud environments along with their frequently found restrictions (CECs) is possible with CEM.

A further approach that also combines SOA and cloud computing elements to define a cloud computing architecture is described by Zhang and Zhou [2009]. Their **Cloud Computing Open Architecture (CCOA)** consists of ten architectural modules. Similarly to the previously described SOCCA approach, CCOA also constitutes a high-level architecture that does not exhibit the level of detail of our meta-model CEM. CCOA contains elements for a *cloud core infrastructure*, such as utilized hardware and software, as well as *cloud IT infrastructure management*, *cloud provisioning service*, and *cloud subscription service*, for instance.

Similarly to SOCCA, CCOA also comprises several aspects that are out of CEM's scope, such as elements for modeling a *cloud partner dashboard* or *cloud ecosystem management*. The latter covers the management of memberships, for instance. Furthermore, CCOA does also not consider the specifics of PaaS-based cloud environments. Indeed, it includes an element that represents value-added services, such as business process as a service, but PaaS is not mentioned.

13.1.4 Cloud Application Modeling

Related work from cloud application modeling is listed in Table 13.4. The related work is examined according to the following five characteristics. The abbreviations in parentheses are used for referencing purposes.

Characteristics of Approaches for Cloud Application Modeling:

1. (IaaS) Covers IaaS-based cloud applications? Does the approach cover software systems that utilize and build on IaaS-based cloud environments?

✓: yes, high level of detail; (✓): yes, low level of detail; −: no

2. (PaaS) Covers PaaS-based cloud applications? Does the approach cover software systems that utilize and build on PaaS-based cloud environments?

✓: yes, high level of detail; (✓): yes, low level of detail; −: no

Table 13.4. Related work for cloud application modeling

Approach	IaaS[1]	PaaS[2]	Depl.[3]	Arch.[4]	Code[5]	Comments
Ardagna et al. [2012]	(✓)	(✓)	(✓)	(✓)	-	MODAClouds; Model-driven approach for the design and execution of applications on multiple clouds
Binz et al. [2011]	(✓)	(✓)	(✓)	(✓)	-	CMotion; Also considers adapters for incorporating cloud services
Brandtzæg et al. [2012a]	(✓)	(✓)	(✓)	(✓)	-	PIM4Cloud DSL; Internal DSL in Scala developed in the context of the REMICS project
Chapman et al. [2012]	✓	-	(✓)	✓	-	Techniques for defining cloud-based application architectures; Reservoir project context
Fehling et al. [2011]	(✓)	(✓)	(✓)	(✓)	-	Patterns for cloud-based application architectures
Guillén et al. [2013]	(✓)	(✓)	(✓)	-	-	Framework for developing cloud-agnostic applications
Hamdaqa et al. [2011]	(✓)	-	(✓)	(✓)	-	Cloud application meta-model
Koziolek [2010]	-	(✓)	-	(✓)	-	SPOSAD architectural style for multi-tenant software applications
Mietzner et al. [2009]	(✓)	(✓)	✓	(✓)	-	Cafe; Composite Application Framework
Nguyen et al. [2011]	(✓)	(✓)	(✓)	(✓)	-	Blueprint template for describing cloud-based services
SOFTEAM [2012]	(✓)	-	(✓)	(✓)	-	PIM4Cloud UML profile; Developed in the context of the REMICS project
Tianfield [2011]	(✓)	-	(✓)	-	-	Cloud Application Architecture (CAA)

3. (Depl.) Covers deployment aspects of cloud applications? Does the approach cover the deployment of application artifacts to cloud-based resources? For example, CloudMIG's CDOs include services that are created from a status quo deployment model and a mapping of the services to a number of VMs of specific VM instance types.

✓: yes, high level of detail; (✓): yes, low level of detail; −: no

4. (Arch.) Covers the software architecture of cloud applications? Does the approach cover the software architecture of cloud-based applications, such as their architectural components and corresponding connectors?

✓: yes, high level of detail; (✓): yes, low level of detail; −: no

5. (Code) Covers a source code model of cloud applications? Does the approach cover a source code model of cloud-based applications? CloudMIG utilizes KDM models for representing an application's source code elements, for example, its classes, packages, loops, and methods.

✓: yes, high level of detail; (✓): yes, low level of detail; −: no

Approaches for Cloud Application Modeling:

The **MODAClouds** project,[16] partially funded through EU's FP7, aims to provide methods and tools that enable applications to use multiple clouds [Ardagna et al. 2012]. At the time of writing the project just started. MODAClouds stands for *MOdel-Driven Approach for design and execution of applications on multiple Clouds*. Using several cloud environments to deploy applications may be reasonable for increasing reliability, scalability, or to exploit specific cloud environment capabilities for different systems of a large software landscape, for instance. In contrast, CloudMIG allows to compare various cloud environments and supports CloudMIG users to select a single target cloud environment. An overview of the MODAClouds approach is shown in Figure 13.13.

MODAClouds comprises a *Decision Support System (DSS)* that supports the comparison of different cloud providers. Similarly to the information

[16] http://www.modaclouds.eu/

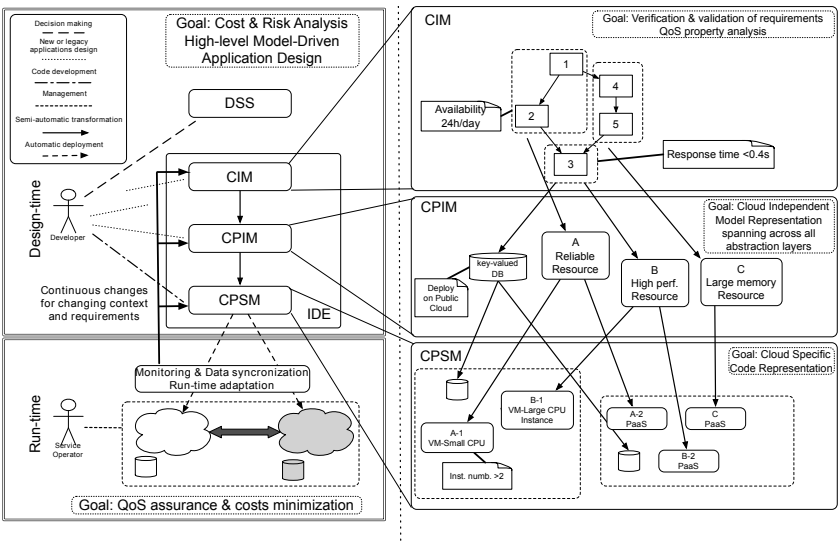


Figure 13.13. MODAClouds overview (from [Ardagna et al. 2012])

contained in CloudMIG's cloud profiles, DSS allows to review costs and non-functional characteristics of cloud environment candidates, for instance. For developing new applications on the basis of cloud technologies or migrating existing applications, MODAClouds provides a corresponding IDE. According to the MDE paradigm, this IDE allows to edit and analyze the three following types of models (cf. Figure 13.13).

- *Cloud-enabled Computation Independent Model (CIM)*: An application model augmented with QoS properties. CIM can be compared to CloudMIG's extracted KDM models that are enriched by transformed SMM models from monitoring log files for describing performance and usage characteristics, for instance. However, CIM does not seem to include the level of details provided by CloudMIG's KDM models.
- *Cloud-Provider Independent Model (CPIM)*: The CIM is transformed to CPIM that integrates abstract cloud patterns or resources. This model

is central to MODAClouds' idea of facilitating the use of several cloud environments, as is enables processing a cloud-based application model without having to decide for a specific cloud provider.

- ▻ *Cloud-Provider Specific Model (CPSM)*: If a target cloud environment has been identified, the CPIM is transformed to a CPSM that describes the binding to a provider in the form of specific deployment artifacts.

Furthermore, MODAClouds also covers the operation of cloud-based applications through its *run-time* layer. Actual operation support for cloud-based systems is not addressed by CloudMIG. However, both MODAClouds and CloudMIG cover IaaS and PaaS-based cloud environments as targets for cloud-based systems.

The Cloud Motion Framework (**CMotion**) [Binz et al. 2011] that considers the migration of applications in and between clouds was already described in Section 13.1.1. It also allows to model cloud-based applications. More specifically, it allows to define how the SUAs are deployed to resources of a specific cloud environment. However, in contrast to our approach CloudMIG that utilizes extracted KDM models, CMotion's application models are rather coarse-grained. They contain the artifacts of composite applications. Basic building blocks are operating systems, databases, and application archive files, for instance.

Deployment models can include substitutions of status quo components with cloud services. For example, a relational database that is used on premise may be substituted by a specific persistence service of a particular cloud provider. Similarly to CloudMIG's CDOs can different alternatives be generated. CMotion also enables to incorporate *adapters* that are needed for integrating specific cloud services. A similar element is provided by CloudMIG's CEM as well (cf. the `ServiceAdapter` class in CEM's *Mapping* package in Appendix A). However, in addition to it CMotion also provides so-called *manual adapters* that have to be built manually by developers for incorporating a specific cloud service if complex changes are required.

The REMICS project that addresses methods and tools for supporting migrations to the cloud was already mentioned in Section 13.1.1. In the context

of REMICS, several techniques are developed that also cover the modeling and specification of cloud-based applications and their actual deployment to the cloud. **PIM4Cloud** is, on the one hand, a UML profile [SOFTEAM 2012] for describing the deployment of applications to cloud environments. On the other hand, there exists a DSL named *PIM4Cloud DSL* [Brandtzæg et al. 2012a] that is implemented as an internal DSL in Scala and apparently originates from the **CloudML** modeling language [Brandtzæg et al. 2012b; Brandtzæg 2012].

The PIM4Cloud DSL also addresses the deployment of applications to cloud platforms and follows a component-based approach. In contrast to the UML profile, it is possible to actually perform cloud deployments with the PIM4Cloud DSL. The PIM4Cloud DSL uses a minimal component model for describing cloud-based applications. The approach requires two artifacts. A *PIM4Cloud DSL descriptor* constitutes the application model, an *infrastructure descriptor* provides a coarse-grained cloud environment model. An *interpreter* component is used to map both artifacts and to actually deploy the application. In contrast to CloudMIG, REMICS also considers the operation of migrated applications. Hence, after deploying an application, the interpreter returns a runtime model (PIM4Models@Runtime) of the application that can be used to query QoS properties, for instance.

The PIM4Cloud DSL and the PIM4Cloud UML profile differ in several ways. For example, the former also addresses PaaS-based cloud environments [Brandtzæg et al. 2012a], whereas the PIM4Cloud UML profile explicitly excludes PaaS [SOFTEAM 2012]. Furthermore, the meta-model described by the PIM4Cloud UML profile contains substantially more details than the DSL. In contrast to the PIM4Cloud UML profile and the DSL, CloudMIG employs a fine-grained application model, as the extracted KDM models also represent the source code of an SUA. This allows for in-depth analyses, for example, for automatically checking the conformance regarding a specific cloud environment.

Comprehensive techniques for describing cloud-based applications are also developed in the context of the **Reservoir** EU project that was already mentioned in Section 13.1.3. Chapman et al. [2012] define the following

six high-level requirements that should be met for being able to specify cloud-based applications.

- *Software composition*: It has to be possible to represent different computing nodes that host specific sets of components that form a service. These nodes may exhibit different resources, such as CPUs, memory, or operating systems. For describing the nodes and the service requirements, Chapman et al. [2012] propose a declarative *manifest language* that enables to specify application servers, virtual machines, and components, for instance. CloudMIG's pendant is given by the combination of CEM and KDM. The KDM models even contain fine-grained code models that can than be mapped to cloud resources originating from cloud profiles (instances of CEM). Furthermore, CloudMIG explicitly separates the on premise deployment architecture and the envisioned cloud-based target architecture. The former is specified with the help of a status quo deployment model. For the latter, CDOs are used that can be defined manually or generated automatically for finding near-optimal solutions.
- *Network topology*: Defining network topologies is also possible with CloudMIG. Instead of the mentioned *manifests* that are also used by the approach from Chapman et al. [2012] for this purpose, network connections can be defined in a status quo deployment model as well well as in a CDO.
- *Capacity adjustment*: For dynamic resource scaling, Chapman et al. [2012] define the elasticity rule syntax shown in Figure 13.14. This syntax and CloudMIG's reconfiguration rules (cf. Section 8.3) are similar to some extent. Both allow to add and remove VMs based on certain conditions. However, their elasticity rule syntax does not consider vertical scaling (cf. Section 8.3.1). In return, they allow to migrate running VMs from one host to another (see the `migrateVM` operation in Figure 13.14). This primitive is not supported by CloudMIG's reconfiguration rules.
- *Dependencies*: Chapman et al. [2012] also consider dependencies regarding the order of starting and stopping certain components. Those dependencies are not covered by our approach.

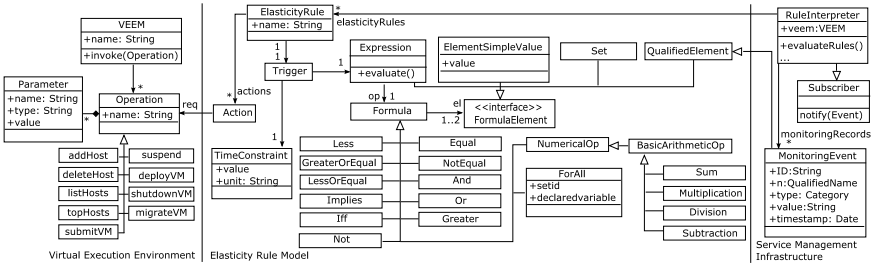


Figure 13.14. Reservoir's elasticity rules syntax (from [Chapman et al. 2012])

- *Location constraints*: Those constraints restrict the placement of components to physical locations, for example, for reasons of reliability. In contrast to Reservoir, CloudMIG does not consider deployments with more than one involved cloud environment. However, CloudMIG users (cf. Section 6.3.3) can manually define CDOs that adhere to those kinds of restrictions regarding different data centers of a chosen cloud environment.
- *Customization*: This requirement demands configurability of VM instances at runtime, for example, for dynamically connecting application server VM instances with a separate database VM instance. As CloudMIG does not cover the actual migration execution (cf. Section 6.3.2), such customizations are out of its scope.

Several **patterns for cloud-based application architectures** are described by Fehling et al. [2011]. These patterns are accompanied by patterns for different cloud types and cloud service models. The patterns for cloud-based application architectures propose high-level solutions for cloud-specific challenges in the four sub areas *basic architectural patterns*, *elasticity patterns*, *availability patterns*, and *multi-tenancy patterns*. For example, the *loose coupling* basic architectural pattern advises that an application's components should be decoupled, as this facilitates the distribution of the components to separate VMs, for instance. The patterns provide no detailed modeling concepts for specifying cloud-based applications. Hence, considering the

meta-models for describing cloud-based applications that are covered in this section, the patterns rather provide guidelines for effectively building or restructuring applications on the basis of these meta-models. However, a CloudMIG user may take the patterns into account when manually creating a CDO out of a built status quo deployment model.

A further approach that addresses the construction of applications that can be deployed to multiple cloud environments is presented by Guillén et al. [2013]. The approach aims to mitigate the commonly found vendor lock-in problem and to enable developers to distribute application components to separate cloud environments. The authors describe a framework for building new cloud-agnostic applications, i.e., applications that do not have to be tailored for specific cloud environments. In contrast, CloudMIG addresses the migration of existing software systems to the cloud. Guillén et al. [2013] use the notion of *cloud artifacts*. Cloud artifacts are generated automatically by the framework for each addressed cloud environment. They comprise the source code of an application component and also contain generated *adapters* and *interoperability services/clients*. Adapters are generated by the so-called *cloud adaptation engine* and enable a cloud artifact to use cloud services, for example, storage or messaging services of a specific cloud environment. As described in the context of the CMotion approach above, CloudMIG's CEM also provides means for modeling those adapters. Interoperability services/clients are generated by the *service generation engine* and enable cross-cloud interaction with other cloud artifacts. The framework provides a coarse-grained *cloud variability model* that describes all supported cloud environments as feature models. Furthermore, a developer is required to build a deployment plan that is processed by the framework to generate the cloud artifacts.

Compared with CloudMIG, the framework does not analyze an application's source code, for example, for detecting CEC violations or augmenting included method calls with historical usage data. However, our approach CloudMIG and the framework presented by Guillén et al. [2013] both address PaaS and IaaS-based cloud environments. Though, their cloud artifact building blocks do not consider a cloud's elasticity, whereas the CDOs that are generated by CDOXplorer include optimized reconfiguration rules.

For modeling cloud-based applications and to facilitate reasoning about the software architecture of those applications, Hamdaqa et al. [2011] propose a **cloud application meta-model**. The meta-model consists of a set of associated classes that represent certain cloud-related concepts. For example, a software system that is deployed to a cloud environment is represented by the class `CloudApplication` that consists of several `CloudTasks`. The latter describes abstract functionalities provided by a cloud-based application. Specific tasks are, for example, given by the subclasses `CloudFrontTask` and `CloudRotorTask`. The former describes services that are directly accessed by users, such as a web application, whereas the latter describes computations running in the background. Furthermore, the meta-model mixes the cloud provider and cloud user perspective. For example, it also contains elements for describing a cloud environment's VM instance types and offered persistence services. `CloudTasks` always have to be accompanied by an element (`ConfigurationData`) that describes, among others, the size of a VM. Hence, the meta-model does not allow to define deployments to PaaS-based cloud environments that do not employ VMs as building block resources.

In general, the meta-model provides a rather coarse-grained view on a cloud-based application. In contrast, CloudMIG employs a combination of full-fledged KDM models that are mapped to elements from the cloud domain which are defined in cloud profiles with the help of CEM. Compared to CloudMIG's CDO model, the meta-model presented by Hamdaqa et al. [2011] does also not allow to define reconfiguration rules for dynamic resource scaling.

An architectural style for a specific class of cloud-based applications is presented by Koziolek [2010]. The so-called **Shared, Polymorphic, Scalable Application and Data (SPOSAD)** style addresses multi-tenant software applications that are deployed to PaaS-based cloud environments. Figure 13.15 shows an overview of the SPOSAD style. It extends the n-tier architectural style as it comprises a client tier, application tier, and database tier. Multiple tenants can be served by a corresponding application. Each tenant may exhibit multiple clients that access the multi-tenant application via a *web browser*, for instance. The application is built from a single code base for all tenants and runs on a common infrastructure. Basic building blocks of the

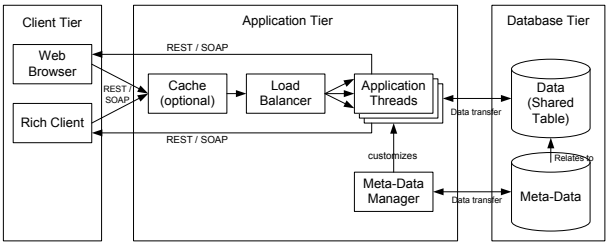


Figure 13.15. SPOSAD architectural style (from [Koziolek 2010])

application tier are several *applications threads* and a *meta-data manager*. Each application thread runs the same code and enables the application to scale out. The meta-data manager is used for customizing the application threads with regard to the different tenants. User requests are scheduled with the help of a *load balancer* component. It is also possible to integrate further optional components, such as a *cache* component (cf. Figure 13.15). The components *meta-data* and *data* in the database tier are used for storing the tenants' meta configuration data and application data, respectively.

Comparing SPOSAD with CloudMIG's means for modeling cloud-based applications, our approach addresses common enterprise applications, whereas SPOSAD targets multi-tenant applications that run in PaaS-based cloud environments. Furthermore, CloudMIG's CDOs support IaaS-based cloud environments. When automatically generating CDO candidates, a status quo deployment model is used as a basis for creating optimized target architectures. Cloud services, such as the *cache* component shown in Figure 13.15, can be integrated in cloud-based application architectures with CEM's `ServiceAdapter` element (cf. the *Mapping* package in Appendix A).

A framework that also considers multi-tenant cloud-based applications is the **Composite Application Framework (Cafe)** [Mietzner et al. 2009]. Besides describing cloud-based applications, it can be used to model cloud infrastructures and corresponding deployment aspects. Cafe also supports automatic provisioning of the specified applications. Applications are modeled as graphs. The nodes represent components, edges represent commu-

nication or deployment relationships, i.e., components can be deployed on other components. Components can also be augmented with properties, for example, the programming language used to implement the component or the employed *multi-tenancy pattern* can be specified. An exemplary, simple multi-tenancy pattern is *single instance*. It determines that a component is used together by multiple tenants.

The provisioning procedure aims to fulfill defined dependencies and also can deploy components across different cloud environments. For actually deploying components, Cafe builds on the authors' previous work and uses *provisioning services* [Mietzner and Leymann 2008]. Those provisioning services can start new VM instances or set up runtime environments for components, for instance. In contrast to CloudMIG, the model for describing cloud-based applications is rather coarse-grained as it does not include details regarding the implementation of its components. CloudMIG uses KDM models that exhibit those details. They get mapped to resources included in cloud profiles. However, CloudMIG does not address the actual deployment of SUAs (cf. Section 6.3.2).

For specifying cloud-based applications that are provided to consumers as services, Nguyen et al. [2011] propose the **Blueprint template**. Providers of those services can use the Blueprint template to describe their offering in a *Blueprint*, i.e., an instance of the Blueprint template. The Blueprint template is similar to AWS CloudFormation[17] but Nguyen et al. [2011] aim to provide a vendor-neutral solution. The Blueprint template is structured in several sections. For example, the *resource requirements section* lists the cloud resources that are required to run the service, e.g., a specific application server. Furthermore, so-called *Blueprint extensions* can be used to provide additional information for the sections' elements. For example, it is possible to specify *QoS profiles* and *policy profiles*. The former can be used to describe QoS characteristics of cloud resources, e.g., a guaranteed throughput of requests. The latter may restrict the deployment of applications to specific

[17] AWS CloudFormation (`http://aws.amazon.com/cloudformation/`) allows to describe the composition of resources from the Amazon cloud in a reusable *template*. For example, a template may define which VM image has to be used in each specific geographical region and what VM instance type should be utilized to run an application.

geographical regions for reasons of compliance, for instance. On the one hand, these profiles may be either used to specify properties of the offered service with respect to its consumers. On the other hand, the profiles can be utilized to pose requirements to the underlying cloud platform. Both aspects are covered by CloudMIG as well. Simulating CDOs with the help of CDOSim also yields properties such as expected response times. This view is relevant for users of the migrated system. Furthermore, CloudMIG also incorporates workload profiles from monitoring data that describe the system usage prior to migration. Hence, the workload profiles that augment the application architecture also implicitly constitute requirements regarding potential cloud environments, as a CDO candidate has to be able to handle a previous usage pattern. Furthermore, in contrast to the Blueprint template, CloudMIG's CDOs focus on the application architecture that is composed of several services and accompanied reconfiguration rules (cf. Section 8.3). Moreover, each service is also described by a detailed KDM code model. In comparison, the Blueprint template does only list coarse-grained *implementation artifacts* that make up the service.

In the previous Section 13.1.3, we already mentioned the Cloud Platform Architecture (CPA) from Tianfield [2011]. The author separates cloud environment and cloud application concerns. For describing cloud environments the CPA is used. For modeling cloud-based applications, Tianfield [2011] proposes the **Cloud Application Architecture (CAA)**. It consists of the following three layers.

- *Business Service and Process (BSP) Layer*: This layer manages cloud resources, for example, it can dynamically allocate and deallocate resources. Furthermore, this layer provides means for measuring QoS characteristics and ensuring conformance with stipulated SLAs.
- *Cloud Broker Layer*: Cloud brokers constitute agents that select cloud service providers (CSPs) for deploying specific services from the BSP layer.
- *Cloud Service Provider (CSP) Layer*: CSPs offer infrastructure cloud resources. These cloud resources run *virtual appliances*. For example, a

virtual appliance may be implemented as VM image that contains all components that build a service from the BSP layer.

In contrast to CloudMIG, the basic building blocks for modeling the actual software system that constitutes a service are therefore given by virtual appliance elements. This model is rather coarse-grained as it does not even define how components and corresponding connectors are represented.

Compared with CloudMIG, also no detailed source code models are mapped to the virtual appliances, for example, for statically analyzing raised CEC violations. Furthermore, CloudMIG does not consider a broker role that selects a cloud environment and a specific deployment architecture, as CloudMIG aims to support future SaaS providers to compare well-suited cloud environments and CDOs.

13.2 Conformance Checking

CloudMIG's conformance checking approach examines whether a software system complies to a specification of a cloud environment that is represented by a cloud profile. More precisely, the cloud profile specification contains a set of CEC definitions that also constitute specific requirements regarding a software system because it has to comply with these CECs.

In the general case, *checking the conformance* of a software system (or more general, of an *object under analysis*) means to merely evaluate whether the system complies to some kind of *specification* that includes a set of requirements. Hence, the general notion of conformance checking is broadly defined and is therefore also used in a plethora of contexts. Some examples are listed below.

- *API protocol conformance*: Checking the conformance with an API protocol ensures that the API's methods are only used in a defined way, for instance, that they are called in a specific order (e.g., see [Li et al. 2010b; Ball et al. 2011]). For example, after acquiring resources with the method

`acquire()`, an API protocol may state that the resources have to be released with the corresponding method `release()`.

- *Business process conformance*: Data from monitoring log files can be used to check a software system's conformance with specified business processes. That means, it can be checked whether the models of business processes correspond to the way they are actually implemented (e.g., see [Rozinat and van der Aalst 2008; van der Aalst 2011a]).
- *Coding rule conformance*: Coding rules define the structure of source code files and issues related to the type face, for example, regarding the indentation of statements in the source code. Checking the conformance of coding rules may support to retain maintainability, for instance (e.g., see [Boogerd and Moonen 2008; Marpons et al. 2008]).
- *Security policy conformance*: Security policies can specify access rights or enforce data encryption for storing specific file types, for instance. Checking the conformance with those security policies may support preventing data theft, for instance (e.g., see [Hansen and Oleshchuk 2005; Hu et al. 2007]).

There exist few cloud migration approaches that consider checking the technical conformance of SUAs regarding cloud environments. They were already described in Section 13.1. However, these approaches only address conformance checking on a conceptual level and neither describe nor provide means for automatically detecting CEC violations. Furthermore, there exist cloud migration approaches that also use the notion of *constraints*, but the term is used with another meaning (cf. Section 13.1 for details). Hence, to the best of our knowledge, there does not exist directly comparable related work that covers conformance checking regarding CECs in the sense it is provided by CloudMIG.

However, as described above, conformance checking can be defined with a broader scope for the general case as evaluating whether a software system complies with requirements that are contained in a specification. Therefore, considering that the approach CloudMIG addresses conformance checking

in a software evolution and also cloud computing context, this section describes related work from the following two subareas.

Conformance Checking in the Context of Software Evolution: CloudMIG supports the migration of software systems to the cloud. Migrating an on premise software system to a cloud environment pervasively modifies and evolves the system. Hence, CloudMIG addresses conformance checking in the context of software evolution.

Conformance Checking in the Context of Cloud Computing: As CloudMIG aims to detect an SUA's violations regarding constrains that are exhibited by cloud environments, the characteristics of its conformance checking approach are closely linked to the cloud domain. Hence, we also report on related work regarding conformance checking in the context of cloud computing.

Please note that even in these general forms, conformance checking still covers a large body of work. Hence, we do not make claims of being complete but present an overview and core subjects from both subareas.

The related work in both subareas is examined according to the following four characteristics. The abbreviations in parentheses are used for referencing purposes.

Characteristics of Approaches for Conformance Checking in the Context of Software Evolution and Cloud Computing:

1. (Spec.) What constitutes the specification the object under analysis has to comply with? As described above, when checking the conformance of an object under analysis, it has to comply with a specification. This characteristic states the specification.

2. (OUA) What constitutes the object under analysis that has to comply with the specification? As described above, when checking the

conformance against a specification, an object under analysis is evaluated. This characteristic states the object under analysis.

3. (Cre.) Is the specification created automatically? What is the level of automation usually employed to create the specification? Is is usually created automatically?

✓: automatically; (✓): semi-automatically; −: manually

4. (Check.) Is the conformance checking procedure performed automatically? What is the level of automation usually employed for the actual conformance checking procedure? Is it usually performed automatically?

✓: automatically; (✓): semi-automatically; −: manually

13.2.1 Conformance Checking in the Context of Software Evolution

Related work from conformance checking in the context of software evolution is listed in Table 13.5. The related work is examined according to the four previously described characteristics.

Software reflexion models [Murphy et al. 1995] enable comparing high-level, structural models of a software system with its actual source code. For example, a common problem in software evolution is design erosion [van Gurp and Bosch 2002], i.e., the envisioned design of an application and its actual implementation tend to diverge over time. If both artifacts (design and source code) are available, a reflexion model can be created automatically. A reflexion model shows the difference between both artifacts, for example, where associations between classes exist in the source code that are not present in the design model. Considering the notion of conformance checking, reflexion models enable to check the source code (object under analysis) for conformance with the envisioned design (specification).

In common with CloudMIG's conformance checking approach, the specifications are created manually. As described before, the specifications for

Table 13.5. Related work for conformance checking in the context of software evolution

Approach	Spec.[1]	OUA[2]	Cre.[3]	Check.[4]	Comments
Bittencourt [2010]	Software architecture	Source code	-	✓	Evolutionary reflexion models (ERMs)
Brunet et al. [2009]	Software architecture	Source code	-	✓	Design tests
Chowdhury and Meyers [1993]	Stylistic, implementation, and design CCEL expressions	C++ source code	-	✓	C++ Constraint Expression Language (CCEL)
Davis et al. [2003]	Software architecture	COTS components	(✓)	(✓)	Interoperability conflicts
Garlan et al. [1995]	Software architecture	Reusable entities	(✓)	(✓)	Architectural mismatch
Itsykson and Zozulya [2011]	Present library specification with program annotation language (PanLang)	Target library specification with PanLang	-	✓	Application migration; Conformance check of target libraries
Jung and Saglietti [2005]	Application properties described with interface and constraint language	Reusable components	(✓)	✓	Software reuse; Static and dynamic inconsistency detection
Ma et al. [2007]	Set of existing system services and dependencies	Service hosting environment candidate	(✓)	✓	Migration of services between service hosting environments
Murphy et al. [1995]	Software architecture	Source code	-	✓	Software reflexion models
Passos et al. [2010]	Software architecture/ queries/ design rules	Source code	-	✓	Comparison of static architecture-conf. checking techniques
Zhao et al. [2012]	Set of structural constraints	Runtime architecture	-	✓	Conformance checking of SOA topologies

CloudMIG's conformance checking process constitute the cloud profiles that contain CEC definitions. There exist methods for automatically reconstructing a software's design and architecture (e.g., see [Koschke 2009]), however, the reflexion models target handcrafted high-level software models. Nevertheless, the conformance checking processes themselves can be performed automatically in both cases, i.e., computing reflexion models and detecting CEC violations.

An extension of the classical reflexion models that aligns with agile development processes is proposed by Bittencourt [2010]. To cope with the specific challenges posed by software evolution, Bittencourt [2010] presents evolutionary reflexion models (ERMs). ERMs cover the three following modifications to the original software reflexion models.

- ERMs use the concept of *design tests* that was proposed by Brunet et al. [2009]. Design tests integrate the conformance checking process, that compares the envisioned software architecture or design with the source code of a software system, in executable tests. These tests can be incorporated in a continuous integration build process, for instance. Hence, deviations between source code and high-level, structural models of a software system can be checked automatically and often. Therefore, design tests are well-suited for coping with specific concerns of software evolution.
- ERMs support developers in semi-automatically adapting mappings between source code and design.
- ERMs can propose architectural restructuring operations based on deteriorative source code changes.

As with the original reflexion models and our approach CloudMIG, the actual conformance checking process is also performed automatically in the case of design tests and ERMs.

In the context of statically checking the conformance of an application's implementation to a specified software architecture, Passos et al. [2010] compare software reflexion models with source code query languages and

dependency-structure matrices. Source code query languages can be used to query a range of diverse source code properties, for example, it is often possible to retrieve software metrics, specific statements, or information regarding the employed coding style. Source code query languages often also can be used to extract structural information that is relevant for checking the conformance regarding a defined architecture.

A dependency-structure matrix is a two-dimensional matrix that contains information regarding dependency properties of two classes of an application. For example, a cell might represent the number of method calls between both classes. Those properties might be extracted automatically, however, architectural constraints have to be defined manually. Compared with software reflexion models, for both techniques the specifications used for conformance checking exhibit a lower level of abstraction. Software reflexion models employ an architecture description provided by software architects and therefore require mappings to the source code. In contrast, using source code query languages, each architectural constraint is represented by at least one query. Dependency-structure matrices employ views on the package and class level and require design rules for each constraint. Compared with CloudMIG's conformance checking approach, the specifications are given by cloud profiles that provide fine-grained CEC definitions.

An approach that not only can be used to check the conformance with an envisioned design or architecture when evolving a software system, but also with stylistic and implementation aspects, is contributed by Chowdhury and Meyers [1993]. The authors propose the C++ Constraint Expression Language (CCEL) that allows to define constraints regarding the design, coding style, and implementation properties of C++ programs.

Listing 13.1 shows an example that poses an implementation constraint (`VirtualDtorInBase`) and requires each C++ base class to contain a virtual destructor [Oualline 2003]. The example uses two classes `B` and `D` where `D` is a descendant of class `B` (Line 3) and therefore, `B` constitutes the base class. The `Assert` statement (Lines 4-6) states that the base class `B` has to contain a member function `bmf` that complies with C++'s naming scheme of a virtual

```
1 VirtualDtorInBase{
2   Class B;
3   Class D | D.is_descendant(B);
4   Assert(MemberFunction B::bmf; |
5     bmf.name() == "~" + B.name() &&
6     bmf.is_virtual());
7 };
```

Listing 13.1. CCEL example that requires each C++ base class to have a virtual destructor (from [Chowdhury and Meyers 1993])

destructor, i.e., that the method name is given by a tilde followed by the name of the corresponding class (Line 5). Furthermore, the method has to be declared as virtual function using the keyword `virtual` (Line 6). In common with CloudMIG's conformance checking approach, the conformance of a C++ program (the object under analysis) with CCEL expressions can be checked automatically. The specifications are given by the CCEL expressions that state stylistic, implementation, and design constraints that have to be defined manually.

A further area in the context of software evolution that necessitates checking the conformance of software entities is addressed by, for example, Davis et al. [2003] and Garlan et al. [1995]. For developing applications, reusing existing components promises to decrease development effort and lower costs. However, software reuse bears numerous difficulties if parts of complex systems have to be integrated in new systems. Garlan et al. [1995] coined the term *architectural mismatch*. It describes implicit assumptions many system components make about their environment that are hard to analyze. Those implicit dependencies, such as expecting specific powerful file systems or particular versions of third-party libraries that may not be available on a target platform, can impede and eliminate potential benefits.

Davis et al. [2003] classify those interoperability conflicts and focus on the integration of COTS components. A common interoperability problem class is *invalid data*, for instance, that describes the problems of dealing with different data formats. For addressing the difficulties arising from the reuse of software that are described by Garlan et al. [1995] and Davis et al. [2003], the conformance of the reusable parts with an existing system has to be

checked. For example, Jung and Saglietti [2005] propose an interface and constraint language that describes global properties of the application that should reuse existing components. Depending on the complexity and types of inconsistencies, constraint violations may be detectable statically or only during runtime.

CloudMIG's conformance checking approach distinguishes CEC violations due to their detectability as well. We propose the five detectability categories DC1-DC5 (cf. Section 7.1.3). Compared with CloudMIG, the approach presented by Jung and Saglietti [2005] also allows to automatically detect constraint violations. However, due to the inherent complexity, some of the interoperability conflicts and problems of reusing existing components that are described by Garlan et al. [1995] and Davis et al. [2003] are not yet covered by automated techniques. Furthermore, there exists an essential difference between checking the conformance of source code and the envisioned design with the help of software reflexion models that was described above, and checking for incompatibilities in the context of software reuse. The latter allows to use software architecture models that were extracted automatically. When checking the compatibility of a component with an existing application, it is reasonable to check against the application's actual architecture.

An approach that addresses conformance checking in the context of application migration is presented by Itsykson and Zozulya [2011]. The approach focuses on applications written in C and the substitution of libraries that are used by those applications. For example, when migrating from a Windows to a Linux operating system, utilized libraries have to be substituted. Itsykson and Zozulya [2011] represent applications as abstract semantic graphs (ASGs).

The authors present a program annotation language (PanLang) that can be used to manually specify the libraries. Such specifications can then be employed to check the conformance of target libraries. For each existing library, this conformance checking procedure utilizes execution traces from both the old and the new library and compares them. Hence, the new libraries constitute the objects under analysis that have to comply with the

specification. In common with CloudMIG, the objects under analysis can be checked automatically for conformance with the specification.

Ma et al. [2007] consider the evolution of Service-Oriented Architectures (SOAs). Services in a SOA are hosted in *service hosting environments* that themselves are composed of a set of *system services*, such as a file system or Domain Name System (DNS). The authors address the migration of services between different service hosting environments. Corresponding service hosting environment models can be generated automatically and differ in terms of provided QoS properties, for instance. As the system services exhibit complex dependencies, Ma et al. [2007] propose a model-based dependency management. This dependency management is employed for checking the conformance of a specific, generated service hosting environment model. Hence, the specification is given by the set of the existing system services and their dependencies. A potential service hosting environment candidate has to comply with this specification. In contrast, CloudMIG checks the conformance of the existing system with a given specification (cloud profile).

The evolution of SOA-based systems is also addressed by Zhao et al. [2012]. The authors cover modifications to the topology of SOA-based systems during runtime. For example, new services may be added or the communication channels between services may be altered. The resulting runtime architectures have to comply with topological constraints. Similarly to CDOXplorer that disregards infeasible CDO candidates (cf. Section 8.3.4), the conformance checking approach from Zhao et al. [2012] rejects invalid runtime architectures. Compared with CloudMIG's conformance checking approach that aims to detect CEC violations, the objects under analysis of the approach from Zhao et al. [2012] are given by potential runtime architectures. Runtime architectures have to comply with a specification that is composed of a set of structural constraints.

13.2.2 Conformance Checking in the Context of Cloud Computing

Related work from conformance checking in the context of cloud computing is listed in Table 13.6. The related work is examined according to the same four characteristics that were used to describe the related work regarding conformance checking in the context of software evolution (cf. Section 13.2.1).

Business processes that are implemented on the basis of cloud infrastructures are considered by van der Aalst [2011c]. *Configurable process models* basically define the arrangement of activities that are relevant for various tenants. These process models can be tailored according to the needs of the tenants and are therefore also called *multi-tenant processes*. The author uses *causal nets (C-nets)* [van der Aalst 2011b] as a means for describing business processes. For example, as a primitive for configuring processes, C-nets consider the removal of behavior instead of refining or adding activities. C-nets are also used as a basis for conformance checking.

In the context of cloud-based, multi-tenant processes, conformance checking mainly addresses the comparison of event log data with process models. A measure for evaluating the conformance is *fitness*, i.e., the degree the event log data complies with a given process model. However, it is also possible to compare different process models that were configured by tenants. These process models have to conform to the configurable process model. Hence, the specification can either be given by the configurable process model or by a configured instance of it, i.e., a concrete process model. In common with the specifications covered by CloudMIG's conformance checking process (cloud profiles), both kinds of specifications have to be created manually.[18]

Cloud-based business processes are also addressed by Accorsi et al. [2011], their approach is called **Comcert**. However, the authors check the conformance regarding compliance requirements, such as privacy and security

[18] In the context of conformance checking, automatically discovered process models are not used as specifications, but at most as objects under analysis that are checked regarding their conformance.

Table 13.6. Related work for conformance checking in the context of cloud computing

Approach	Spec.[1]	OUA[2]	Cre.[3]	Check.[4]	Comments
van der Aalst [2011c]	Configurable process model or concrete process model	Process model or event log data	-	✓	Cloud-based multi-tenant processes; Conformance checking on the basis of C-nets
Accorsi et al. [2011]	Compliance rules	Design time process models	-	✓	Comcert; Cloud-based business processes; Conformance checking on the basis of Petri nets
Brandic et al. [2010]	Compliance Level Agreements (CLAs)	Applications running in C3-aware cloud environments	(✓)	✓	Compliant Cloud Computing (C3)
Chazalet [2010]	Service level agreement	Service at runtime	-	✓	Service level checking
Du et al. [2010]	Integrity attestation graph	Multi-cloud dataflow processing system	✓	✓	RunTest; Checking integrity of dataflow processing systems
Jenkins et al. [2011]	Cloud API test cases	Cloud environment	-	✓	Framework for testing cloud environments
Kourtesis and Paraskakis [2011]	Policies of PaaS-based cloud environments	Cloud application artifacts	-	✓	CAST platform; Policy conformance checking
Rings et al. [2011]	Telecommunication standard	Implementation under test (IUT)	-	✓	Conformance checking vs. interoperability testing
Ullah [2012]	ISO/IEC 27002:2005	Infrastructure that hosts cloud environment software	-	(✓)	Security compliance tool for cloud providers

concerns. Furthermore, the compliance check is accomplished at design time, i.e., during the modeling of the business processes. Moreover, in contrast to the approach from van der Aalst [2011c] that was described above, Comcert uses Petri nets instead of C-nets for specifying the business processes. Accorsi et al. [2011] also utilize Petri nets to define the compliance rules.

Therefore, compared with CloudMIG, the conformance checking process by Accorsi et al. [2011] employs the same technology for representing the specification (compliance rules) and the objects under analysis (cloud-based business processes). In contrast, CloudMIG's specification artifacts are given by cloud profiles that are instances of CEM, whereas the objects under analysis are represented by KDM models.

Compliance regarding security, privacy, and trust concerns is also considered by Brandic et al. [2010]. However, compared with the two approaches described before, the authors do not address business process modeling. In contrast, Brandic et al. [2010] propose an approach for compliance management regarding cloud-based applications. The approach is termed **Compliant Cloud Computing (C3)**. It supports the specification of compliance requirements for an application via so-called Compliance Level Agreements (CLAs). CLAs are similar to SLAs, but extend those with elements for covering the certification and auditing of requirement definitions regarding security, privacy, and trust. CLAs can be generated from requirement descriptions that are created with the help of specific DSLs.

The C3 approach also aims at facilitating the cloud provider selection process, i.e., selecting a cloud environment that is capable of fulfilling cloud user requirements. The C3 architecture includes a middleware component that is installed to cloud environment resources. This middleware component supports the application deployment to those *C3-aware cloud providers* and enforces adherence to compliance requirements at runtime. For example, the middleware could check that specific user data might only be consumed by components that are running in particular locations. Comparing the conformance checking processes of C3 and CloudMIG, the specifications are given by CLAs and cloud profiles, respectively. The objects

under analysis correspond to the applications running in C3-aware cloud environments and extracted KDM models in the cases of C3 and CloudMIG, respectively.

As described before, CLAs are, to some degree, similar to conventional SLAs. Validating SLAs is a common task during the operation of cloud-based applications. SLAs may determine a set of Service Level Objectives (SLOs), such as a maximum median network latency or response time (per timeframe) for a service, for instance. However, in the context of cloud computing, SLAs can be declared on various levels. For example, an IaaS-based cloud environment may offer a storage service and therefore guarantee a specific read/write speed for low-level read/write operations. In contrast, a SaaS provider may use this IaaS-based service to build a web-based application that is offered to SaaS users. Furthermore, the SaaS provider might also offer an SLA to SaaS users but himself having to consider the SLA from the IaaS-based service while settling the corresponding SLOs.

Considering those scenarios, Chazalet [2010] describes an approach for *service level checking*. The approach proposes an architecture that consists of the following three layers.

- *Service monitoring*: Highest layer in the architecture. Retrieves data from the data collector layer and checks the conformance to an SLA.
- *Data collector*: Obtains data from core monitoring layer. Performs computations and provides data to service monitoring layer according to the abstraction level of SLOs that are defined in a corresponding SLA.
- *Core monitoring*: Lowest layer in the architecture. Collects low-level data from service probes. Additionally performs at most simple tasks such as basic data aggregation or filtering.

The specifications used in the service level checking process are therefore given by SLAs and the included SLOs, whereas CloudMIG addresses cloud profile specifications and included CEC definitions. The service level checking process and the conformance checking process of CloudMIG also differ

as the former performs dynamic analyses during regular operation (online), whereas CloudMIG focuses on static (offline) analyses on the basis of KDM models.

An approach that also, in contrast to CloudMIG, performs online conformance checking is presented by Du et al. [2010]. The authors consider a specific class of cloud-based applications that the authors call *dataflow processing systems*, i.e., systems that perform computations on large-scale data streams. Du et al. [2010] present their framework **RunTest** that can check the integrity of dataflow processing systems that are deployed to multiple cloud environments. The approach aims to ensure trustworthiness and to identify inconsistent results and malicious service offerings. According to the authors, RunTest is constructed to be a lightweight approach as it does not require a trustworthy, third-party control station.

The specification used by RunTest for its conformance checking process—assuring the integrity of dataflow processing systems—is not static. Du et al. [2010] propose a dynamic *integrity attestation graph* that is updated at runtime and serves as a basis for detecting results that are not trustworthy. In contrast, the specifications used for CloudMIG's conformance checking approach (cloud profiles) are not modified during the detection of CEC violations. However, compared with CloudMIG's cloud profiles, the integrity attestation graphs can be created automatically.

In an abstract sense, the further approach from Jenkins et al. [2011] also aims to check the conformance of cloud environments to their specifications. More precisely, the authors present a framework that can be used to test the functionality of APIs that are provided by cloud environments. Underlying cloud infrastructures and platforms play an essential role when considering the reliability and also the correctness of cloud-based applications. However, as the internals of most cloud environments are out of cloud users' control, Jenkins et al. [2011] argue that it is important to also test their offered functionality for being able to provide reliable services on top of IaaS or PaaS-based clouds. For example, corresponding tests could be considered as black box tests and also be included in a continuous integration build process.

For testing a specific cloud environment, Jenkins et al. [2011] propose to build a custom application that can run in the corresponding cloud environment as client application. For each specific API that is exhibited by the cloud environment, a plugin is added that tests the API. For example, the authors provide a prototype implementation that tests Google App Engine (GAE) for Java that is also used in our evaluation of CloudMIG's conformance checking approach (cf. Chapter 11). Among others, GAE for Java provides APIs for storing data and fetching URLs. The prototype from Jenkins et al. [2011] includes plugins for those APIs, for instance.

In contrast to CloudMIG's conformance checking approach, the framework from Jenkins et al. [2011] has to perform a dynamic analysis for checking the conformance of a cloud environment regarding its specification. The specification itself is given by test cases that check the API description. Hence, the framework has to be deployed and run for providing results. In comparison, our conformance checking approach processes extracted KDM models and performs an offline check with regard to a specification (a cloud profile).

An approach that also addresses conformance checking for ensuring the reliability and stability of cloud environments is presented by Kourtesis and Paraskakis [2011]. However, their approach takes a cloud provider perspective. Instead of testing cloud environments by cloud users as is covered by Jenkins et al. [2011], Kourtesis and Paraskakis [2011] aim to support cloud providers that build and offer a PaaS-based cloud environment in establishing corresponding governance processes. The approach originates from the context of the research project CAST [Kourtesis et al. 2011] that investigates methods and techniques for building PaaS-based cloud environments. The *CAST platform* was developed within the frame of this research project. It is a PaaS-based cloud environment that enables to develop and deploy enterprise applications. Kourtesis and Paraskakis [2011] address impairments of reliability and stability that may arise when deploying deficient cloud-based applications that do not comply with the platform's specification. Hence, Kourtesis and Paraskakis [2011] propose a governance system that includes five core functionalities, such as tracking the dependencies of cloud-based applications and managing their lifecycle.

A further functionality refers to automatically checking the conformance of cloud-based application artifacts with policies that are defined within a platform. For example, such a policy may determine the structure of configuration files. Corresponding conformance checks are triggered when those artifacts are created or changed. In contrast, CloudMIG's conformance checking procedure is performed apart from the live system.

Rings et al. [2011] contribute a testing framework for assessing the interoperability of grid computing and cloud computing infrastructures in the context of the telecommunication domain. The authors address the simultaneous deployment of applications to those distinct infrastructures to facilitate, for example, failover and maximization of resource provisioning scenarios. In this context, the authors elaborate on the difference between conformance checking and testing for interoperability. Rings et al. [2011] emphasize that *conformance checking* generally refers to evaluating an implementation regarding requirements that are included in a specification. This notion of conformance checking matches with the concept used for describing corresponding related work in this Section 13.2. In contrast, *interoperability testing* is considered by the authors as examining whether "[...] implementations provide end-to-end functionality as described or implied by a specification." [Rings et al. 2011]

The general conformance checking procedure of the testing framework from Rings et al. [2011] utilizes test drivers to analyze an *implementation under test (IUT)*. To test the conformance regarding a specification, the IUT is regarded as a black box that is deployed to a *system under test (SUT)*. Hence, the tests do not make any assumptions regarding the IUT's specific implementation that exceed the characteristics that are determined by the specification. This concept corresponds to CloudMIG's conformance checking procedure, as the scope for detecting CEC violations is exclusively determined by the CEC definitions contained in a cloud profile.

An important aspect when migrating software systems to the cloud is compliance regarding security regulations and policies. However, auditing processes and the assessment of cloud providers' security compliance most often have to be performed manually. In this context, Ullah [2012] developed

an application that aims to automatically evaluate the security compliance of cloud providers. The application has to be deployed by cloud providers as some of its data collection modes need access to system components that are most often not available to cloud users. The high-level architecture of the application includes the following three basic components.

- *Data Collection Engine*: This component is deployed to the infrastructure employed for hosting the cloud environment. It collects data used for assessing the cloud environment's security compliance. Ullah [2012] describes four possible approaches for collecting relevant data, for example, querying APIs that are provided by the cloud environment or delegation to a third-party vulnerability assessment tool. These two approaches are also implemented in the application from Ullah [2012].
- *Verification Engine*: This component uses data from the Data Collection Engine to assess the cloud environment's security compliance regarding security controls from the ISO/IEC 27002:2005 information security standard [ISO/IEC 2008].
- *CloudAudit API*: CloudAudit[19] is a framework from the Cloud Security Alliance[20] that aims to provide a common interface for, among others, representing and querying auditing information. The application from Ullah [2012] provides the results from the Verification Engine component to cloud users according to the CloudAudit API definition.

In contrast to CloudMIG's conformance checking process, the compliance checking procedure from Ullah [2012] may incorporate a manual step. Cloud providers may also need to manually enter information to the Data Collection Engine in cases automatic gathering is not possible or is insufficient. Furthermore, CloudMIG's conformance checking approach processes KDM models of an object under analysis (the SUA), whereas the application from Ullah [2012] requires access to the infrastructure that is hosting cloud environment software.

[19] https://cloudsecurityalliance.org/research/cloudaudit/

[20] https://cloudsecurityalliance.org/

13.3 Deployment and Reconfiguration Optimization

This section discusses related work concerning the optimization of deployment architectures and reconfiguration rules. Solely approaches that tackle related problems from a cloud user perspective are considered. Cloud users want to deploy software to the cloud under given constraints. Especially, a cloud environment's internal structure is transparent to cloud users. As there exists a large body of work in this regard, we limit the description to selected approaches in the following three subareas.

Deployment Optimization in Non-Cloud Scenarios: Optimizing the deployment of software systems is a vital research area not just since the advent of the cloud computing paradigm. However, the corresponding objectives that should be optimized similarly comprise the number and types of hardware resources, QoS characteristics, and costs, for instance.

Non-Evolutionary Cloud Deployment Optimization: Considering the optimization of deployment and reconfiguration aspects of cloud-based applications, there exist several approaches that do not rely on evolutionary optimization methods. Instead, those approaches apply, for example, custom-made optimization algorithms or integer linear programming [Padberg 2010].

Evolutionary Cloud Deployment Optimization: Approaches of this subarea bear the most resemblance to CDOXplorer as they address the optimization of deployment and reconfiguration rules of cloud-based applications with the help of evolutionary optimization methods.

The optimization of deployment architectures is often accompanied by modifying a software system's actual component structure. Hence, considering the 4+1 view model of software architecture from Kruchten [1995], many of the approaches that are presented in this section address both the physical and logical view. A comprehensive overview regarding software architec-

ture optimization methods is contributed in a systematic literature review by Aleti et al. [2013].

The related work in all three subareas is examined according to the following five characteristics. The abbreviations in parentheses are used for referencing purposes.

Characteristics of Approaches for Deployment and Reconfiguration Optimization:

1. (Sim.) Incorporates simulation? Does the optimization procedure incorporate simulation, for example, for assessing the fitness of candidate solutions?

✓: yes; −: no

2. (Costs) Considers costs? Does the optimization procedure consider financial costs of a potential deployment architecture or of a complete CDO?

✓: yes, costs are optimized; (✓): yes, costs are considered in the optimization process, but not as an objective that is optimized; −: no

3. (RT) Considers response times? Does the optimization procedure consider response times of a potential deployment architecture or of a complete CDO?

✓: yes, response times are optimized; (✓): yes, response times are considered in the optimization process, but not as an objective that is optimized; −: no

4. (SLAs) Considers SLAs? Does the optimization procedure consider whether a potential deployment architecture or a complete CDO conforms with SLAs?

✓: yes, conformance regarding an SLA is optimized; (✓): yes, conformance regarding an SLA is considered in the optimization process, but not as an objective that is optimized; −: no

5. (Usg.) Considers an SUA's usage patterns? Does the optimization procedure take into account the specific usage patterns of a status quo deployment?

✓: yes, high level of detail; (✓): yes, low level of detail; −: no

13.3.1 Deployment Optimization in Non-Cloud Scenarios

Related work from approaches for deployment optimization in non-cloud scenarios is listed in Table 13.7. The related work is examined according to the five previously described characteristics.

Arshad et al. [2003] propose their tool **Planit** for optimizing the deployment and reconfiguration of software systems. Planit aims to find near-optimal plans for transitioning a software system (1) into an initial/normal deployment state and also (2) into states for recovering if specific events occur, for instance, when machines or components fail. A plan consists of a list of activities that have to be accomplished for transitioning between states. Plans are optimized regarding the following quality characteristics: plan execution time, induced costs, and allocated resources. For example, plans may last a different amount of time because of varying orders for starting components or machines.

Planit uses the temporal planner LPG [Gerevini and Serina 2002]. *Planners* are a common technique from the artificial intelligence area that explore plan spaces. *Temporal planners* are specific planners that, beyond the order of events, also consider the particular points in time the events occur [Ghallab et al. 2004]. Hence, in contrast to CDOXplorer, the optimization procedure from Arshad et al. [2003] addresses a precise timing of included deployment and reconfiguration steps. Furthermore, Planit does not consider prior system usage in its optimization process, whereas a software system's former usage patterns are essential in the case of CloudMIG for optimizing CDOs with the help of CDOXplorer and CDOSim.

A single-objective approach that optimizes the deployment of enterprise applications regarding costs in accordance with QoS constraints is proposed

Table 13.7. Related work from approaches for deployment optimization in non-cloud scenarios

Approach	Sim.[1]	Costs[2]	RT[3]	SLAs[4]	Usg.[5]	Comments
Arshad et al. [2003]	-	✓	-	-	-	Planit; Deployment and reconfiguration planning with the temporal planner LPG
Balogh et al. [2005]	-	✓	-	-	(✓)	Single-objective optimization (costs); Custom-made backtracking algorithm
Boone et al. [2008]	-	-	✓	-	(✓)	Deployment optimization with Mixed Integer Linear Programming (MILP)
Csorba et al. [2008]	-	-	-	(✓)	-	Initial service deployment optimization uses Cross Entropy Ant System (CEAS)
Jung et al. [2008]	-	-	(✓)	(✓)	-	Consolidation of multi-tier applications to virtual machines; Combination of bin packing and gradient search
Kichkaylo and Karamcheti [2004]	-	-	-	-	-	Extension of the AI planning approach Sekitei; Component placement problem (CPP)
Malek et al. [2012]	✓	-	✓	-	(✓)	Extensible framework and visual modeling and analysis environment; Deployment architecture improvement
Martens et al. [2010]	✓	✓	✓	-	(✓)	Software and deployment architecture optimization for component-based systems concerning performance, reliability, and costs
Zhang et al. [2007]	-	-	(✓)	(✓)	✓	Optimization of number of machines for service deployment in a SOA context

by Balogh et al. [2005]. The approach uses UML models to describe a software system. For annotating QoS properties to the UML model, Balogh et al. [2005] utilize OMG's UML Profile for Schedulability, Performance, and Time [Object Management Group 2005] and UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms [Object Management Group 2006]. For example, the workload and availability of components are specified through augmenting those components with corresponding `QOS_Workload` and `QOS_Availability` tagged values, respectively. Furthermore, components have a tagged value termed `TCO` that allows to define their total cost of ownership.

For specifying the computing capabilities of the available hardware, the TPC-W transactional web e-commerce benchmark from the Transaction Processing Performance Council[21] is proposed. The resulting requests per minute have to be annotated to each server model element through a tagged value called `performance`. In contrast, CDOXplorer uses the CDO simulation tool CDOSim that requires runs from the MIPIPS and weights benchmark (cf. Section 8.2). The optimization procedure from Balogh et al. [2005] employs a custom-made backtracking algorithm. For example, initially, the aggregate workload is determined for each service i according to Formula 13.3.1 (see [Balogh et al. 2005]).

$$Workload(i) = Capacity\_need(i) + \sum_{j \in depends(i)} Workload(j) \tag{13.3.1}$$

$Capacity\_need(i)$ determines the number of direct client requests to the service i. $depends(i)$ stands for the services that depend on service i. For example, considering a service k that depends on service i and suppose that clients call the service k directly x times, the simple model from Balogh et al. [2005] assumes that each call to service k causes a single subsequent call to service i. Hence, service i would also be called x-times from service k. Similarly to CDOXplorer's constraints for building feasible CDOs (cf. Section 8.3.4), the backtracking algorithm from Balogh et al. [2005] also has to consider such constraints for creating valid deployment archi-

[21] `http://www.tpc.org/`

tectures. For example, the following *workload constraint* in Formula 13.3.2 ensures that the capacity of each machine m of the available hardware HW (denoted $Capacity(m)$) is higher than the workload induced by all services deployed to m (denoted $deployed(m)$). SF refers to a saturation factor that enables to determine the maximum proportion of workload for the machines (see [Balogh et al. 2005]).

$$\forall m \in HW : Capacity(m) \cdot SF \geqslant \sum_{s \in deployed(m)} Workload(s) \tag{13.3.2}$$

Basically, the backtracking algorithm optimizes the overall total cost of ownership according to the objective function that is shown in Formula 13.3.3 (see [Balogh et al. 2005]).

$$TCO_{System} = \sum_{m \in HW} TCO(m) \cdot number\_used(m) \tag{13.3.3}$$

In the above Formula 13.3.3, $TCO(m)$ denotes the total cost of ownership for the machine m. $number\_used(m)$ stands for the number of machines of the type m that are actually used.

Comparing the approach from [Balogh et al. 2005] with CDOXplorer, both approaches address enterprise applications. However, CDOXplorer is a multi-objective optimization algorithm, whereas the approach from [Balogh et al. 2005] only optimizes a system's TCO. Furthermore, their approach assumes the generation of the target software system from the built UML model via model-driven technologies. In contrast, CloudMIG addresses the migration of software systems to the cloud. Moreover, CloudMIG incorporates actual monitored (varying) usage patterns of an SUA's status quo deployment in the deployment optimization process. As opposed to this, their approach requires a user to specify a fixed workload factor for each service.

A further approach that also starts with UML models for optimizing the deployment of components for distributed systems is presented by Boone et

Figure 13.16. Deployment optimization using UML models and MILP (from [Boone et al. 2008])

al. [2008]. An overview of this approach is shown in Figure 13.16. The given hardware resources and software components have to be modeled with a UML deployment diagram. The software components are also included in a UML activity diagram for modeling the messages that are sent between components. Hardware properties, such as a server's computing capacity or network bandwidth, are annotated to UML model elements with tagged values.

The approach from Boone et al. [2008] uses the optimization technique Mixed Integer Linear Programming (MILP) for optimizing the deployment of components to the given hardware resources while taking into account replication and also QoS aspects. MILP is a special case of linear programming (cf. Section 4.1.1), where optimization problems are formulated with the help of linear relationships and additionally some or all included variables are integers [Padberg 2010].

After describing a software system with the mentioned UML models, they are transformed to a MILP model. Similarly to CDOXplorer that restricts the automatic creation of CDOs to feasible candidates with the help of certain constraints (cf. Section 8.3.4), the approach from Boone et al. [2008] also uses specific constraints to limit the search space for the MILP solver. For example, the constraint shown in Formula 13.3.4 states that each software component i is either replicated or not and that a replicated component is deployed to exactly one resource (see [Boone et al. 2008]).

$$\sum_{r \in R} S_{ir0} + S_{ir1} = 1, \forall i \in I \tag{13.3.4}$$

In the above Formula 13.3.4 I denotes the set of all available software components. These components can be deployed to the available hardware

resources R. Furthermore, if S_{ir0} equals 0, the software component i is not deployed to the hardware resource r. Otherwise if S_{ir0} equals 1, it is deployed to the hardware resource r without replication. Likewise, if S_{ir1} equals 0, the software component i is not deployed to the hardware resource r. Otherwise if S_{ir1} equals 1, it is deployed to the hardware resource r and a replication of i exists that is also deployed to r. After performing the optimization procedure, the UML models can be updated with the optimal solution that is found by the MILP solver (cf. Figure 13.16). In addition to it, a deployment tool may be used for automatically accomplishing the found, optimal deployment.

In contrast to the approach from Boone et al. [2008], CDOXplorer takes into account an SUA's actual dynamic usage for finding optimized CDOs, as monitored log data can be used instead of fixed arrival rates. Furthermore, the system model from Boone et al. [2008] employs a coarse-grained logical view as the main building blocks are components and servers. Instead, CDOXplorer uses simulation results from CDOSim that are produced from fine-grained KDM models. Moreover, runtime reconfiguration is not considered by Boone et al. [2008].

UML models are also used for modeling the system components in the context of the approach from Csorba et al. [2008]. UML collaboration diagrams are utilized to specify components and network links. Furthermore, the corresponding diagrams are augmented with QoS properties. The approach from Csorba et al. [2008] aims to find the best-suited deployment of components to a set of nodes while satisfying given QoS requirements. The approach is not restricted to specific objectives. Any QoS requirement may be used as long as a corresponding cost function can be formulated. The optimization procedure uses diverse node types. For example, a *client node* represents the only node type which can handle client requests, whereas a *server node* can host service components. In contrast, CDOXplorer can differentiate the node resources through assigning different *Initial Start Config* elements to a *Node Configuration* (cf. Section 8.3).

Similarly to CDOXplorer, the approach also uses an evolutionary optimization algorithm. The Cross Entropy Ant System (CEAS) [Helvik and Wittner

2001] is utilized. It constitutes an ant colony optimization system and can be executed in a distributed fashion. The heuristic algorithm searches for an optimal initial deployment of the components. The authors also mention that the optimization of component redeployment at runtime constitutes future work, but this aspect does not seem to be covered yet. In contrast, CDOXplorer also optimizes runtime reconfiguration rules.

A hybrid approach that considers the consolidation of multi-tier applications to virtual machines is presented by Jung et al. [2008]. Applications are modeled and optimized offline to generate well-suited system configurations. These are transformed to adaptation policies that can be consumed online by rule engines. In this context, node replications, assignment of replicas to resources, and the maximum ratio of the resources that are allowed to be used can dynamically be modified. The goal is to maximize resource utilization while satisfying response time thresholds that are defined in SLAs.

Compared with CDOXplorer, the approach by Jung et al. [2008] only allows to define a single resource type and homogeneous resource instances instead of considering distinct VM instance types that can also be used in parallel. An optimization technique that uses a combination of bin packing and gradient search (cf. Section 4.1.1) is used in their approach. Jung et al. [2008] also generate different workloads that are used in the optimization procedure. In contrast, CDOXplorer constitutes a simulation-based genetic algorithm and employs a system's actually monitored usage patterns for optimizing CDOs. However, both approaches consider the offline optimization of reconfiguration rules.

To limit search space size, CDOXplorer uses discrete values and tight boundaries for the design of its genes (cf. Section 8.3.3). Similarly, the approach from Kichkaylo and Karamcheti [2004] introduces the notion of *resource levels* to reduce the number of potential deployment plans. Those resource levels specify resource properties in terms of intervals instead of broad ranges of values. For example, the network bandwidth could by defined in resource levels of $[0, 20)$, $[20, 40)$, and $[40, \infty)$ GB/s. With the help of the resource levels, Kichkaylo and Karamcheti [2004] address the

component placement problem (CPP), i.e., finding best-suited placements of components regarding optimal resource consumption.

In contrast to CDOXplorer, Kichkaylo and Karamcheti [2004] do not employ a genetic algorithm, but build on the authors' previous work and use and extend the AI planning approach Sekitei [Kichkaylo et al. 2003]. The Sekitei AI planning algorithm considers two types of actions: placement of components on nodes and crossing of data streams over network links. Compared with our approach, CDOXplorer's deployment model covers more details. For example, the CDO model addresses different cloud environments, VM instance types, and its reconfiguration rules include diverse scaling types and scaling conditions.

For improving the deployment architecture of distributed systems regarding arbitrary QoS properties, Malek et al. [2012] propose an extensible framework and visual modeling and analysis environment. The framework incorporates continuous system monitoring and changing the deployment architecture at runtime by actually executing redeployment operations. A further mode allows for offline simulation. Arbitrary deployment constraints and QoS properties can be formally defined by manually specifying utility functions. The approach provides four predefined deployment improvement algorithms, among those is a genetic algorithm. The utility functions are used as fitness functions, whereas CDOXplorer uses simulation runs to obtain fitness values. Furthermore, Malek et al. [2012] do not consider the integration of reverse-engineered code models, as is supported by our approach.

Martens et al. [2010] focus on component-based systems and at finding optimized software and deployment architectures concerning performance, reliability, and costs. Similar to our approach, a genetic algorithm is used and simulations are, partially, employed for assessing solutions. Their software PerOpteryx provides tool support and explores four degrees of freedom, e.g., processor speeds. No dynamic resource scaling is supported, but further degrees of freedom can be added. In contrast, CDOXplorer currently explores 17 fixed degrees of freedom (the number of node configurations and 16 genes) and optimizes runtime reconfiguration rules.

Zhang et al. [2007] propose a QoS-aware approach for finding the optimal number of machines for deploying services in the context of service oriented architectures (SOAs). A greedy algorithm is used that maximizes the throughput. Unlike in our approach, dynamic resource scaling is not supported. However, Zhang et al. [2007] consider inferring arrival rates from actual monitoring log data. CDOXplorer uses the simulation tool CDOSim that relies on workload profiles, which can be automatically created from historical monitoring log data as well.

13.3.2 Non-Evolutionary Cloud Deployment Optimization

Related work from approaches for non-evolutionary deployment optimization in cloud scenarios is listed in Table 13.8. The related work is examined according to the five characteristics that were also used in the previous Section 13.3.1 to describe the related work in the context of deployment optimization in non-cloud scenarios.

CDOXplorer is a multi-objective optimization algorithm that targets enterprise applications and aims to optimize CDOs regarding costs, response times, and SLA conformance. In contrast, the approach from Bittencourt and Madeira [2011] addresses a single-objective optimization problem that merely covers the minimization of financial costs for deploying workflow-based systems. The authors address the deployment to hybrid cloud environments, i.e., selecting the best-suited resources from public and private clouds that enable adherence with execution time constraints at minimal costs. For tackling this optimization problem, Bittencourt and Madeira [2011] propose the **Hybrid Cloud Optimized Cost (HCOC)** scheduling algorithm. In general, HCOC works as follows (cf. [Bittencourt and Madeira 2011]).

In a first initial step, HCOC schedules the workflow solely in the private cloud. If an execution time constraint is violated—i.e., if the complete workflow lasts longer than permitted—workflow tasks have to be rescheduled. This task rescheduling (1) selects tasks that should be rescheduled, (2) selects resources from the public cloud, and (3) reschedules the selected tasks

Table 13.8. Related work from approaches for non-evolutionary deployment optimization in cloud scenarios

Approach	Sim.[1]	Costs[2]	RT[3]	SLAs[4]	Usg.[5]	Comments
Bittencourt and Madeira [2011]	-	✓	-	-	-	HCOC: Hybrid Cloud Optimized Cost scheduling algorithm; Workflow-based systems
Liew and Su [2012]	-	✓	✓	(✓)	✓	CloudGuide; Integer linear programming; Estimation of costs and performance of cloud deployments
Lucas Simarro et al. [2011]	-	✓	-	-	-	Optimization of VM deployment to multiple cloud environments via cloud brokers; Integer programming problem
Mao et al. [2010]	-	✓	✓	-	✓	Scientific computing jobs; Integer programming problem; Dynamic resource scaling
San Aniceto et al. [2011]	-	✓	-	-	✓	Single-objective cost optimization; Balancing usage of reserved and on-demand VM instances
Smit and Stroulia [2011]	✓	✓	✓	(✓)	-	Simulation-generated state-transition models; These models are used for runtime reconfiguration
Trummer et al. [2010]	-	✓	-	-	-	Single-objective deployment optimization (costs); Constraint optimization problem
Villegas et al. [2012]	-	✓	-	-	-	Optimizing the deployment of computational jobs to IaaS-based cloud resources; Provisioning and allocation policies
Wu et al. [2011]	-	✓	(✓)	✓	-	Resource usage optimization for SaaS providers conc. costs and SLA violations

to the available resources (the hybrid cloud that consists of the private cloud and the resources from the public cloud that were selected in (2)). Regarding (1), Bittencourt and Madeira [2011] propose different policies for selecting the workload tasks that have to be rescheduled, for example, according to task priorities. In comparison with HCOC, CloudMIG does not cover hybrid clouds. However, our approach optimizes runtime reconfiguration rules and also enables to compare different service compositions.

An approach termed **CloudGuide** that supports users to compare different CDOs regarding web-based systems is proposed by Liew and Su [2012]. CDOs are termed *cloud configurations* in the context of CloudGuide. The approach exhibits several similarities with CloudMIG. It also enables comparing various combinations of IaaS-based cloud environments, VM instance types, and numbers of VMs regarding resulting costs. An overview of CloudGuide is shown in Figure 13.17. In its first step, CloudGuide surveys potential IaaS-based cloud environments and benchmarks their VM instance types. The second step also benchmarks the local deployment of the application that should be migrated to the cloud.

Furthermore, similarly to CloudMIG, application behavior profiles are created, for example, from monitoring log files. The output from the first and second step are used as input in the third step. Here, the capacities of the VM instance types are modeled with the help of the so-called *CloudGuide Suggestion Engine*. Based on each pair of measurements regarding a local deployment and VM instance type, Liew and Su [2012] propose to calculate speedup factors. These speedup factors enable estimating the performance if the respective software system was deployed on a VM of the corresponding VM instance type. CloudMIG also uses measurements of the status quo deployment and VM instance types from the MIPIPS and weights benchmark (cf. Section 8.2). However, the measurement results are then used by CDOSim to simulate potential CDOs.

Instead of simulating CDOs, CloudGuide uses queuing models. The inter-arrival times are extracted from application logs, for instance. This is also similar to CloudMIG that builds workload profiles from monitoring log data. The fourth step estimates the costs of potential cloud deployments,

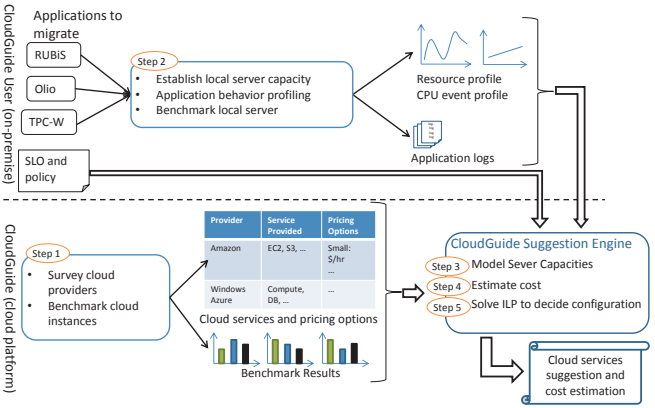


Figure 13.17. CloudGuide overview (from [Liew and Su 2012])

the fifth step suggests well-suited cloud deployments. For this purpose, users of CloudGuide can select one out of three policies that are used by the *CloudGuide Suggestion Engine* to optimize the cloud deployments. CloudGuide includes the following three policies (cf. [Liew and Su 2012]):

1. Minimize deployment cost while satisfying the peak incoming request rate
2. Maximize throughput given budget M
3. Improve peak throughput by 50% and minimize cost

The optimization problem is formulated as an integer linear program. For example, considering a common three-tier application with tiers $T_1 - T_3$ and overall k VM instance types from all of the surveyed cloud providers (step 1 in Figure 13.17), a potential *cloud configuration* C is denoted as shown in Formula 13.3.5 (cf. [Liew and Su 2012]).

$$C = \{ \sum_{j=1}^{k} n_{1j}, \sum_{j=1}^{k} n_{2j}, \sum_{j=1}^{k} n_{3j} \} \tag{13.3.5}$$

In Formula 13.3.5, n_{ij} denotes the number of VMs of VM instance type $j, 1 \leqslant j \leqslant k$ that host tier $i, 1 \leqslant i \leqslant 3$.

Besides the similarities between CloudGuide and CloudMIG, the approaches differ in several aspects. For example, as can be seen in Formula 13.3.5, the application model is coarse-grained in the case of CloudGuide. In the above-mentioned example, the application merely consists of three abstract components (T_1-T_3). In contrast, CloudMIG employs fine-grained KDM code models. Furthermore, for the optimization of cloud configurations, CloudGuide solely considers the three policies that were also mentioned above.

In comparison with our approach, CDOXplorer constitutes a multi-objective genetic algorithm that aims to find near-optimal solutions regarding the objectives costs, response times, and conformance to SLAs. Moreover, CDOXplorer also can optimize runtime reconfiguration rules, whereas CloudGuide merely provides support for manually comparing different reconfiguration strategies. However, both approaches consider scaling out/in as well as scaling up/down strategies. Moreover, there does not seem to be tool support available for CloudGuide, whereas our tool *CloudMIG Xpress* is publicly available as open source software.

Lucas Simarro et al. [2011] address the optimization of financial costs for deploying VMs to federated cloud environments. The authors consider an architecture that benefits cloud users as those can partition their virtual clusters and utilize best-suited cloud resources from multiple cloud environments such that the overall costs decrease. The architecture includes a cloud broker role that not only provides a uniform access to the distinct cloud environments, but also cost-efficiently schedules VMs across cloud boundaries in a way that is transparent to the cloud users. The cloud broker not only takes into account fixed prices, but also prices that vary according to the actual demand or according to the bids of cloud users (or cloud brokers). Those highly dynamic pricing models are currently not addressed by CDOXplorer.

The cloud broker role proposed by Lucas Simarro et al. [2011] also covers a price prediction algorithm for coping with those highly dynamic prices.

The price prediction aims to approximate the so-called *Oracle* prices, i.e., the prices that are actually charged by cloud environments. Lucas Simarro et al. [2011] consider a *scheduling period* of one hour. That means, the VMs may be relocated every hour to a different cloud environment. Furthermore, each hour the quality of the prediction for the scheduling period price can be assessed. However, in contrast to CloudMIG, their approach only enables to use a single VM instance type per virtual cluster. Lucas Simarro et al. [2011] formulate the optimization problem as an integer program. For example, $X_{i,k}(t)$ denotes whether a VM instance v_i is deployed to cloud c_k during scheduling period t. If this is the case, $X_{i,k}(t) = 1$, otherwise, $X_{i,k}(t) = 0$. The real price of a VM during period t in a cloud c_k is denoted as $P_k(t)$. Hence, the price prediction aims to approximate the Oracle price function $TIC_{Oracle}(t)$ that is shown in Formula 13.3.6 below.

$$TIC_{Oracle}(t) = \sum_{i}^{n} \sum_{k}^{m} X_{i,k}(t) \cdot P_k(t) \tag{13.3.6}$$

In Formula 13.3.6, *TIC* stands for *Total Infrastructure Cost*. The authors formulate the full optimization problem with the AMPL language [Fourer et al. 2002] and use the MINOS solver [Neumaier et al. 2005] to perform the optimization. In contrast to the approach from Lucas Simarro et al. [2011], CDOXplorer's basic building blocks are not restricted to VMs, but also consider service assignment to VMs, different VM instance types, and varying numbers of used VMs, for instance. However, CloudMIG does not address deployment architectures that incorporate multiple cloud environments. Though, their approach does not cover dynamic resource scaling, whereas CDOXplorer enables the optimization of reconfiguration rules.

Dynamic scaling is, in contrast, also supported by the approach of Mao et al. [2010]. Similarly to Lucas Simarro et al. [2011], the authors also address integer programming problems. However, Mao et al. [2010] aim to reduce costs and maximize the performance of scientific computing jobs. Scaling policies are derived at runtime by learning from previous job executions. The approach is validated with an implementation that uses Microsoft Windows Azure. We also use this cloud environment for our evaluation of CloudMIG (cf. Chapters 11 and 12). Compared with the

approach from Mao et al. [2010], CloudMIG targets enterprise software and uses simulations for assessing different CDO candidates before actually deploying an application to a specific cloud environment.

San Aniceto et al. [2011] use a custom-made single-objective optimization algorithm for reducing the costs of leased VMs. Besides on-demand instances—that can be started and stopped at any time—it also considers reserved instances. Those are frequently offered for a single payment per time period. In turn, a discount is given in the hour rates. The algorithm aims at finding the best suited combination of on-demand and reserved instances based on historical workload data. However, it does not support dynamic reconfiguration.

An approach that also incorporates simulations in the deployment and reconfiguration optimization process is described by Smit and Stroulia [2011]. The authors build on a simulator from their previous work [Smit et al. 2008] and consider a two-dimensional optimization problem where they strive to minimize costs and response times. Their approach addresses SOA-based systems and is structured into the following three steps.

1. Various simulations are performed that assess different combinations of workload, SLA constraints, and service configurations. The latter refers to different numbers of service instances or servers, for instance.

2. The data gathered from the first step is used to build a state-transition model of the SOA-based system. The model describes the expected system's behavior if it were deployed to a cloud environment. The states result from *snapshots* taken during simulation. If similar conditions are detected, these are clustered to a state. Regarding a given SLA constraint, a state may be *satisfactory* if it complies to the constraint, otherwise it is supposed to be *unsatisfactory*. A state can also be classified as *boundary* if it meets an SLA constraint, but has transitions to one or more unsatisfactory states. Each state is augmented with a cost value. This cost value comes from the minimal value from all of the snapshots that are represented by this state. Transitions also result from the simulations in the case a snapshot that leads to a source state is followed by a snapshot that

leads to a destination state. Transitions may represent varying conditions, such as an increasing workload.

3. When the SOA-based system is actually deployed to a cloud environment, an autonomic manager monitors the system. According to this data obtained during runtime and comparison with the state-transition model from the second step, the autonomic manager can infer the current state of the system. Hence, according to the state-transition model, it can also infer beneficial actions due to the current state and changing conditions. However, Smit and Stroulia [2011] solely report on adding or removing VM instances of the same VM instance type, whereas CloudMIG also supports vertical scaling (cf. Section 8.3.1).

Though also utilizing simulations in the course of the deployment and reconfiguration optimization procedure, the approach from Smit and Stroulia [2011] and CloudMIG differ substantially. Their approach performs several simulation runs to establish a repository of rated system configurations and to infer a state-transition model. This model is used during the operation phase to maintain cost-efficiency and responsiveness through performing adequate reconfiguration actions. In contrast, CloudMIG uses CDOSim runs as a fitness function and to guide the way through the search space. CDOXplorer also concentrates on workload profiles that originate from actual monitoring data, whereas the approach from Smit and Stroulia [2011] simulates arbitrary usage patterns.

Trummer et al. [2010] contribute an algorithm for computing an application's cost-optimal deployment architecture for IaaS-based clouds. This single-objective optimization is described as a constraint optimization problem and is tackled with an existing constraint solver. As opposed to this, we use multi-objective optimization and support dynamic resource scaling. Nevertheless, Trummer et al. [2010] also consider constraints that are similar to those employed by CDOXplorer for constructing feasible CDOs (cf. Section 8.3.4). Furthermore, they use application templates that serve, from a high-level perspective, the same purpose as our status quo deployment models. However, the application templates model software systems at a coarse-grained level. In contrast, our status quo deployment models and

the resulting CDOs that are used for simulation purposes include mapped, fine-grained KDM application code models.

Optimizing the deployment of computational jobs to IaaS-based cloud resources is investigated by Villegas et al. [2012]. The authors examine eight provisioning and four allocation policies. Their goal is to optimize costs and performance. In the context of deploying computational jobs to IaaS-based clouds, *provisioning* refers to the activity of starting and stopping VM instances. The simplest provisioning policies are called *startup* and *on-demand, single VM*. The policy *startup* starts as many VM instances as possible during the system startup. The policy *on-demand, single VM* starts a new VM for every job that cannot be assigned.

The VM instances are shutdown according to a parameter that specifies a timeframe, i.e., if a VM instance is idle for at least the timeframe specified by this parameter, then the VM is shutdown. In contrast to provisioning, *allocation* means the assignment of computational jobs to specific VM instances. A simple exemplary allocation policy is *first-come, first-served*, where each job is assigned to a single VM instance according to the order in which the jobs were created.

Compared with the work from Villegas et al. [2012], CloudMIG addresses enterprise software systems. Furthermore, CDOXplorer also searches for, among others, well-suited service compositions, adequate cloud environments, and appropriate cloud resources (VM instance types). Those aspects are not considered by Villegas et al. [2012]. However, their provisioning policies resemble our reconfiguration rules, as they control leasing and releasing of cloud resources according to defined conditions. Nevertheless, in contrast, CDOXplorer regards the parameters of its reconfiguration rules as parts of the search space. For example, it searches a well-suited CPU utilization threshold for starting new VM instances with regard to a specific usage pattern.

As with our approach CloudMIG, Wu et al. [2011] consider resource usage optimization for SaaS providers that build upon leased VMs. Wu et al. [2011] assume that a maximum service utilization is defined per customer in SLA agreements. SLA violations and infrastructure costs are minimized by two

custom-made algorithms that are evaluated with the tool CloudSim. Our simulator CDOSim also builds on CloudSim, but it is used by CDOXplorer as a part of the optimization procedure itself. In contrast to CDOXplorer, Wu et al. do not consider distributed applications, runtime reconfiguration, and arbitrary workload.

13.3.3 Evolutionary Cloud Deployment Optimization

Related work from approaches for evolutionary deployment optimization in cloud scenarios is listed in Table 13.9. The related work is examined according to the five characteristics that were also used in the Section 13.3.1 to describe the related work in the context of deployment optimization in non-cloud scenarios.

Related work that employs evolutionary optimization techniques in cloud deployment scenarios almost exclusively takes a cloud provider perspective or requires corresponding knowledge regarding the internal structure of a cloud environment (e.g., Nakada et al. 2009; Csorba et al. 2010; Lee et al. 2010; Zheng et al. 2011; Yusoh and Tang 2012). In contrast, Jiang et al. [2011] address cloud users and enable the optimization of CDOs with the help of a genetic algorithm. Jiang et al. [2011] name a CDO a *deployment configuration*. A deployment configuration consists of several *deployment plans*, whereas a deployment plan represents a VM instance of a specific VM instance type. A deployment plan also contains one ore more services of an application that are deployed to this VM instance.

In contrast to CloudMIG, the approach from Jiang et al. [2011] focuses on online optimization of deployment configurations. That means, the genetic algorithm is executed at runtime after so-called *reconfiguration intervals* to obtain well-suited deployment configurations that adapt to changed environment conditions, such as increased workload. Hence, their approach does not require optimizing reconfiguration rules. Our optimization procedure is typically performed during the planning phase of a cloud migration (cf. Section 6.1). The system from Jiang et al. [2011] monitors the deployed system and employs queuing theory to estimate future throughput, latency,

Table 13.9. Related work from approaches for evolutionary deployment optimization in cloud scenarios

Approach	Sim.[1]	Costs[2]	RT[3]	SLAs[4]	Usg.[5]	Comments
Jiang et al. [2011]	-	✓	✓	✓	✓	Deployment configuration optimization; Genetic algorithm; Online optimization
Pandey et al. [2010]	-	✓	-	-	-	Optimizing mapping of scientific tasks to cloud resources; Particle swarm optimization-based heuristic
Szabo and Kroeger [2012]	-	✓	-	-	-	Scientific workflow tasks; Optimizing mapping to cloud resources and task execution order; Genetic algorithm
Wada et al. [2011]	-	✓	✓	(✓)	✓	Optimizing deployment configurations conc. costs and services' QoS attributes; Genetic algorithm E^3-R

CPU usage, and costs. Those properties are regarded as SLA metrics that are used as a combined fitness function for the genetic algorithm. As opposed to this, CDOXplorer uses CDOSim runs to assess the fitness of CDOs.

The genetic algorithm from Jiang et al. [2011] selects parent individuals according to a binary tournament. The mutation operator increases or decreases the amount of used resources. However, the selection of specific cloud environments is not optimized as their approach performs the optimization online. In contrast, CDOXplorer employs two tournament rounds for the selection of parent individuals (cf. Section 8.1.1). Furthermore, CDOXplorer's mutation operator can modify the whole range of defined genes (cf. Section 8.4.2), among others, a disruptive change may switch the used cloud environment, for instance.

A particle swarm optimization-based heuristic is introduced by Pandey et al. [2010]. Those heuristics are closely related to evolutionary computation and

rely on the social behavior of particles, e.g., fishes in a swarm [Kennedy and Eberhart 1995]. The single-objective optimization algorithm maps scientific tasks to cloud resources for minimizing costs. In comparison, we target optimal CDOs for enterprise software, formulate the search as a multi-objective optimization problem, and facilitate dynamic resource scaling. Furthermore, Pandey et al. assume that the task execution times are known for all available cloud resources. In contrast, CDOXplorer uses the tool CDOSim for simulating dynamic properties of potential cloud deployments, such as response times of client requests.

The deployment optimization of scientific workflow tasks is also addressed by Szabo and Kroeger [2012]. More specifically, the authors aim at finding well-suited allocations of tasks to VM instances and also well-suited task execution orders. They consider communication overhead as an important factor, as the scientific tasks often produce large files that then have to be transmitted to other tasks for further processing. Szabo and Kroeger investigate two single-objective problem variants that optimize either costs or total runtime, and also a multi-objective variant that optimizes both. The standard genetic algorithm NSGA-II is used. Our CDOXplorer algorithm also uses NSGA-II for its selection operation (cf. Section 8.1.1).

Candidate solutions are encoded with two different types of chromosomes, whereas CDOXplorer uses a single chromosome type (cf. Section 8.3.3). Their first chromosome models the task allocations to VM instances and the second chromosome represents the order in which the tasks are executed. Due to the construction of the chromosomes, the first chromosome can be processed with a standard single-point crossover operator. However, the second chromosome needs a domain-specific chromosome operator that retains the defined chromosome structure (task order) and prevents infeasible candidates. Furthermore, also two different mutation operators are required and also the mutation operator for the second chromosome type has to retain task order that is valid for a given workflow definition.

In comparison with our approach, CloudMIG addresses enterprise applications. CDOXplorer also uses crossover and mutation operators that prevent infeasible candidate solutions (cf. Section 8.4). Moreover, CDOXplorer uses

CDOSim runs to assess CDOs. In contrast, the approach from Szabo and Kroeger [2012] uses a fitness function that can be calculated and that comprises communication overhead, runtime, and costs.

Wada et al. [2011] follow, in common with CloudMIG, a strict cloud user perspective and contribute the genetic algorithm **E^3-R** that explores deployment configurations for optimizing costs and services' QoS attributes. Similarly to Jiang et al. [2011], *deployment configurations* name CDOs and contain *deployment plans*. The latter describe VM instances of a specific VM instance type in combination with mapped services. E^3-R can reduce redundant QoS objectives and estimate the performance of deployment configurations using queueing theory and historic mean arrival rates. However, E^3-R does not support varying workload and dynamic resource scaling, and it regards services as black boxes, whereas our approach simulates reverse-engineered and transformed architectural models.

13.4 Summary

This chapter describes the related work that comes from the three areas of *cloud migration*, *conformance checking*, and *deployment and reconfiguration optimization*. Each area is structured in two or more subareas. To provide a concise overview, the description of each subarea includes a table with the corresponding related work and central characteristics. The related work from the three areas is summarized below.

Cloud Migration: The approach CloudMIG aims to support future SaaS providers to plan the migration of enterprise applications to the cloud (cf. Chapter 6). Hence, related work comes from the area of cloud migration. There exist various *cloud migration approaches* that also propose a process for migrating applications to a cloud environment. However, most approaches focus on specific activities that support the migration to cloud technologies and various approaches are also limited to specific cloud environments. Furthermore, most of the examined cloud migration approaches do not take the specific usage patterns of SUAs into account. In contrast, CloudMIG

considers the construction of workload profiles, for example, from actual monitoring log data, for aligning and specifically optimizing the target architecture and reconfiguration rules.

CloudMIG also proposes the Cloud Suitability and Alignment (CSA) hierarchy (cf. Section 7.4.1) for analyzing an application's suitability regarding a migration to the cloud. Hence, the subarea *cloud suitability analysis* is a further source for additional related work. However, most related work in this subarea examines the corresponding suitability regarding non-technical constraints or regulatory conformance, for instance. Considering CloudMIG's CEM, our approach also provides means for modeling cloud environments and cloud-based applications as a foundation for conformance checking and the optimization of CDOs. Thus, related work also comes from the subareas *cloud environment modeling* and *cloud application modeling*. However, there exist few approaches that provide detailed meta-models that actually enable the modeling of cloud environments or cloud-based applications. Most approaches rather describe abstract architectures from a high-level point of view.

Conformance Checking: Almost all approaches that check the conformance of software systems in the context of cloud computing consider checking the conformance regarding organizational, regulatory, or governance issues. There exist few approaches that address the technical analysis of restrictions that are posed by cloud environments. Moreover, these approaches rather cover conformance checking from a high-level point of view as a black-box-type step of a cloud migration process. Hence, this area expands the search space and describes related work that checks the conformance regarding specific technical issues in the following subareas: *Conformance checking in the context of software evolution* and *conformance checking in the context of cloud computing*. Thus, these subareas compare specific detection mechanisms—for example, in the context of cloud API validation or cloud interoperability testing—with those proposed by CloudMIG.

Deployment and Reconfiguration Optimization: Related work regarding deployment and reconfiguration optimization comes from three subareas. The optimization of application deployment, resource provisioning, and

component mapping is relevant in various research fields apart from the cloud computing context. Hence, optimizing the deployment of, for example, classic multi-tier and service-oriented systems is addressed in the *deployment optimization in non-cloud scenarios* subarea. When turning to related work in the context of cloud computing, this area investigates the subareas *non-evolutionary cloud deployment optimization* and *evolutionary cloud deployment optimization*. The latter subarea examines related work that incorporates evolutionary—and also related—optimization techniques, such as genetic algorithms and particle swarm optimization. In contrast to CDOXplorer, many approaches that cover cloud-based systems do not take into account the optimization of runtime reconfiguration rules.

Part IV

Conclusions and Future Work

Chapter 14

Conclusions and Future Work

This chapter concludes the thesis and summarizes core aspects of the CloudMIG approach and of its experimental evaluation. It also reports on open issues that are left for future work. The chapter is structured as follows. Section 14.1 draws the conclusions and Section 14.2 describes future work.

14.1 Conclusions

This thesis describes the CloudMIG approach that aims to support SaaS providers to leverage cloud computing technologies also for existing software systems and to migrate those systems to the cloud. This section summarizes the key concepts of the CloudMIG approach and overviews the conducted experiments that demonstrate CloudMIG's feasibility and practicality. Then, the section reports on the lessons that were learned during the conception, development, and assessment of the approach.

Summary

Cloud computing can be seen as the realization of the long-desired utility computing paradigm that was already envisioned decades ago and that introduced the notion of computing resources that are available in the required amount anywhere and anytime (cf. Section 2.1.1). Its means for dynamically provisioning resources according to a varying user demand

are compelling also for software service providers who want to exploit the cloud's capabilities for existing applications. However, migrating an existing (on premise) software system to the cloud and also aligning the system with provided infrastructure and platform cloud services to optimally benefit from, for example, dynamic resource scaling and the frequently used pay-per-use pricing model, poses considerable challenges to (future) SaaS providers.

To cope with these challenges, this thesis proposes the CloudMIG approach that supports SaaS providers in planning the migration and that relieves SaaS providers from several time-consuming and costly manual tasks (cf. Chapter 6). Figure 14.1 gives an overview on important aspects of CloudMIG that are investigated in detail in the previous chapters. CloudMIG builds on the cloud concepts as described by the National Institute of Standards and Technology (NIST, cf. Sections 2.2 and 2.3) and addresses a migration of software systems to the corresponding Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) cloud service models. CloudMIG also draws on methods and techniques from the software modernization (cf. Chapter 3) and Search-Based Software Engineering (SBSE, cf. Chapter 4) fields and focuses on supporting the planning phase of a migration to the cloud (cf. Section 6.3).

As the usage patterns of enterprise software systems often vary significantly, those systems can benefit to a great extent from the cloud's enabled elasticity. Hence, CloudMIG focuses on enterprise software systems. Moreover, it aims to align existing enterprise software systems to a cloud environment based on its specific, historic usage patterns. CloudMIG follows a model-based approach and builds on application models that get reverse-engineered from an existing system. For this purpose, CloudMIG utilizes models from OMG's Architecture-Driven Modernization (ADM) initiative (cf. Section 3.3), i.e., the Knowledge Discovery Meta-Model (KDM) and Structured Metrics Metamodel (SMM).

Cloud environments are modeled with the help of cloud profiles. Cloud profiles are instances of the so-called Cloud Environment Model (CEM, cf. Section 6.3.4) and include specifications regarding the resources and ser-

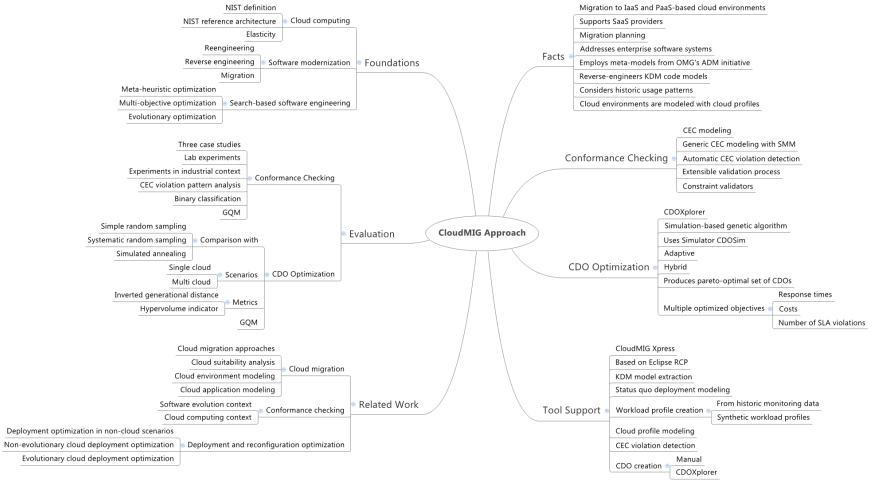


Figure 14.1. Core aspects of the CloudMIG approach investigated in this thesis

vices that are offered by a cloud environment, its availability in geographical regions, and a used pricing model, for instance. CloudMIG actually describes a number of specific activities along with corresponding models for planning a migration to the cloud (cf. Chapter 6). Besides this overall approach, this thesis focuses on the following two core components of CloudMIG.

The included automatic *conformance checking* approach (cf. Chapter 7) allows to detect violations regarding restrictions that are imposed by cloud environments. Those restrictions may prohibit writing data to the filesystem, for instance. We call those restrictions Cloud Environment Constraints (CECs). They can be modeled as part of a cloud profile. There exist several pre-built CEC types. For modeling arbitrary CECs, it is possible to use a generic CEC type that builds on SMM-based measures. Instead of manually reviewing the source code of an existing application, a SaaS provider may use CloudMIG's conformance checking process to automatically detect violations of CECs that are called CEC violations. Each CEC type can be detected with a

specific *constraint validator* component. Additional constraint validators can be plugged into the validation process as needed.

The second core component of CloudMIG besides its conformance checking approach is the *creation and optimization of Cloud Deployment Options* (CDOs, cf. Chapter 8). A CDO describes a set of decisions a SaaS provider has to make for deploying and operating a system in the cloud. For example, a CDO describes which cloud environment, cloud resources, and mapping of system components to cloud resources should be used. Furthermore, a CDO defines a set of so-called runtime reconfiguration rules that determine strategies for leveraging the cloud's dynamic resource scaling. There exists a vast amount of potential CDOs, but it is not viable for SaaS providers to implement and compare them all to select the best candidate.

Hence, CloudMIG enables to automatically create a set of pareto-optimal CDOs from which a SaaS provider can select the CDO that best suits its specific needs. This approach uses a simulation-based genetic algorithm for exploring the search space of all possible CDOs. The genetic algorithm is called *CDOXplorer* and uses our simulation tool CDOSim (cf. Section 8.2) for simulating response times, costs, and the number of SLA violations of a CDO. CDOXplorer uses CDOSim as a fitness function. Furthermore, CDOXplorer also exhibits adaptivity and hybridity characteristics (cf. Section 8.5) that actually make CDOXplorer a simulation-based, adaptive, and hybrid genetic algorithm.

A proof of concept implementation of CloudMIG is provided in the form of the tool *CloudMIG Xpress* (cf. Chapter 10). It is an Eclipse RCP-based application that provides support for creating the models that are used by CloudMIG, such as KDM-based code models or status quo deployment models that describe the current on premise deployment of an application. It also allows to create synthetic workload profiles or extract workload profiles from actual monitoring log data. A cloud profile editor can be used to build cloud profiles. *CloudMIG Xpress* utilizes these cloud profiles as a basis for performing CloudMIG's conformance checking process and for applying the genetic algorithm CDOXplorer.

An extensive experimental evaluation demonstrates CloudMIG's feasibility and practicality. The evaluation is planned, performed, and analyzed following the Goal Question Metric (GQM) method, whereas a focus is on CloudMIG's conformance checking and CDO optimization approaches. For the former, two case studies that comprise lab experiments and one case study in an industrial context are performed to investigate CEC violation patterns in existing applications (cf. Chapter 11). Binary classification techniques are used to determine the (excellent) precision of the conformance checking approach.

CDOXplorer's performance is compared with three other state-of-the-art search and optimization algorithms (cf. Chapter 12): Simple random sampling, systematic random sampling, and simulated annealing. Two different scenarios (single cloud and multi cloud) are used in conjunction with two well-known performance metrics (Inverted Generational Distance and Hypervolume Indicator) for assessing all of the multi-objective algorithms. The results show that CDOXplorer can produce solutions that are superior to all of the other approaches. Furthermore, this thesis provides an extensive review of related work that comes from the areas of cloud migration, conformance checking, and deployment and reconfiguration optimization.

Lessons Learned

The design, implementation, and evaluation of CloudMIG led to four observations regarding (1) reverse-engineering system models, (2) CEC modeling, (3) software architecture generation, and (4) simulation-based optimization. Those observations are briefly described as *lessons learned* in the following.

Reverse-engineering System Models As revealed by the review of further approaches that also address the migration of existing applications to the cloud (cf. Section 13.1.1), most approaches employ rather coarse-grained models of those applications. That means, corresponding models may only include existing application tiers, such as a database and a business logic element, or describe the high-level components of those applications. In

contrast, CloudMIG's models that describe an existing software system are automatically extracted with the help of a static analysis and comprise a fine-grained model of the system's source code.

On the one hand, this approach could be seen as an additional hurdle when reasoning about diverse migration options. On the other hand, the extracted KDM models allow for a fine-grained analysis of potential CEC violations. Considering the CEC violations that were detected in the context of the evaluation in the investigated applications, almost none of those CEC violations could have been detected if only high-level system models had been used. Furthermore, our simulation tool CDOSim also relies on the fine-grained KDM models as a basis for simulating CDOs. Hence, the benefit of using these models is clear.

However, our experiments show that extracting KDM models is also subject to some issues. The major issue is the size of those models that can increase quickly. KDM provides very useful means for representing software systems on various levels of abstraction. It is especially useful in software modernization contexts. Nevertheless, model sizes can become an issue depending on the size of an application's source code and the hardware used for model extraction. Corresponding problems could be mitigated by, for instance, using more capable machines with more memory or excluding some application artifacts, such as included unit tests. However, a more sophisticated way to handle large KDM models or to reduce their general size remains an open issue.

CEC Modeling CECs are modeled as part of a cloud profile and by detailing one of the existing CEC types (cf. Section 7.2). However, if a cloud environment exhibits a CEC that does not match a common pre-built CEC type, it has to be modeled with either using the generic `BasicOCLConstraint` or `SMMConstraint` requiring the definition of an OCL expression or an SMM measure, respectively. Both options require expert knowledge for being able to use OCL and SMM. The concept of SMM to allow the modeling of arbitrary measures and computing those measures on the basis of given models is compelling.

However, the SMM instances can become complex and hard to create, read, and interpret. At the time of writing, our domain-specific language Metrics Definition Language (MDL), that aims to mitigate the complexity of creating SMM measures, is under development (cf. [Frey et al. 2012]). Hence, an additional CEC type could then be added that allows to process measures that are defined with MDL. An MDL specification would then be automatically transformed in an SMM instance that could be delegated to the existing `SMMConstraintValidator` (cf. Figure 10.7).

Software Architecture Generation Besides detecting CEC violations, a central aspect of CloudMIG is its deployment and reconfiguration optimization approach that relieves SaaS providers from having to design a well-suited CDO manually. As described in Section 6.3.2, the CloudMIG method also sketches a further approach G2 for generating potential target architectures that uses a rule-based restructuring approach. A drawback of this approach is that it requires a manual intervention. A SaaS provider has to configure and prioritize the rules for restructuring an extracted system model based on specific preferences. This requires a considerable amount of expert knowledge.

Hence, using methods from the Search-Based Software Engineering (SBSE) field, as is done with CDOXplorer, turned out to be more effective, as they allow for a fully automatic architecture creation and optimization. Moreover, the increasing body of knowledge in the area of architecture optimization (cf. [Aleti et al. 2013]) clearly demonstrates the general usefulness of incorporating those methods in architecture design processes. By integrating mature optimization techniques, they allow to automatically search near-optimal solutions and to cope with the prevalent problem of design space explosion.

Simulation-based Optimization As described above, employing an architecture optimization approach and following the idea of SBSE to regard the software engineering problem of designing architectures as a search and optimization problem is intuitive and works very well. However, in the specific case of CDO optimization this was just one side of the coin. The

other side turned out to be employing the methodology of simulation-based optimization that proved to be very useful in the context of creating and optimizing CDOs. Using simulation results as fitness values is an intuitive means for assessing the quality of candidate solutions. Instead of having to map other measures to quality characteristics, incorporating simulation is a more direct and natural way to enable quality assessment.

However, using simulation runs as part of an optimization algorithm has obviously also some drawbacks. The major challenge is the cost of each simulation run in terms of time needed to complete. Hence, to be of practical use, trade-offs often have to be admitted, such as reducing the population sizes and number of generations, which is also done for the genetic algorithm CDOXplorer. Therefore, great care should be taken when admitting those trade-offs as they can easily render well-known and proven optimization algorithms useless. Thus, the evaluation of CDOXplorer also performs a fundamental assessment and evaluates whether the algorithm can provide well-suited results reliably (cf. Section 12.3.2), i.e., whether it converges at all.

14.2 Future Work

As described in the previous chapters of this thesis, CloudMIG covers a wide area of techniques that are used for supporting the migration of existing enterprise software systems to the cloud. Among those techniques are the extraction of KDM models, construction of status quo deployment models, creation of synthetic workload profiles, extraction of workload profiles from actual monitoring log data, and, ultimately, the execution of the conformance checking approach and the optimization of CDOs, for instance. However, there are also some open issues that are left for future work. These open issues are structured into four categories: (1) CloudMIG approach, (2) conformance checking, (3) CDO optimization, and (4) *CloudMIG Xpress*. The future work is described according to these categories in the following.

CloudMIG Approach For detecting CEC violations and mapping elements of an SUA to cloud resources, CloudMIG builds on extracted KDM models. More specifically, it utilizes KDM's `Core`, `KDM`, `Source`, `Code`, `Action`, and to some extent its `Platform` and `Structure` packages (cf. Section 3.3.2). This is due to the fact that CloudMIG focuses on representing code model elements that are extracted from an application's source code.

Giving other system artifacts, such as databases and user interface specifications, greater attention during the migration planning process is a worthwhile option for future extensions of the approach. KDM also provides corresponding packages that could be used (packages `Data` and `UI` in the mentioned example).

A further beneficial extension to CloudMIG could add support for migration projects that target hybrid cloud scenarios, i.e., that want to migrate and distribute an application to several cloud environments. This would also allow the CDO optimization approach to create CDOs of even higher quality. For example, a CDO could comprise two different cloud environments and map large data chunks that are rarely accessed to the cloud storage service of the cloud environment that offers lower costs.

Furthermore, CloudMIG currently supports IaaS and PaaS-based cloud environments as potential targets of a migration. An interesting approach that uses characteristics of both service models comes from the authors of Leitner et al. [2012]. They contribute the approach CloudScale for transparently scaling applications on the basis of IaaS-based cloud environments. Extending CloudMIG to provide corresponding support seems to be a worthwhile option that could be addressed in future work.

CloudMIG focuses on the planning phase of migration projects that consider and evaluate the migration of an existing enterprise application to the cloud. As described in Section 13.1.1, there also exist cloud migration approaches that provide support for executing the actual migration (e.g., the ARTIST project [Bergmayr et al. 2013]). CloudMIG could also be extended in this direction. For example, when a SaaS provider selects a CDO from CDOXplorer's set of pareto-optimal CDOs, model-driven techniques could be used to transform the application according to the configuration described

in that CDO. Then, the modified application could be deployed using the specific APIs of the chosen cloud environment.

Similarly, besides supporting the actual migration process, the ARTIST project mentioned above also covers a quality assurance step that follows the actual migration. Employing existing test cases for ensuring functional equivalence of the migrated system would also constitute a worthwhile enhancement of the CloudMIG approach.

Conformance Checking As described in Section 7.4.2, CloudMIG also includes an approach for prioritizing the correction of CEC violations that are detected by the conformance checking process. The corresponding prioritization function `prio` basically utilizes the severity levels of the detected CEC violations and also an architectural coupling metric for incorporating the potential impact of specific CEC violation fixes. The corresponding results presented in the context of the experiments in Section 11.5.1 are promising, as they match with the intuition of narrowing the focus to particularly critical classes. However, as stated in Section 11.6.1, the generalizability of the particular results is limited as the found CEC violations are not corrected due to the construction of the specific case study. Therefore, `prio` is not listed as a contribution of this thesis in Section 1.3. Additional experiments should be conducted as part of the future work to further validate the prioritization function `prio`.

CDO Optimization The simulation-based genetic algorithm CDOXplorer currently considers the three objectives costs, response times, and number of SLA violations when optimizing CDOs. This is due to the fact that our simulation tool CDOSim is used as a fitness function and CDOSim delivers results for these objectives. Extending CDOSim and CDOXplorer to include additional objectives could further improve the overall applicability of produced CDOs and incorporate further aspects that are also of interest to SaaS providers. For example, to lower future costs of maintaining the migrated software systems, metrics that indicate maintainability could also be used to extend the fitness function (cf. [Harman and Clark 2004]).

Furthermore, CDOXplorer produces optimized CDOs on the basis of IaaS cloud environments. This is due to the fact that the code container structures of PaaS-based cloud environments are far more heterogeneous than those of IaaS-based cloud environments. Nevertheless, enhancing CDOXplorer in this direction would significantly increase the number of potential CDOs and therefore offer more possibilities to SaaS providers to choose from.

CloudMIG Xpress The proof of concept implementation of CloudMIG (*CloudMIG Xpress*) also has several open issues. Section 6.3.3 sketches the concept of a repository for publicly storing and exchanging cloud profiles, CEC validators, monitoring format readers, and KDM discoverers. At the time of writing, several components of this repository already exist. However, functionalities to access this repository need to be included in *CloudMIG Xpress* in the future work.

Regarding the implementation of CloudMIG's conformance checking approach, the existing constraint validators all employ a static analysis on the basis of extracted KDM models. However, there exist further possibilities to design constraint validators. For example, dynamic analyses could be considered in the future work to cover a greater range of detectability categories (cf. Section 7.1.3).

Furthermore, there currently exist those cloud profiles, KDM discoverers, monitoring format readers, and CEC validators that were used in the context of the experimental evaluation in the Chapters 11 and 12. Additional cloud profiles, KDM discoverers, monitoring format readers, and CEC validators should be added to cover further potential migration use cases.

The genetic algorithm CDOXplorer uses CDOSim as a fitness function. CDOXplorer and also CDOSim itself should be parallelized as part of the future work to better exploit multi-core systems. Adding support for automatically distributing simulation jobs to a set of available machines could enable further performance improvements.

Appendix A

The Packages of the Cloud Environment Model

The Cloud Environment Model is a meta-model for describing PaaS- and IaaS-based cloud environments from the perspective of a cloud user. As described in Section 6.3.4, it consists of packages that are organized in a layered structure. With $l(p)$ being the layer number of a package p, it means that an element of package A can only access an element of package B, if $l(A) \geqslant l(B)$. There exist the following eight packages.

Core Package Provides basic elements that are utilized or extended by elements of several other packages.

Mapping Package The included elements enable the assignment of extracted source code models to cloud resources.

Constraints Package Provides elements for the definition of a specific cloud environment's CECs.

Usage Package Defines generic elements for specifying parts of the utilization model that utilize OMG's SMM.

IaaS Package Elements for the definition of services and resources of a cloud environment that follows the IaaS service model.

PaaS Package Elements for the definition of services and resources of a cloud environment that follows the PaaS service model.

Pricing Package Provides elements for the definition of a cloud environment's pricing model, i.e., the prices of the provided services.

Cloud Profile Package Includes fundamental elements for the specification of a particular cloud environment, i.e., a cloud profile.

The packages are described in the following with the help of UML class diagrams. Please note that we describe the version of CEM that was modeled using Ecore. We do not make use of the, semantically identical, KDM version of CEM that results by transforming this Ecore-based version (cf. Section 6.3.4). The Ecore class diagrams are better suited for illustrating the concepts and elements of the packages than the rather unwieldy representation as an internal KDM DSL.

However, some details of the packages are missing as the single class diagrams do not include references to the other packages for reasons of readability. The complete Ecore model `CEM.ecore` can be found online in the source code version of the tool *CloudMIG Xpress*.[1]

[1] `http://www.cloudmig.org`

Core Package

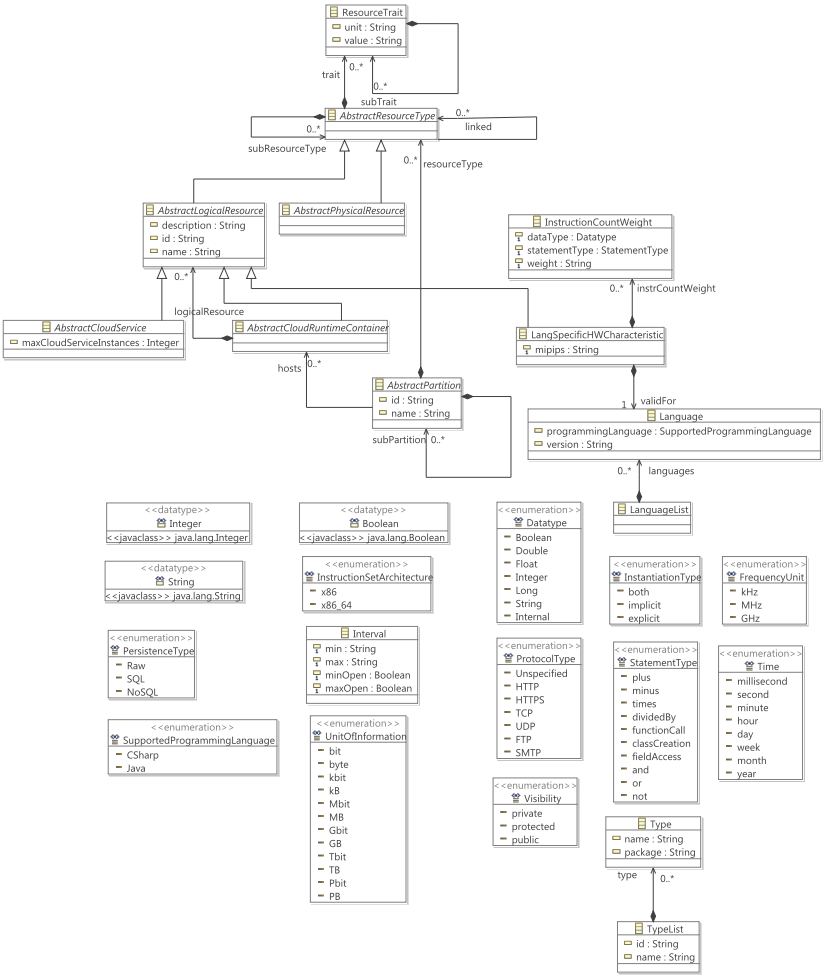


Figure A.1. The *Core* Package of CEM

The *Core* package contains several abstract types, as it functions as a foundation for the other packages and provides many elements that are used or extended by other packages. Similarly to the KDM representation of CEM, that can be obtained via transforming the Ecore model of CEM to a KDM-based DSL, the Ecore model builds on the notion of *resources* that form a cloud platform.[2]

A central basic element is `AbstractResourceType` that is the base class of most of CEM's elements. It can contain several `ResourceTraits` that describe arbitrary properties of a resource, i.e., properties that are not included explicitly in the CEM. Further essential types are given by two of its subtypes `AbstractPhysicalResource` and `AbstractLogicalResource`. The former is the basis for physical resources that are offered to cloud users, such as storage space and network bandwidth, where `AbstractLogicalResources` are logical entities. Specific `AbstractLogicalResources` are services, container structures, or hardware characteristics. Services, such as a database or Virtual Private Network (VPN) service, derive from `AbstractCloudService` and runtime container structures (e.g., VM instances) extend `AbstractCloudRuntimeContainer`. The specific type `LangSpecificHWCharacteristic` models language-specific properties of a runtime container structure. Currently, just the MIPIPS value of the VM instance types are stored. As described in Section 8.2.1, different statement types of a programming language are weighted in the course of benchmarking a status quo node or a VM instance type. Those weights can be modeled with the help of the class `InstructionCountWeight`.

Furthermore, a `LangSpecificHWCharacteristic` refers to a specific `Language`. `Languages` can be grouped in a `LanguageList` and are determined by a programming language and a version of this language. Different programming languages are given by the `SupportedProgrammingLanguage` enumeration. The class `AbstractPartition` can describe a geographic or logic partitioning of a cloud environment (see the classes `Realm` and `Location` in CEM's *IaaS* package). Further classes included for reasons of convenience are `Interval` and `Type`. The latter facilitates a lean specification of a programming language type. Similar to `Languages`, `Types` can be grouped in a corresponding list

[2]The KDM-based DSL mainly utilizes elements of KDM's *Platform* package.

(`TypeList`). Besides the wrapper for the Java data types `String`, `Integer`, and `Boolean`, the rest of the package's elements are enumerations. They are briefly described below.

`Datatype`
: Data types of a programming language.

`FrequencyUnit`
: Units of frequencies such as GHz.

`InstantiationType`
: Determines if runtime code containers (e.g., VMs) are started explicitly, implicitly, or both. When selecting "explicit," the containers have to be started by the cloud user via the command line, web console, or user-provided capacity management tool, for instance. In contrast, the containers are started by the cloud platform without intervention of the cloud user when selecting "implicit." The cloud environment provides the possibility to select from both instantiation types when using "both."

`InstructionSetArchitecture`
: Describes an instruction set architecture such as `x86`.

`PersistenceType`
: A type of persistency such as database models.

`ProtocolType`
: A network protocol type.

`StatementType`
: The type of a programming language statement.

`Time`
: A time unit such as second and minute.

`UnitOfInformation`
: A unit of information, for example, MB and TB.

`Visibility`
: Visibility identifiers of a programming language's elements.

Mapping Package

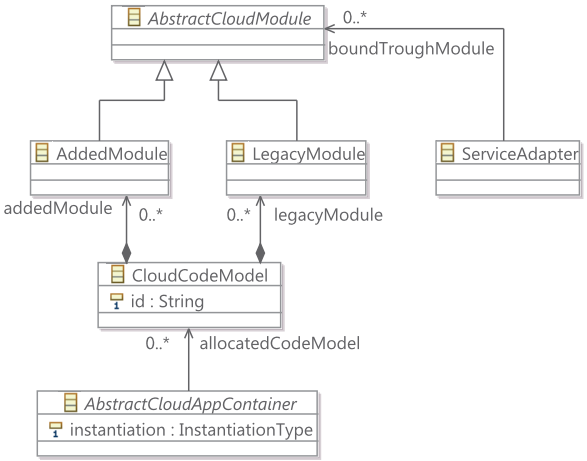


Figure A.2. The *Mapping* Package of CEM

The *Mapping* package uses the notion of *modules* to group parts of a software system's source code. Modules that run on a cloud platform derive from `AbstractCloudModule`. There exist the two child classes `LegacyModule` and `AddedModule`. The former represents modules that are part of an existing enterprise software system that should be migrated to the cloud (SUA). The latter describes modules that did not exist in that enterprise software system before and have to be added during the actual migration activity. Several `AbstractCloudModules` can be grouped to a `CloudCodeModel`.

The element `AbstractCloudAppContainer` models a container structure that includes a migrated application that is given by zero to more `CloudCodeModels`. For example, an `AbstractCloudAppContainer` can be a VM image. The class `ServiceAdapter` constitutes an adapter that connects an `AbstractCloudModule` to a service that is provided by the cloud environment. For example, for not only saving processed document files to a VM instance's local storage, but additionally to a cloud environment's BLOB storage service.

Constraints Package

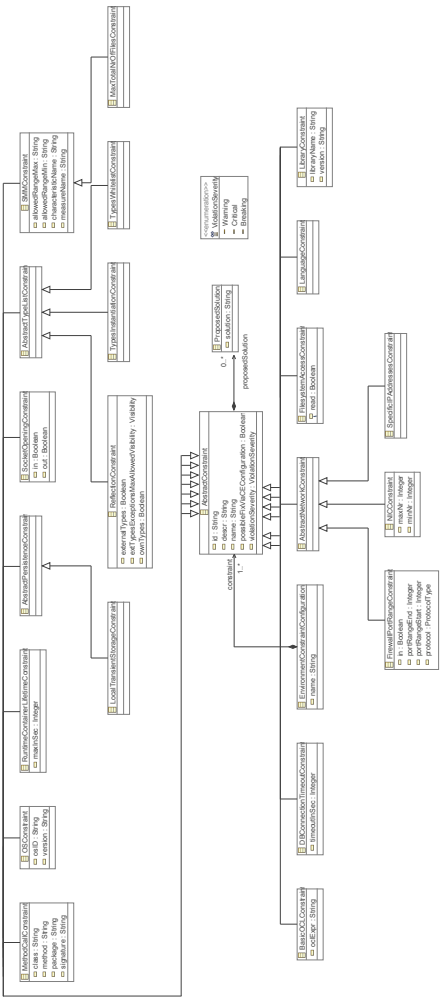


Figure A.3. The *Constraints* Package of CEM

The *Constraints* package provides means for specifying CECs. A cloud profile defines its CECs with the help of the `EnvironmentConstraintConfiguration` container class, that includes instances of the central element `AbstractConstraint` (the actual CECs). Among others, the violation severity of CECs can be defined with the help of the enumeration `ViolationSeverity`, or solutions can be proposed with instances of the class `ProposedSolution` in the case those solutions exist for frequent causes of corresponding *CEC violations*. For example, if a CEC prevents an application from writing to the file system, a corresponding proposed solution could be to use one of the cloud's specific storage APIs for persisting the data instead.

Most of the classes modeled in the *Constraints* package define specific CECs that can be detected by corresponding constraint validators. The constraints are briefly described below, the CECs that are already covered by corresponding CEC validators in the tool *CloudMIG Xpress* are indicated by (*).

`AbstractNetworkConstraint`
: Abstract supertype for constraints posing network-related restrictions.

`AbstractPersistenceConstraint (*)`
: Abstract supertype for constraints posing persistence-related restrictions.

`AbstractTypeListConstraint (*)`
: Abstract supertype for constraints that use a list of types for posing their restrictions.

`BasicOCLConstraint (*)`
: Allows to specify constraints using OCL. The constraint fails if a defined OCL invariant does not hold.

`DBConnectionTimeoutConstraint`
: Restricts connections to databases by specifying a timeout.

`FilesystemAccessConstraint (*)`
: Forbids the read or write access to the file system.

FirewallPortRangeConstraint
Restricts the usage of network ports to a specific port range.

LanguageConstraint (∗)
Marks the cloud environment as usable with specific programming languages.

LibraryConstraint
Restricts the usage of a particular software library to a specific version of that library.

LocalTransientStorageConstraint (∗)
Indicates that locally stored data is not saved persistently.

MaxTotalNrOfFilesConstraint (∗)
Indicates that there exists a maximum number of files that can be part of an application.

MethodCallConstraint (∗)
Forbids the calling of specific methods.

NICConstraint
Sets a minimum and maximum number of Network Interface Cards (NICs) that can be used by an application.

OSConstraint
Specifies operating systems that can be used for running an application.

ReflectionConstraint (∗)
Restricts the usage of reflection operations.

RuntimeContainerLifetimeConstraint
Sets a maximum lifetime of an application runtime container. For example, a worker thread in a PaaS cloud environment may be terminated after a specific number of seconds.

SocketOpeningConstraint (∗)
Network sockets for incoming or outgoing traffic may not be openable for applications.

SMMConstraint (*)
A constraint that executes an SMM `DirectMeasure` on the extracted KDM model and restricts valid values to a specific range.

SpecificIPAddressConstraint
Restricts the usage of IP addresses to a specific address range.

TypesInstantiationConstraint (*)
Forbids the instantiation of specific types.

TypesWhitelistConstraint (*)
Only types specified by this constraint are allowed to be used.

Usage Package

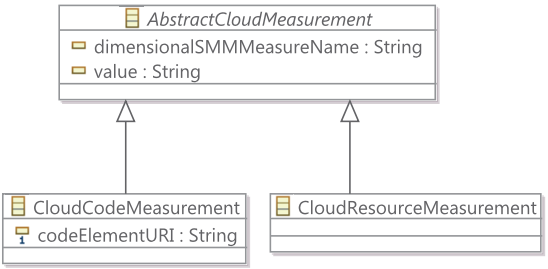


Figure A.4. The *Usage* Package of CEM

The *Usage* package defines elements for supporting the creation of a utilization model in CloudMIG's activity A1. It utilizes SMM and provides a lean mechanism to link CDOs and cloud resources with SMM measurements. The `AbstractCloudMeasurement` class refers to a corresponding SMM `DimensionalMeasure` and includes the measurement result. The `CloudCodeMeasurement` provides measurement results referring to a KDM element. The `CloudResourceMeasurement` provides measurement results referring to a specific cloud resource.

IaaS Package

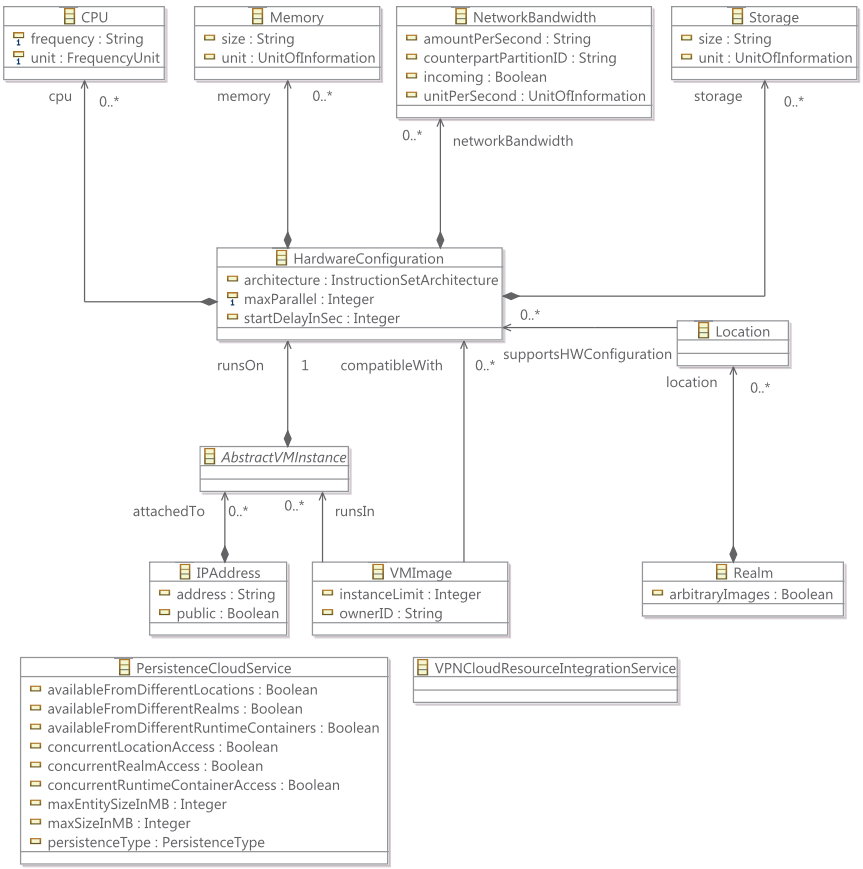


Figure A.5. The *IaaS* Package of CEM

The *IaaS* package contains elements for describing characteristics of IaaS-based cloud environments. To model geographical and logical structuring of computational resources that are provided by a cloud environment, the *IaaS*

package contains the Realm and the Location element. For example, a user of the cloud environment Amazon EC2[3] can choose a so-called *Region* where a VM instance should be started. A *Region* is a data center located at a specific geographical position and is mapped in the cloud profile of Amazon EC2 to a Realm element. The provided data centers of Amazon EC2 are globally distributed to lower the latency for local users. Each *Region* is divided into several *Availability Zones* that provide insulation from failures in other *Availability Zones*. Considering the exemplary cloud profile of Amazon EC2, an *Availability Zone* maps to a Location element.

Central building blocks of IaaS-based cloud environments are the offered VM instance types. CEM's *IaaS* package models VM instance types with the help of the HardwareConfiguration element. A HardwareConfiguration element specifies the provided hardware resources, i.e., included CPUs, Memory, NetworkBandwidth, and Storage with the corresponding model elements. VM images contain an installed operating system and applications. They are modeled by the class VMImage. VM instances are started from a specific VM instance type—i.e., using a specific HardwareConfiguration—and from a particular VMImage. VM instances are modeled with instances of a subclass of AbstractVMInstance. As CEM just models the static structure of a cloud environment, concrete dynamic *VM* instances (i.e., a concrete subtype of AbstractVMInstance) are not included. However, specific IPAddresses can be modeled for attaching to an AbstractVMInstance.

The further class PersistenceCloudService models services of a cloud environment that can be used for storing data persistently. Considering the exemplary cloud environment Amazon EC2, this could be Amazon S3, Amazon RDS, or Amazon SimpleDB, for instance.[4] Furthermore, the VPNCloudResourceIntegrationService element allows to model integration of a VPN service.

[3] http://aws.amazon.com/ec2/

[4] http://aws.amazon.com/products/

PaaS Package

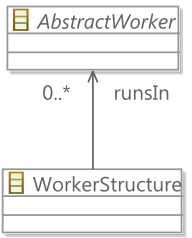


Figure A.6. The *PaaS* Package of CEM

The *PaaS* package contains elements for describing characteristics of PaaS-based cloud environments. The included `WorkerStructure` *runs in* an `AbstractWorker`. The first refers to a container for KDM elements, the latter describes worker elements of a PaaS-based cloud environment, such as processes and threads.

Pricing Package

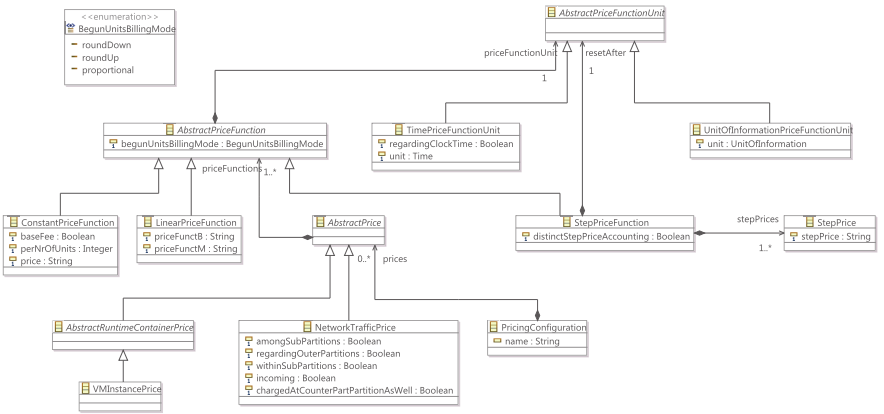


Figure A.7. The *Pricing* Package of CEM

The *Pricing* package allows to build a pricing model regarding a cloud environment's provided services and resources. A price is represented by subclasses of `AbstractPrice`. Several prices are grouped in a `PricingConfiguration`. There exist two subclasses of `AbstractPrice`: `NetworkTrafficPrice` and `AbstractRuntimeContainerPrice`. The former models a price that is billed for transferring a specific amount of data to or from a specific location. Subclasses of the latter enable the specification of prices regarding `AbstractCloudRuntimeContainers` (see the `Core` package). The current version of CEM contains the specific subclass `VMInstancePrice` that models the price for using a particular VM instance type for a certain time.

To enable, to some degree, flexibility in the definition of a pricing model, the *Pricing* package describes prices with the help of mathematical functions that refer to specific units. For example, the more a VM instance is used, the evenly more it costs (linear function). Considering the cloud environment Amazon EC2, VM instances are billed on an hourly basis. Hence, the unit of the linear function corresponds to *hours*. Price functions are modeled with `AbstractPriceFunctions` and their units are given by

`AbstractPriceFunctionUnits`. There exist two specific `AbstractPriceFunctionUnits`: `TimePriceFunctionUnit` and `UnitOfInformationPriceFunctionUnit`. The former corresponds to prices that are billed per time unit (such as the example of Amazon EC2 mentioned above). The latter refers to units of certain amounts of data, for example, *GB*.

CEM includes three concrete price functions that are briefly described below.

`ConstantPriceFunction`
: Enables the modeling of constant prices, e.g., a fixed price per month.

`LinearPriceFunction`
: A `LinearPriceFunction` models variable, linear increasing costs, e.g., the usage of VM instance type X could cost $0.6/h.

`StepPriceFunction`
: Allows modeling prices according to a step function, e.g., $0.2 per TB network traffic for the first ten TB, $0.15 per TB for 10-20 TB.

Furthermore, the `BegunUnitsBillingMode` enumeration enables specifying the exact price calculation mode for begun units. For example, consider hourly VM instance type prices and a usage time of 65 minutes. The cloud provider could charge (1) one hour, (2) two hours, or (3) one hour and five minutes, where the excessive five minutes are billed proportionally to the full hour price, for instance.

Cloud Profile Package

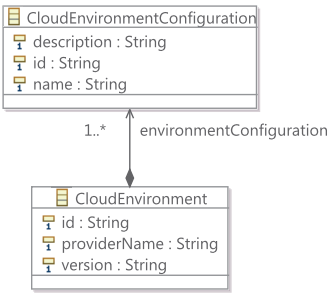


Figure A.8. The *Cloud Profile* Package of CEM

The *Cloud Profile* package builds the umbrella for specifying cloud profiles. It consists of the elements `CloudEnvironment` and `CloudEnvironmentConfiguration`. The former represents the specific cloud environment that is offered by a cloud provider, the latter refers to one or more cloud environment configurations that can be part of a single cloud environment. We refer to Section 6.3.4 for further details.

Appendix B

Cloud Profile Excerpts

The evaluation of CloudMIG employs four cloud profiles that model PaaS and IaaS-based cloud environments with the help of CEM (cf. Chapters 11 and 12). The tool *CloudMIG Xpress* internally uses a transformed KDM-based representation of cloud profiles (cf. Section 6.3.4). However, this appendix describes the four cloud profile specifications using the original XMI representation to improve readability. Though, only a small excerpt of the cloud profiles is presented due to space constraints. The complete cloud profiles are included in the released version of *CloudMIG Xpress*.[1] The four cloud environments are briefly described below.

- **Amazon EC2** Amazon EC2[2] is an IaaS-based cloud environment. The cloud profile is used for evaluating the simulation-based evolutionary algorithm CDOXplorer (cf. Chapter 12) and for assessing the CDO simulation tool CDOSim [Fittkau 2012; Fittkau et al. 2012a; b].
- **Eucalyptus Cluster** The cloud profile describes a private IaaS-based cloud environment of the Software Engineering Group, Kiel University, Germany, that utilizes the cloud software Eucalyptus.[3] The cloud profile is used for evaluating CDOXplorer (cf. Chapter 12) and assessing the CDO simulation tool CDOSim [Fittkau 2012; Fittkau et al. 2012a; b].

[1] http://www.cloudmig.org

[2] http://aws.amazon.com/ec2

[3] http://www.eucalyptus.com

- **Google App Engine (GAE)** The cloud profile of Google App Engine[4] describes V. 1.3.6. of the Google App Engine for Java API. GAE is a PaaS-based cloud environment. The corresponding cloud profile is used for evaluating CloudMIG's conformance checking approach (cf. Chapter 11).
- **Microsoft Windows Azure** The VM role of Microsoft Windows Azure[5] constitutes an IaaS-based cloud environment. The corresponding cloud profile is used for evaluating CloudMIG's conformance checking approach (cf. Chapter 11) and the simulation-based evolutionary algorithm CDOXplorer (cf. Chapter 12).

For each cloud environment, a corresponding cloud profile excerpt is presented in the following. Please refer to Appendix A and the user guide documentation that comes with the tool *CloudMIG Xpress* for a detailed description of the included elements.

[4] https://developers.google.com/appengine

[5] http://www.windowsazure.com

Amazon EC2

The excerpt of the Amazon EC2 cloud profile shown in Listing B.1 contains several noteworthy cloud environment characteristics that also demonstrate the appropriate usage of CEM (cf. Appendix A). Among others, the following elements of the cloud environment are included (the numbers refer to line numbers in Listing B.1).

- **[2] Provider name:** Besides the definition of used namespaces, line 2 specifies the name of the provider that offers the corresponding cloud environment.
- **[5] Cloud services:** The cloud environment provides several services that can be used separately or in conjunction with VMs, e.g., SimpleDB, Simple Notification Service (SNS), and Simple Workflow Service (SWS). The cloud service in line 5 refers to the Elastic Block Store (EBS) service that provides block level storage capabilities.
- **[6] Constraints:** The CECs of the cloud environment are defined in a so-called `EnvironmentConstraintConfiguration` container. This element is specified in CEM's `Constraints` package (cf. Appendix A). The cloud profile excerpt in Listing B.1 shows a single CEC example in line 7 (a `FirewallPortRangeConstraint`) that is included in this container.
- **[10] VM instance types:** CEM specifies basic hardware resource types of IaaS-based cloud environments, such as VM instance types from Amazon EC2, with the help of the `HardwareConfiguration` element. Starting from line 10, the cloud profile excerpt defines Amazon EC2's *m1.small* VM instance type.
- **[12] Computational capabilities:** Amazon EC2's MIPIPS and weights values for the *m1.small* VM instance type that were measured with CDOSim's MIPIPS and weights benchmark (cf. Section 8.2).
- **[19] Data centers:** Amazon EC2's data centers and the offered availability zones are modeled with CEM's `Realm` and `Location` elements, respectively.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <cloudprofile:CloudEnvironment xmi:version="2.0" xmlns:xmi="..." xmlns:xsi="..."
       xmlns:cloudprofile="..." xmlns:constraints="..." xmlns:iaas="..." xmlns:pricing="..." id="org.
       cloudmig.cloudprofiles.amazon" providerName="Amazon Web Services LLC" version="0.1">
3    <environmentConfiguration description="..." id="org.cloudmig.cloudprofiles.amazon.ec2" name="
         Amazon Elastic Compute Cloud (EC2)">
4      <appDataContainer xsi:type="iaas:VMImage" description="..." id="..." name="Basic 64-bit
           Amazon Linux AMI 2011.09" compatibleWith="..." instanceLimit="-1" ownerID="amzn"/>
5      <cloudService xsi:type="iaas:PersistenceCloudService" description="..." id="..." name="Amazon
           Elastic Block Store (EBS)" maxCloudServiceInstances="-1"
           availableFromDifferentRuntimeContainers="true" maxEntitySizeInMB="-1"
           maxSizeInMB="1000000"/>
6      <constraintConfiguration name="Constraints">
7        <constraint xsi:type="constraints:FirewallPortRangeConstraint" id="..." descr="..." name="Initial
             incoming" in="true" portRangeEnd="-1" portRangeStart="-1">
8        </constraint>
9      </constraintConfiguration>
10     <hardwareConfiguration description="Standard Instance - Small Instance" id="m1.small" name=
           "Small Instance" maxParallel="-1" startDelayInSec="149">
11       <cpu frequency="1.2" unit="GHz"/>
12       <langSpecificHWChar description="" id="..." name="" mipips="52.29">
13         <instrCountWeight statementType="and" weight="1.201088956280618"/>
14       </langSpecificHWChar>
15       <memory size="1.7" unit="GB"/>
16       <networkBandwidth amountPerSecond="1000" unitPerSecond="Mbit"/>
17       <storage size="160" unit="GB"/>
18     </hardwareConfiguration>
19     <partition xsi:type="iaas:Realm" id="org.cloudmig.cloudprofiles.amazon.realms.uswest1" name="
           US-West-1" arbitraryImages="true">
20       <location id="..." name="us-west-1a" supportsHWConfiguration="..."/>
21       <location id="..." name="us-west-1b" supportsHWConfiguration="..."/>
22     </partition>
23     <pricingConfiguration name="Amazon EC2 US-East VM Instances On-Demand Pricing Conf.">
24       <prices xsi:type="pricing:VMInstancePrice" validInPartitions="..." runtimeContainerBasesOn="
             ..." validForHardwareConfigurations="...">
25         <priceFunctions xsi:type="pricing:ConstantPriceFunction" baseFee="false"
               begunUnitsBillingMode="roundUp" perNrOfUnits="1" price="0.085">
26           <priceFunctionUnit xsi:type="pricing:TimePriceFunctionUnit" regardingClockTime="false"
                 unit="hour"/>
27         </priceFunctions>
28       </prices>
29     </pricingConfiguration>
30   </environmentConfiguration>
31 </cloudprofile:CloudEnvironment>
```

Listing B.1. Amazon EC2 cloud profile (excerpt)

Eucalyptus Cluster

The excerpt of the Eucalyptus cluster cloud profile shown in Listing B.2 contains several noteworthy cloud environment characteristics that also demonstrate the appropriate usage of CEM (cf. Appendix A). Among others, the following elements of the cloud environment are included (the numbers refer to line numbers in Listing B.2).

- **[4] Available VM images:** A VM image that can be used to deploy further software. In the context of CloudMIG, extracted KDM elements from an SUA are mapped to VM images. VM instances are then started from these mappings.
- **[5] Constraints:** The `EnvironmentConstraintConfiguration` container includes a single `LocalTransientStorageConstraint` (cf. CEM's `Constraints` package in Appendix A).
- **[16] VM instance types' memory:** The amount of main memory that is available to an exemplary VM instance type of the Eucalyptus configuration.
- **[17] VM instance types' network bandwidth:** The network bandwidth that is available to an exemplary VM instance type of the Eucalyptus configuration.
- **[18] VM instance types' storage:** The local storage that is available to an exemplary VM instance type of the Eucalyptus configuration.
- **[23] Pricing models:** The example pricing model defines the price for using a VM instance of a specific VM instance type for one hour.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <cloudprofile:CloudEnvironment xmi:version="2.0" xmlns:xmi="..." xmlns:xsi="..."
       xmlns:cloudprofile="..." xmlns:constraints="..." xmlns:iaas="..." xmlns:pricing="..." id="org.
       cloudmig.cloudprofiles.eucalyptus" providerName="SE Group Kiel" version="0.1">
3    <environmentConfiguration description="..." id="org.cloudmig.cloudprofiles.eucalyptus" name="
         Eucalyptus">
4      <appDataContainer xsi:type="iaas:VMImage" description="..." id="emi-7f418316" name="Basic
           32-bit Eucalyptus Linux EMI 2011.09" compatibleWith="..." instanceLimit="-1" ownerID="
           euca"/>
5      <constraintConfiguration name="Constraints">
6        <constraint xsi:type="constraints:LocalTransientStorageConstraint" id="..." descr="..." name="
             Local Transient Storage" possibleFixViaCEConfiguration="true">
7        </constraint>
8      </constraintConfiguration>
9      <hardwareConfiguration description="Small Instance" id="m1.small" name="Small Instance"
           architecture="x86_64" maxParallel="8" startDelayInSec="89">
10       <cpu frequency="2.7" unit="GHz"/>
11       <langSpecificHWChar id="m1.small.langSpecificHWCharacteristic.java.6" description="" name="
             " mipips="210.55825546622216">
12         <validFor programmingLanguage="Java" version="1.6.0_30"/>
13         <instrCountWeight dataType="Long" statementType="minus" weight="2.339003823904304"/>

14         <instrCountWeight dataType="Internal" statementType="functionCall" weight="
               1.71776644769095"/>
15       </langSpecificHWChar>
16       <memory size="1" unit="GB"/>
17       <networkBandwidth amountPerSecond="1000" unitPerSecond="Mbit"/>
18       <storage size="12" unit="GB"/>
19     </hardwareConfiguration>
20     <partition xsi:type="iaas:Realm" id="..." name="Own" arbitraryImages="true">
21       <location id="..." name="Kiel" supportsHWConfiguration=""/>
22     </partition>
23     <pricingConfiguration name="Own VM Instances On-Demand Pricing Configuration">
24       <prices xsi:type="pricing:VMInstancePrice" validInPartitions="..." runtimeContainerBasesOn="
             ..." validForHardwareConfigurations="...">
25         <priceFunctions xsi:type="pricing:ConstantPriceFunction" baseFee="false"
               begunUnitsBillingMode="roundUp" perNrOfUnits="1" price="0.53">
26           <priceFunctionUnit xsi:type="pricing:TimePriceFunctionUnit" regardingClockTime="false"
                 unit="hour"/>
27         </priceFunctions>
28       </prices>
29     </pricingConfiguration>
30   </environmentConfiguration>
31 </cloudprofile:CloudEnvironment>
```

Listing B.2. Eucalyptus cluster cloud profile (excerpt)

Google App Engine

The excerpt of the Google App Engine (GAE) cloud profile shown in Listing B.3 contains several noteworthy cloud environment characteristics that also demonstrate the appropriate usage of CEM (cf. Appendix A). Among others, the following elements of the cloud environment are included (the numbers refer to line numbers in Listing B.3).

- **[3] Environment configurations:** Besides GAE for Java, GAE offers sandbox environments for software written in PHP, Python, and Go, too. Hence, for extending the cloud profile to also include those three sandbox environments, a cloud profile contributor (cf. Section 6.3.3) would have to add three more `CloudEnvironmentConfiguration` elements (see CEM's `Cloud Profile` package in Appendix A).
- **[6] Proposed solutions:** A cloud profile contributor can include proposed solutions in cloud profiles when specifying CECs. In line 6 of the cloud profile excerpt in Listing B.3, a proposed solution is described that may assist CloudMIG users (cf. Section 6.3.3) if a CEC violation regarding the CEC type `FileSystemAccessConstraint` is detected when checking their SUA.
- **[17] Type lists:** Type lists are container structures for types that can be used in the context of CEC specifications, for instance. Applications running in GAE for Java can only access JRE classes that are included in the type list starting in line 17 of the cloud profile excerpt in Listing B.3. The corresponding CEC definition in line 11 specifies a `TypesWhitelistConstraint` that references this type list via its `types` attribute (omitted in Listing B.3 for brevity). Hence, during execution of the conformance checking process, if an access to a type is detected that is not included in this list, a corresponding CEC violation is raised.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <cloudprofile:CloudEnvironment xmi:version="2.0" xmlns:xmi="..." xmlns:xsi="..."
       xmlns:cloudprofile="..." xmlns:constraints="..." xmlns:iaas="..." id="org.cloudmig.
       cloudprofiles.gae" providerName="Google Inc." version="0.1">
3    <environmentConfiguration description="..." id="org.cloudmig.cloudprofiles.gae.java" name="Google
         App Engine for Java">
4      <constraintConfiguration name="Constraints">
5        <constraint xsi:type="constraints:FilesystemAccessConstraint" id="..." descr="..." name="No
             Filesystem Write" violationSeverity="Critical">
6          <proposedSolution solution="Utilize the GAE datastore for storing persistent data."/>
7        </constraint>
8        <constraint xsi:type="constraints:FilesystemAccessConstraint" id="..." descr="..." name="Limited
             Filesystem Read" read="true">
9          <proposedSolution solution="Where feasible: Upload needed files with the application to read
               them from the filesystem."/>
10       </constraint>
11       <constraint xsi:type="constraints:TypesWhitelistConstraint" id="..." descr="..." name="Allowed
             JRE Types" violationSeverity="Critical" types="..." closure="..."/>
12       <constraint xsi:type="constraints:ReflectionConstraint" id="..." descr="..." name="Restricted
             Reflection" violationSeverity="Critical" types="..." extTypesExceptionsMaxAllowedVisibility=
             "public" ownTypes="true"/>
13       <constraint xsi:type="constraints:SocketOpeningConstraint" id="..." descr="..." name="Not
             Directly" violationSeverity="Critical" in="true" out="true">
14         <proposedSolution solution="Refactor application to utilize one of the allowed ports, use the URL
               Fetch API and let Google App Engine open ports implicitly."/>
15       </constraint>
16       <constraint xsi:type="constraints:MethodCallConstraint" id="..." descr="..." name="
             SecurityManager.getThreadGroup" violationSeverity="Critical" class="SecurityManager"
             method="getThreadGroup" package="java.lang" signature=""/>
17       <typeLists id="..." name="Google App Engine JRE Whitelist">
18         <!-- Whitelist contains 1,405 entries (excluded for readability) -->
19         <type name="DataFlavor" package="java.awt.datatransfer"/>
20         <type name="MimeType" package="java.awt.datatransfer"/>
21         <type name="Transferable" package="java.awt.datatransfer"/>
22         <type name="AppletInitializer" package="java.beans"/>
23       </typeLists>
24       <typeLists name="Google App Engine Thread Restriction Types List">
25         <type name="Thread" package="java.lang"/>
26         <type name="Timer" package="java.util"/>
27       </typeLists>
28       <typeLists id="..." name="Google App Engine Java Platform SE6 Typelist Reference"/>
29     </constraintConfiguration>
30   </environmentConfiguration>
31 </cloudprofile:CloudEnvironment>
```

Listing B.3. Google App Engine (GAE) cloud profile (excerpt)

Microsoft Windows Azure

The excerpt of the Microsoft Windows Azure cloud profile shown in Listing B.4 contains several noteworthy cloud environment characteristics that also demonstrate the appropriate usage of CEM (cf. Appendix A). Among others, the following elements of the cloud environment are included (the numbers refer to line numbers in Listing B.4).

- **[5] Environment configurations:** The VM role of Microsoft Windows Azure is defined with the help of a `CloudEnvironmentConfiguration` element starting in line 5 (see CEM's `Cloud Profile` package in Appendix A). As Microsoft Windows Azure also offers the Web role and Worker role (cf. Section 11.2.3), two further `CloudEnvironmentConfiguration` elements are included in the cloud profile excerpt in Listing B.4 (lines 3 and 4) for demonstration purposes.
- **[8] Constraints:** The `EnvironmentConstraintConfiguration` container element starting in line 8 contains a single `LocalTransientStorageConstraint` (cf. CEM's `Constraints` package in Appendix A). This constraint provokes a CEC violation if an application writes data to the local storage.
- **[20] Pricing models:** The pricing model of Microsoft Windows Azure's VM role is defined with the help of a `PricingConfiguration` element that starts in line 20 (cf. CEM's `Pricing` package in Appendix A). This container element includes the distinct prices defined for this cloud environment, e.g., the price for using a VM instance of a specific VM instance type for one hour.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <cloudprofile:CloudEnvironment xmi:version="2.0" xmlns:xmi="..." xmlns:xsi="..."
       xmlns:cloudprofile="..." xmlns:constraints="..." xmlns:iaas="..." xmlns:pricing="..." id="org.
       cloudmig.cloudprofiles.mswindowsazure" providerName="Microsoft Corporation" version="0.1">
3    <environmentConfiguration description="..." id="org.cloudmig.cloudprofiles.mswindowsazure.
         webrole" name="Microsoft Windows Azure Web Role"/>
4    <environmentConfiguration description="..." id="org.cloudmig.cloudprofiles.mswindowsazure.
         workerrole" name="Microsoft Windows Azure Worker Role"/>
5    <environmentConfiguration description="..." id="org.cloudmig.cloudprofiles.mswindowsazure.
         virtualmachinerole" name="Microsoft Windows Azure Virtual Machine Role">
6      <appDataContainer xsi:type="iaas:VMImage" description="..." id="..." name="Windows Server
           2008 R2 Standard" compatibleWith="..." instanceLimit="-1" ownerID="microsoft"/>
7      <cloudService xsi:type="iaas:PersistenceCloudService" description="..." id="..." name="Microsoft
           BLOB Storage" maxCloudServiceInstances="-1" availableFromDifferentLocations="true"
           availableFromDifferentRealms="true" availableFromDifferentRuntimeContainers="true"
            concurrentLocationAccess="true" concurrentRealmAccess="true"
           concurrentRuntimeContainerAccess="true" maxSizeInMB="1000000"/>
8      <constraintConfiguration name="Constraints">
9        <constraint xsi:type="constraints:LocalTransientStorageConstraint" id="..." descr="..." name="
             Local Transient Storage" possibleFixViaCEConfiguration="true"/>
10     </constraintConfiguration>
11     <hardwareConfiguration description="Extra Small Instance" id="extra.small" name="Extra Small
           Instance" architecture="x86_64" maxParallel="20" startDelayInSec="240">
12       <cpu frequency="1.8" unit="GHz"/>
13       <memory size="768" unit="MB"/>
14     <networkBandwidth amountPerSecond="5" counterpartPartitionID="" unitPerSecond="Mbit"/
           >
15       <storage size="20" unit="GB"/>
16     </hardwareConfiguration>
17     <partition xsi:type="iaas:Realm" id="..." name="Zone 1" arbitraryImages="true">
18       <location id="..." name="North Europe" supportsHWConfiguration="..."/>
19     </partition>
20     <pricingConfiguration name="MS Windows Azure VM Instances On-Demand Pricing Configuration">
21       <prices xsi:type="pricing:VMInstancePrice" validInPartitions="..."
             validForHardwareConfigurations="...">
22         <priceFunctions xsi:type="pricing:ConstantPriceFunction" baseFee="false"
               begunUnitsBillingMode="roundUp" perNrOfUnits="1" price="0.02">
23           <priceFunctionUnit xsi:type="pricing:TimePriceFunctionUnit" regardingClockTime="true"
                 unit="hour"/>
24         </priceFunctions>
25       </prices>
26     </pricingConfiguration>
27   </environmentConfiguration>
28 </cloudprofile:CloudEnvironment>
```

Listing B.4. Microsoft Windows Azure cloud profile (excerpt)

Bibliography

[Accorsi et al. 2011] R. Accorsi, L. Lowis, and Y. Sato. Automated Certification for Compliant Cloud-based Business Processes. *Business & Information Systems Engineering* 3.3 (2011), pages 145–154. (Cited on pages 472–474)

[Adamov and Erguvan 2009] A. Adamov and M. Erguvan. The truth about cloud computing as new paradigm in IT. In: *International Conference on Application of Information and Communication Technologies, 2009. AICT 2009.* 2009, pages 1–3. (Cited on page 36)

[Ahern et al. 2008] D. M. Ahern, A. Clouse, and R. Turner. CMMI Distilled: A Practical Introduction to Integrated Process Improvement (3rd Edition). 3rd edition. Addison-Wesley Professional, May 2008. (Cited on page 435)

[Aksanli et al. 2012] B. Aksanli, J. Venkatesh, and T. Rosing. Using Datacenter Simulation to Evaluate Green Energy Integration. *Computer* 45.9 (2012), pages 56–64. (Cited on page 251)

[Al-Aomar 2002] R. Al-Aomar. A Robust Simulation-Based Multicriteria Optimization Methodology. In: *Proceedings of the Winter Simulation Conference, 2002.* Volume 2. 2002, 1931–1939 vol.2. (Cited on page 100)

[Albrecht and Gaffney 1983] A. Albrecht and J. Gaffney J.E. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering* SE-9.6 (1983), pages 639–648. (Cited on page 438)

[Aleti et al. 2013] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software Architecture Optimization Methods: A Systematic Literature Review. *IEEE Transactions on Software Engineering* 39.5 (2013), pages 658–683. (Cited on pages 11, 481, and 515)

[Alonso et al. 2004] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. Web Services: Concepts, Architectures and Applications (Data-Centric Systems and Applications). 1st edition. Springer, 2004. (Cited on page 144)

[Ardagna et al. 2012] D. Ardagna, E. Di Nitto, G. Casale, D. Pectu, P. Mohagheghi, S. Mosser, P. Matthews, A. Gericke, C. Balligny, F. D'Andria, C.-S. Nechifor, and C. Sheridan. MODACLOUDS, A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds. In: *Modelling in Software Engineering @ ICSE(MiSE)*. IEEE/ACM. Zurich, 2012, pages 1–7. (Cited on pages 450–452)

[Armbrust et al. 2009] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical report UCB/EECS-2009-28. EECS Department, University of California, Berkeley, 2009. (Cited on pages 2, 3, 36, 38, 39, 41, 51, 52, and 129)

[Armbrust et al. 2010] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A View of Cloud Computing. *Communications of the ACM* 53.4 (2010), pages 50–58. (Cited on pages 37, 39, 40)

[Arshad et al. 2003] N. Arshad, D. Heimbigner, and A. Wolf. Deployment and Dynamic Reconfiguration Planning for Distributed Software Systems. In: *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence, 2003.* 2003, pages 39–46. (Cited on pages 482, 483)

[Avison et al. 1999] D. E. Avison, F. Lau, M. D. Myers, and P. A. Nielsen. Action Research. *Communications of the ACM* 42 (1 1999), pages 94–97. (Cited on page 107)

[Avizienis et al. 2004] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing* 1.1 (2004), pages 11–33. (Cited on pages 140 and 182)

[Avriel 2003] M. Avriel. Nonlinear Programming: Analysis and Methods. Dover Publications, 2003. (Cited on page 81)

[Azadivar 1999] F. Azadivar. Simulation Optimization Methodologies. In: *Proceedings of the 1999 Winter Simulation Conference*. WSC '99. Phoenix, Arizona, United States, 1999, pages 93–100. (Cited on page 100)

[Babar and Chauhan 2011] M. A. Babar and M. A. Chauhan. A Tale of Migration to Cloud Computing for Sharing Experiences and Observations. In: *Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing (SECLOUD 2011)*. SECLOUD '11. ACM, 2011, pages 50–56. (Cited on pages 2 and 137)

[Baglietto et al. 2012] P. Baglietto, M. Maresca, M. Stecca, and C. Moiso. Towards a CAPEX-free Service Delivery Platform. In: *16th International Conference on Intelligence in Next Generation Networks (ICIN), 2012*. 2012, pages 8–14. (Cited on page 39)

[Bakshi 2011] K. Bakshi. Considerations for Cloud Data Centers: Framework, Architecture and Adoption. In: *IEEE Aerospace Conference, 2011*. 2011, pages 1 –7. (Cited on pages 440, 441)

[Ball et al. 2011] T. Ball, V. Levin, and S. K. Rajamani. A Decade of Software Model Checking with SLAM. *Communications of the ACM* 54 (7 2011), pages 68–76. (Cited on page 462)

[Balogh et al. 2005] A. Balogh, D. Varró, and A. Pataricza. Model-Based Optimization of Enterprise Application and Service Deployment. In: *Service Availability*. Edited by M. Malek, E. Nett, and N. Suri. Volume 3694. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005, pages 84–98. (Cited on pages 483–485)

[Banks et al. 2004] D. Banks, L. House, F. R. McMorris, P. Arabie, and W. A. Gaul, editors. Classification, Clustering, and Data Mining Applications (Studies in Classification, Data Analysis, and Knowledge Organization). Softcover reprint of the original 1st ed. 2004. Springer, 2004. (Cited on page 326)

[Barroso and Hölzle 2009] L. A. Barroso and U. Hölzle. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. *Synthesis Lectures on Computer Architecture* 4.1 (2009), pages 1–108. (Cited on page 251)

[Basili 1992] V. R. Basili. Software Modeling and Measurement: The Goal/Question/Metric Paradigm. Technical report CS-TR-2956. University of Maryland (College Park, MD), 1992. (Cited on pages 324, 325)

[Basili and Rombach 1988] V. Basili and H. Rombach. The TAME Project: Towards Improvement-Oriented Software Environments. *IEEE Transactions on Software Engineering* 14.6 (1988), pages 758–773. (Cited on pages 17, 30, 321, and 324)

[Bass et al. 2002] L. Bass, M. Klein, and F. Bachmann. Quality Attribute Design Primitives and the Attribute Driven Design Method. English. In: *Software Product-Family Engineering*. Edited by F. Linden. Volume 2290. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, pages 169–186. (Cited on pages 16, 29, 119, 281, 282, and 297)

[Bass et al. 2003] L. Bass, P. Clements, and R. Kazman. Software Architecture in Practice (2nd Edition). 2nd edition. Addison-Wesley Professional, Apr. 2003. (Cited on pages 58, 281–283)

[Bauer and King 2006] C. Bauer and G. King. Java Persistence with Hibernate. Revised. Manning Publications, Nov. 2006. (Cited on page 334)

[Baumöl et al. 1996] U. Baumöl, J. Borchers, S. Eicker, K. Hildebrand, R. Jung, and F. Lehner. Einordnung und Terminologie des Software Reengineering. *Informatik-Spektrum* 19 (4 1996), pages 191–195. (Cited on page 58)

[Becker et al. 2006] S. Becker, W. Hasselbring, A. Paul, M. Boskovic, H. Koziolek, J. Ploski, A. Dhama, H. Lipskoch, M. Rohr, D. Winteler, S. Giesecke, R. Meyer, M. Swaminathan, J. Happe, M. Muhle, and T. Warns. Trustworthy Software Systems: A Discussion of Basic Concepts and Terminology. *ACM SIGSOFT Software Engineering Notes* 31.6 (2006), pages 1–18. (Cited on page 182)

[Begin 2002] C. Begin. Implementing the Microsoft .NET Pet Shop using Java. 2002. URL: `http://www.clintonbegin.com/downloads/JPetStore-1-2-0.pdf`. (Cited on page 334)

[BenMrad 1994] M. BenMrad. First International Conference on the World-Wide Web. *SIGWEB Newsl.* 3.2 (Sept. 1994), pages 14–15. (Cited on page 35)

[Bennett and Rajlich 2000] K. H. Bennett and V. T. Rajlich. Software Maintenance and Evolution: A Roadmap. In: *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*. Limerick, Ireland: ACM, 2000, pages 73–87. (Cited on pages 54 and 152)

[Bergey et al. 2001] J. Bergey, L. O'Brien, and D. Smith. DoD Software Migration Planning. Technical Note CMU/SEI-2001-TN-012. Software Engineering Institute, Carnegie Mellon University, 2001. (Cited on page 63)

[Bergmayr et al. 2013] A. Bergmayr, H. Bruneliere, J. L. C. Izquierdo, J. Gorronogoitia, G. Kousiouris, D. Kyriazis, P. Langer, A. Menychtas, L. Orue-Echevarria, C. Pezuela, and M. Wimmer. Migrating Legacy Software to the Cloud with ARTIST. In: *17th European Conference on Software Maintenance and Reengineering (CSMR 2013), European Projects Track*. 2013, pages 465–468. (Cited on pages 414, 416, and 517)

[Beserra et al. 2012] P. V. Beserra, A. Camara, R. Ximenes, A. B. Albuquerque, and N. C. Mendonca. Cloudstep: A Step-by-Step Decision Process to Support Legacy Application Migration to the Cloud. In: *IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2012*. 2012, pages 7 –16. (Cited on pages 414 and 417)

[Bhuvana and Aravindan 2011] J. Bhuvana and C. Aravindan. Preferential local search with adaptive weights in Evolutionary Algorithms for Multiobjective Optimization Problems. In: *International Conference of Soft Computing and Pattern Recognition (SoCPaR), 2011*. 2011, pages 358–363. (Cited on pages 99 and 274)

[Bielefeld 2012] T. C. Bielefeld. Online Performance Anomaly Detection for Large-Scale Software Systems. Diploma Thesis, Kiel University. Diplomarbeit. Department of Computer Science, Kiel University, Germany, 2012. (Cited on page 330)

[Biggerstaff 1989] T. Biggerstaff. Design Recovery for Maintenance and Reuse. *Computer* 22.7 (1989), pages 36 –49. (Cited on page 56)

[Binkley 2007] D. Binkley. Source Code Analysis: A Road Map. In: *Future of Software Engineering, 2007. FOSE '07*. 2007, pages 104–119. (Cited on page 63)

[Binz et al. 2011] T. Binz, F. Leymann, and D. Schumm. CMotion: A Framework for Migration of Applications into and between Clouds. Englisch. In: *IEEE International Conference on Service-Oriented Computing and Applications (SOCA), 2011*. IEEE Computer Society, 2011. (Cited on pages 42, 414, 418, 419, 450, and 453)

[Birman et al. 2009] K. Birman, G. Chockler, and R. van Renesse. Toward a Cloud Computing Research Agenda. *SIGACT News* 40.2 (2009), pages 68–80. (Cited on page 35)

[Bisbal et al. 1999a] B. Bisbal, D. Lawless, B. Wu, and J. Grimson. Legacy Information System Migration: A Brief Review of Problems, Solutions and Research Issues. Technical report TCD-CS-1999-38. Trinity College, Dublin, Ireland, 1999. (Cited on page 63)

[Bisbal et al. 1997] J. Bisbal, D. Lawless, B. Wu, J. Grimson, V. Wade, R. Richardson, and D. O'Sullivan. An Overview of Legacy Information System Migration. In: *Proceedings of the Asia Pacific Software Engineering Conference (APSEC '97) and International Computer Science Conference (ICSC '97)*. 1997, pages 529–530. (Cited on pages 63, 64)

[Bisbal et al. 1999b] J. Bisbal, D. Lawless, B. Wu, and J. Grimson. Legacy Information Systems: Issues and Directions. *Software, IEEE* 16.5 (1999), pages 103–111. (Cited on page 66)

[Bittencourt and Madeira 2011] L. Bittencourt and E. Madeira. HCOC: A Cost Optimization Algorithm for Workflow Scheduling in Hybrid Clouds. *Journal of Internet Services and Applications* 2 (3 2011), pages 207–227. (Cited on pages 490–492)

[Bittencourt 2010] R. Bittencourt. Conformance Checking during Software Evolution. In: *17th Working Conference on Reverse Engineering (WCRE), 2010*. 2010, pages 289 –292. (Cited on pages 466, 467)

[Bloch 2008] J. Bloch. Effective Java (2nd Edition). 2nd edition. Addison-Wesley, May 2008. (Cited on page 44)

[Blum and Roli 2003] C. Blum and A. Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys* 35.3 (Sept. 2003), pages 268–308. (Cited on page 86)

[Boehm 1976] B. Boehm. Software Engineering. *IEEE Transactions on Computers* 25 (1976), pages 1226–1241. (Cited on page 53)

[Bojanova et al. 2013] I. Bojanova, J. Zhang, and J. Voas. Cloud Computing. *IT Professional* 15.2 (2013), pages 12–14. (Cited on page 38)

[Boogerd and Moonen 2008] C. Boogerd and L. Moonen. Assessing the Value of Coding Standards: An Empirical Study. In: *IEEE International Conference on Software Maintenance, 2008. ICSM 2008.* 2008, pages 277–286. (Cited on page 463)

[Boone et al. 2008] B. Boone, F. De Turck, and B. Dhoedt. Automated Deployment of Distributed Software Components with Fault Tolerance Guarantees. In: *Sixth International Conference on Software Engineering Research, Management and Applications, 2008. SERA '08.* 2008, pages 21–27. (Cited on pages 483, 485–487)

[Bowler 2009] P. J. Bowler. Evolution: The History of an Idea. 3 Anv. University of California Press, Sept. 2009, page 496. (Cited on page 88)

[Brade et al. 1992] K. Brade, M. Guzdial, M. Steckel, and E. Soloway. Whorf: A Visualization Tool for Software Maintenance. In: *Proceedings of the IEEE Workshop on Visual Languages, 1992*. 1992, pages 148–154. (Cited on page 62)

[Brandic et al. 2010] I. Brandic, S. Dustdar, T. Anstett, D. Schumm, F. Leymann, and R. Konrad. Compliant Cloud Computing (C3): Architecture and Language Support for User-Driven Compliance Management in Clouds. In: *IEEE 3rd International Conference on Cloud Computing (CLOUD), 2010*. 2010, pages 244 –251. (Cited on pages 473, 474)

[Brandtzæg 2012] E. Brandtzæg. CloudML - A DSL for model-based realization of applications in the cloud. Master's thesis. Oslo, Norway: University of Oslo, Department of Informatics, 2012. (Cited on page 454)

[Brandtzæg et al. 2012a] E. Brandtzæg, P. Mohagheghi, and S. Mosser. Towards a Domain-Specific Language to Deploy Applications in the Clouds. In: *Proceedings of the Third International Conference on Cloud Computing, GRIDs, and Virtualization (Cloud Computing 2012)*. Nice, France, 2012, pages 213–218. (Cited on pages 450 and 454)

[Brandtzæg et al. 2012b] E. Brandtzæg, S. Mosser, and P. Mohagheghi. Towards CloudML, a Model-based Approach to Provision Resources in the Clouds. In: *First International Workshop on Model-Driven Engineering for and on the Cloud (co-located with ECMFA'12) (CloudMDE'12)*. Copenhaghen, Danemark: DTU, 2012, pages 18–27. (Cited on page 454)

[Brinkkemper et al. 1996] S. Brinkkemper, K. Lyytinen, and R. J. Welke, editors. Proceedings of the IFIP TC8, WG8.1/8.2 working conference on method engineering on Method engineering : principles of method construction and tool support. Atlanta, Georgia, United States: Chapman & Hall, Ltd., 1996. (Cited on page 108)

[Brodie and Stonebraker 1993] M. L. Brodie and M. Stonebraker. DARWIN: On the Incremental Migration of Legacy Information Systems. Technical report TR-0222-10-92-165. University of California, Berkeley, 1993. (Cited on pages 63, 67–70)

[Bruneliere et al. 2010] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot. MoDisco: A Generic And Extensible Framework For Model Driven Reverse Engineering. In: *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*. ASE '10. Antwerp, Belgium: ACM, 2010, pages 173–174. (Cited on pages 77, 197, and 296)

[Brunet et al. 2009] J. Brunet, D. Guerrero, and J. Figueiredo. Design Tests: An Approach to Programmatically Check your Code Against Design Rules. In: *31st International Conference on Software Engineering - Companion Volume, 2009. ICSE-Companion 2009.* 2009, pages 255–258. (Cited on pages 466, 467)

[Brusco and Stahl 2005] M. J. Brusco and S. Stahl. Branch-and-Bound Applications in Combinatorial Data Analysis. Springer Verlag, New York, 2005, page 222. (Cited on page 86)

[Buckley 1996] M. Buckley. Linear Array Synthesis Using a Hybrid Genetic Algorithm. In: *Antennas and Propagation Society International Symposium, 1996. AP-S. Digest*. Volume 1. 1996, 584–587 vol.1. (Cited on pages 99 and 274)

[Burke and Kendall 2010] E. K. Burke and G. Kendall, editors. Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques. Springer, Oct. 2010. (Cited on page 86)

[Buyya et al. 2008] R. Buyya, C. S. Yeo, and S. Venugopal. Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In: *10th IEEE International Conference on High Performance Computing and Communications, 2008. HPCC '08.* 2008, pages 5–13. (Cited on pages 1, 35, 37, 38)

[Buyya et al. 2009] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems* 25.6 (2009), pages 599–616. (Cited on page 38)

[Byrne 1992] E. Byrne. A Conceptual Foundation for Software Re-engineering. In: *Proceerdings of the Conference on Software Maintenance, 1992.* Nov. 1992, pages 226–235. (Cited on pages 56–58)

[Calheiros et al. 2009] R. N. Calheiros, R. Ranjan, C. A. F. D. Rose, and R. Buyya. CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. *CoRR* abs/0903.2525 (2009). (Cited on page 251)

[Calheiros et al. 2011] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience* 41.1 (2011), pages 23–50. (Cited on pages 167 and 251)

[Canfora and Di Penta 2007] G. Canfora and M. Di Penta. New Frontiers of Reverse Engineering. In: *Future of Software Engineering, 2007. FOSE '07*. 2007, pages 326 –341. (Cited on page 62)

[Canfora et al. 2005] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. An Approach for QoS-aware Service Composition based on Genetic Algorithms. In: *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*. GECCO '05. Washington DC, USA: ACM, 2005, pages 1069–1075. (Cited on page 236)

[Canfora et al. 2008] G. Canfora, A. R. Fasolino, G. Frattolillo, and P. Tramontana. A wrapping approach for migrating legacy system interactive functionalities to Service Oriented Architectures. *Journal of Systems and Software* 81.4 (2008), pages 463–480. (Cited on page 65)

[Canfora et al. 2011] G. Canfora, M. Di Penta, and L. Cerulo. Achievements and Challenges in Software Reverse Engineering. *Communications of the ACM* 54 (4 2011), pages 142–151. (Cited on pages 60–63)

[Carlshamre 2002] P. Carlshamre. Release Planning in Market-Driven Software Product Development: Provoking an Understanding. English. *Requirements Engineering* 7.3 (2002), pages 139–151. (Cited on page 85)

[Catteddu and Hogben 2009] D. Catteddu and G. Hogben. Cloud Computing. Benefits, risks and recommendations for information security. European Network and Information Security Agency (ENISA). 2009. (Cited on page 41)

[Chapin et al. 2001] N. Chapin, J. E. Hale, K. M. Khan, J. F. Ramil, and W.-G. Tan. Types of software evolution and software maintenance. *Journal of Software Maintenance and Evolution: Research and Practice* 13.1 (2001), pages 3–30. (Cited on page 54)

[Chapman et al. 2010] C. Chapman, W. Emmerich, F. Marquez, S. Clayman, and A. Galis. Elastic Service Definition in Computational Clouds. In: *IEEE/IFIP Network Operations and Management Symposium Workshops (NOMS Wksps), 2010*. 2010, pages 327–334. (Cited on pages 43, 44)

[Chapman et al. 2012] C. Chapman, W. Emmerich, F. Márquez, S. Clayman, and A. Galis. Software architecture definition for on-demand cloud provisioning. *Cluster Computing* 15 (2 2012). 10.1007/s10586-011-0152-0, pages 79–100. (Cited on pages 450, 454–456)

[Chauhan and Babar 2011] M. Chauhan and M. Babar. Migrating Service-Oriented System to Cloud Computing: An Experience Report. In: *IEEE International Conference on Cloud Computing (CLOUD), 2011*. 2011, pages 404–411. (Cited on page 2)

[Chazalet 2010] A. Chazalet. Service Level Checking in the Cloud Computing Context. In: *IEEE 3rd International Conference on Cloud Computing (CLOUD), 2010*. 2010, pages 297 –304. (Cited on pages 473 and 475)

[Chen et al. 1990] Y.-F. Chen, M. Nishimoto, and C. Ramamoorthy. The C Information Abstraction System. *IEEE Transactions on Software Engineering* 16.3 (1990), pages 325–334. (Cited on page 61)

[Chidamber et al. 1998] S. Chidamber, D. Darcy, and C. Kemerer. Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis. *IEEE Transactions on Software Engineering* 24.8 (1998), pages 629–639. (Cited on page 373)

[Chikofsky and Cross 1990] E. Chikofsky and I. Cross J.H. Reverse Engineering and Design Recovery: A Taxonomy. *Software, IEEE* 7.1 (1990), pages 13–17. (Cited on pages 55–57, 60, 61)

[Chou and Ghaboussi 2001] J.-H. Chou and J. Ghaboussi. Genetic algorithm in structural damage detection. *Computers & Structures* 79.14 (2001), pages 1335–1353. (Cited on page 81)

[Chowdhury and Meyers 1993] A. Chowdhury and S. Meyers. Facilitating Software Maintenance by Automated Detection of Constraint Violations. In: *Proceedings of the Conference on Software Maintenance,1993. CSM-93,* 1993, pages 262–271. (Cited on pages 466, 468, 469)

[Clarke et al. 2003] J. Clarke, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd. Reformulating software engineering as a search problem. *Software, IEE Proceedings* 150.3 (2003), pages 161–175. (Cited on pages 81 and 83)

[Clements and Northrop 2001] P. Clements and L. Northrop. Software Product Lines: Practices and Patterns. 3rd. Addison-Wesley Professional, Aug. 2001. (Cited on page 58)

[CloudMIG Xpress Project 2012] CloudMIG Xpress Project. CloudMIG Xpress 0.5 Beta User Guide. Software Engineering Group, Kiel University, Germany. June 2012. URL: http://www.cloudmig.org/. (Cited on pages 156 and 330)

[Coello Coello 2006] C. A. Coello Coello. Evolutionary Multi-Objective Optimization: A Historical View of the Field. *IEEE Computational Intelligence Magazine* 1.1 (2006), pages 28–36. (Cited on page 90)

[Comella-Dorda et al. 2000] S. Comella-Dorda, K. Wallnau, R. Seacord, and J. Robert. A Survey of Black-Box Modernization Approaches for Information Systems. In: *Proceedings of the International Conference on Software Maintenance, 2000.* 2000, pages 173–183. (Cited on page 55)

[Consens et al. 1992] M. Consens, A. Mendelzon, and A. Ryman. Visualizing And Querying Software Structures. In: *Proceedings of the International Conference on Software Engineering, 1992, (ICSE 1992)*. 1992, pages 138 –156. (Cited on page 62)

[Cook et al. 2001] S. Cook, J. He, and R. Harrison. Dynamic and Static Views of Software Evolution. In: *Proceedings of the IEEE International Conference on Software Maintenance, 2001.* 2001, pages 592–601. (Cited on page 53)

[Cooper 1998] H. M. Cooper. Synthesizing Research: A Guide for Literature Reviews. 3. SAGE Publications, Inc., 1998. (Cited on page 107)

[Cornelissen et al. 2009] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke. A Systematic Survey of Program Comprehension through Dynamic Analysis. *IEEE Transactions on Software Engineering* 35.5 (2009), pages 684 –702. (Cited on page 63)

[Creswell 2008] J. W. Creswell. Research Design: Qualitative, Quantitative, and Mixed Methods Approaches (3rd Edition). 3nd. Sage Publications, Inc, July 2008. (Cited on page 107)

[Cryer and Chan 2010] J. D. Cryer and K.-S. Chan. Time Series Analysis: With Applications in R (Springer Texts in Statistics). Springer, Nov. 2010. (Cited on page 247)

[Csorba et al. 2010] M. J. Csorba, H. Meling, and P. E. Heegaard. Ant System for Service Deployment in Private and Public Clouds. In: *Proceedings of the 2nd Workshop on Bio-inspired Algorithms for Distributed Systems*. BADS '10. Washington, DC, USA: ACM, 2010, pages 19–28. (Cited on page 499)

[Csorba et al. 2008] M. Csorba, P. Heegaard, and P. Herrmann. Cost-Efficient Deployment of Collaborating Components. In: *Distributed Applications and Interoperable Systems*. Edited by R. Meier and S. Terzis. Volume 5053. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, pages 253–268. (Cited on pages 483 and 487)

[Davis et al. 2003] L. Davis, D. Flagg, R. Gamble, and C. Karatas. Classifying Interoperability Conflicts. In: *COTS-Based Software Systems*. Edited by H. Erdogmus and T. Weng. Volume 2580. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2003, pages 62–71. (Cited on pages 466, 469, 470)

[De Lucia et al. 1997] A. De Lucia, G. Di Lucca, A. Fasolino, P. Guerra, and S. Petruzzelli. Migrating Legacy Systems towards Object-Oriented Platforms. In: *Proceedings of the International Conference on Software Maintenance, 1997*. Oct. 1997, pages 122 –129. (Cited on page 65)

[Deb et al. 2002] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pages 182–197. (Cited on pages 96–99, and 239)

[Deb 2011] K. Deb. Multi-Objective Optimization Using Evolutionary Algorithms: An Introduction. Technical report KanGAL Report Number 2011003. Department of Mechanical Engineering, Indian Institute of Technology Kanpur, Kanpur, PIN 208016, India, 2011. (Cited on page 89)

[Deb 2012] K. Deb. Advances in Evolutionary Multi-objective Optimization. In: *Search Based Software Engineering*. Edited by G. Fraser and J. Teixeira de Souza. Volume 7515. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2012, pages 1–26. (Cited on page 92)

[Demeyer et al. 2004] S. Demeyer, S. Ducasse, K. Mens, A. Trifu, R. Vasa, and F. Van Rysselberghe. Object-Oriented Reengineering. In: *Object-Oriented Technology. ECOOP 2003 Workshop Reader*. Edited by F. Buschmann, A. Buchmann, and M. Cilia. Volume 3013. Lecture Notes in Computer Science. 10.1007/978-3-540-25934-3_8. Springer Berlin / Heidelberg, 2004, pages 72–85. (Cited on page 371)

[Deng 2007] G. Deng. Simulation-Based Optimization. PhD thesis. University of Wisconsin-Madison, USA, 2007. (Cited on page 100)

[Dikaiakos et al. 2009] M. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali. Cloud Computing: Distributed Internet Computing for IT and Scientific Research. *IEEE Internet Computing* 13.5 (2009), pages 10–13. (Cited on pages 35, 37, and 40)

[Dillon et al. 2010] T. Dillon, C. Wu, and E. Chang. Cloud Computing: Issues and Challenges. In: *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*. 2010, pages 27–33. (Cited on pages 35 and 42)

[Du et al. 2010] J. Du, W. Wei, X. Gu, and T. Yu. RunTest: Assuring Integrity of Dataflow Processing in Cloud Computing Infrastructures. In: *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. ASIACCS '10. Beijing, China: ACM, 2010, pages 293–304. (Cited on pages 473 and 476)

[Dustdar et al. 2011] S. Dustdar, Y. Guo, B. Satzger, and H.-L. Truong. Principles of Elastic Processes. *IEEE Internet Computing* 15.5 (2011), pages 66–71. (Cited on page 36)

[Dustdar et al. 2012] S. Dustdar, Y. Guo, R. Han, B. Satzger, and H.-L. Truong. Programming Directives for Elastic Computing. *IEEE Internet Computing* 16.6 (2012), pages 72–77. (Cited on pages 43, 44, and 248)

[Easterbrook et al. 2008] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. Selecting Empirical Methods for Software Engineering Research. In: *Guide to Advanced Empirical Software Engineering*. Edited by F. Shull, J. Singer, and D. Sjøberg. Springer London, 2008, pages 285–311. (Cited on page 107)

[Ebert et al. 2008] J. Ebert, V. Riediger, and A. Winter. Graph Technology in Reverse Engineering: The TGraph Approach. In: *Workshop Software Reengineering*. Edited by R. Gimnich, U. Kaiser, J. Quante, and A. Winter. Volume 126. LNI. GI, 2008, pages 67–81. (Cited on page 61)

[Ebneter et al. 2010] D. Ebneter, S. Grivas, T. Kumar, and H. Wache. Enterprise Architecture Frameworks for Enabling Cloud Computing. In: *IEEE 3rd International Conference on Cloud Computing (CLOUD), 2010*. 2010, pages 542–543. (Cited on page 431)

[Efftinge et al. 2011] S. Efftinge, S. Frey, W. Hasselbring, and J. Köhnlein. Einsatz domänenspezifischer Sprachen zur Migration von Datenbankanwendungen. In: *Datenbanksysteme für Business, Technologie und Web (BTW 2011)*. Edited by T. Härder, W. Lehner, B. Mitschang, H. Schöning, and H. Schwarz. Volume P-180. Lecture Notes in Informatics. Kaiserslautern: Köllen Druck+Verlag, Mar. 2011, pages 554–573. (Cited on page 22)

[Eick et al. 1992] S. Eick, J. Steffen, and J. Sumner E.E. Seesoft-A Tool For Visualizing Line Oriented Software Statistics. *IEEE Transactions on Software Engineering* 18.11 (1992), pages 957–968. (Cited on page 62)

[Endres and Rombach 2003] A. Endres and D. Rombach. A Handbook of Software and Systems Engineering. Empirical Observations, Laws and Theories. The Fraunhofer IESE Series on Software Engineering. Pearson Education Limited, 2003. (Cited on page 111)

[Ernst et al. 2001] M. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin. Dynamically Discovering Likely Program Invariants to Support Program Evolution. *IEEE Transactions on Software Engineering* 27.2 (2001), pages 99–123. (Cited on page 63)

[Eysholdt et al. 2009] M. Eysholdt, S. Frey, and W. Hasselbring. EMF Ecore Based Meta Model Evolution and Model Co-Evolution. In: *Softwaretechnik-Trends*. Volume 29. 2. (Proceedings of the 11th Workshop Software-Reengineering (WSR 2009)). 2009, pages 20–21. (Cited on page 21)

[Fahrner and Vossen 1995] C. Fahrner and G. Vossen. Transforming Relational Database Schemas into Object-Oriented Schemas according to ODMG-93. In: *Deductive and Object-Oriented Databases*. Edited by T. Ling, A. Mendelzon, and L. Vieille. Volume 1013. Lecture Notes in Computer

Science. Springer Berlin / Heidelberg, 1995, pages 429–446. (Cited on page 66)

[Fehling et al. 2011] C. Fehling, F. Leymann, R. Mietzner, and W. Schupeck. A Collection of Patterns for Cloud Types, Cloud Service Models, and Cloud-based Application Architectures. Englisch. Technical Report 2011/05. University of Stuttgart, Germany, 2011, page 61. (Cited on pages 450 and 456)

[Fenner 2011] S. Fenner. Migration of Software Systems to Platform as a Service based Cloud Environments. Diploma Thesis. Kiel University, 2011. (Cited on pages 323, 331, 352, 363–367, and 377)

[Ferenc et al. 2002] R. Ferenc, A. Beszedes, M. Tarkiainen, and T. Gyimothy. Columbus - Reverse Engineering Tool and Schema for C++. In: *Proceedings of the International Conference on Software Maintenance, 2002.* 2002, pages 172–181. (Cited on page 63)

[Fittkau 2012] F. Fittkau. Simulating Cloud Deployment Options for Software Migration Support. Master's thesis. Software Engineering Group, Kiel University, Kiel, Germany, 2012. (Cited on pages 15, 238, 251–254, 257, 393, and 539)

[Fittkau et al. 2012a] F. Fittkau, S. Frey, and W. Hasselbring. CDOSim: Simulating Cloud Deployment Options for Software Migration Support. In: *IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2012*. Riva del Garda, Italy, 2012, pages 37–46. (Cited on pages 20, 167, 235, 236, 238, 251, 252, 254, 256–258, 293, and 539)

[Fittkau et al. 2012b] F. Fittkau, S. Frey, and W. Hasselbring. Cloud User-Centric Enhancements of the Simulator CloudSim to Improve Cloud Deployment Option Analysis. In: *Proceedings of the European Conference on Service-Oriented and Cloud Computing (ESOCC)*. Edited by F. De Paoli, E. Pimentel, and G. Zavattaro. Volume 7592. Lecture Notes in Computer Science. Bertinoro, Italy: Springer Berlin / Heidelberg, 2012, pages 200–207. (Cited on pages 20, 236, 251, 257, 293, and 539)

[Fleischer 2003] M. Fleischer. The Measure of Pareto Optima Applications to Multi-objective Metaheuristics. In: *Evolutionary Multi-Criterion Optimization*. Edited by C. Fonseca, P. Fleming, E. Zitzler, L. Thiele, and K. Deb. Volume 2632. Lecture Notes in Computer Science. 10.1007/3-540-36970-8_37. Springer Berlin / Heidelberg, 2003, pages 74–74. (Cited on page 389)

[Flyvbjerg 2011] B. Flyvbjerg. Case Study. In: *The Sage Handbook of Qualitative Research*. Edited by N. K. Denzin and Y. S. Lincoln. 4th. Sage Publications, 2011, pages 301–306. (Cited on page 107)

[Folkerts et al. 2013] E. Folkerts, A. Alexandrov, K. Sachs, A. Iosup, V. Markl, and C. Tosun. Benchmarking in the Cloud: What It Should, Can, and Cannot Be. In: *Selected Topics in Performance Evaluation and Benchmarking*. Edited by R. Nambiar and M. Poess. Volume 7755. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pages 173–188. (Cited on page 42)

[Fonseca et al. 2006] C. Fonseca, L. Paquete, and M. Lopez-Ibanez. An Improved Dimension-Sweep Algorithm for the Hypervolume Indicator. In: *IEEE Congress on Evolutionary Computation, 2006. CEC 2006.* 2006, pages 1157–1163. (Cited on pages 388, 389)

[Foster et al. 2008] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. In: *Grid Computing Environments Workshop, 2008. GCE '08*. 2008, pages 1–10. (Cited on pages 35, 37, 38)

[Fourer et al. 2002] R. Fourer, D. M. Gay, and B. W. Kernighan. AMPL: A Modeling Language for Mathematical Programming. 2nd edition. Duxbury Press, Nov. 2002. (Cited on page 495)

[Frakes and Kang 2005] W. Frakes and K. Kang. Software Reuse Research: Status and Future. *IEEE Transactions on Software Engineering* 31.7 (2005), pages 529 –536. (Cited on page 58)

[Freeman and Munro 1992] R. M. Freeman and M. Munro. Redocumentation for the Maintenance of Software. In: *Proceedings of the 30th Annual Southeast Regional Conference*. ACM-SE 30. Raleigh, North Carolina: ACM, 1992, pages 413–416. (Cited on page 56)

[Frey and Hasselbring 2010a] S. Frey and W. Hasselbring. Model-Based Migration of Legacy Software Systems into the Cloud: The CloudMIG Approach. In: *Softwaretechnik-Trends*. Volume 30. 2. (Proc. WSR 2010). 2010, pages 84–85. (Cited on page 18)

[Frey and Hasselbring 2010b] S. Frey and W. Hasselbring. Model-Based Migration of Legacy Software Systems to Scalable and Resource-Efficient Cloud-Based Applications: The CloudMIG Approach. In: *Proceedings of the First International Conference on Cloud Computing, GRIDs, and Virtualization (Cloud Computing 2010)*. Lisbon, Portugal, Nov. 2010, pages 155–158. (Cited on page 18)

[Frey and Hasselbring 2011a] S. Frey and W. Hasselbring. An Extensible Architecture for Detecting Violations of a Cloud Environment's Constraints During Legacy Software System Migration. In: *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR 2011)*. Edited by T. Mens, Y. Kanellopoulos, and A. Winter. Oldenburg, Germany: IEEE Computer Society, 2011, pages 269–278. (Cited on pages 19, 182, and 212)

[Frey and Hasselbring 2011b] S. Frey and W. Hasselbring. The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications. *International Journal on Advances in Software* 4.3 and 4 (2011), pages 342–353. (Cited on pages 19, 131–134, 147, 159, and 227)

[Frey et al. 2011] S. Frey, A. van Hoorn, R. Jung, W. Hasselbring, and B. Kiel. MAMBA: A Measurement Architecture for Model-Based Analysis. Technical report TR-1112. Department of Computer Science, Kiel University, Germany, Dec. 2011. (Cited on pages 23, 78, 79, 212, 309, and 311)

[Frey et al. 2012] S. Frey, A. van Hoorn, R. Jung, B. Kiel, and W. Hasselbring. MAMBA: Model-Based Software Analysis Utilizing OMG's SMM. In: *Proceedings of the 14. Workshop Software-Reengineering (WSR '12)*. May 2012, pages 37–38. (Cited on pages 23, 212, 309, 311, and 515)

[Frey et al. 2013a] S. Frey, W. Hasselbring, and B. Schnoor. Automatic conformance checking for migrating software systems to cloud infrastructures and platforms. *Journal of Software: Evolution and Process* 25.10

(2013), pages 1089–1115. (Cited on pages 20, 171, 173, 174, 182, 205, 212, 214, 220, 222, 309, 310, 331, 351, 355–358, 360, 362, 363)

[Frey et al. 2013b] S. Frey, F. Fittkau, and W. Hasselbring. Search-Based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud. In: *Proceedings of the 2013 International Conference on Software Engineering (ICSE 2013)*. ICSE '13. San Francisco, CA, USA: IEEE Press, 2013, pages 512–521. (Cited on pages 21, 396, 402, 403, and 407)

[Fu 2002] M. C. Fu. Optimization for Simulation: Theory vs. Practice. *INFORMS Journal on Computing* 14.3 (2002), pages 192–215. (Cited on page 100)

[Fuhr et al. 2010] A. Fuhr, T. Horn, and A. Winter. Model-Driven Software Migration. In: *Software Engineering 2010: Fachtagung des GI-Fachbereichs Softwaretechnik 22.-26.02. 2010 in Paderborn*. Edited by G. Engels, M. Luckey, and W. Schäfer. Volume P-159. Bonn: Gesellschaft für Informatik, 2010, pages 69–80. (Cited on page 65)

[Galán et al. 2009] F. Galán, A. Sampaio, L. Rodero-Merino, I. Loy, V. Gil, and L. M. Vaquero. Service Specification in Cloud Environments Based on Extensions to Open Standards. In: *Proceedings of the Fourth International ICST Conference on Communication System Software and Middleware*. COMSWARE '09. Dublin, Ireland: ACM, 2009, 19:1–19:12. (Cited on pages 43 and 236)

[Galante and de Bona 2012] G. Galante and L. de Bona. A Survey on Cloud Computing Elasticity. In: *IEEE Fifth International Conference on Utility and Cloud Computing (UCC), 2012*. 2012, pages 263–270. (Cited on pages 36 and 43)

[Gao and Wang 2008] J. Gao and Q. Wang. Simulation-based Optimization Method for Three-echelon Network Inventory System of a Supply Chain. In: *Chinese Control and Decision Conference, 2008. CCDC 2008.* 2008, pages 2406–2410. (Cited on page 101)

[Garlan et al. 1995] D. Garlan, R. Allen, and J. Ockerbloom. Architectural Mismatch: Why reuse is so hard. *Software, IEEE* 12.6 (1995), pages 17 –26. (Cited on pages 466, 469, 470)

[Gartner, Inc. 2013] Gartner, Inc. Forecast Overview: Public Cloud Services, Worldwide, 2011-2016, 4Q12 Update. 2013. URL: http://www.gartner.com/id=2332215. (Cited on page 35)

[Gendreau and Potvin 2012] M. Gendreau and J.-Y. Potvin, editors. Handbook of Metaheuristics (International Series in Operations Research & Management Science). 2nd ed. 2010. Springer, Nov. 2012. (Cited on pages 86, 87)

[Gentry 2010] C. Gentry. Computing Arbitrary Functions of Encrypted Data. *Commun. ACM* 53.3 (2010), pages 97–105. (Cited on page 127)

[Gerevini and Serina 2002] A. Gerevini and I. Serina. LPG: A Planner Based on Local Search for Planning Graphs with Action Costs. In: *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*. 2002, pages 13–22. (Cited on page 482)

[Getoor and Taskar 2007] L. Getoor and B. Taskar. Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning series). The MIT Press, 2007, page 586. (Cited on pages 326, 327)

[Ghallab et al. 2004] M. Ghallab, D. Nau, and P. Traverso. Automated Planning: Theory & Practice (The Morgan Kaufmann Series in Artificial Intelligence). 1st edition. Morgan Kaufmann, May 2004. (Cited on page 482)

[Giesecke 2008] S. Giesecke. Architectural Styles for Early Goal-driven Middleware Platform Selection. PhD thesis. Department of Computer Science, Carl von Ossietzky University of Oldenburg, Germany, 2008, pages 1–279. (Cited on page 109)

[Godfrey and German 2008] M. Godfrey and D. German. The Past, Present, and Future of Software Evolution. In: *Frontiers of Software Maintenance, 2008. FoSM 2008.* 2008, pages 129–138. (Cited on page 53)

[Goldberg 1989] D. E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. 1st edition. Addison-Wesley Professional, Jan. 1989. (Cited on page 93)

[Goncalves 2010] A. Goncalves. Beginning Java EE 6 with GlassFish 3 (Expert's Voice in Java Technology). 2nd edition. Apress, Aug. 2010. (Cited on page 334)

[Gonzalez-Perez and Henderson-Sellers 2008] C. Gonzalez-Perez and B. Henderson-Sellers. Metamodelling for Software Engineering. Wiley, 2008. (Cited on page 107)

[Goyal 2010] P. Goyal. Enterprise Usability of Cloud Computing Environments: Issues and Challenges. In: *19th IEEE International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010*. 2010, pages 54 –59. (Cited on page 123)

[Grosskopf et al. 2009] A. Grosskopf, G. Decker, and M. Weske. The Process: Business Process Modeling using BPMN. 1st edition. Meghan-Kiffer Press, Feb. 2009. (Cited on page 74)

[Grossman 2009] R. Grossman. The Case for Cloud Computing. *IT Professional* 11.2 (2009), pages 23–27. (Cited on pages 35 and 40)

[Group 2012] T. O. Group. SOA Reference Architecture Technical Standard. online. 2012. URL: http://www.opengroup.org/soa/source-book/soa_refarch/index.htm. (Cited on page 443)

[Grundy et al. 2012] J. Grundy, G. Kaefer, J. Keong, and A. Liu. Guest Editors' Introduction: Software Engineering for the Cloud. *IEEE Software* 29 (2012), pages 26–29. (Cited on pages 6, 10, and 236)

[Gruschka and Jensen 2010] N. Gruschka and M. Jensen. Attack Surfaces: A Taxonomy for Attacks on Cloud Services. In: *IEEE 3rd International Conference on Cloud Computing (CLOUD), 2010*. 2010, pages 276 –279. (Cited on page 41)

[Guikema et al. 2004] S. Guikema, R. Davidson, and Z. Cagnan. Efficient simulation-based discrete optimization. In: *Proceedings of the 2004 Winter Simulation Conference, 2004.* Volume 1. 2004, pages –544. (Cited on page 100)

[Guillén et al. 2013] J. Guillén, J. Miranda, J. M. Murillo, and C. Canal. A service-oriented framework for developing cross cloud migratable software. *Journal of Systems and Software* 0 (2013). (Cited on pages 450 and 457)

[Hajjat et al. 2010] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani. Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud. In: *Proceedings of the ACM SIGCOMM 2010 Conference on SIGCOMM*. SIGCOMM '10. New Delhi, India: ACM, 2010, pages 243–254. (Cited on pages 137, 414, and 419)

[Hale et al. 1990] D. Hale, D. Haworth, and S. Sharpe. Empirical Software Maintenance Studies During the 1980s. In: *Proceedings of the Conference on Software Maintenance, 1990*. 1990, pages 118–123. (Cited on page 53)

[Hall et al. 2005] G. A. Hall, W. Tao, and J. C. Munson. Measurement and Validation of Module Coupling Attributes. *Software Quality Journal* 13 (3 2005), pages 281–296. (Cited on page 373)

[Hamdaqa et al. 2011] M. Hamdaqa, T. Livogiannis, and L. Tahvildari. A Reference Model for Developing Cloud Applications. In: *CLOSER*. Edited by F. Leymann, I. Ivanov, M. van Sinderen, and B. Shishkov. SciTePress, 2011, pages 98–103. (Cited on pages 450 and 458)

[Hansen and Oleshchuk 2005] F. Hansen and V. Oleshchuk. Conformance Checking of RBAC Policy and its Implementation. In: *Information Security Practice and Experience*. Edited by R. Deng, F. Bao, H. Pang, and J. Zhou. Volume 3439. Lecture Notes in Computer Science. 10.1007/978-3-540-31979-5_13. Springer Berlin / Heidelberg, 2005, pages 144–155. (Cited on page 463)

[Hansen et al. 2011] K. M. Hansen, K. Jonasson, and H. Neukirchen. An empirical study of software architectures' effect on product quality. *Journal of Systems and Software* 84.7 (2011), pages 1233–1243. (Cited on page 117)

[Hansen and Jaszkiewicz 1998] M. P. Hansen and A. Jaszkiewicz. Evaluating the quality of approximations to the non-dominated set. Technical report IMM-REP-1998-7. Technical University of Denmark, Institute of Mathematical Modelling, 1998. (Cited on page 386)

[Harman 2007] M. Harman. The Current State and Future of Search Based Software Engineering. In: *Future of Software Engineering, 2007. FOSE '07*. 2007, pages 342–357. (Cited on pages 15, 81, 83–86, 92, 93)

[Harman 2011] M. Harman. Software Engineering Meets Evolutionary Computation. *Computer* 44.10 (2011), pages 31–39. (Cited on pages 81, 82, 93, 94, 237, and 239)

[Harman and Clark 2004] M. Harman and J. Clark. Metrics Are Fitness Functions Too. In: *Proceedings of the 10th International Symposium on Software Metrics, 2004.* 2004, pages 58–69. (Cited on pages 83 and 518)

[Harman and Jones 2001] M. Harman and B. F. Jones. Search-based software engineering. *Information and Software Technology* 43.14 (2001), pages 833–839. (Cited on page 81)

[Harman et al. 2009] M. Harman, S. A. Mansouri, and Y. Zhang. Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications. Technical report TR-09-03. Department of Computer Science, King's College London, 2009. (Cited on pages 81–83)

[Harman et al. 2012a] M. Harman, K. Lakhotia, J. Singer, D. R. White, and S. Yoo. Cloud engineering is Search Based Software Engineering too. *Journal of Systems and Software* 0 (2012), pages –. (Cited on page 15)

[Harman et al. 2012b] M. Harman, S. A. Mansouri, and Y. Zhang. Search-Based Software Engineering: Trends, Techniques and Applications. *ACM Computing Surveys* 45.1 (Dec. 2012), 11:1–11:61. (Cited on pages 81–83, 86, 91, 92)

[Harmer et al. 2009] T. Harmer, P. Wright, C. Cunningham, and R. Perrott. Provider-Independent Use of the Cloud. In: *Euro-Par 2009 Parallel Processing*. Edited by H. Sips, D. Epema, and H.-X. Lin. Volume 5704. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, pages 454–465. (Cited on page 439)

[Hasselbring et al. 2004] W. Hasselbring, R. Reussner, H. Jaekel, J. Schlegelmilch, T. Teschke, and S. Krieghoff. The Dublo Architecture Pattern for Smooth Migration of Business Information Systems: An Experience Report. In: *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2004, pages 117–126. (Cited on page 69)

[Hasselbring et al. 2008] W. Hasselbring, A. Büdenbender, S. Grasmann, S. Krieghoff, and J. März. Muster zur Migration betrieblicher Informationssysteme. In: *Software Engineering*. Edited by K. Herrmann and B. Brügge. Volume 121. LNI. GI, 2008, pages 80–84. (Cited on page 69)

[Haupt and Haupt 2004] R. L. Haupt and S. E. Haupt. Practical Genetic Algorithms. 2nd edition. Wiley-Interscience, May 2004, page 272. (Cited on page 93)

[Hayes 2008] B. Hayes. Cloud Computing. *Communications of the ACM* 51 (7 2008), pages 9–11. (Cited on page 35)

[Haziza et al. 1992] M. Haziza, J.-F. Voidrot, E. Minor, L. Pofelski, and S. Blazy. Software Maintenance: An Analysis of Industrial Needs and Constraints. In: *Proceerdings of the Conference on Software Maintenance, 1992.* 1992, pages 18–26. (Cited on page 53)

[Helvik and Wittner 2001] B. Helvik and O. Wittner. Using the Cross-Entropy Method to Guide/Govern Mobile Agent's Path Finding in Networks. In: *Mobile Agents for Telecommunication Applications*. Edited by S. Pierre and R. Glitho. Volume 2164. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2001, pages 255–268. (Cited on page 487)

[Heričko et al. 2008] M. Heričko, A. Živkovič, and I. Rozman. An approach to optimizing software development team size. *Information Processing Letters* 108.3 (2008), pages 101–106. (Cited on page 84)

[Hickey and Rahmouni 2010] M. Hickey and M. Rahmouni. Modelling Cloud Computing Infrastructure. In: *Mechanisms for Autonomous Management of Networks and Services*. Edited by B. Stiller and F. De Turck. Volume 6155. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, pages 106–109. (Cited on pages 440 and 442)

[Höfer and Karagiannis 2011] C. Höfer and G. Karagiannis. Cloud computing services: taxonomy and comparison. English. *Journal of Internet Services and Applications* 2.2 (2011), pages 81–94. (Cited on page 42)

[Hofmeister 1994] C. R. Hofmeister. Dynamic Reconfiguration of Distributed Applications. Technical report CS-TR-3210. Department of Computer Science, University of Maryland, 1994. (Cited on page 261)

[Holland 1975] J. H. Holland. Adaptation in Natural and Artificial Systems. 1st ed. Ann Arbor, MI: The University of Michigan Press, 1975. (Cited on page 93)

[Holt and Pak 1996] R. Holt and J. Pak. GASE: Visualizing Software Evolution-in-the-Large. In: *Proceedings of the Third Working Conference on Reverse Engineering, 1996*. 1996, pages 163 –167. (Cited on page 62)

[Hong and Nelson 2009] L. J. Hong and B. L. Nelson. A Brief Introduction to Optimization via Simulation. In: *Winter Simulation Conference*. WSC '09. Austin, Texas: Winter Simulation Conference, 2009, pages 75–85. (Cited on page 100)

[Hu et al. 2007] V. Hu, E. Martin, J. Hwang, and T. Xie. Conformance Checking of Access Control Policies Specified in XACML. In: *31st Annual International Computer Software and Applications Conference, 2007. COMPSAC 2007.* Volume 2. 2007, pages 275–280. (Cited on page 463)

[Huber et al. 2012] N. Huber, A. van Hoorn, A. Koziolek, F. Brosig, and S. Kounev. S/T/A: Meta-modeling Run-time Reconfiguration in Component-Based System Architectures. In: *Proceedings of the 9th IEEE International Conference on e-Business Engineering (ICEBE 2012)*. Accepted for publication. 2012. (Cited on page 43)

[Huebscher and McCann 2008] M. C. Huebscher and J. A. McCann. A survey of Autonomic Computing-Degrees, Models, and Applications. *ACM Computing Surveys* 40.3 (Aug. 2008), 7:1–7:28. (Cited on page 245)

[Hunter and Crawford 2001] J. Hunter and W. Crawford. Java Servlet Programming (Java Series). Second Edition. O'Reilly Media, Apr. 2001. (Cited on pages 189 and 336)

[Iachan 1982] R. Iachan. Systematic Sampling: A Critical Review. *International Statistical Review / Revue Internationale de Statistique* 50.3 (1982), pages 293–303. (Cited on pages 383, 384)

[IBM 2003] C. IBM. An architectural blueprint for autonomic computing. Technical report. 2003. (Cited on page 245)

[IEEE 1998] IEEE. IEEE Standard for Software Maintenance. *IEEE Std 1219-1998* (1998). (Cited on page 53)

[Iosup et al. 2011] A. Iosup, N. Yigitbasi, and D. Epema. On the Performance Variability of Production Cloud Services. In: *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2011*. 2011, pages 104 –113. (Cited on page 41)

[Iqbal et al. 2009] W. Iqbal, M. Dailey, and D. Carrera. SLA-Driven Adaptive Resource Management for Web Applications on a Heterogeneous Compute Cloud. In: *CloudCom*. Edited by M. G. Jaatun, G. Zhao, and C. Rong. Volume 5931. Lecture Notes in Computer Science. Springer, 2009, pages 243–253. (Cited on page 130)

[Ishibuchi and Murata 1999] H. Ishibuchi and T. Murata. Local Search Procedures in a Multi-Objective Genetic Local Search Algorithm for Scheduling Problems. In: *IEEE International Conference on Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999*. Volume 1. 1999, 665 –670 vol.1. (Cited on pages 99 and 274)

[Ishibuchi et al. 2003] H. Ishibuchi, T. Yoshida, and T. Murata. Balance Between Genetic Search and Local Search in Memetic Algorithms for Multiobjective Permutation Flowshop Scheduling. *IEEE Transactions on Evolutionary Computation* 7.2 (2003), pages 204–223. (Cited on page 274)

[Islam et al. 2012] S. Islam, K. Lee, A. Fekete, and A. Liu. How A Consumer Can Measure Elasticity for Cloud Platforms. In: *Proceedings of the third joint WOSP/SIPEW International Conference on Performance Engineering*. ICPE '12. Boston, Massachusetts, USA: ACM, 2012, pages 85–96. (Cited on pages 36, 233, 234)

[ISO/IEC 2001] ISO/IEC. ISO/IEC 9126-1. Software engineering – Product quality. ISO/IEC, 2001. (Cited on pages 16 and 287)

[ISO/IEC 2008] ISO/IEC. ISO/IEC 27002:2005 - Information technology – Security techniques – Code of practice for information security management. ISO/IEC, 2008. (Cited on page 479)

[ISO/IEC/IEEE 2006] ISO/IEC/IEEE. International Standard - ISO/IEC 14764 IEEE Std 14764-2006 Software Engineering - Software Life Cycle Processes - Maintenance. 2006. (Cited on pages 70–72, 138, and 416)

[Itsykson and Zozulya 2011] V. Itsykson and A. Zozulya. Automated Program Transformation for Migration to New Libraries. In: *7th Central and Eastern European Software Engineering Conference in Russia (CEE-SECR), 2011*. 2011, pages 1–7. (Cited on pages 466 and 470)

[Jahnke et al. 1996] J. Jahnke, W. Schafer, and A. Zundorf. A Design Environment for Migrating Relational to Object Oriented Database Systems. In: *Proceedings of the International Conference on Software Maintenance 1996*. 1996, pages 163 –170. (Cited on page 66)

[Jain et al. 2000] A. Jain, R. Duin, and J. Mao. Statistical Pattern Recognition: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.1 (2000), pages 4–37. (Cited on page 198)

[Jenkins et al. 2011] W. Jenkins, S. Vilkomir, P. Sharma, and G. Pirocanac. Framework for Testing Cloud Platforms and Infrastructures. In: *International Conference on Cloud and Service Computing (CSC), 2011*. 2011, pages 134 –140. (Cited on pages 473, 476, 477)

[Jensen 2003] M. Jensen. Reducing the Run-Time Complexity of Multiobjective EAs: The NSGA-II and Other Algorithms. *IEEE Transactions on Evolutionary Computation* 7.5 (2003), pages 503–515. (Cited on page 96)

[Jesson et al. 2011] J. Jesson, L. Matheson, and F. M. Lacey. Doing Your Literature Review: Traditional and Systematic Techniques. London: SAGE, 2011. (Cited on page 107)

[Jeusfeld et al. 2009] M. A. Jeusfeld, M. Jarke, and J. Mylopoulos. Metamodeling for Method Engineering. The MIT Press, 2009. (Cited on page 107)

[Jiang et al. 2011] J. Jiang, J. Lu, and G. Zhang. An Innovative Self-Adaptive Configuration Optimization System in Cloud Computing. In: *IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC), 2011*. 2011, pages 621–627. (Cited on pages 499, 500, and 502)

[Jones et al. 2002] D. Jones, S. Mirrazavi, and M Tamiz. Multi-objective meta-heuristics: An overview of the current state-of-the-art. *European Journal of Operational Research* 137.1 (2002), pages 1–9. (Cited on page 86)

[Jr. 2011] S. O. Jr. The Problem with Cloud-Computing Standardization. *Computer* 44 (2011), pages 13–16. (Cited on page 8)

[Jung et al. 2008] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu. Generating Adaptation Policies for Multi-tier Applications in Consolidated Server Environments. In: *International Conference on Autonomic Computing, 2008. ICAC '08.* 2008, pages 23–32. (Cited on pages 483 and 488)

[Jung and Saglietti 2005] M. Jung and F. Saglietti. Supporting Component and Architectural Re-usage by Detection and Tolerance of Integration Faults. In: *Ninth IEEE International Symposium on High-Assurance Systems Engineering, 2005. HASE 2005.* 2005, pages 47 –55. (Cited on pages 466 and 470)

[Kaisler and Money 2011] S. Kaisler and W. Money. Service Migration in a Cloud Architecture. In: *44th Hawaii International Conference on System Sciences (HICSS), 2011.* 2011, pages 1 –10. (Cited on pages 123, 143, 414, and 420)

[Kan 2002] S. H. Kan. Metrics and Models in Software Quality Engineering (2nd Edition). 2nd edition. Addison-Wesley Professional, Sept. 2002. (Cited on pages 77, 84, 167, 229, and 324)

[Kappel et al. 2011] G. Kappel, Z. Maamar, and H. R. Motahari Nezhad, editors. Service Oriented Computing. Berlin Heidelberg: Springer, LNCS, 2011. (Cited on page 38)

[Kazman et al. 1998] R. Kazman, S. G. Woods, and S. J. Carrière. Requirements for Integrating Software Architecture and Reengineering Models: CORUM II. In: *WCRE '98: Proceedings of the Working Conference on Reverse Engineering (WCRE'98).* Washington, DC, USA: IEEE Computer Society, 1998, page 154. (Cited on pages 56, 58, 59)

[Kemerer and Slaughter 1999] C. Kemerer and S. Slaughter. An Empirical Approach to Studying Software Evolution. *IEEE Transactions on Software Engineering* 25.4 (1999), pages 493–509. (Cited on page 53)

[Kennedy and Eberhart 1995] J. Kennedy and R. Eberhart. Particle Swarm Optimization. In: *In Proceedings IEEE International Conference on Neural Networks, 1995.* Volume 4. 1995, pages 1942–1948. (Cited on page 501)

[Khajeh-Hosseini et al. 2010] A. Khajeh-Hosseini, D. Greenwood, and I. Sommerville. Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS. *CoRR* abs/1002.3492 (2010). (Cited on pages 2 and 137)

[Khajeh-Hosseini et al. 2011] A. Khajeh-Hosseini, D. Greenwood, J. W. Smith, and I. Sommerville. The Cloud Adoption Toolkit: supporting cloud adoption decisions in the enterprise. *Software: Practice and Experience* (2011). (Cited on pages 431–433)

[Kichkaylo and Karamcheti 2004] T. Kichkaylo and V. Karamcheti. Optimal Resource-Aware Deployment Planning for Component-based Distributed Applications. In: *Proceedings of the 13th IEEE International Symposium on High performance Distributed Computing, 2004.* 2004, pages 150–159. (Cited on pages 483, 488, 489)

[Kichkaylo et al. 2003] T. Kichkaylo, A. Ivan, and V. Karamcheti. Constrained Component Deployment in Wide-Area Networks Using AI Planning Techniques. In: *Proceedings of the International Parallel and Distributed Processing Symposium, 2003.* 2003. (Cited on page 489)

[Kiczales et al. 2001] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An Overview of AspectJ. In: *Proceedings of the 15th European Conference on Object-Oriented Programming*. ECOOP '01. London, UK, UK: Springer-Verlag, 2001, pages 327–353. (Cited on page 210)

[Kim et al. 2009] W. Kim, S. D. Kim, E. Lee, and S. Lee. Adoption Issues for Cloud Computing. In: *Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services*. iiWAS '09. Kuala Lumpur, Malaysia: ACM, 2009, pages 3–6. (Cited on pages 137, 431, 433, 434)

[King and Ganti 2010] T. King and A. Ganti. Migrating Autonomic Self-Testing to the Cloud. In: *Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW), 2010*. 2010, pages 438–443. (Cited on page 2)

[Knowles and Corne 2000] J. D. Knowles and D. W. Corne. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evol. Comput.* 8.2 (June 2000), pages 149–172. (Cited on page 386)

[Kolda et al. 2003] T. G. Kolda, R. M. Lewis, and V. Torczon. Optimization by Direct Search: New Perspectives on Some Classical and Modern Methods. *SIAM Review* 45 (2003), pages 385–482. (Cited on page 382)

[Konak et al. 2006] A. Konak, D. W. Coit, and A. E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering and System Safety* 91.9 (2006), pages 992–1007. (Cited on pages 93, 96, 97)

[Koschke 2009] R. Koschke. Software Engineering: International Summer Schools, ISSSE 2006-2008, Salerno, Italy, Revised Tutorial Lectures. Architecture Reconstruction (2009), pages 140–173. (Cited on page 467)

[Kourtesis and Paraskakis 2011] D. Kourtesis and I. Paraskakis. Governance in Cloud Platforms for the Development and Deployment of Enterprise Applications. In: *3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2011)*. 2011. (Cited on pages 473 and 477)

[Kourtesis et al. 2011] D. Kourtesis, V. Kuttruff, and I. Paraskakis. Optimising Development and Deployment of Enterprise Software Applications on PaaS: The CAST Project. In: *Towards a Service-Based Internet. ServiceWave 2010 Workshops*. Edited by M. Cezon and Y. Wolfsthal. Volume 6569. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2011, pages 14–25. (Cited on page 477)

[Koza 1995] J. R. Koza. Survey of Genetic Algorithms and Genetic Programming. In: *WESCON/'95. Conference record. 'Microelectronics Communications Technology Producing Quality Products Mobile and Portable Power Emerging Technologies'*. 1995, pages 589–. (Cited on page 93)

[Koziolek 2010] H. Koziolek. Towards an Architectural Style for Multi-tenant Software Applications. In: *Software Engineering*. Edited by G. Engels, M. Luckey, and W. Schäfer. Volume 159. LNI. GI, 2010, pages 81–92. (Cited on pages 450, 458, 459)

[Kruchten 1995] P. Kruchten. The 4+1 View Model of Architecture. *IEEE Software* 12.6 (1995), pages 42–50. (Cited on pages 158 and 480)

[Krueger 1992] C. W. Krueger. Software Reuse. *ACM Computing Surveys* 24 (2 1992), pages 131–183. (Cited on page 58)

[Kund 2013] S. Kund. Design and Implementation of an Eclipse P2-based Online Repository for Exchanging Cloud Profiles. Bachelor's Thesis. Kiel University, 2013. (Cited on page 296)

[Kuperberg et al. 2011] M. Kuperberg, N. Herbst, J. v. Kistowski, and R. Reussner. Defining and Quantifying Elasticity of Resources in Cloud Computing and Scalable Platforms. 2011. (Cited on pages 36 and 236)

[Lanza and Ducasse 2003] M. Lanza and S. Ducasse. Polymetric Views - A Lightweight Visual Approach to Reverse Engineering. *IEEE Transactions on Software Engineering* 29.9 (2003), pages 782 –795. (Cited on page 62)

[Lanza 2001] M. Lanza. The Evolution Matrix: Recovering Software Evolution using Software Visualization Techniques. In: *Proceedings of the 4th International Workshop on Principles of Software Evolution*. IWPSE '01. Vienna, Austria: ACM, 2001, pages 37–42. (Cited on page 62)

[Laszewski and Nauduri 2011] T. Laszewski and P. Nauduri. Migrating to the Cloud: Oracle Client/Server Modernization. 1st edition. Syngress, Oct. 2011. (Cited on pages 414 and 420)

[Law and McComas 2000] A. Law and M. McComas. Simulation-based Optimization. In: *Proceedings of the Winter Simulation Conference, 2000*. Volume 1. 2000, pages 46–49. (Cited on pages 15 and 100)

[Law and Szeto 2007] N. L. Law and K. Y. Szeto. Adaptive Genetic Algorithm with Mutation and Crossover Matrices. In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. IJCAI'07. Hyderabad, India: Morgan Kaufmann Publishers Inc., 2007, pages 2330–2333. (Cited on pages 99 and 273)

[Leavitt 2009] N. Leavitt. Is Cloud Computing Really Ready for Prime Time? *Computer* 42.1 (2009), pages 15–20. (Cited on page 41)

[Lee et al. 2010] C. Lee, J. Suzuki, A. Vasilakos, Y. Yamamoto, and K. Oba. An Evolutionary Game Theoretic Approach to Adaptive and Stable Application Deployment in Clouds. In: *Proceeding of the 2nd Workshop on Bio-inspired Algorithms for Distributed Systems*. BADS '10. Washington, DC, USA: ACM, 2010, pages 29–38. (Cited on page 499)

[Lee et al. 2003] E. Lee, B. Lee, W. Shin, and C. Wu. A Reengineering Process for Migrating from an Object-oriented Legacy System to a Component-based System. In: *Proceedings of the 27th Annual International Computer Software and Applications Conference, 2003. COMPSAC 2003*. 2003, pages 336 –341. (Cited on page 65)

[Lehman 1980] M. Lehman. Programs, Life Cycles, and Laws of Software Evolution. *Proceedings of the IEEE* 68.9 (1980), pages 1060 –1076. (Cited on pages 9, 54, and 56)

[Leiner et al. 1997] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. S. Wolff. The Past and Future History of the Internet. *Communications of the ACM* 40.2 (Feb. 1997), pages 102–108. (Cited on page 35)

[Leitner et al. 2011] P. Leitner, W. Hummer, and S. Dustdar. Cost-Based Optimization of Service Compositions. *IEEE Transactions on Services Computing* PP.99 (2011), page 1. (Cited on page 10)

[Leitner et al. 2012] P. Leitner, B. Satzger, W. Hummer, C. Inzinger, and S. Dustdar. CloudScale: a Novel Middleware for Building Transparently Scaling Cloud Applications. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. SAC '12. Trento, Italy: ACM, 2012, pages 434–440. (Cited on page 517)

[Leymann 2009] F. Leymann. Cloud Computing: The Next Revolution in IT. Englisch. In: *Proc. 52th Photogrammetric Week*. Wichmann Verlag, 2009, pages 3–12. (Cited on pages 3 and 39)

[Leymann et al. 2011] F. Leymann, C. Fehling, R. Mietzner, A. Nowak, and S. Dustdar. Moving Applications to the Cloud: An Approach based on Application Model Enrichment. Englisch. *International Journal of Cooperative Information Systems (IJCIS)* 20.3 (2011), pages 307–356. (Cited on pages 414 and 421)

[Li et al. 2010a] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: Comparing Public Cloud Providers. In: *Proceedings of the 10th annual conference on Internet measurement*. IMC '10. Melbourne, Australia: ACM, 2010, pages 1–14. (Cited on page 41)

[Li and Henry 1993] W. Li and S. Henry. Object-oriented metrics that predict maintainability. *Journal of Systems and Software* 23.2 (1993), pages 111–122. (Cited on page 288)

[Li et al. 2010b] X. Li, H. Hoover, and P. Rudnicki. API Conformance Verification for Java Programs. In: *Formal Methods and Software Engineering*. Edited by J. Dong and H. Zhu. Volume 6447. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, pages 188–203. (Cited on page 462)

[Liew and Su 2012] S. H. Liew and Y.-Y. Su. CloudGuide: Helping Users Estimate Cloud Deployment Cost and Performance for Legacy Web Applications. In: *IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom), 2012*. 2012, pages 90–98. (Cited on pages 491–493)

[Lin and Miller 2004] B. Lin and D. Miller. Tabu search algorithm for chemical process optimization. *Computers & Chemical Engineering* 28.11 (2004), pages 2287 –2306. (Cited on page 81)

[Liu et al. 2011] F. Liu, J. Tong, R. B. Mao J.and Bohn, J. V. Messina, M. L. Badger, and D. M. Leaf. NIST Cloud Computing Reference Architecture. NIST Special Publication 500-292. 2011. (Cited on pages 48–50, and 440)

[Liu et al. 2012] J. Liu, L.-J. Zhang, B. Hu, and K. He. CCRA: Cloud Computing Reference Architecture. In: *IEEE Ninth International Conference on Services Computing (SCC), 2012*. 2012, pages 657–665. (Cited on pages 440, 443, 444)

[Loebbecke et al. 2011] C. Loebbecke, B. Thomas, and T. Ullrich. Assessing Cloud Readiness: Introducing the Magic Matrices Method Used by Continental AG. In: *Governance and Sustainability in Information Systems. Managing the Transfer and Diffusion of IT*. Edited by M. Nüttgens, A. Gadatsch, K. Kautz, I. Schirmer, and N. Blinn. Volume 366. IFIP Advances in Information and Communication Technology. Springer Berlin Heidelberg, 2011, pages 270–281. (Cited on pages 431, 434, 435)

[Löffler 2011] P. Löffler. Migration von Softwaresystemen auf IaaS-basierte Cloud Umgebungen. Bachelor's Thesis. Kiel University, Kiel, Germany, 2011. (Cited on page 36)

[Lucas Simarro et al. 2011] J. Lucas Simarro, R. Moreno-Vozmediano, R. Montero, and I. Llorente. Dynamic Placement of Virtual Machines for Cost Optimization in Multi-Cloud Environments. In: *International Conference on High Performance Computing and Simulation (HPCS), 2011*. 2011, pages 1–7. (Cited on pages 491, 494, 495)

[Lukasiewycz et al. 2011] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich. Opt4J: A Modular Framework for Meta-Heuristic Optimization. In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. GECCO '11. Dublin, Ireland: ACM, 2011, pages 1723–1730. (Cited on page 315)

[Luke 2011] S. Luke. Essentials of Metaheuristics. lulu.com, Mar. 2011. (Cited on pages 81, 84–88)

[Luo 2010] Y. Luo. Network I/O Virtualization for Cloud Computing. *IT Professional* 12.5 (2010), pages 36–41. (Cited on page 43)

[Ma et al. 2007] Q. Ma, Y. Li, K. Sun, and L. Liu. Model-Based Dependency Management for Migrating Service Hosting Environment. In: *Services Computing, 2007. SCC 2007. IEEE International Conference on*. 2007, pages 356 –363. (Cited on pages 466 and 471)

[MacDonald and Freeman 2010] M. MacDonald and A. Freeman. Pro ASP.NET 4 in C# 2010. 4th edition. Apress, June 2010. (Cited on page 337)

[Malek et al. 2012] S. Malek, N. Medvidovic, and M. Mikic-Rakic. An Extensible Framework for Improving a Distributed Software System's Deployment Architecture. *IEEE Transactions on Software Engineering* 38.1 (2012), pages 73–100. (Cited on pages 483 and 489)

[Maletic et al. 2003] J. Maletic, A. Marcus, and L. Feng. Source Viewer 3D (sv3D) - A Framework for Software Visualization. In: *Proceedings. 25th International Conference on Software Engineering, 2003, (ICSE 2003)*. 2003, pages 812 –813. (Cited on page 62)

[Man et al. 2008] Z. Man, T. Wei, L. Xiang, and K. Lishan. Research on Multi-project Scheduling Problem Based on Hybrid Genetic Algorithm. In: *International Conference on Computer Science and Software Engineering, 2008*. Volume 1. 2008, pages 390–394. (Cited on pages 99 and 274)

[Mao et al. 2010] M. Mao, J. Li, and M. Humphrey. Cloud Auto-scaling with Deadline and Budget Constraints. In: *Proceedings of the 11th IEEE/ACM International Conference on Grid Computing (GRID)*. 2010, pages 41–48. (Cited on pages 491, 495, 496)

[Marler and Arora 2004] R. Marler and J. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization* 26.6 (2004), pages 369–395. (Cited on page 90)

[Marpons et al. 2008] G. Marpons, J. Mariño, M. Carro, n. Herranz, J. Moreno-Navarro, and L.-k. Fredlund. Automatic Coding Rule Conformance Checking Using Logic Programming. In: *Practical Aspects of Declarative Languages*. Edited by P. Hudak and D. Warren. Volume 4902. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, pages 18–34. (Cited on page 463)

[Marshall et al. 2012] P. Marshall, H. Tufo, and K. Keahey. Provisioning Policies for Elastic Computing Environments. In: *IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012*. 2012, pages 1085–1094. (Cited on page 10)

[Marston et al. 2011] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi. Cloud computing – the business perspective. *Decision Support Systems* 51.1 (2011), pages 176–189. (Cited on pages 35 and 123)

[Martens et al. 2010] A. Martens, H. Koziolek, S. Becker, and R. Reussner. Automatically Improve Software Architecture Models for Performance, Reliability, and Cost Using Evolutionary Algorithms. In: *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*. WOSP/SIPEW '10. San Jose, California, USA: ACM, 2010, pages 105–116. (Cited on pages 483 and 489)

[Mattoon et al. 2011] S. Mattoon, B. Hensle, and J. Baty. Cloud Computing Maturity Model - Guiding Success with Cloud Capabilities (An Oracle White Paper). Technical report. Oracle Corporation, 2011. (Cited on pages 431 and 435)

[Mayer et al. 1995] R. J. Mayer, J. W. Crump, R. Fernandes, A. Keen, and M. K. Painter. Information Integration for Concurrent Engineering (IICE) Compendium of Methods Report. Interim Technical Paper for Period February 1991 to March 1995. Air Force Materiel Command, Wright-Patterson Air Force Base, Ohio, 1995. (Cited on pages 108, 112–114)

[McAffer et al. 2010] J. McAffer, J.-M. Lemieux, and C. Aniszczyk. Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications, 2ND EDITION. Addison-Wesley Longman, Amsterdam, 2010. (Cited on pages 16 and 303)

[Mele et al. 2006] F. D. Mele, G. Guillén, A. Espuña, and L. Puigjaner. A Simulation-Based Optimization Framework for Parameter Optimization of Supply-Chain Networks. *Industrial & Engineering Chemistry Research* 45.9 (2006), pages 3133–3148. (Cited on page 101)

[Mell and Grance 2011] P. Mell and T. Grance. The NIST Definition of Cloud Computing. NIST Special Publication 800-145. 2011. (Cited on page 42)

[Melski et al. 2009] D. Melski, T. Teitelbaum, and T. Reps. Static Analysis of Software Executables. In: *Cybersecurity Applications Technology Conference For Homeland Security, 2009. CATCH '09.* 2009, pages 97–102. (Cited on page 63)

[Meng et al. 2010] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis. Efficient Resource Provisioning in Compute Clouds via VM Multiplexing. In: *Proceedings of the 7th International Conference on Autonomic Computing*. ICAC '10. Washington, DC, USA: ACM, 2010, pages 11–20. (Cited on page 43)

[Mens 2012] T. Mens. On the Complexity of Software Systems. *Computer* 45.8 (2012), pages 79–81. (Cited on page 152)

[Mens et al. 2005] T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri. Challenges in Software Evolution. In: *Eighth International Workshop on Principles of Software Evolution*. 2005, pages 13–22. (Cited on page 53)

[Menzel and Ranjan 2011] M. Menzel and R. Ranjan. CloudGenius: Automated Decision Support for Migrating Multi-Component Enterprise Applications to Clouds. *CoRR* abs/1112.3880 (2011). (Cited on pages 414 and 422)

[Michie et al. 1994] D. Michie, D. J. Spiegelhalter, and C. Taylor. Machine Learning, Neural and Statistical Classification. 1994. (Cited on page 326)

[Mietzner and Leymann 2008] R. Mietzner and F. Leymann. Towards Provisioning the Cloud: On the Usage of Multi-Granularity Flows and Services to Realize a Unified Provisioning Infrastructure for SaaS Applications. In: *IEEE Congress on Services - Part I, 2008*. 2008, pages 3–10. (Cited on page 460)

[Mietzner and Leymann 2010] R. Mietzner and F. Leymann. A Self-Service Portal for Service-Based Applications. In: *IEEE International Conference on Service-Oriented Computing and Applications (SOCA), 2010*. 2010, pages 1–8. (Cited on page 419)

[Mietzner et al. 2009] R. Mietzner, T. Unger, and F. Leymann. Cafe: A Generic Configurable Customizable Composite Cloud Application Framework. In: *On the Move to Meaningful Internet Systems: OTM 2009*. Edited by R. Meersman, T. Dillon, and P. Herrero. Volume 5870. Lecture Notes in Computer Science. 10.1007/978-3-642-05148-7_24. Springer Berlin / Heidelberg, 2009, pages 357–364. (Cited on pages 450 and 459)

[Miranda et al. 2012] J. Miranda, J. Guillén, J. M. Murillo, and C. Canal. Enough About Standardization, Let's Build Cloud Applications. In: *Proceedings of the WICSA/ECSA 2012 Companion Volume*. WICSA/ECSA '12. Helsinki, Finland: ACM, 2012, pages 74–77. (Cited on page 115)

[Misra and Mondal 2011] S. C. Misra and A. Mondal. Identification of a company's suitability for the adoption of cloud computing and modelling its corresponding Return on Investment. *Mathematical and Computer Modelling* 53.3-4 (2011), pages 504–521. (Cited on pages 137, 431, 436, 437)

[Mitchell 1998] M. Mitchell. An Introduction to Genetic Algorithms (Complex Adaptive Systems). Third Printing. A Bradford Book, Feb. 1998. (Cited on pages 93 and 95)

[Mohagheghi and Sæther 2011] P. Mohagheghi and T. Sæther. Software Engineering Challenges for Migration to the Service Cloud Paradigm: Ongoing Work in the REMICS Project. In: *IEEE World Congress on Services (SERVICES), 2011*. 2011, pages 507–514. (Cited on pages 414, 422, 423)

[Mohan 2011] T. S. Mohan. Migrating into a Cloud. In: *Cloud Computing: Principles and Paradigms*. John Wiley & Sons, Inc., 2011. Chapter 2, pages 43–56. (Cited on pages 414 and 424)

[Montesinos et al. 1999] P. Montesinos, A. Garcia-Guzman, and J. L. Ayuso. Water distribution network optimization using a modified genetic algorithm. *Water Resources Research* 35.11 (1999), pages 3467–3473. (Cited on page 81)

[Moreno-Vozmediano et al. 2012] R. Moreno-Vozmediano, R. Montero, and I. Llorente. IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures. *Computer* 45.12 (2012), pages 65–72. (Cited on pages 440, 443, and 445)

[Moscato et al. 2011] F. Moscato, R. Aversa, B. Di Martino, T. Fortis, and V. Munteanu. An Analysis of mOSAIC ontology for Cloud Resources annotation. In: *Federated Conference on Computer Science and Information Systems (FedCSIS), 2011*. 2011, pages 973–980. (Cited on pages 440 and 445)

[Muhlenbein and Mahnig 2002] H. Muhlenbein and T. Mahnig. A comparison of stochastic local search and population based search. In: *Proceedings of the 2002 Congress on Evolutionary Computation, 2002. CEC '02.* Volume 1. 2002, pages 255 –260. (Cited on page 274)

[Müller et al. 2000] H. A. Müller, J. H. Jahnke, D. B. Smith, M.-A. Storey, S. R. Tilley, and K. Wong. Reverse Engineering: A Roadmap. In: *Proceedings of the Conference on The Future of Software Engineering*. ICSE '00. Limerick, Ireland: ACM, 2000, pages 47–60. (Cited on page 60)

[Murphy et al. 1995] G. C. Murphy, D. Notkin, and K. Sullivan. Software Reflexion Models: Bridging the Gap between Source and High-Level Models. In: *Proceedings of the 3rd ACM SIGSOFT Symposium on Foundations of Software Engineering*. SIGSOFT '95. Washington, D.C., USA: ACM, 1995, pages 18–28. (Cited on pages 465, 466)

[Murugesan 2013] S. Murugesan. Cloud Computing: The New Normal? *Computer* 46.1 (2013), pages 77–79. (Cited on page 35)

[Musa 1993] J. Musa. Operational Profiles in Software-Reliability Engineering. *IEEE Software* 10.2 (1993), pages 14–32. (Cited on page 156)

[Nakada et al. 2009] H. Nakada, T. Hirofuchi, H. Ogawa, and S. Itoh. Toward Virtual Machine Packing Optimization Based on Genetic Algorithm. In: *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*. Edited by S. Omatu, M. Rocha, J. Bravo, F. Fernández, E. Corchado, A. Bustillo, and J. Corchado. Volume 5518. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2009, pages 651–654. (Cited on page 499)

[Nasir and Niazi 2011] U. Nasir and M. Niazi. Cloud computing adoption assessment model (CAAM). In: *Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement*. Profes '11. Torre Canne, Brindisi, Italy: ACM, 2011, pages 34–37. (Cited on pages 42, 431, and 437)

[Neumaier et al. 2005] A. Neumaier, O. Shcherbina, W. Huyer, and T. Vinkó. A comparison of complete global optimization solvers. *Mathematical Programming* 103 (2 2005), pages 335–356. (Cited on page 495)

[Nguyen et al. 2011] D. Nguyen, F. Lelli, Y. Taher, M. Parkin, M. Papazoglou, and W.-J. van den Heuvel. Blueprint Template Support for Engineering Cloud-Based Services. In: *Towards a Service-Based Internet*. Edited by W. Abramowicz, I. Llorente, M. Surridge, A. Zisman, and J. Vayssière. Volume 6994. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2011, pages 26–37. (Cited on pages 450 and 460)

[Nie and Deng 2008] Y. Nie and W. Deng. A Hybrid Genetic Learning Algorithm for Pi-Sigma Neural Network and the Analysis of Its Convergence. In: *Fourth International Conference on Natural Computation, 2008. ICNC '08.* Volume 3. 2008, pages 19 –23. (Cited on pages 99 and 274)

[Nurmi et al. 2009] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. In: *Proceedings of the 9th IEEE/ACM Interna-*

tional Symposium on Cluster Computing and the Grid, 2009. CCGRID '09. 2009, pages 124–131. (Cited on pages 129 and 254)

[Object Management Group 2005] Object Management Group. UML Profile for Schedulability, Performance and Time (SPT) V.1.1. http://www.omg.org/spec/SPTP/1.1/. 2005. (Cited on page 484)

[Object Management Group 2006] Object Management Group. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms V.1.0. http://www.omg.org/spec/QFTP/1.0/. 2006. (Cited on page 484)

[Object Management Group 2010] Object Management Group. Unified Modeling Language (UML) Superstructure Version 2.3. http://www.omg.org/spec/UML/2.3/. 2010. (Cited on page 141)

[Object Management Group 2011a] Object Management Group. Architecture-Driven Modernization (ADM): Knowledge Discovery Meta-Model (KDM) Version 1.3. http://www.omg.org/spec/KDM/1.3/. 2011. (Cited on pages 74 and 206)

[Object Management Group 2011b] Object Management Group. Architecture-Driven Modernization (ADM): Abstract Syntax Tree Meta-Model (ASTM) Version 1.0. http://www.omg.org/spec/ASTM/. 2011. (Cited on page 73)

[Object Management Group 2011c] Object Management Group. OMG Meta Object Facility (MOF) Core Specification Version 2.4.1. http://www.omg.org/spec/MOF/2.4.1/. 2011. (Cited on page 77)

[Object Management Group 2012] Object Management Group. Architecture-Driven Modernization (ADM): Structured Metrics Meta-Model (SMM) Version 1.0. http://www.omg.org/spec/SMM/1.0/. 2012. (Cited on pages 13, 77–79, and 156)

[Osman and Kelly 1996] I. Osman and J. Kelly. Meta-Heuristics: An Overview. English. In: *Meta-Heuristics*. Edited by I. Osman and J. Kelly. Springer US, 1996, pages 1–21. (Cited on page 86)

[Oualline 2003] S. Oualline. Practical C++ Programming, Second Edition. 2nd. O'Reilly Media, Jan. 2003. (Cited on page 468)

[Padberg 2010] M. Padberg. Linear Optimization and Extensions (Algorithms and Combinatorics). Springer, Dec. 2010. (Cited on pages 480 and 486)

[Palankar et al. 2008] M. R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel. Amazon S3 for Science Grids: a Viable Solution? In: *Proceedings of the 2008 International Workshop on Data-aware Distributed Computing*. DADC '08. Boston, MA, USA: ACM, 2008, pages 55–64. (Cited on page 2)

[Pallis 2010] G. Pallis. Cloud Computing: The New Frontier of Internet Computing. *IEEE Internet Computing* 14.5 (2010), pages 70–73. (Cited on page 42)

[Pandey et al. 2010] S. Pandey, L. Wu, S. M. Guru, and R. Buyya. A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments. In: *Proc. 24th IEEE Int Advanced Information Networking and Applications (AINA) Conference*. 2010, pages 400–407. (Cited on pages 500, 501)

[Parkhill 1966] D. Parkhill. The Challenge of the Computer Utility. Addison-Wesley Educational Publishers Inc, 1966. (Cited on page 37)

[Parnas 1994] D. L. Parnas. Software Aging. In: *Proceedings of the 16th International Conference on Software Engineering*. ICSE '94. Sorrento, Italy: IEEE Computer Society Press, 1994, pages 279–287. (Cited on page 56)

[Parr and Quong 1995] T. J. Parr and R. W. Quong. ANTLR: A Predicated-LL(k) Parser Generator. *Software: Practice and Experience* 25.7 (1995), pages 789–810. (Cited on page 307)

[Passos et al. 2010] L. Passos, R. Terra, M. Valente, R. Diniz, and N. Mendonça. Static Architecture-Conformance Checking: An Illustrative Overview. *IEEE Software* 27.5 (2010), pages 82–89. (Cited on pages 466, 467)

[Pearson and Benameur 2010] S. Pearson and A. Benameur. Privacy, Security and Trust Issues Arising from Cloud Computing. In: *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), 2010*. 2010, pages 693–702. (Cited on page 41)

[Pérez-Castillo et al. 2011] R. Pérez-Castillo, I. G.-R. de Guzmán, and M. Piattini. Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems. *Computer Standards & Interfaces* 33.6 (2011), pages 519–532. (Cited on pages 13, 73–75)

[Phaphoom et al. 2012] N. Phaphoom, N. Oza, X. Wang, and P. Abrahamsson. Does Cloud Computing Deliver the Promised Benefits for IT Industry? In: *Proceedings of the WICSA/ECSA 2012 Companion Volume*. WICSA/ECSA '12. Helsinki, Finland: ACM, 2012, pages 45–52. (Cited on pages 35, 37, 42, and 123)

[Pigoski and Nelson 1994] T. Pigoski and L. Nelson. Software Maintenance Metrics: A Case Study. In: *Proceedings of the International Conference on Software Maintenance, 1994.* 1994, pages 392–401. (Cited on page 53)

[Pocuca et al. 2012] S. Pocuca, B. Marsic, and A. Grgic. Possible scenarios for cloud-based offering by operators. In: *Proceedings of the 35th International Convention MIPRO, 2012*. 2012, pages 506–511. (Cited on page 39)

[Popovic and Hocenski 2010] K. Popovic and Z. Hocenski. Cloud computing security issues and challenges. In: *Proceedings of the 33rd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2010*. 2010, pages 344–349. (Cited on page 41)

[Potts 1993] C. Potts. Software-Engineering Research Revisited. *IEEE Software* 10.5 (1993), pages 19–28. (Cited on page 107)

[Powers 2011] D. M. W. Powers. Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation. *Journal of Machine Learning Technologies* 2.1 (2011), pages 37–63. (Cited on pages 327–329)

[Price et al. 1993] B. A. Price, R. M. Baecker, and I. S. Small. A Principled Taxonomy of Software Visualization. *Journal of Visual Languages & Computing* 4.3 (1993), pages 211–266. (Cited on page 62)

[Prinz 2012] O. Prinz. Transformation of Java Bytecode to KDM Models as a Foundation for Dependency Analysis. Diploma Thesis. Kiel University, Kiel, Germany, 2012. (Cited on page 306)

[Prodan and Ostermann 2009] R. Prodan and S. Ostermann. A Survey and Taxonomy of Infrastructure as a Service and Web Hosting Cloud Providers. In: *10th IEEE/ACM International Conference on Grid Computing, 2009*. 2009, pages 17–25. (Cited on pages 124 and 235)

[Rafique et al. 2011] K. Rafique, A. Tareen, M. Saeed, J. Wu, and S. Qureshi. Cloud Computing Economics Opportunities and Challenges. In: *4th IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT), 2011*. 2011, pages 401–406. (Cited on page 40)

[Rajan et al. 2011] D. Rajan, A. Canino, J. A. Izaguirre, and D. Thain. Converting a High Performance Application to an Elastic Cloud Application. In: *IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), 2011*. 2011, pages 383 –390. (Cited on page 2)

[Rangan 2008] K. Rangan. The Cloud Wars: $100+ Billion at Stake. Technical report. Merrill Lynch, 2008. (Cited on pages 2 and 39)

[Ranganathan and Jouppi 2005] P. Ranganathan and N. Jouppi. Enterprise IT Trends and Implications for Architecture Research. In: *11th International Symposium on High-Performance Computer Architecture, 2005. HPCA-11*. 2005, pages 253 –256. (Cited on page 175)

[Rardin 1997] R. L. Rardin. Optimization in Operations Research. 1st edition. Prentice Hall, Aug. 1997. (Cited on page 86)

[Ren et al. 2012] K. Ren, C. Wang, and Q. Wang. Security Challenges for the Public Cloud. *Internet Computing, IEEE* 16.1 (2012), pages 69 –73. (Cited on pages 41 and 419)

[Richardson et al. 1997] R. Richardson, D. Lawless, B. Bisbal, B. Wu, J. Grimson, V. Wade, and D. O'Sullivan. A Survey of Research into Legacy System Migration. Technical report TCD-CS-1997-01. Trinity College, Dublin, Ireland, 1997. (Cited on pages 63, 64, 70, and 140)

[Räihä 2010] O. Räihä. A Survey on Search-Based Software Design. *Computer Science Review* 4.4 (2010), pages 203–249. (Cited on page 83)

[Rings et al. 2011] T. Rings, J. Grabowski, and S. Schulz. A Testing Framework for Assessing Grid and Cloud Infrastructure Interoperability. *International Journal On Advances in Systems and Measurements* 4.1&2 (2011), pages 95–108. (Cited on pages 473 and 478)

[Rocha et al. 2011] F. Rocha, S. Abreu, and M. Correia. The Final Frontier: Confidentiality and Privacy in the Cloud. *Computer* 44.9 (2011), pages 44 –50. (Cited on page 41)

[Rochwerger et al. 2009] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan. The Reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development* 53.4 (2009), 4:1–4:11. (Cited on pages 440 and 446)

[Rohr et al. 2008] M. Rohr, A. van Hoorn, J. Matevska, N. Sommer, L. Stoever, S. Giesecke, and W. Hasselbring. Kieker: Continuous Monitoring and on demand Visualization of Java Software Behavior. In: *Proceedings of the IASTED International Conference on Software Engineering 2008 (SE'08)*. Edited by C. Pahl. Anaheim, CA, USA: ACTA Press, 2008, pages 80–85. (Cited on page 156)

[Rosado et al. 2012] D. G. Rosado, R. Gómez, D. Mellado, and E. Fernández-Medina. Security Analysis in the Migration to Cloud Environments. *Future Internet* 4.2 (2012), pages 469–487. (Cited on page 419)

[Rozinat and van der Aalst 2008] A. Rozinat and W. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems* 33.1 (2008), pages 64–95. (Cited on page 463)

[Ruhe and Saliu 2005] G. Ruhe and M. Saliu. The Art and Science of Software Release Planning. *IEEE Software* 22.6 (2005), pages 47–53. (Cited on page 85)

[San Aniceto et al. 2011] I. San Aniceto, R. Moreno-Vozmediano, R. Montero, and I. Llorente. Cloud Capacity Reservation for Optimal Service Deployment. In: *Proceedings of the Second International Conference on Cloud Computing, GRIDs, and Virtualization (Cloud Computing 2011)*. Rome, Italy: IARIA Conference, 2011, pages 52–59. (Cited on pages 491 and 496)

[Schmidt et al. 2010] M. Schmidt, N. Fallenbeck, M. Smith, and B. Freisleben. Efficient Distribution of Virtual Machines for Cloud Computing. In: *18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2010*. 2010, pages 567–574. (Cited on page 46)

[Schoen 1991] F. Schoen. Stochastic Techniques for Global Optimization: A Survey of Recent Advances. *Journal of Global Optimization* 1 (3 1991). 10.1007/BF00119932, pages 207–228. (Cited on pages 382–385)

[Schrijver 1998] A. Schrijver. Theory of Linear and Integer Programming. Wiley, June 1998. (Cited on page 81)

[Seacord et al. 2001] R. C. Seacord, S. Comella-Dorda, G. Lewis, P. Place, and D. Plakosh. Legacy System Modernization Strategies. Technical report CMU/SEI-2001-TR-025. Carnegie Mellon University, Software Engineering Institute, 2001. (Cited on page 151)

[Seacord et al. 2003] R. C. Seacord, D. Plakosh, and G. A. Lewis. Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices (SEI Series in Software Engineering). Addison-Wesley Longman, Amsterdam, Feb. 2003. (Cited on page 57)

[Seaman 1999] C. B. Seaman. Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Transactions on Software Engineering* 25.4 (July 1999), pages 557–572. (Cited on page 108)

[Sha and Xu 2011] J. Sha and M. Xu. Applying Hybrid Genetic Algorithm to Constrained Trajectory Optimization. In: *International Conference on Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011*. Volume 7. 2011, pages 3792 –3795. (Cited on pages 99 and 274)

[Sharp et al. 1999] H. Sharp, A. Finkelstein, and G. Galal. Stakeholder Identification in the Requirements Engineering Process. In: *Proceedings of the Tenth International Workshop on Database and Expert Systems Applications, 1999*. 1999, pages 387 –391. (Cited on page 140)

[Shimba 2010] F. Shimba. Cloud Computing: Strategies for Cloud Computing Adoption. Master's thesis. Dublin Institute of Technology, 2010. (Cited on pages 137, 414, and 425)

[Siegele 2008] L. Siegele. Let it Rise: A Special Report on Corporate IT. *The Economist* (2008). (Cited on pages 2 and 39)

[Silva and Lucredio 2012] E. A. N. d. Silva and D. Lucredio. Software Engineering for the Cloud: A Research Roadmap. In: *26th Brazilian Symposium on Software Engineering (SBES), 2012*. 2012, pages 71 –80. (Cited on page 42)

[Simon et al. 2003] G. Simon, P. Volgyesi, M. Maroti, and A. Ledeczi. Simulation-based optimization of communication protocols for large-scale wireless sensor networks. In: *Proceedings of the 2003 IEEE Aerospace Conference, 2003.* Volume 3. 2003. (Cited on page 101)

[Sjoberg et al. 2007] D. Sjoberg, T. Dyba, and M. Jorgensen. The Future of Empirical Methods in Software Engineering Research. In: *Future of Software Engineering, 2007. FOSE '07*. 2007, pages 358–378. (Cited on pages 107, 108)

[Smit and Stroulia 2011] M. Smit and E. Stroulia. Autonomic Configuration Adaptation Based on Simulation-generated State-transition Models. In: *37th EUROMICRO Conference on Software Engineering and Advanced Applications*. 2011. (Cited on pages 491, 496, 497)

[Smit et al. 2008] M. Smit, A. Nisbet, E. Stroulia, A. Edgar, G. Iszlai, and M. Litoiu. Capacity Planning for Service-oriented Architectures. In: *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds*. CASCON '08. Ontario, Canada: ACM, 2008, 11:144–11:156. (Cited on page 496)

[Sneed 1995] H. Sneed. Planning the Reengineering of Legacy Systems. *IEEE Software* 12.1 (1995), pages 24 –34. (Cited on page 58)

[SOFTEAM 2012] T. SOFTEAM SINTEF. PIM4Cloud. EU Project Deliverable. REMICS Consortium, 2012. (Cited on pages 450 and 454)

[Solingen 1999] B. Solingen. Goal/Question/Metric Method. Mcgraw Hill Higher Education, Jan. 1999. (Cited on pages 324, 325, 338, 339, 342, 395, 396)

[Spinellis 2001] D. Spinellis. Notable design patterns for domain-specific languages. *Journal of Systems and Software* 56.1 (2001), pages 91–99. (Cited on page 173)

[Srinivas and Patnaik 1994a] M. Srinivas and L. Patnaik. Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms. *IEEE Transactions on Systems, Man and Cybernetics* 24.4 (1994), pages 656–667. (Cited on pages 99 and 273)

[Srinivas and Patnaik 1994b] M. Srinivas and L. Patnaik. Genetic Algorithms: A Survey. *Computer* 27.6 (1994), pages 17–26. (Cited on pages 93, 96, and 98)

[Srinivas and Deb 1994] N. Srinivas and K. Deb. Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation* 2.3 (Sept. 1994), pages 221–248. (Cited on page 96)

[Sriram and Khajeh-Hosseini 2010] I. Sriram and A. Khajeh-Hosseini. Research Agenda in Cloud Technologies. *CoRR* abs/1001.3259 (2010). (Cited on pages 35, 37, and 42)

[Steinberg et al. 2009] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. EMF: Eclipse Modeling Framework 2.0. 2nd. Addison-Wesley Professional, 2009. (Cited on pages 170, 308, 309)

[Storey and Müller 1995] M.-A. Storey and H. Müller. Manipulating and Documenting Software Structures Using SHriMP Views. In: *Proceedings of the International Conference on Software Maintenance, 1995, (ICSM 1995)*. 1995, pages 275–284. (Cited on page 62)

[Stroulia and Systä 2002] E. Stroulia and T. Systä. Dynamic Analysis for Reverse Engineering and Program Understanding. *SIGAPP Appl. Comput. Rev.* 10 (1 2002), pages 8–17. (Cited on page 63)

[Subashini and Kavitha 2011] S. Subashini and V. Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications* 34.1 (2011), pages 1 –11. (Cited on pages 41 and 419)

[Suleiman et al. 2012] B. Suleiman, S. Sakr, R. Jeffery, and A. Liu. On understanding the economics and elasticity challenges of deploying business applications on public cloud infrastructure. *Journal of Internet Services and Applications* 3 (2 2012). 10.1007/s13174-011-0050-y, pages 173–193. (Cited on pages 11 and 36)

[Systä 2000] T. Systä. Static And Dynamic Reverse Engineering Techniques for Java Software Systems. ISBN 951-44-4811-1. PhD thesis. Tampere: University of Tampere, Department of Computer and Information Sciences, 2000. (Cited on page 63)

[Szabo and Kroeger 2012] C. Szabo and T. Kroeger. Evolving Multi-objective Strategies for Task Allocation of Scientific Workflows on Public Clouds. In: *IEEE Congress on Evolutionary Computation (CEC), 2012*. 2012, pages 1–8. (Cited on pages 500–502)

[Talbi 2009] E.-G. Talbi. Metaheuristics: From Design to Implementation (Wiley Series on Parallel and Distributed Computing). Wiley, June 2009. (Cited on pages 86, 87)

[Tang and Breckon 2011] I. Tang and T. Breckon. Automatic Road Environment Classification. *IEEE Transactions on Intelligent Transportation Systems* 12.2 (2011), pages 476–484. (Cited on page 326)

[Tao 2001] L. Tao. Shifting Paradigms with the Application Service Provider Model. *Computer* 34.10 (2001), pages 32–39. (Cited on page 37)

[Taylor et al. 2009] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. Software Architecture: Foundations, Theory, and Practice. Wiley, Jan. 2009. (Cited on pages 281, 289, 290)

[Tchifilionova 2011] V. Tchifilionova. Security and Privacy Implications of Cloud Computing – Lost in the Cloud. In: *Open Research Problems in Network Security*. Edited by J. Camenisch, V. Kisimov, and M. Dubovitskaya. Volume 6555. Lecture Notes in Computer Science. 10.1007/978-3-642-19228-9_14. Springer Berlin / Heidelberg, 2011, pages 149–158. (Cited on page 41)

[Teckelmann et al. 2011] R. Teckelmann, C. Reich, and A. Sulistio. Mapping of Cloud Standards to the Taxonomy of Interoperability in IaaS. In: *IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), 2011*. 2011, pages 522–526. (Cited on page 42)

[Thakar and Szalay 2010] A. Thakar and A. Szalay. Migrating a (Large) Science Database to the Cloud. In: *HPDC '10: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. Chicago, Illinois: ACM, 2010, pages 430–434. (Cited on page 2)

[Tianfield 2011] H. Tianfield. Cloud Computing Architectures. In: *IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2011*. 2011, pages 1394 –1399. (Cited on pages 440, 447, 450, and 461)

[Tietjens 2010] B.-P. Tietjens. Modellbasierte Architektur-Rekonstruktion mit Hilfe von KDM und EMF. Diploma Thesis. Kiel University, Kiel, Germany, 2010. (Cited on page 306)

[Tonella 2005] P. Tonella. Reverse Engineering of Object Oriented Code. In: *Proceedings of the 27th International Conference on Software Engineering*. ICSE '05. St. Louis, MO, USA: ACM, 2005, pages 724–725. (Cited on page 63)

[Tran et al. 2011a] V. Tran, J. Keung, A. Liu, and A. Fekete. Application Migration to Cloud: A Taxonomy of Critical Factors. In: *Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing (SECLOUD 2011)*. SECLOUD '11. New York, NY, USA: ACM, 2011, pages 22–28. (Cited on pages 2 and 137)

[Tran et al. 2011b] V. Tran, K. Lee, A. Fekete, A. Liu, and J. Keung. Size Estimation of Cloud Migration Projects with Cloud Migration Point (CMP). In: *The Fifth International Symposium on Empirical Software Engineering and Measurement*. Banff, Alberta, Canada, 2011. (Cited on pages 431 and 438)

[Trummer et al. 2010] I. Trummer, F. Leymann, R. Mietzner, and W. Binder. Cost-Optimal Outsourcing of Applications into the Clouds. In: *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), 2010*. 2010, pages 135 –142. (Cited on pages 491 and 497)

[Tsai et al. 2010] W.-T. Tsai, X. Sun, and J. Balasooriya. Service-Oriented Cloud Computing Architecture. In: *Seventh International Conference on Information Technology: New Generations (ITNG), 2010*. 2010, pages 684 –689. (Cited on pages 440, 447, 448)

[Ullah 2012] K. W. Ullah. Automated Security Compliance Tool for the Cloud. Master's thesis. Norwegian University of Science and Technology, Department of Telematics, 2012. (Cited on pages 473, 478, 479)

[Ulrich 2004] W. Ulrich. A Status on OMG Achitecture-Driven Modernization Task Force. In: *Proceedings of the 8th International IEEE Enterprise Distributed Object Computing Conference (EDOC), Workshop on Model-driven Evolution of Legacy Systems (MELS)*. 2004. (Cited on pages 54 and 73)

[Van Gurp and Bosch 2002] J. van Gurp and J. Bosch. Design erosion: problems and causes. *Journal of Systems and Software* 61.2 (2002), pages 105 –119. (Cited on page 465)

[Van Hoorn et al. 2009] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey, and D. Kieselhorst. Continuous Monitoring of Software Services: Design and Application of the Kieker Framework. Technical report TR-0921. Department of Computer Science, Kiel University, Germany, Nov. 2009. (Cited on pages 21, 156, and 296)

[Van Hoorn et al. 2011] A. van Hoorn, S. Frey, W. Goerigk, W. Hasselbring, H. Knoche, S. Köster, H. Krause, M. Porembski, T. Stahl, M. Steinkamp, and N. Wittmüss. DynaMod project: Dynamic analysis for model-driven software modernization. In: *Proceedings of the 1st International Workshop on Model-Driven Software Migration (MDSM 2011), Project Presentations Track*. 2011. (Cited on pages 22 and 153)

[Van Hoorn et al. 2012] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis. In: *Proc. of the 3rd ACM/SPEC Int'l Conference on Performance Engineering (ICPE 2012)*. ACM, Apr. 2012, pages 247–248. (Cited on pages 156 and 296)

[Van der Aalst 2011a] W. van der Aalst. Using Process Mining to Bridge the Gap between BI and BPM. *Computer* 44.12 (2011), pages 77–80. (Cited on page 463)

[Van der Aalst 2011b] W. M. P. van der Aalst. Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer, 2011, pages I–XVI, 1–352. (Cited on page 472)

[Van der Aalst 2011c] W. van der Aalst. Business Process Configuration in the Cloud: How to Support and Analyze Multi-tenant Processes? In: *Ninth IEEE European Conference on Web Services (ECOWS), 2011*. 2011, pages 3–10. (Cited on pages 472–474)

[Van Latum et al. 1998] F. Van Latum, R. Van Solingen, M. Oivo, B. Hoisl, D. Rombach, and G. Ruhe. Adopting GQM-Based Measurement in an Industrial Environment. *Software, IEEE* 15.1 (1998), pages 78–86. (Cited on page 324)

[Van Veldhuizen and Lamont 2000] D. Van Veldhuizen and G. Lamont. On Measuring Multiobjective Evolutionary Algorithm Performance. In: *Proceedings of the 2000 Congress on Evolutionary Computation, 2000.* Volume 1. 2000, pages 204–211. (Cited on pages 386, 387)

[Vaquero et al. 2009] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A Break in the Clouds: Towards a Cloud Definition. *ACM SIGCOMM Computer Communication Review* 39.1 (2009), pages 50–55. (Cited on page 129)

[Venugopal et al. 2011] S. Venugopal, S. Desikan, and K. Ganesan. Effective Migration of Enterprise Applications in Multicore Cloud. In: *Fourth IEEE International Conference on Utility and Cloud Computing (UCC), 2011.* 2011, pages 463–468. (Cited on pages 414 and 425)

[Vergilio et al. 2011] S. Vergilio, T. Colanzi, A. Pozo, and W. Assuncao. Search Based Software Engineering: A Review from the Brazilian Symposium on Software Engineering. In: *25th Brazilian Symposium on Software Engineering (SBES), 2011*. 2011, pages 50–55. (Cited on page 81)

[Villegas et al. 2012] D. Villegas, A. Antoniou, S. Sadjadi, and A. Iosup. An Analysis of Provisioning and Allocation Policies for Infrastructure-as-a-Service Clouds. In: *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2012*. 2012, pages 612 –619. (Cited on pages 491 and 498)

[Voinea 2007] S. Voinea. Software Evolution Visualization. PhD thesis. Eindhoven University of Technology, Department of Mathematics and Computer Science, 2007. (Cited on page 62)

[Wada et al. 2011] H. Wada, J. Suzuki, Y. Yamano, and K. Oba. Evolutionary deployment optimization for service-oriented clouds. *Software: Practice and Experience* 41.5 (2011), pages 469–493. (Cited on pages 500 and 502)

[Waldspurger and Rosenblum 2012] C. Waldspurger and M. Rosenblum. I/O virtualization. *Communications of the ACM* 55.1 (Jan. 2012), pages 66–73. (Cited on pages 35 and 43)

[Walls 2011] C. Walls. Spring in Action. Third Edition. Manning Publications, June 2011. (Cited on page 334)

[Wang et al. 2012] L. Wang, J. Zhan, W. Shi, and Y. Liang. In Cloud, Can Scientific Communities Benefit from the Economies of Scale? *IEEE Transactions on Parallel and Distributed Systems* 23.2 (2012), pages 296–303. (Cited on page 40)

[Wang et al. 2010] L. Wang, G. von Laszewski, A. Younge, X. He, M. Kunze, J. Tao, and C. Fu. Cloud Computing: a Perspective Study. *New Generation Computing* 28 (2 2010). 10.1007/s00354-008-0081-5, pages 137–146. (Cited on pages 37, 38)

[Wang et al. 2005] W. Wang, H. Rivard, and R. Zmeureanu. An object-oriented framework for simulation-based green building design optimization with genetic algorithms. *Advanced Engineering Informatics* 19.1 (2005), pages 5–23. (Cited on page 101)

[Ward et al. 2010] C. Ward, N. Aravamudan, K. Bhattacharya, K. Cheng, R. Filepp, R. Kearney, B. Peterson, L. Shwartz, and C. Young. Workload Migration into Clouds Challenges, Experiences, Opportunities. In: *IEEE 3rd International Conference on Cloud Computing (CLOUD), 2010*. 2010, pages 164 –171. (Cited on pages 123, 137, 414, 426, 427)

[Weinhardt et al. 2009] C. Weinhardt, A. Anandasivam, B. Blau, N. Borissov, T. Meinl, W. Michalk, and J. Stößer. Cloud Computing – A Classification, Business Models, and Research Directions. *Business & Information Systems Engineering* 1 (5 2009). 10.1007/s12599-009-0071-2, pages 391–399. (Cited on page 37)

[Wettel and Lanza 2007] R. Wettel and M. Lanza. Visualizing Software Systems as Cities. In: *4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007.* 2007, pages 92 –99. (Cited on page 62)

[Whitley 1994] D. Whitley. A genetic algorithm tutorial. *Statistics and Computing* 4 (2 1994). 10.1007/BF00175354, pages 65–85. (Cited on pages 93–95, and 99)

[Winter and Ziemann 2007] A. Winter and J. Ziemann. Model-based Migration to Service-oriented Architectures - A Project Outline. In: *CSMR 2007, 11th European Conference on Software Maintenance and Reengineering,*

Workshops. Edited by H. Sneed. Vrije Universiteit Amsterdam. 2007, pages 107–110. (Cited on pages 63, 64)

[Wu et al. 1997] B. Wu, D. Lawless, J. Bisbal, R. Richardson, J. Grimson, V. Wade, and D. O'Sullivan. The Butterfly Methodology: A Gateway-free Approach for Migrating Legacy Information Systems. In: *Proceedings of the Third IEEE International Conference on Engineering of Complex Computer Systems, 1997*. 1997, pages 200–205. (Cited on page 70)

[Wu et al. 2005] L. Wu, H. Sahraoui, and P. Valtchev. Coping with Legacy System Migration Complexity. In: *Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems, 2005. ICECCS 2005*. 2005, pages 600–609. (Cited on pages 63–65)

[Wu et al. 2011] L. Wu, S. Garg, and R. Buyya. SLA-Based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments. In: *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2011*. 2011, pages 195–204. (Cited on pages 491, 498, 499)

[Wulf 2012] C. Wulf. Automatic Conformance Checking of C#-based Software Systems for Cloud Migration. Master's thesis. Software Engineering Group, Kiel University, Germany, 2012. (Cited on pages 307, 323, 331, 352–354, 368–370, 375, and 377)

[Wulf et al. 2012] C. Wulf, S. Frey, and W. Hasselbring. A Three-Phase Approach to Efficiently Transform C# into KDM. Technical report TR-1211. Department of Computer Science, Kiel University, Germany, Aug. 2012. (Cited on pages 23, 75, 76, and 308)

[Yau et al. 1988] S. Yau, R. Nicholl, J.-P. Tsai, and S.-S. Liu. An Integrated Life-Cycle Model for Software Maintenance. *IEEE Transactions on Software Engineering* 14.8 (1988), pages 1128–1144. (Cited on page 53)

[Younge et al. 2010] A. Younge, G. von Laszewski, L. Wang, S. Lopez-Alarcon, and W. Carithers. Efficient Resource Management for Cloud Computing Environments. In: *Proceedings of the International Conference on Green Computing (GREENCOMP '10)*. 2010, pages 357 –364. (Cited on page 37)

[Youseff et al. 2008] L. Youseff, M. Butrico, and D. Da Silva. Toward a Unified Ontology of Cloud Computing. In: *Grid Computing Environments Workshop, 2008. GCE '08*. 2008, pages 1–10. (Cited on page 38)

[Yu 2012] Z. Yu. Cloud Computing-Conversion Technology for Interoperability. In: *Fourth International Conference on Multimedia Information Networking and Security (MINES), 2012*. 2012, pages 179–182. (Cited on page 42)

[Yun and Gen 2003] Y. Yun and M. Gen. Performance Analysis of Adaptive Genetic Algorithms with Fuzzy Logic and Heuristics. English. *Fuzzy Optimization and Decision Making* 2 (2 2003), pages 161–175. (Cited on pages 99 and 273)

[Yusoh and Tang 2012] Z. I. M. Yusoh and M. Tang. Composite SaaS Placement and Resource Optimization in Cloud Computing Using Evolutionary Algorithms. *2012 IEEE Fifth International Conference on Cloud Computing* 0 (2012), pages 590–597. (Cited on pages 117 and 499)

[Zardari and Bahsoon 2011] S. Zardari and R. Bahsoon. Cloud Adoption: A Goal-Oriented Requirements Engineering Approach. In: *Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing (SECLOUD 2011)*. SECLOUD '11. ACM, 2011, pages 29–35. (Cited on page 137)

[Zeigler et al. 2000] B. P. Zeigler, H. Praehofer, and T. G. Kim. Theory of Modeling and Simulation, Second Edition. 2nd edition. Academic Press, Jan. 2000, page 510. (Cited on page 100)

[Zhang et al. 2007] C. Zhang, R. Chang, C.-S. Perng, E. So, C. Tang, and T. Tao. QoS-Aware Optimization of Composite-Service Fulfillment Policy. In: *IEEE International Conference on Services Computing, 2007. SCC 2007.* 2007, pages 11–19. (Cited on pages 483 and 490)

[Zhang and Liu 2011] G. Zhang and L. Liu. Why Do Migrations Fail and What Can We Do about It? In: *Proceedings of Usenix 25th Large Installation System Administration Conference (LISA '11)*. USENIX Association, 2011. (Cited on pages 414 and 428)

[Zhang and Zhou 2009] L.-J. Zhang and Q. Zhou. CCOA: Cloud Computing Open Architecture. In: 2009, pages 607–616. (Cited on pages 38, 440, and 449)

[Zhang et al. 2010] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications* 1 (1 2010). 10.1007/s13174-010-0007-6, pages 7–18. (Cited on pages 37, 40, and 42)

[Zhang et al. 2008] Q. Zhang, A. Zhou, and Y. Jin. RM-MEDA: A Regularity Model-Based Multiobjective Estimation of Distribution Algorithm. *IEEE Transactions on Evolutionary Computation* 12.1 (2008), pages 41–63. (Cited on page 388)

[Zhang and Liu 2010] R. Zhang and L. Liu. Security Models and Requirements for Healthcare Application Clouds. In: *IEEE 3rd International Conference on Cloud Computing (CLOUD), 2010*. 2010, pages 268–275. (Cited on page 40)

[Zhang et al. 2009] W. Zhang, A. J. Berre, D. Roman, and H. A. Huru. Migrating Legacy Applications to the Service Cloud. In: *14th Conference companion on Object Oriented Programming Systems Languages and Applications (OOPSLA 2009)*. Orlando, Florida, USA, 2009, pages 59–68. (Cited on pages 414, 427, 428)

[Zhao et al. 2012] B. Zhao, Y. Zhao, and D. Ma. A Constraint Mechanism for Dynamic Evolution of Service Oriented Systems. In: *IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2012*. 2012, pages 103–110. (Cited on pages 466 and 471)

[Zheng et al. 2011] Z. Zheng, R. Wang, H. Zhong, and X. Zhang. An Approach for Cloud Resource Scheduling Based on Parallel Genetic Algorithm. In: *3rd International Conference on Computer Research and Development (ICCRD), 2011*. Volume 2. 2011, pages 444–447. (Cited on page 499)

[Zhou et al. 2010a] H. Zhou, H. Yang, and A. Hugill. An Ontology-Based Approach to Reengineering Enterprise Software for Cloud Computing. In: *IEEE 34th Annual Computer Software and Applications Conference (COMPSAC), 2010*. 2010, pages 383 –388. (Cited on pages 137, 414, and 430)

[Zhou et al. 2010b] M. Zhou, R. Zhang, W. Xie, W. Qian, and A. Zhou. Security and Privacy in Cloud Computing: A Survey. In: *Sixth International Conference on Semantics Knowledge and Grid (SKG), 2010*. 2010, pages 105–112. (Cited on page 41)

[Zhu 2005] H. Zhu. Software Design Methodology: From Principles to Architectural Styles. 1st edition. Butterworth-Heinemann, June 2005. (Cited on page 289)

[Zitzler and Thiele 1999] E. Zitzler and L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation* 3.4 (1999), pages 257–271. (Cited on pages 89 and 91)

[Zitzler and Thiele 1998] E. Zitzler and L. Thiele. Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study. In: *Parallel Problem Solving from Nature - PPSN V*. Edited by A. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel. Volume 1498. Lecture Notes in Computer Science. 10.1007/BFb0056872. Springer Berlin / Heidelberg, 1998, pages 292–301. (Cited on page 388)

[Zitzler et al. 2000] E. Zitzler, K. Deb, and L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation* 8.2 (June 2000), pages 173–195. (Cited on page 96)

[Zitzler et al. 2003] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation* 7.2 (2003), pages 117–132. (Cited on pages 92, 386, and 399)

[Zou and Kontogiannis 2002] Y. Zou and K. Kontogiannis. Migration to Object Oriented Platforms: A State Transformation Approach. In: *Proceedings of the International Conference on Software Maintenance, 2002.* 2002, pages 530 –539. (Cited on page 65)