

Secure Distributed Information Management Based on Semantic Web Technologies

Dissertation

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften

der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel

vorgelegt von
Falko Schönteich, M.Sc.
aus Kiel

Kiel, 2021

Dekan:	Prof. Dr. Lorenz Kienle
1. Gutachter:	Prof. Dr. Ansgar Scherp
2. Gutachter:	Prof. Dr. Peer Kröger
Datum der mündlichen Prüfung:	02. September 2021

Erklärung

Hiermit versichere ich,

1. dass diese Arbeit – abgesehen von der Beratung durch den Betreuer Prof. Dr. Ansgar Scherp – nach Inhalt und Form meine eigene ist,
2. dass Vorversionen einiger Teile dieser Arbeit bereits veröffentlicht wurden oder zur Veröffentlichung vorgelegt wurden, nämlich
 - Falko Schönteich, Andreas Kasten, and Ansgar Scherp. A Pattern-Based Core Ontology for Product Lifecycle Management based on DUL. In *Workshop on Ontology Design and Patterns (WOP) colocated at International Semantic Web Conference (ISWC)*, CEUR Workshop Proceedings. CEUR-WS.org, 2018
 - Falko Schönteich, Andreas Kasten, and Ansgar Scherp. Secure Product Lifecycle Management with CO-PLM. In *Advances in Pattern-based Ontology Engineering*. IOS Press, 2021
 - Falko Schönteich, Andreas Kasten, and Ansgar Scherp. Distributed Identity Management for Semantic Entities. In *International Conference on Information Management and Big Data (SIMBIG)*. Springer, 2020
 - Falko Schönteich, Ansgar Scherp, and Andreas Kasten. Distributed Identity Management for Semantic Entities based on Graph Signatures and owl:sameAs. *International Journal of Semantic Computing (IJSC)*, 2021

Als Erstautor dieser Publikationen übernahm ich die methodische und inhaltliche Verantwortung der Forschungsarbeit. Die weiteren Autoren übernahmen eine überwiegend beratende Funktion und gaben stellenweise Verbesserungsvorschläge.

Die Grundideen der in den Publikationen präsentierten Forschung stammen von mir. Die Ausformulierung von Texten, die Gestaltung von Abbildungen, die Literaturrecherche sowie die Durchführung von Experimenten erfolgte fast ausschließlich durch mich. Desweiteren versichere ich,

3. dass kein Teil dieser Arbeit bereits einer anderen Stelle im Rahmen eines Prüfungsverfahrens vorgelegen hat,
4. dass diese Arbeit unter Einhaltung der Regeln guter wissenschaftlicher Praxis der Deutschen Forschungsgemeinschaft entstanden ist,
5. und dass mir kein akademischer Grad entzogen wurde.

Kiel

2021

Falko Schönteich

Acknowledgments

Throughout the writing of this dissertation, I have received great support and assistance. I would like to express my sincere gratitude to Prof. Dr. Ansgar Scherp for supervising me as a doctoral candidate. This thesis would not exist without you. Your advice was invaluable to transform my initial ideas formed by practical experience into profound academic research. You always encouraged me to continue my research and helped me grow as a research scientist. Thanks to your insightful guidance, this thesis has a level of quality and clear structure I would not have reached on my own. Your insightful feedback pushed me to sharpen my thinking and form my work into this thesis. You also taught me the important lesson to ultimately trust my own judgment when things get difficult.

Special appreciation belongs to Dr. Andreas Kasten. Thank you for sharing your extensive knowledge about Information Security, Semantic Web technologies, and, particularly, combining both. Your valuable feedback and input helped a great deal when writing the publications this thesis is based on.

I want to thank Till Blume for sacrificing valuable time when he was actually working on his own thesis. Thank you for helping me find opportunities for some impactful, last-minute improvements of my thesis and the “Schrevenpark-Coffee-Times.”

Words cannot express how grateful I am to my mother Birgit, my father Winfrid, and my sister Carina for all their love and support. Thank you for always believing in me.

Finally, I could not have completed this dissertation without the patience and encouragement of Jacqueline. You were the light that guided me through difficult times.

Abstract

Multi-organizational collaboration, e.g., international industrial projects, requires the integration of distributed information systems. With increasing system integration, security risks affecting the fundamental security goals, confidentiality, integrity, and availability, as well as administrative challenges arise. The Semantic Web offers technologies for information representations universally processable by humans and machines. Therefore, Semantic Web technologies aid in system integration, but the security features required for multi-organizational collaboration are missing.

This thesis fills this gap by presenting semDIM, the first semantic framework offering secure distributed management of semantic identities, groups, and roles. The message exchange protocol developed for semDIM features a novel pattern-based concurrency control mechanism for ensuring consistency in distributed pattern-based data stores managed by multiple organizations.

The framework is developed according to the requirements of a real-life international industrial engineering and manufacturing project, but it is applicable to any other domain. This flexibility is ensured by using ontology patterns that provide a formal adaptable knowledge representation. Combining semDIM with the respective ontologies allows it to be aligned to different use cases. For the industrial scenario, this thesis presents CO-PLM, the first pattern-based product-data-related core ontology covering all lifecycle phases and integrating workflows as well as security policies. This thesis evaluates CO-PLM and semDIM against requirements derived from the collaboration scenario. In contrast to the state-of-the-art solutions, combined, CO-PLM and semDIM cover all requirements for secure semantic information exchange in industrial collaboration projects, and they can also be used independently for other use cases.

Zusammenfassung

Die organisationsübergreifende Zusammenarbeit, z. B. internationale Industrieprojekte, erfordert die Integration verteilter Informationssysteme. Mit zunehmender Systemintegration entstehen Sicherheitsrisiken, die die grundlegenden Sicherheitsziele, Vertraulichkeit, Integrität und Verfügbarkeit, betreffen sowie administrative Herausforderungen. Das Semantic Web bietet Technologien, welche Daten universell von Menschen und Maschinen verarbeitbar machen. Semantische Web-Technologien helfen daher bei der Systemintegration, aber es fehlen Sicherheitsmechanismen, die für die multiorganisatorische Zusammenarbeit erforderlich sind.

Die vorliegende Arbeit füllt diese Lücke, indem sie semDIM vorstellt, das erste semantische Framework, das eine sichere verteilte Verwaltung von semantischen Identitäten, Gruppen und Rollen bietet. Das für semDIM entwickelte Nachrichtenaustauschprotokoll verfügt über einen neuartigen Mechanismus (Pattern-based Concurrency Control) zur Sicherstellung der Konsistenz in verteilten, musterbasierten Datenspeichern, die von mehreren Organisationen verwaltet werden.

Das Framework wurde anhand der Anforderungen eines realen internationalen Projekts im Bereich des industriellen Maschinenbaus entwickelt. Es ist aber auch auf jede andere Domäne anwendbar. Diese Flexibilität wird durch die Verwendung von Ontology Patterns gewährleistet, die eine formale, anpassbare Wissensrepräsentation bieten. Durch die Kombination von semDIM mit den entsprechenden Ontologien können unterschiedliche Anwendungsfälle adressiert werden. Für das industrielle Szenario wird in dieser Arbeit CO-PLM vorgestellt, die erste musterbasierte produktdatenbezogene Core Ontology, die alle Lebenszyklusphasen abdeckt und Workflows sowie Sicherheitsrichtlinien integriert. Sowohl CO-PLM als auch semDIM werden gegen die aus dem Szenario abgeleiteten Anforderungen evaluiert. CO-PLM und semDIM kombiniert erfüllen alle Anforderungen eines sicheren semantischen Informationsaustausch in industriellen Kollaborationsprojekten. Sie können aber auch unabhängig voneinander für andere Anwendungsfälle verwendet werden.

Contents

List of Figures	viii
List of Tables	xiii
Listings	xiv
Acronyms	xv
1 Introduction	1
1.1 Motivating Scenario: The <i>Blue Train</i> Project	4
1.1.1 Challenge 1: Managing Product-related Information	7
1.1.2 Challenge 2: Distributed Identity Management	8
1.2 Research Questions	9
1.3 Approach	10
1.4 Contribution	13
2 Secure Product Lifecycle Management with CO-PLM	15
2.1 Background and Structure of This Chapter	16
2.2 Related Work	18
2.3 Problem Description	20
2.3.1 Engineering and Manufacturing View of Parts	21
2.3.2 Engineering and Manufacturing Product Structure	23
2.4 Requirements	25
2.4.1 R1: Differentiating between Product Concepts and Product Instances	25
2.4.2 R2: Different Views of Parts depending on Context	25
2.4.3 R3: Distributed Workflow Models and Workflow Executions	25
2.4.4 R4: Integration of Security Policies	26
2.4.5 Non-functional Requirements	26
2.5 Overview of the CO-PLM Core Ontology	27

2.6	Design of the CO-PLM Core Ontology	29
2.6.1	Product Part Information Entity Pattern	31
2.6.2	Product Part Information Entity Qualities Pattern	32
2.6.3	Product Part Information Object Pattern	33
2.6.4	Part Entry Pattern	33
2.6.5	Product Part Information Realization Pattern	34
2.6.6	Pattern for R1: Product Part Pattern	36
2.6.7	Pattern for R2: Product Part Description Pattern.	38
2.6.8	Patterns for R3: PLM Workflow Patterns	40
2.6.9	Pattern for R4: Access Regulation Rule Pattern	41
2.6.10	Fulfillment of Non-Functional Requirements	43
2.7	Evaluation of Practical Use	44
2.7.1	Joint Software and Semantic Engineering Process	44
2.7.2	Reasoning Experiments on the <i>Blue Train</i> Data and Large-Scale Data	46
2.8	Synopsis	49
3	Semantic Distributed Identity Management with semDIM	50
3.1	Background and Structure of this Chapter	51
3.2	Related Work	55
3.3	Problem Description	58
3.3.1	DIM Challenges in Distributed Engineering	58
3.3.2	DIM for the <i>Blue Train</i> Project	59
3.4	Requirements	61
3.4.1	R1: Provision of Unique Identifiers for Persons, Groups, and Roles across Namespaces	62
3.4.2	R2: Support for Distributed Identities	62
3.4.3	R3: Support for Distributed Groups	63
3.4.4	R4: Support for Distributed Roles	63
3.4.5	R5: Logging of DIM Activities	64
3.4.6	R6: Guarantee of Information Security	64
3.4.7	R7: Guarantee for Consistent Distributed Identity Information	65
3.4.8	R8: Support for Different Degrees of Technical Coupling	65
3.4.9	R9: Concurrent Operations	66
3.5	Design of the semDIM Core Ontology	67
3.5.1	Pattern for R1: Namespace Pattern for Unique Identification of Entities and Namespace Ownership	68
3.5.2	Pattern for R2–R4: Same Entity Pattern	69
3.5.3	Pattern for R3: Distributed Groups Pattern	70

3.5.4	Pattern for R4: Permission Patterns (for Distributed Roles)	70
3.5.5	Patterns for R5–R6: Signature Pattern, Certificate Pattern, and Graph Signing Pattern	72
3.6	semDIM’s Architecture and Secure Identity Information Exchange	74
3.6.1	Component-based Architecture of semDIM	75
3.6.2	Message Passing between the semDIM Components	77
3.6.3	Pattern-based Concurrency Control in semDIM	80
3.6.4	Data Ownership in semDIM	85
3.6.5	Summary	90
3.7	Application of semDIM to the Scenario	90
3.7.1	Example for R1: Namespaces and Unique Identifiers	92
3.7.2	Example for R2: Distributed Identifiers for Agents	93
3.7.3	Example for R3: Distributed Identifiers for Groups	94
3.7.4	Example for R4: Distributed Identifiers for Roles	94
3.7.5	Example for R5–R6: Signed Triples Using Graph Signatures	97
3.7.6	Example for R7–R9: Concurrency Control	97
3.7.7	Summary	98
3.8	Security Analysis of semDIM	98
3.8.1	Spoofing	99
3.8.2	Tampering	100
3.8.3	Repudiation	101
3.8.4	Information disclosure	101
3.8.5	Denial of Service	102
3.8.6	Elevation of Privilege	102
3.8.7	Summary	103
3.9	Synopsis	103
4	Conclusion	105
4.1	Lessons Learned	106
4.2	Future Work	108

List of Figures

1.1	The <i>Blue Train</i> project. This big picture depicts the core aspects of the scenario discussed in this thesis. It combines distributed identity management with managing different views on product-related information across organizations. The two companies, A and B, each have their own information system landscape and operate their own identity management system. For the project, these systems need to exchange identity information. Also, different views on the joint product <i>Blue Train</i> must be managed across companies. This scenario is inspired by a real-life industrial engineering and manufacturing project.	5
1.2	The <i>Blue Train</i> , fully assembled.	6
1.3	Different structure views of <i>Blue Train</i> . The dotted lined boxes represent the companies' responsibilities. The red arrow highlights a part switching responsibilities across lifecycle phases.	7
1.4	Semantic Web Stack, ordering Semantic Web technologies into multiple levels, adapted from [7]. The research focus for this thesis is highlighted in gray.	10
1.5	Ontology levels categorize ontologies regarding the abstraction level of their concepts, adapted from [70]. Core ontologies are highlighted, as the ontologies developed in this thesis belong to this category.	12
2.1	<i>Blue Train</i> with Bill of Materials (extract).	21
2.2	Design and element IDs of BRICK 2X2. The design ID belongs to the engineering view. The element IDs belong to the manufacturing view.	22
2.3	Product variants of <i>Blue Train</i> . The engineering view consists of only the functional aspects of the train. The manufacturing view also includes color.	23
2.4	Different structure views of <i>Blue Train</i>	24

2.5	Example information objects in a part-centered PLM model with four lifecycle phases.	28
2.6	Overview of the Design Patterns of CO-PLM organized into packages. The box with the bent corner contains the legend. This notation is used in all following figures describing ontology patterns. The numbered “R”s stand for the requirements R1–R4 from Section 2.4.	30
2.7	Product Part Information Entity Pattern. Entities of class <code>InformationEntity</code> are divided into <code>InformationObject</code> and <code>InformationRealization</code> realizing the former. Subsequently, <code>ProductPartInformationEntity</code> is divided into <code>ProductPartInformationObject</code> and its realization <code>ProductPartInformationRealization</code>	31
2.8	Product Part Information Entity Qualities Pattern. Entities of class <code>ProductPartInformationEntity</code> can have multiple PLM-related <code>Quality</code> s, for example, <code>Version</code> , <code>ApprovalStatus</code> , and <code>LevelOfConfidentiality</code> with their respective <code>Regions</code>	32
2.9	Part Information Object Pattern. Common examples of <code>ProductPartInformationObjects</code> are presented.	33
2.10	Part Entry Pattern. Each <code>PartEntry</code> consists of exactly one <code>ProductPartInformationEntity</code> providing a reference to the listed part as well as one <code>Measure</code> specifying the required quantity. A set of <code>PartEntries</code> form a BOM describing a <code>ProductPart</code>	34
2.11	Product Part Information Realization Pattern. Realizations for the Product Part Information Objects introduced in Figure 2.9 are presented. Concepts from IOLite [18] are reused and missing concepts are added.	36
2.12	Product Part Pattern. <code>ProductParts</code> can be either <code>ProductPartMasters</code> or their realization, <code>PhysicalProductParts</code>	36

2.13	Train example to illustrate the fundamental difference between the two kinds of Product Part Information Entities and the relations between them. The entities on the top row encapsulated by clouds, <code>ProductPartMaster</code> and <code>BillOfMaterials</code> , are <code>InformationObjects</code> , that is, abstract objects. The clouds represent the fact that these objects are not physical, just like a thought or idea about the entities they represent. The entities on the bottom row, <code>PhysicalProductPart</code> and <code>PrintedDocument</code> are <code>InformationRealizations</code> , that is, tangible physical objects.	37
2.14	DUL DnS Pattern. Created according to the axioms in [18].	38
2.15	Product Part Description Pattern. PLM-related roles and parameters can be defined by a <code>ProductPartDescription</code> describing a <code>ProductPart</code> . Such a description can be satisfied by a concrete <code>ProductPartSituation</code>	39
2.16	<i>Blue Train</i> Example Product Part Description Pattern. In this example instantiation of the Product Part Description Pattern, the <i>Blue Train</i> version 1.0 is described.	40
2.17	PLM Structured Workflow Pattern. A <code>PLMStructuredWorkflow</code> is a <code>StructuredWorkflow</code> extended by PLM-related concepts.	41
2.18	<i>Blue Train</i> Example PLM Weakly Structured Workflow Pattern. This example instantiation of the PLM Weakly Structured Workflow represents a workflow within a larger change management process. In the example, change board meetings led by the configuration manager are defined. In the first of these meetings, version 1.0 of <i>Blue Train</i> 's EBOM is defined.	42
2.19	Access Regulation Rule Pattern. An <code>AccessRegulationRuleMethod</code> extends a <code>Method</code> by concepts required for regulating access.	42
2.20	Example Application of the Access Regulation Rule Pattern. This example rule instantiation allows the agent <code>sa-1</code> to read the file <code>df-1</code> via the SSL-secured channel <code>ssl-1</code>	44
2.21	Semantic Engineering Process combining software and ontology engineering. The boxes with a bent corner represent artifacts. Regular boxes stand for information systems generating, transmitting (arrows), and processing these artifacts. Here, API means the API specification.	46

2.22	Reasoning time related to number of part types added step-wise from the <i>Blue Train</i> example.	47
2.23	Reasoning series for synthetic data. Each colored line represents a reasoning series. A series consists of multiple reasoning runs with increasing numbers of parts. As part numbers increase, the reasoning time of the run increases. . . .	48
3.1	Distributed identity management. In classical identity management (IM), all identity information of one organization is handled by one system. In multi-organizational collaboration, each organization may have its own system landscape and identity management. Distributed identity management (DIM) addresses integrating such independent identity management systems.	52
3.2	Identity management scenario with two companies A and B (with the corresponding namespaces <code>a:</code> and <code>b:</code>). Person 1 owns two identities, one in each namespace. The <i>Blue Train</i> project group is represented by two technical groups.	59
3.3	The two companies jointly designing the <i>Blue Train</i> . Company A designs the steam engine and the chassis, and Company B is responsible for the cabin.	60
3.4	Overview of the pattern of the semDIM core ontology, the imported ontologies, and how they match to functional requirements (see Section 3.4). The box with the bent corner contains the legend. This notation is used in all following figures describing ontology patterns. Red patterns are only needed for Scenario Variant II. The numbered “R”s stand for the requirements R1–R6 from Section 3.4.	68
3.5	Namespace Pattern. Entities can belong to multiple namespaces, including subnamespaces. Namespaces may have multiple owners, that is, agents having full control over defining entities in the respective namespace.	69
3.6	Same Entity Pattern. Subproperties to <code>owl:sameAs</code> are introduced for explicitly connecting related instances of persons, groups, and roles even across namespaces. The <code><instance></code> annotation indicates that these properties are used in the context of instances and does not mean the classes themselves are <code>owl:sameAs</code> or <code>owl:equivalentClass</code> . The latter is inherently true, however, as classes are always equivalent to themselves.	69

3.7	Distributed Groups Pattern. Distributed groups have members and administrating agents from potentially multiple namespaces.	70
3.8	Permission Pattern. Based on the DnS, all concepts required to define permission assignment workflows and their executions are introduced.	71
3.9	Group Permission Pattern. The Permission Pattern is extended by group-related concepts.	72
3.10	Group Permission Admin Example Instantiation. In the <i>Blue Train</i> scenario, the root identity <code>root-a-1</code> assigns <code>a-1</code> admin permissions for <code>group-1</code>	72
3.11	Signature Pattern. Concepts for signing arbitrary <code>dul:Things</code> , such as graphs, are presented. Adapted from [40].	73
3.12	Graph Signing Method Pattern. Different methods required for graph signing are presented. Adapted from [40].	73
3.13	Certificate Pattern. PGP and X509 certificates are supported. Adapted from [40].	74
3.14	semDIM client-server architecture featuring two companies, each owning one DIM Server and an arbitrary number of clients. Components in red are only required for the Scenario Variant II.	76
3.15	semDIM Message Model presented as a UML sequence diagram showing the client and server communication within and across company borders. The red parts are only required for Scenario Variant II.	79
3.16	Additional steps in the semDIM Sequence Diagram to allow locking.	81
3.17	Taxonomy of concurrency control algorithms, adapted from [44] and [87]. The algorithms combined in our approach are highlighted in gray.	83

List of Tables

- 3.1 Literature comparison against requirements from Section 3.4. . 57
- 3.2 Concurrency control approaches compared against requirements. 85

Listings

3.1	Namespace definitions defined by <code>a:root-a-1</code> for Company A.	92
3.2	Namespace definitions defined by <code>b:root-b-1</code> for Company B.	92
3.3	First identities and groups defined by <code>a:root-a-1</code> .	93
3.4	Identity connection defined by <code>a:root-1</code> . This is an instantiation of the Same Entity Pattern illustrated in Figure 3.6.	94
3.5	Group connection defined by <code>a:root-a-1</code> . This is an instantiation of the Same Entity Pattern illustrated in Figure 3.6.	94
3.6	Role connections defined by <code>a:root-a-1</code> declaring <code>a:a-1</code> to be the group administrator of <code>group-1</code> . These are instantiations of the Same Entity Pattern illustrated in Figure 3.6.	94
3.7	Group Permission Pattern instantiation showing Role connections defined by <code>a:root-a-1</code> declaring <code>a:a-1</code> to group administrator of <code>group-1</code> .	95
3.8	Role assignment defined by <code>b:b-1</code> , which is Person 1 acting in Company B's namespace, declaring <code>b:b-2</code> to be a group member of <code>group-1</code> .	96
3.9	Example of a signed graph applied on user data defined by <code>a:a-1</code> .	97

Acronyms

2PC	two-phase commit
2PL	two-phase locking
3PC	three-phase commit
ABAC	attribute-based access control
API	application programming interface
BOM	bill of materials
CAD	computer-aided design
CEO	chief executive officer
CIA	confidentiality, integrity, and availability
CO	core ontology
CO-PLM	Core Ontology for Product Lifecycle Management
CRM	customer relationship management
DDoS	distributed denial of service
DID	decentralized identifiers
DIM	distributed identity management
DO	domain ontology
DoS	denial of service
DUL	DOLCE+DnS Ultralight
EBOM	engineering bill of materials
ERP	enterprise resource management
FO	foundational ontology
FOAF	Friend of a Friend
IM	identity management
JoSSEP	Joint Software and Semantic Engineering Process
LED	Linked Enterprise Data
LOD	Linked Open Data
MBOM	manufacturing bill of materials
MGL	multiple granularity locking
MitM	man-in-the-middle (attack)

OM	Ontology of units of Measure
OS	operating system
OWL	Web Ontology Language
PBCC	pattern-based concurrency control
PDM	product data management
PLM	product lifecycle management
POM	project object model
PS2PL	primary site two-phase locking
RBAC	role-based access control
RDF	Resource Description Framework
SAML	Security Assertion Markup Language
semDIM	Semantic Distributed Identity Management
SPARQL	SPARQL Protocol and RDF Query Language
SSI	self-sovereign identities
SSL	Secure Sockets Layer
STEP	Standard for the Exchange of Product model data
STRIDE	spoofing, tampering, repudiation, information disclosure, denial of service, elevation of privilege
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	universal unique identifier
VPN	virtual private network
XACML	eXtensible Access Control Markup Language

Chapter 1

Introduction

Processing power and interconnectivity between computers increase rapidly. Organizations, e.g., companies, governments, universities, and non-governmental organizations, face an exponentially growing amount of different kinds of information. Relational databases are a well-tried approach to structure large quantities in a machine-processable way [52]. However, such databases are rigid in their structure, as they can only store data in the columns defined in advance. In the real world, objects have more dynamic and complex relationships with each other. Depending on the context, some information might be relevant in one domain, and at the same time, be considered negligible in another domain. For example, a person has general attributes, like name, birth date, and address. From a financial perspective, this person may also have one or multiple banking accounts, which have, in turn, a bank identification number, account number, and balance. From an academic point of view, the person has relationships with a university, subject of study, and a final degree. Data in relational databases can hardly dynamically depict such contexts.

A representation closer to the natural structure of relations between objects is graph data. Nodes are connected by edges that represent relationships between nodes. Graph data does not rely on a rigid number of possible properties and the data schema can be adapted according to an application's requirements [3]. Graph data can be considered as the inherent structure of data on the World Wide Web, as the relationships between elements form in an organic, non-predefined way [47]. Also, the meaning of data—the semantics—can be formally defined in a universal way, as graph data does not rely on a certain technical database implementation [76]. On the other hand, graph data possesses some major downsides compared to database structures, as most processing tasks, such as indexing, serialization, sorting, and search, can be computationally more

1. INTRODUCTION

complex in graphs. However, graph databases can outperform relational databases with increasing numbers of relationships [86].

Technologies addressing the challenges of structuring and processing information in a semantic way are called Semantic Web technologies [27]. Linked Open Data (LOD) describes the principle of unambiguously identifying and referring to resources or “things” on the World Wide Web using uniform resource identifiers (URI) [5]. LOD allows integrating information from various sources and in different forms, such as information about people and their relationships with each other (Friend of a Friend¹) [9], information about geographical positions², or information about documents³. The benefit of automatically harnessing information from different sources is most apparent on the public Internet, where hosts are easily accessible and provide resources with links to other hosts’ resources. Because of this, Semantic Web technologies are most commonly applied in the context of public network applications.

However, there is increasing interest in using these techniques in private or protected networks, such as corporate intranets. Linked Enterprise Data (LED) is the counterpart to LOD. LED also relies on URIs but addresses technical and administrative differences related to connecting data in a company context in contrast to public data [28, 65]. LED applications can be divided according to three scenario categories: data integration (i. e., for knowledge portals or semantic search), import data from the (public) Linked Data Cloud, and publish data to the Linked Data Cloud [6]. The need for security mechanisms in such applications, particularly in a business context, is recognized in the literature [4]. However, the Semantic Web reference architecture only offers “a rather abstract representation of security” [40].

In this work, the term “protected network” refers to a network in which most systems are not directly connected or accessible via the Internet. While this is a rather vague definition, protected networks are generally synonymous with settings like company intranets and site-to-site virtual private networks of project partners. In a protected network setting, data processing is often more regulated. Policies, such as corporate guidelines and instructions, restrict the officially allowed forms of data structure. However, such restrictions often only are present in a human-understandable format, i. e., text written in prose. This lacks sufficient formalization to enable machines to automatically process that data or even correlate multiples of

¹<http://www.foaf-project.org/>, last accessed on 2021-01-16

²<https://www.w3.org/2003/01/geo/>, last accessed on 2021-01-16

³<http://dublincore.org/documents/dcmi-terms/>, last accessed on 2021-01-16

1. INTRODUCTION

such structures. In contrast, Semantic Web technologies aim to provide information representations readable by both humans and machines.

This thesis focuses on comparing Semantic Web technologies in the context of multi-organizational collaboration in detail. Applying Semantic Web technologies into protected networks was proposed and executed previously, according to the literature [27]. However, the literature is often limited to adapting existing solutions into protected environments, for example, implementing a semantically enriched Wiki-based knowledge management system or a semantic search processor into a company’s intranet. These are legitimate generic uses of Semantic Web technologies in protected networks. However, this thesis focuses on addressing problems specific to protected networks.

Complex policy and regulation frameworks implemented in companies are an example of problems specific to protected networks. In general, applications on the public web also have policies; for example, only users who “had an account for about 4 days and have made at least 10 edits”⁴ may start new articles. Policies in companies are usually far more complex, involving several attributes of an employee, such as security clearances, department, job position, and tenure. Furthermore, information systems may have varying security requirements in terms of confidentiality, integrity, and availability.

This thesis focuses on multi-organizational collaboration. Here, the above-mentioned challenges of managing information processing across distributed information systems become most apparent. Interfaces, data protocols, and security policies all must be matched to allow for functional and secure processes.

The following section introduces a motivating scenario, scoped to a limited—yet self-contained—problem space. The scenario is inspired by a real-life joint venture constellation⁵, featuring multiple involved parties with varying forms of relationships. This scenario serves as a “big picture” for further discussion on the challenges mentioned above. Chapters 2 and 3 will pick up the scenario and give further details in their problem description. Sections 2.3 and 3.3 guide the reader in recognizing the connections between the research in the two main chapters and the scenario.

The remainder of this introductory chapter is structured as follows: Section 1.1 presents the scenario mentioned before and identifies the two main challenges within this scenario. Next, Section 1.2 uses these two main challenges to formulate two respective research questions about addressing

⁴https://en.wikipedia.org/wiki/Wikipedia:Why_create_an_account%3F, last accessed on 2021-01-16

⁵The author of this thesis has over ten years of professional experience in industrial manufacturing- and engineering-related projects.

the challenges with semantic technologies. Section 1.3 then offers a general discussion of the Semantic Web technology stack and identifies technologies relevant to the research questions. Based on the above two sections, Section 1.4 formulates the contribution of this thesis, summarizing the approaches to address the two challenges.

1.1 Motivating Scenario: The *Blue Train* Project

Businesses rely on a diverse landscape of processes, information systems, and digital services. Almost none of the usual business processes, such as procurement, sales or production, are managed without any form of digital support. Examples of essential business information systems common to many industries are enterprise resource planning (ERP) for data with a commercial focus, product lifecycle management (PLM) for handling all product-related data over all product lifecycle phases, and customer relationship management (CRM) to track interactions with and data about customers. However, most companies do not rely only on out-of-the-box IT services but use an individual orchestration of different information systems based on distinct architectures, offering a range of standard and custom services. Interoperability and data exchange become even more complicated when considering the processes and information systems of partners, customers, suppliers, and service providers. In an international project involving multiple parties from different countries, even more technical, legal, language-related, and cultural differences must be considered when connecting information systems. Project-specific information, such as product information, must be exchanged in a form processable by multiple parties' information systems. Furthermore, identities, groups, and roles as the basis for security policies must be managed across organizations. Identity management (IM) regarding the systems of one company is a challenging task. However, this thesis addresses the even more complex setting of distributed identity management (DIM) involving the integrated identity management of multiple organizations.

The following scenario (see Figure 1.1 describes a typical setting for an international industrial engineering and manufacturing project. It is inspired by a real-life project but uses pseudonymized company names and products. The scenario features two companies, Company A and Company B, collaborating on a joint industrial project. The steps required to establish a

1.1. MOTIVATING SCENARIO: THE *BLUE TRAIN* PROJECT

collaboration environment for these companies and the processes during the project realization are the subject of this scenario.

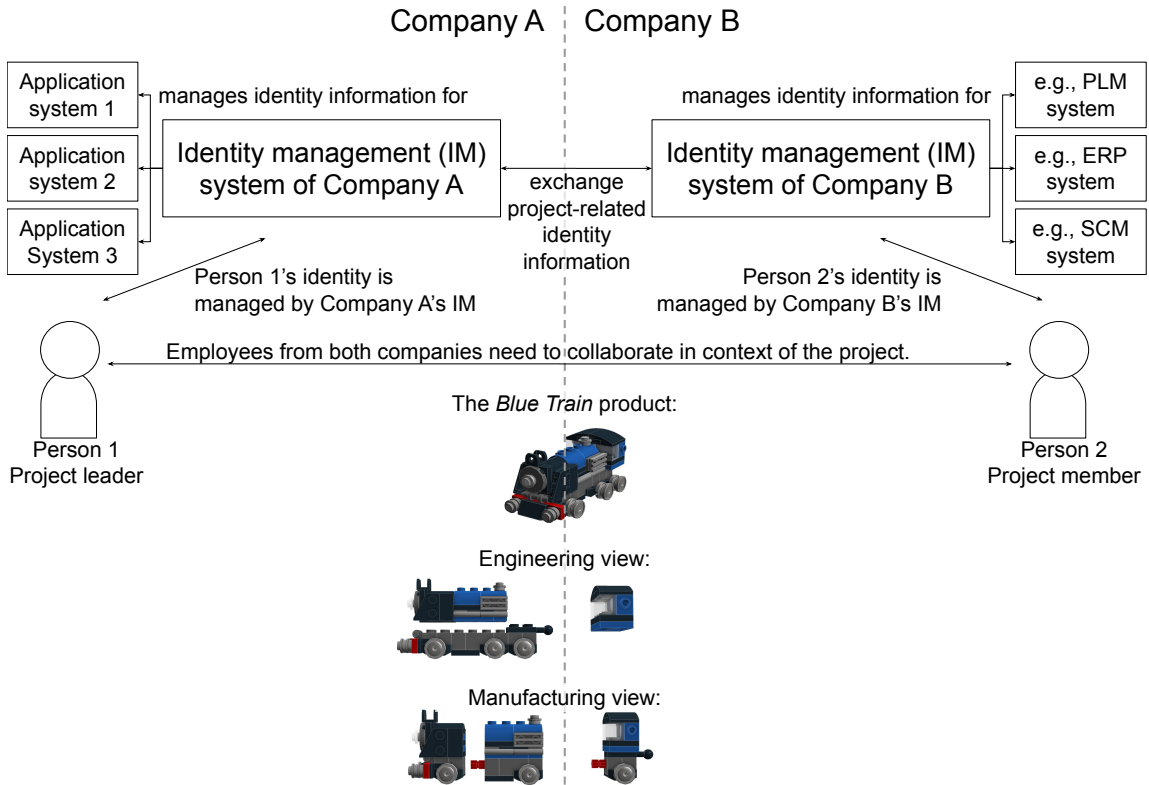


Figure 1.1: The *Blue Train* project. This big picture depicts the core aspects of the scenario discussed in this thesis. It combines distributed identity management with managing different views on product-related information across organizations. The two companies, A and B, each have their own information system landscape and operate their own identity management system. For the project, these systems need to exchange identity information. Also, different views on the joint product *Blue Train* must be managed across companies. This scenario is inspired by a real-life industrial engineering and manufacturing project.

We use a fictional product represented by a LEGO model. Using such a fictional demonstration object helps when discussing fundamental PLM concepts and challenges without the danger of potentially disclosing confidential real-life information. Figure 1.2 shows the LEGO Set 31504 “Blue Express.” The fully assembled model, which we will simply refer to as the *Blue Train* product for the remainder of this thesis, is the final product

1.1. MOTIVATING SCENARIO: THE *BLUE TRAIN* PROJECT

in our scenario. This model is particularly suitable for PLM demonstration purposes because its structure can easily be divided into two different views: its functional view and its manufacturing view.

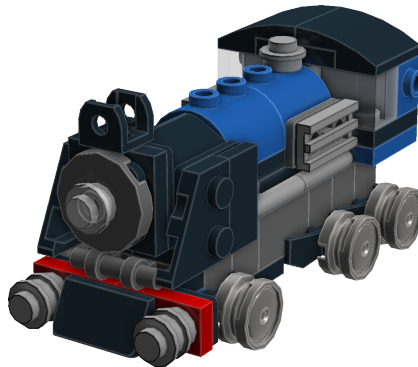
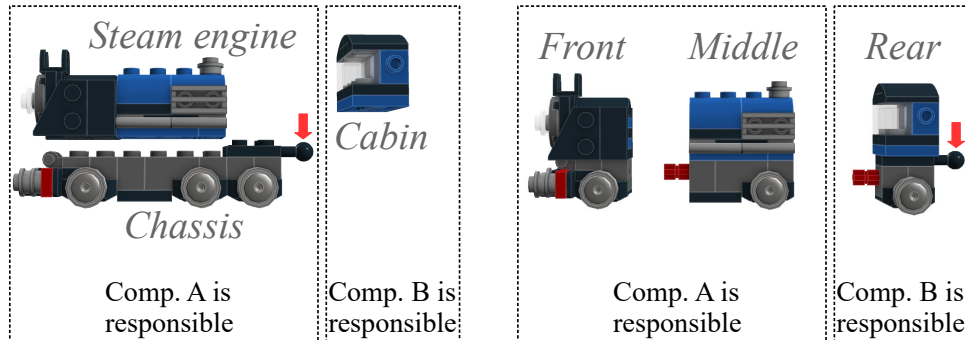


Figure 1.2: The *Blue Train*, fully assembled.

While the functional view features groups of parts belonging together on a technical level, that is, all parts involved in generating power for the train are consolidated in the steam engine, the manufacturing view represents sub-assemblies as they are independently pre-built before being merged into the final product (see Figure 1.3). Further details on such different views, as well as why they are necessary and useful in PLM, are discussed in Section 2.3. A typical challenge in industrial collaboration is the fact that multiple companies are responsible for different components of the final product. In our scenario, Company A is responsible for designing the steam engine and the chassis, and Company B is responsible for the cabin (see Figure 1.3a). The responsibilities for specific parts vary across different life cycle phases. For example, Company A is responsible for the front and middle sub-assemblies, and Company B builds the rear part (see Figure 1.3b). Thereby, some of the product's parts are in the responsibility of Company A in one phase but become the responsibility of Company B in a different phase (see the arrow in the bottom right of the Figures 1.3a and 1.3b marking the same product part switching company responsibility).

This one simple example from our scenario demonstrates that PLM can involve complex relationships between parts across different life cycle phases or views as well as companies. In a real-life setting, the constellations between product parts, companies, and phases are more complex. Most products have significantly more sub-assemblies compared to the *Blue Train*, and more companies are involved, including subsidiaries, sub-contractors, off-set companies (i. e., companies involved due to a contractual minimum



(a) Engineering view of *Blue Train* based on functionality.

(b) Manufacturing view of *Blue Train* based on manufacturing steps.

Figure 1.3: Different structure views of *Blue Train*. The dotted lined boxes represent the companies' responsibilities. The red arrow highlights a part switching responsibilities across lifecycle phases.

for the economic value added in a specific country), or the customers themselves being involved in the product development. For the sake of this demonstration, we reduce the complexity of our scenario to Companies A and B, as this constellation is sufficient to illustrate the main challenges of the scenario.

Two challenges of this collaboration are highlighted in this thesis. We refer to common industry standards that define the subjects of these challenges to illustrate the challenges' general relevance using established terminology.

1.1.1 Challenge 1: Managing Product-related Information

The first challenge addresses **creating, managing, and distributing product-related information over time and across organizational borders, commonly called product lifecycle management (PLM)**. As discussed above, various information systems, agents, organizations, processes or workflows, and technical requirements are involved in the design, manufacturing, usage, and termination of a product. Therefore, PLM is a major challenge in engineering and manufacturing projects, especially in settings involving international collaboration. Section 2.3 elaborates details of PLM in the context of this scenario.

When addressing the central technical challenge of PLM—interoperability between information systems, involved parties, and different business

processes—the classic approach is to refer to standards. For example, IEC 62264 “Enterprise-control system integration” [35] or ISO 10303 “Automation systems and integration—Product data representation and exchange” informally known as “Standard for the Exchange of Product model data” (STEP) [1] are technical standards addressing PLM. Regarding interoperability, such standards generally try to establish common data (file) formats, which result in some major drawbacks: The degree of formalization is either too weak or too strong. The formalization is too weak if it is not sufficient to enforce genuine technical interoperability of system implementations and allows subjective interpretation about its implementation. On the other hand, formalization may be too rigid/excessive for the use case if its full implementation is not compatible with economic or technical capacities. In the latter case, standards may be split into multiple modular parts, for example, the several hundred parts of ISO 10303, possibly resulting in confusion. Furthermore, new and innovative concepts cannot be implemented while ensuring standard compliance. As standards are based on best practices, integration of innovative concepts in standards is often delayed.

This thesis elaborates a semantic approach to these shortcomings by offering a semantic basis to both using standards and formalizing information outside of the scope of the standards used. The advantage of the semantic approach is that the “meaning of data” can either be reused from existing standards or, where necessary, provided with semantic technologies, such as ontologies. The semantic approach thereby allows reusing standards’ concepts while adjusting the degree of formalization as needed and integrating new concepts in a modular way.

1.1.2 Challenge 2: Distributed Identity Management

The second challenge concerns **distributed identity management (DIM), including roles and groups, as a foundation for administrating permissions and enforcing policies to ensure confidentiality, integrity, and availability of project-relevant information and systems**. PLM requires robust identity management. Many workflows in PLM, such as those within change and configuration management, rely on the definition of roles assigned to groups and agents to determine the legitimacy of actions or who is to be notified by certain actions. ISO 24760 describes identity management as containing “processes and policies involved in managing the lifecycle and value, type and optional metadata of attributes [...] in identities [...] known in a particular domain” [36]. This norm defines **identity** as a “set of attributes,” which are

synonymous to properties, “related to an entity.” Examples for **attributes** are entity type, address information, telephone number, a privilege, a MAC address, or a domain name. An entity can be any real-life object, such as a person, a group, an organization or an information system. Every entity may have several identities. **Identifiers** distinguish one entity from another. A commonly used identifier is the universal unique identifier (UUID). An identity is represented in information and communication technology systems by **identity information**, which serves as metadata for the identity. Identity management is **distributed** if multiple points of enrollment and access to registered identity information exist. Management of the above concepts, groups, and roles is needed in our scenario, as the workflows crossing company borders require DIM. The details on how to address DIM in an international industrial collaboration setting are presented in Section 3.3. Based on the two identified challenges, the next section defines two corresponding research questions.

1.2 Research Questions

As stated at the beginning of this chapter, the main goal of this thesis is to develop new solutions based on Semantic Web technologies addressing specifically protected network requirements. With Challenges 1 and 2 from Section 1.1 in mind, the following research questions arise:

Research Question 1: How can Semantic Web technologies be used to support secure creation, exchange, and management of product information across organizations as well as lifecycle phases? This question includes the representation of product information, its change management over time, different organizations’ views on products depending on their role, and a basis for formulating policies regulating the processing, i.e. accessing, transmission, and usage, of product information. The first research question only addresses the general possibility of using identity management entities, including identities, roles, and groups, to regulate the processing of information; in our scenario: product information.

Research Question 2: How can Semantic Web technologies be used to support secure management of distributed identity management in protected environments? How these identity-related entities integrated into a scenario-specific knowledge representation need to be managed is the subject of the second research question. For example, they

need to be created, changed, and exchanged in a distributed setting (see the scenario in Section 1.1).

1.3 Approach

This section discusses the current state-of-the-art of the Semantic Web technology stack and elaborates promising technological approaches for addressing the research questions. In this thesis, a set of technologies from the Semantic Web Stack is used. Figure 1.4 shows the Semantic Web Layer Stack, a typical overview of technologies commonly referred to as Semantic Web technologies [7].

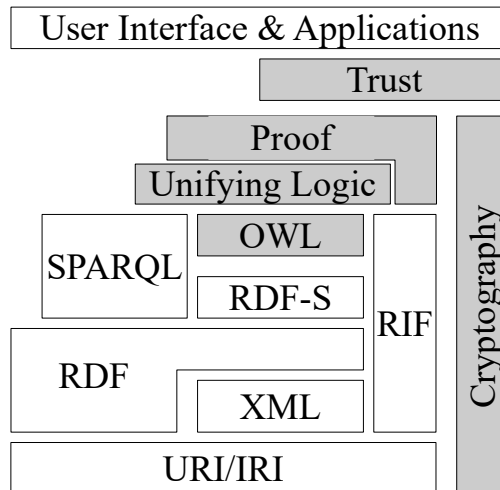


Figure 1.4: Semantic Web Stack, ordering Semantic Web technologies into multiple levels, adapted from [7]. The research focus for this thesis is highlighted in gray.

This thesis addresses the use of all layers depicted in this figure in the context of the scenario from Section 1.1, but some of the layers receive particular attention (see gray background in Figure 1.4). These layers are the most relevant for addressing the identified challenges in distributed collaboration. The Semantic Web Stack is structured from fundamental technologies on the bottom to increasingly complex concepts at the top that build on the lower tiers.

Using fundamental Semantic Web concepts: This work builds on the technologies from the lower levels, although those layers are not the

focus of this thesis' research. For example, the work will use universal resource identifiers (URIs) and format information compliant to the Resource Description Framework (RDF) standard when referring to and describing various entities and properties. Similarly, SPARQL Protocol and RDF Query Language (SPARQL) will be used as the standard query language for RDF data. The Web Ontology Language (OWL) can also be considered a fundamental semantic technology in terms of this categorization, as it is well specified and already commonly used in applications. The OWL layer is highlighted in Figure 1.4 because this thesis heavily relies on using ontology patterns for knowledge patterns as one of the thesis's focus points.

Focus on higher-level Semantic Web concepts: The higher concepts in the stack become increasingly abstract. “Proof” or “Trust” are not really technologies in terms of a technical specification, in contrast to, for example, “RDF”, which is a precisely defined standard. As the top layers are rather coarsely defined, they offer ground for research, especially in the light of a use case that was not within the mainstream of the attention of the Semantic Web community. Trust addresses the question of how to validate if information is provided by a legitimated source. In the context of our scenario, the companies establish trust between one another and must ensure that exchanged data comes from their trusted partners. With proof, exchanged data can be verified, that is, checked for factual correctness. For example, agents claiming to have legitimization for certain actions must provide proof (or that proof must already be available for the controlling system). How such aspects of data management can be supported in a Semantic Web application is subject to ongoing research, such as this thesis.

Ontologies as a central basis for semantic knowledge representation and reasoning: We utilize ontologies to formalize classes of entities and the relationships among them. These ontologies offer a framework in which we can agree on the meaning of various concepts and enable interoperability. Gruber [23] offers a definition: “An ontology is an explicit specification of a conceptualization.” Ontologies form the bridge between human-readable directive texts and machine-processable directives, as they translate the concepts of things—often written in prose by humans—into technical information structures. By formalizing classes of entities and the relationships among them, ontologies offer a framework to commonly agree on the meaning of various concepts and enable interoperability between systems relying on these ontologies. Thereby, ontologies are both a tool for representing knowledge in a formalized way and the foundation of reasoning.

1.3. APPROACH

That is, ontologies allow checking statements for their consistency with the rules of the model and detecting potentially erroneous data and generating new knowledge by using explicitly stated relationships between entities and properties to infer additional, implied relationships, such as those used for classification of entities, using software systems called “reasoners.” In the context of our scenario from Section 1.1, these functionalities of ontologies and reasoners may serve to check product information and structures for consistency, workflows for their validity, and identities and their roles and groups for their correct assignments.

Ontologies can be categorized according to their level of generalization (see Figure 1.5). Foundational ontologies (FO) include generic concepts applicable in various contexts, while domain ontologies (DO) contain specific classes for a certain domain scope. Core ontologies (CO) can serve as a bridge between FOs and DOs. By defining core concepts relying on the foundational concepts of an FO, a CO can offer a reusable, more concrete concept for multiple domains while still ensuring compatibility with the underlying FO. Ontologies can help in integrating different information systems using middleware [61].

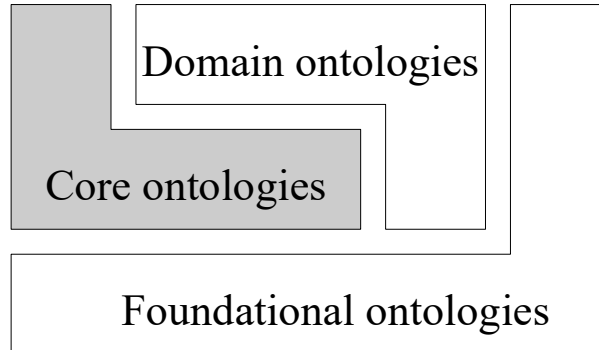


Figure 1.5: Ontology levels categorize ontologies regarding the abstraction level of their concepts, adapted from [70]. Core ontologies are highlighted, as the ontologies developed in this thesis belong to this category.

In our ontology engineering process, we reuse and develop ontology design patterns (in short: ontology patterns). An ontology (design) pattern is “a modeling solution to solve a recurrent ontology design problem”[20]. The use of ontology patterns allows modular and extendable adaptation of our ontologies.

After identifying Challenges 1 and 2 in Section 1.1, deriving research questions in Section 1.2 from those challenges, and analyzing the Semantic

Web Stack in the context of those research questions in this section, we now use the results of the previous sections to formulate the contribution of this thesis.

1.4 Contribution

This thesis develops solutions built on core ontologies to define reusable semantic frameworks of relationships between entities in different domains to support business applications, such as those presented in the scenario in Section 1.1. Two technological approaches were identified. Each results in an implementation of suitable Semantic Web technologies reusing existing approaches and developing new techniques. The contribution of this thesis is to present the results of these approaches, report how they were acquired, and present benefits that no state-of-the-art solution currently can offer regarding the challenges addressed in the research questions.

Technology Approach 1: Using ontologies for modeling products and workflows in product lifecycle management To provide a basis for further research, a core ontology, CO-PLM, is proposed for the elements of PLM. This approach addresses Challenge 1 (managing product-related information). Relying on general concepts, DOLCE+DnS Ultralite [18] is used as a foundational ontology. In Chapter 2, we analyze how a core ontology must be designed to reuse the concepts of the foundational ontology in the context of multi-organizational collaboration. We evaluate existing ontologies and extend or design ontologies suitable to describe the necessary concepts in PLM. Furthermore, we evaluate the application of CO-PLM in the context of PLM and integrate it into a software engineering process.

Technology Approach 2: Using ontologies for representing distributed identities and exchanging them across companies Second, a framework is developed for managing identity management entities, that is, identities, roles, and groups, as well as their relationships. This approach addresses Challenge 2 (DIM). Existing concepts are evaluated and extended to meet the needs of an industrial collaboration setting. A promising existing approach is called “distributed group management using Friend of a Friend.” ($_{dg}$ FOAF) [75]. $_{dg}$ FOAF supports creating and administrating distributed ad-hoc groups but lacks more sophisticated concepts, such as a role model. $_{dg}$ FOAF only distinguishes between administrators and normal users but does not allow a more diverse role model and does not feature a way to formalize the permissions associated

1.4. CONTRIBUTION

with such roles. Also, d_g FOAF only considers processes like authorization, authentication and privilege maintenance in a rudimentary way that is not sufficient for a business application. This thesis will present a more sophisticated approach to also address requirements emerging from a multi-organizational collaboration.

As discussed in the previous sections, it is essential to embed security as a cornerstone into solutions addressing modern information exchange. We consequently rely on best practices regarding ontology engineering, such as extending general concepts by integrating foundational, core, and domain ontologies or ensuring modularity and reusability by implementing ontology patterns. Furthermore, we extend existing concepts and technologies with features not previously supported but relevant to the target applications. Applying Semantic Web technologies to protected networks is not yet a sufficiently researched field of interest. We validate our findings by evaluating them against the requirements in the context of Section 1.1.

Chapter 2

Secure Product Lifecycle Management with CO-PLM

This chapter introduces CO-PLM, a core ontology for secure, pattern-based product lifecycle management. CO-PLM was initially published in [71]. Further discussion, particularly regarding the ontology patterns, was presented in [73]. This chapter is based on these works but adjusts them to the structure of this thesis and enriches parts of them by adding further details and aspects. For example, the chapter provides an extended discussion on the related work and more detailed pattern descriptions.

Global markets and cooperations with international partners provide several benefits but require a complex Product Lifecycle Management (PLM). Several parties take part in the different phases of a product's lifecycle. A major challenge of PLM is the exchange of product data across lifecycle phases, information systems, and parties. Data formats, structure, the quality and form of interfaces, and levels of confidentiality can vary significantly. In scenarios with different confidentiality levels, not every party can access all product data. Furthermore, processes must be adaptable as actors' roles might change depending on a project's phase. Also, versions of information valid at different times, locations, or contexts must be considered. Existing PLM approaches focus only on certain phases of PLM, predominantly design and manufacturing, while the subsequent phases of usage/maintenance and disposal are often ignored. However, the usage phase is becoming increasingly important for revenue as customer expectation for services beyond the initial purchase of a product is growing. This chapter proposes the Core Ontology for PLM (CO-PLM) based on the foundational ontology DOLCE+DnS Ultralite to provide a formal basis for PLM. In contrast to existing approaches, CO-PLM follows a holistic approach covering all phases of PLM and integrates patterns from existing core ontologies.

2.1 Background and Structure of This Chapter

Product Lifecycle Management (PLM) sums up all activities regarding the design, production, usage, and termination of products. It includes the processes required to take a product from the first ideas of its design to the disposal of the product at the end of its life [81]. A central challenge of PLM is sharing information and knowledge between different organizational parties, such as departments, locations, and companies, etc., over the whole lifecycle of a product [58]. This sharing is not a trivial task because companies use many different software systems are used by companies today to support their business processes. These systems often rely on proprietary data models and lack interoperability with one another. The need for international cooperation grew because of two trends.

First, increasingly global markets require businesses to be more competitive in price and quality. This trend extends the supply chain, as remote partners need to be integrated. Furthermore, integration requires more sophisticated concepts for data exchange and workflows with business partners. Such cooperation with international partners can yield significant advantages, such as foreign market access, cost optimization by division of labor, or access to specific know-how. However, corporations face several challenges regarding political, legal, or infrastructural conditions in different countries.

Second, digital devices advance rapidly in computing abilities and interconnectivity. On the one hand, this digitization allows for faster and easier transfer of (product) data. On the other hand, digital accessibility over the Internet endangers the confidentiality and integrity of corporations' information. Today, most organizations' internal networks expose interfaces to the Internet. Organizations need to be able to integrate their processes and information systems with their partners' and customers' processes and information systems. Additionally, mobile clients, such as laptops, mobile phones, and embedded devices integrated into either a company's products or its processing machines require gateways into the internal networks, such as virtual private networks (VPNs) or cloud applications. As a growing amount of information becomes accessible to a wider range of agents, including people, groups, or technical agents, information systems must be able to categorize, classify, and protect information based on an unambiguous framework of rules and data models. Formal ontologies can be used to represent such complex networks of entities and their relations independently from data formats. Multiple types of sensitive data exist, including personally

2.1. BACKGROUND AND STRUCTURE OF THIS CHAPTER

identifiable information (PII) or trade secrets, but this work focuses on product-related data for addressing PLM.

This chapter focuses on “high-technology” products. Technology may be referred to as high-tech if a certain degree of research and development is observed [24]. Several technologies meet this definition, such as machines or subassemblies in astro- and aeronautics, automotive, biotechnology, medicine, and similar sectors. The production processes, data models, and business requirements differ significantly between discrete manufacturing (e. g., assemblies) and process manufacturing (e. g., chemical and pharmaceutical products). Services, software, or chemicals do not require essential concepts of PLM, such as spare parts management or configuration management of sub-components, such as recording serial numbers of exchanged equipment. Chemicals do not have components that could be easily exchanged or tracked separately. Similarly, a software program does not physically wear out by extensive use and does not need to be replaced by an identical copy when losing functionality. Therefore, the ontology developed in this work covers functionality beyond the requirements of products that are produced by discrete manufacturing. Although our ontology can be applied to the former or to any other kind of product, we focus on discrete manufacturing in this work, as this is the targeted application matching our scenario in Section 1.1.

Several models exist for dividing a product’s lifecycle into separate phases. In general, these consist of the **four main phases**, which are product design, manufacturing, usage/maintenance, and disposal, describing the early, middle, and late life of a product. Depending on the specific product, the phases may involve different business processes. In the context of manufacturing, the **design phase** consists of requirements engineering, product specification, and prototyping. This phase starts with the very first idea of a new product. Most of research and product development takes place in this phase. Next, the **production phase** describes the processes around procurement and disposition of the product’s components as well as its assembly. In the **usage phase**, the product is sold to the customers. Most of spare parts management and customer service management takes place in this phase. The product lifecycle ends with the **disposal phase**. The product gets either replaced, recycled, or eliminated when it is no longer useful.

The activities in all phases may differ depending on the components of the product. Most high-tech products consist of multiple types of components, such as electrical, electronic, mechanical, hydraulic/pneumatic, or software. Certain pieces of information are created and must be managed for each of these component types. This process can be digitally supported

by product data management (PDM) software systems. The most common data of a high-tech product’s lifecycle to be managed are its metadata, such as identification numbers, spatial dimensions, weight, or storage conditions. Depending on the industry, further artifacts are relevant, including technical specifications, geometrical drawings and 3D-models usually summarized as computer-aided design (CAD) data, manuals, and test reports.

After discussing the thematic background, this thesis continues in the following way: The related work is discussed in Section 2.2, where we analyze existing ontologies for PLM. Next, Section 2.3 contains a detailed problem description referring to the scenario from Section 1.1 for PLM using the *Blue Train* product. In Section 2.4, the requirements for a core ontology for PLM are described. As none of the ontologies in the related work cover these requirements entirely, we present the pattern-based design of CO-PLM in Section 2.5, followed by a detailed discussion of the ontology design patterns and example instantiations in Section 2.6. Section 2.7 investigates how the use of CO-PLM scales with increasing numbers of product parts in terms of axiom count and reasoning time before the results of this chapter are summarized in Section 2.8.

2.2 Related Work

Several ontologies covering concepts of PLM can be categorized into three groups: engineering [16, 63, 32], manufacturing [51, 63, 32], and lifecycle ontologies [10, 56]. **Engineering-centered ontologies** classify product features, e.g., drilling holes or edges, while **manufacturing-centered ontologies** focus on describing resources and processes on the shop floor level, e.g., machine, schedule, and production steps. In contrast to these, **lifecycle-centered ontologies** try to describe product lifecycle management comprehensively by focusing on more general concepts. They may also refer to products and machines, but not as explicitly as ontologies of the other two types. They rather focus on activities connecting resources and information from different lifecycle phases. In comparison to engineering and manufacturing centered ontologies, lifecycle-centered ontologies also address a more abstract meta-level of Product Data Management. Most literature and software solutions for PLM focus on the design and manufacturing phases, as these two are considered to have the most business impact [50]. The model presented in this work, CO-PLM, shares this view but extends the state of the art by also supporting the remaining phases, especially the usage phase.

2.2. RELATED WORK

The following paragraphs discuss representatives of the related work for each of the three groups a), b), and c).

a) Engineering-centered Ontologies An example of an engineering-centered ontology is the Core Product Model [16]. The central concepts are **Function**, **Form** and **Behaviour**. These concepts connect product information about “what is it intended to do,” “what is its shape and material,” and “how it implements its function.” By focusing on these design-related topics, the Core Product Model is clearly an engineering-centered ontology. It is UML-based, and an XML implementation is demonstrated.

b) Manufacturing-centered Ontologies ADACOR (ADaptive holonic Control aRchitecture for distributed manufacturing systems) [51] describes a model to represent dynamic changes on the shop-floor level in manufacturing. It is based on the holonic manufacturing paradigm, which consists of agents (holons) acting autonomously and cooperatively. ADACOR aims thereby at combining centralized control with decentralized autonomy. As ADACOR focuses on production control at the shop floor, it belongs to the manufacturing-centered ontology category.

Both a) Engineering-centered and b) Manufacturing-centered ontologies Because the following ontologies combine engineering as well as manufacturing aspects, they belong to categories a) and b) at the same time. They address interoperability between these two disciplines but do not offer the abstract concepts required for supporting other phases (especially usage) and cannot be labeled lifecycle-centered ontologies. The product-driven ONTOlogy for Product Data Management (ONTO-PDM) [63] aims at supporting the interoperability of application software (e.g., product data management, enterprise resource planning, and manufacturing execution systems). It is based on the concepts **ProductDefinition**, **Material**, **Equipment**, **Personnel**, **ProcessSegment**, **ProductionSchedule**, **Production Cabability** and **ProductionPerformance** from the norm IEC 62264¹ [35] and extends these by concepts from ISO 10303² [37]. As ONTO-PDM includes engineering- as well as manufacturing-related concepts, it can be assigned to both groups a) and b), but it is not an approach trying to cover all lifecycle phases. Imran and Young [32] suggest an Assembly Reference Ontology for sharing assembly knowledge. It is focused on assembly processes

¹IEC 62264 “Enterprise-control system integration”

²ISO 10303 “Automation systems and integration—Product data representation and exchange,” also known as “Standard for the Exchange of Product model data” (STEP)

2.3. PROBLEM DESCRIPTION

within the manufacturing process. The ontology relies on upper-level ontologies, named Upper Level Ontology and Middle Level Ontology, and was created by Highfleet Inc. It uses the Knowledge Framework Language, which is based on Common Logic, and its Integrated Ontology Development Environment. The work focuses on resolving conflicts in concepts like **Bill of Materials**, **product family**, and **feature** between assembly design and assembly process planning by using a shared reference ontology and therefore belongs to categories a) and b). The work categorizes the abstraction levels of reference ontologies into generic, product lifecycle, design and manufacturing, assembly specific, and assembly design and assembly process planning concepts.

c) Lifecycle-centered Ontologies An example of a lifecycle-centered ontology is suggested by Bruno et al. [10], focusing on small- and medium-sized enterprises. The ontology addresses the problem of linking different pieces of product information by offering a simple and general model, including concepts like **product**, **role**, **file**, and **activity**. The ontology is written in the Web Ontology Language (OWL)³. Matsokis and Kiritsis [56] present an ontology implementation of the UML-based PROMISE Product Data and Knowledge Management (PDKM) system [11], which can also be considered a lifecycle-centered ontology. This model is oriented around product instances or product items, represented by the **physical_product** class, which are related to **as_designed_product** and **events** of the **life-cycle_phases** beginning of life, middle of life, and end of life (of a product). In summary, a variety of core ontologies address different aspects of PLM. However, a) and b) do not cover all phases. Although ontologies of category c) cover the whole product lifecycle, they do not integrate all concepts, such as for workflows and security regulations, required for multi-organizational collaboration.

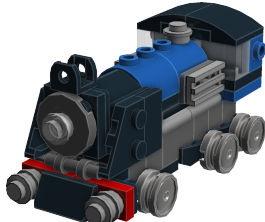
2.3 Problem Description

This sections discusses Challenge 1 from the scenario in Section 1.1 in more detail using the *Blue Train* product. This product is used to illustrate several PLM concepts, including part identification, subpart relationships, and assembly processes. Figure 2.1 shows the *Blue Train* and an extract of its Bill of Materials (BOM). The model consists of 71 individual bricks of 32 different brick types. The model serves as an illustration of both the need

³<https://www.w3.org/TR/owl-semantics/>, last accessed on 2021-01-16

2.3. PROBLEM DESCRIPTION

for a co-design process for pattern-based core ontologies and pattern-based software models.







Element-ID	Name	Picture	Design-ID	Color code	Quantity
4211098	BRICK 1X1		3005	199 - Dark Stone Grey	2
306540	BRICK 1X2 WITHOUT PIN		3065	40 - Transparent	2
4211060	BRICK 2X2		3003	199 - Dark Stone Grey	1
4583862	BRICK 1X1 W. 1 KNOB		87087	23 - Bright Blue	4

Figure 2.1: *Blue Train* with Bill of Materials (extract).

A BOM “lists the subassemblies, intermediate assemblies, component parts, raw materials, and quantities of each needed to produce one final product” [67]. Therefore, the BOM can be different for the design phase, called an engineering BOM (EBOM), in comparison to the production phase’s manufacturing BOM (MBOM). The BOM shown in Figure 2.1 is generated by the LEGO Digital Designer⁴. The most common attributes of a BOM’s entries are an ID and an amount for each ID. Other common content is the color, pictures of the parts, materials, or units, particularly when liquids, gases, or bulk stock are used. The following sections illustrate differences between an EBOM and an MBOM.

2.3.1 Engineering and Manufacturing View of Parts

LEGO bricks have two different IDs, that is, element IDs and design IDs. The design ID defines the geometry of a LEGO brick. For example, a two by two standard brick will always have the same design ID, regardless of color or graphics printed on it (Figure 2.2, left side). The element ID, on the other hand, refers to a specific LEGO brick, including its design ID, color, and any printing, that can be purchased from the parts shop or referenced in a building manual (Figure 2.2, right side).

This distinction illustrates different views of the same part, similar to a real-life scenario. An engineer is more interested in the form, fit, and function of a product or a part, while parts in manufacturing must be identifiable in an unambiguous way for retrieval from a storage system or other manufacturing processes. Therefore, the engineer only needs to specify the design of a part, because it is not necessary to dictate, for example, the color of the

⁴<https://www.lego.com/en-us/ldd>, last accessed on 2021-01-16

2.3. PROBLEM DESCRIPTION

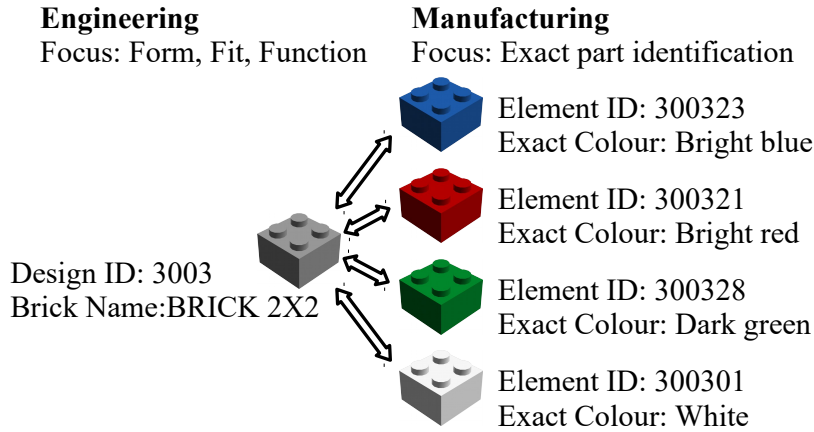


Figure 2.2: Design and element IDs of BRICK 2X2. The design ID belongs to the engineering view. The element IDs belong to the manufacturing view.

part. Besides color, different suppliers may be another reason to have several (element) IDs for the same functional part (design ID). Although information like color or supplier is not relevant for engineering, it may be available in CAD models or in the engineering department’s product data management system nonetheless. A part number from a supplier could imply a (default) color, or an engineer may want to express a recommendation for one supplier. The engineer might know that, while they adhere to the same specification, the quality of one supplier’s parts outperforms the quality of other suppliers’ parts. Thereby, the engineering and manufacturing perspectives are often not clearly distinguishable in reality. In general, the “engineering version” only depicts form, fit, and function, while the “manufacturing version” adds, for example, color, printing, or supplier. This separation implies that one engineering part may refer to many manufacturing parts, while usually, one manufacturing part refers to exactly one engineering part, as depicted in Figure 2.2. This distinction in views may be applied not only on the part level but even on the top level (“whole product” level). For example, there may be region-specific or customer-specific versions of a product. Again, these different product versions are all based on the same functional design (see Figure 2.3).

A more complex scenario arises if both functional and non-functional properties differ between the variants. For example, one product variant’s heating and air control system may be specifically equipped to withstand arid climatic conditions while another one is specialized for arctic environments. In this case, additional versions may exist on the engineering side as well. In the following section, we focus on differences in the engineering and

2.3. PROBLEM DESCRIPTION

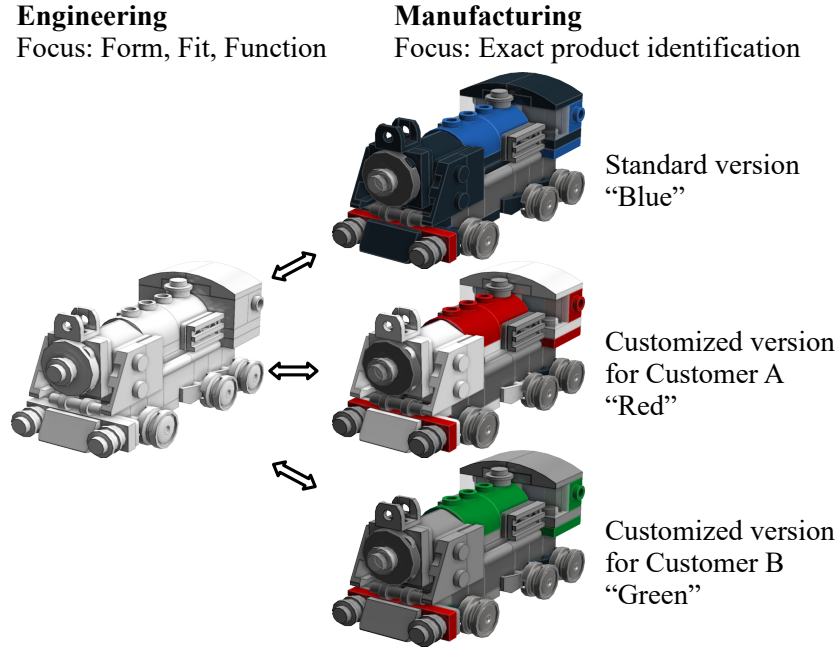


Figure 2.3: Product variants of *Blue Train*. The engineering view consists of only the functional aspects of the train. The manufacturing view also includes color.

manufacturing view of the *Blue Train*'s structure as introduced in the scenario from Section 1.1.

2.3.2 Engineering and Manufacturing Product Structure

Especially for complex products, such as in the automotive, aviation, and similar sectors, the BOMs in engineering and manufacturing differ significantly. In engineering, the product structure is deduced from the functionality of a product. The *Blue Train* can be divided into a steam engine, chassis, and operator's cabin (Figure 2.4a). In a real-life setting, the engineering product structure might be divided into hydraulics, electrics, power train, body parts, and more. The manufacturing structure, on the other hand, is compounded by subassemblies. Complete products are not built from discrete parts in one process; parts are assembled into subassemblies that are then step-wise put together into the final product. The LEGO model's official building instructions define the three segments: front, center, and rear, which are each assembled separately and then joined

2.3. PROBLEM DESCRIPTION

together (Figure 2.4b). Although these are not the proper steps to assemble a real train, the analogy is valid, as engineering and manufacturing product structures also differ in a real-life setting. For example, the functional group containing all electrical elements, such as cables, electric actuators, and power sources, are usually not all assembled together in one separate assembly process, as it would be impracticable to integrate such a cable skeleton into the product. A set of parts grouped by their functions, such as “all electrical elements,” is useful in an EBOM but would not be used in an MBOM as this group will never (except for special testing or demonstration purposes) be assembled into a physical subassembly. There can also be multiple manufacturing structures for the same product. For example, make-or-buy decisions can vary among production locations. Sub-assemblies that are manufactured on-site for use at one location may be purchased from a supplier at another location. Another example might be differently equipped facilities, as there are different production steps necessary depending on which machinery, tools, and so forth are available at different manufacturing sites.

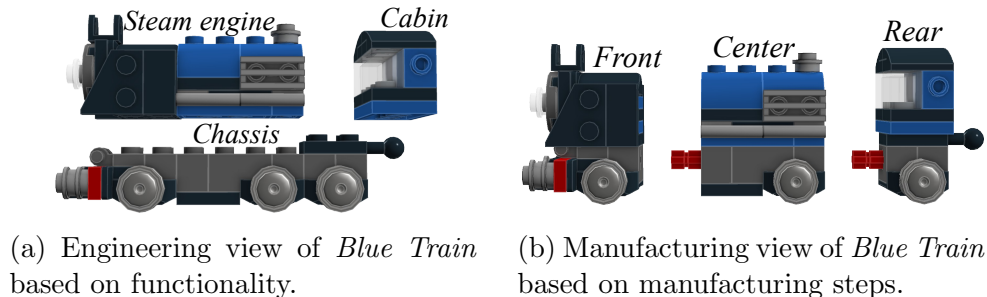


Figure 2.4: Different structure views of *Blue Train*.

Note that when designing models for information systems, part-whole relationships, that is, the relationships between a “whole” and its parts, components and/or constituents, are of major importance [25]. Semantically, it would be slightly more precise to use the term “subcomponent” instead of “subpart,” as “is a component of” usually does not imply transitivity, while “is part of” often does [85]. As a subcomponent of a subassembly is not necessarily existential for the whole product, always calling a subcomponent a “subpart” would be imprecise. However, subpart is the commonly used term in PLM [81, 71] and is therefore adopted in this work.

2.4 Requirements

Based on the above problem description extending the scenario from Section 1.1, we have derived the functional and non-functional requirements for our PLM-related core ontology.

2.4.1 R1: Differentiating between Product Concepts and Product Instances

Typically, the term “product” is used both as the concept of a product, like a model, drawing or idea, and its physical implementation, such as a tangible train with a certain ID where parts like a steam engine with a specific ID are built in. The ontology must be able to explicitly differentiate between such product concepts and product instances.

2.4.2 R2: Different Views of Parts depending on Context

Views of the parts used in products differ depending on PLM phases and parties. A supplier may consider a part as an assembly, while the manufacturer considers the same entity to be a subpart without a sub-structure relevant to the manufacturer. Different requirements exist regarding the information and models needed and created for the parts as these depend heavily on the phases. For example, in the early design phase, parts may not need a parameter set for their procurement method (e. g., built in-house, bought from a supplier, or consignment), while this information is required for production planning in the early manufacturing phase. Thus, the ontology must support separate views of the product data for the several PLM phases.

2.4.3 R3: Distributed Workflow Models and Workflow Executions

In PLM, information about the product and the product itself change across the phases. An ontology needs to support workflows regulating the processes inside a business and across all the organizations involved in those workflows. Information may be accessed, validated, changed, transmitted, or taken as input for manufacturing steps, such as assembling a subassembly according to an MBOM. In some cases, workflows supporting

these processes need to be weakly structured, that is, without rigid pre-defined transitions between steps. For example, processes in the early development of a new product are very dynamic. However, in other cases, processes need to be strongly structured, i. e., with restricted transitions. For example, structured processes are required for approval steps for changing manufacturing documents of a product that is already in production or in use. Therefore, the ontology must support workflows, including different levels of structure for using, creating, and changing product information.

2.4.4 R4: Integration of Security Policies

PLM includes providing the correct information to the authorized person at the right time. Therefore, another key aspect of modern PLM is ensuring the correct access rights to protect the confidentiality, integrity, and availability [2] of all necessary information. This requirement includes flexible permission management that provides and revokes access rights to specific information, such as drawings or models of product parts. Distributed settings that include international collaboration have to be considered, as the parties often reside in different countries where different legislation is applied. The ontology must support establishing rule sets for permission management.

2.4.5 Non-functional Requirements

The central purpose of a core ontology is to provide a formal foundation for data integration in a heterogeneous, possibly distributed infrastructure [70] as is found in PLM applications. Thus, besides the functional requirements above, non-functional requirements also need to be supported by CO-PLM. These non-functional requirements are derived from the experiences in designing core ontologies and are precise semantics, modularity, extensibility, reuseability, and separation of concerns [70]. CO-PLM needs to provide **precise semantics** to ensure **interoperability** with other ontologies. While the CO-PLM ontology can be used as-is, there may be certain circumstances where it is desirable to reuse or extend only parts of it. **Modularity** is a pre-requisite for extensibility, that is, the addition of future functionalities or the integration of new, domain-specific knowledge. The requirement of **extensibility** is needed for CO-PLM to apply it in different sectors such as automotive, aviation, construction, and others. Finally, with **separation of concerns**, we refer to the requirement that the CO-PLM ontology shall be applicable in arbitrary application domains that deal with production [70]. These requirements support the adoption of the ontology.

2.5 Overview of the CO-PLM Core Ontology

Our solution to the requirements stated above is CO-PLM, a novel core ontology for PLM. Unlike existing PLM approaches, our CO-PLM extends the state of the art by supporting all phases of a product’s lifecycle.

Design Approach: The design of CO-PLM follows the most common approach in the PLM literature by applying a part-centered model because product parts are the objects that inherently hold most of the product data [53]. A part is a specific entity within the product, not the documents related to the product. In general, data for the different lifecycle phases is stored in separate software systems that are operated by different departments. These systems expand in functionality over time and focus on the individual requirements of the particular department. Therefore, product models, data formats, and supported processes often differ among these systems. However, a manufacturing company must ensure all product data is precisely assignable to a product or its parts and can be shared across the different departments. CO-PLM offers a semantic basis to formalize information about product-related entities and the relationships among these entities. To this end, we base the design of CO-PLM on the foundational ontology DOLCE+DnS Ultralite (DUL) [18]. Furthermore, we apply a pattern-based ontology design because it supports modularity by enabling reusing individual aspects of the ontology separately. Figure 2.5 illustrates this approach by introducing information objects for every lifecycle phase and the associated BOMs. BOMs are a special case of information objects that give structure composed of further information objects.

In DUL, an information object represents a piece of information in its abstract form, while an information realization is its physical manifestation, such as a paper document or a digital file (consisting of physical bits on a storage or transport medium). The depicted object types illustrate examples of various information objects in different phases. A 4-phase lifecycle model (“Design,” “Manufacturing,” “Usage,” and “Disposal”) is used because this division is the most common in PLM. CO-PLM does not require exactly these phases, nor is it necessary to assign an information object to exactly one phase. In reality, phases may overlap and cannot always be completely separated. Engineering data (**Design Information Object** in Figure 2.5) contain all design-relevant data, such as CAD models, specifications, and changes. EBOM represents the structure of a part’s subparts from a functional perspective. In the manufacturing phase, resource planning must be considered, including personnel, machines, facilities, and goods. Commercial information, such as prices and procurement lead times,

2.5. OVERVIEW OF THE CO-PLM CORE ONTOLOGY

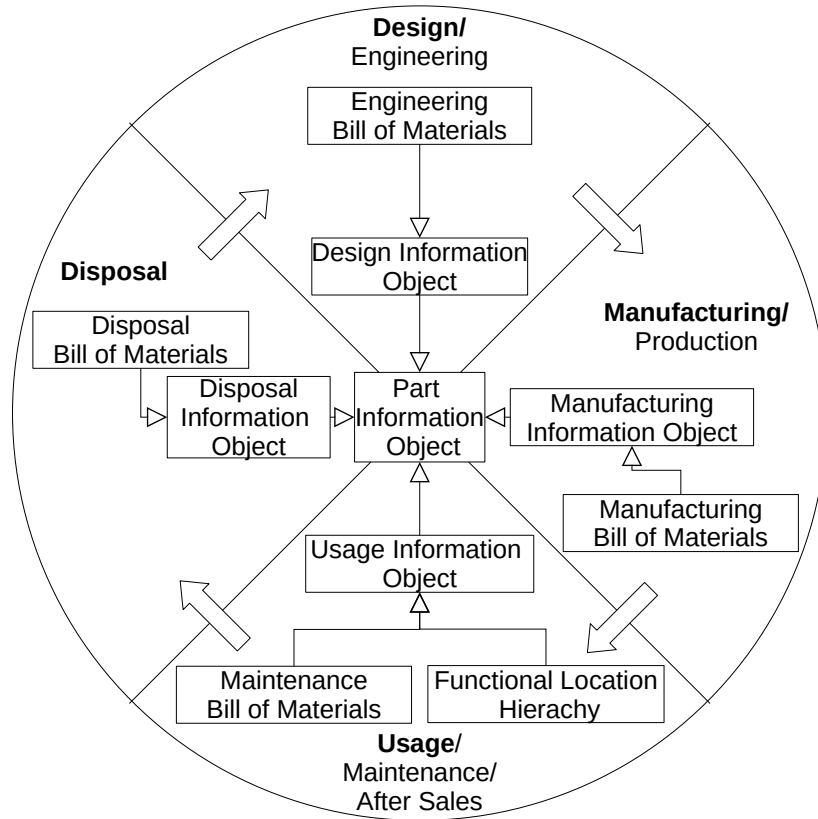


Figure 2.5: Example information objects in a part-centered PLM model with four lifecycle phases.

gains in significance in this phase. MBOM breaks down the part structure into mergeable subassemblies (see Section 2.3).

Spare part management and equipment management are important tasks in the usage phase. Spare parts can differ from manufacturing parts, as the former may be delivered in particular sets or kits. Spare parts could also need special treatment, such as greasing for long-term storage. In many cases, manufacturers of high-tech products consider the manufacturing and the after-sales of a product as separate projects to emphasize the importance of the business after delivery. The disposal phase is often neglected in PLM. However, immense costs can arise for the customer or the manufacturer if formalities regarding proper product disposal are not considered beforehand. Furthermore, the disposal phase contains actions for product substitution, such as the replacement of newer versions or surrogates of the original product

leading to further business opportunities. However, customers may switch to rivals if they are able to cover the disposal phase more sophisticated. Product data for disposal must, therefore, contain all information relevant for the safe disposal or replacement of products. For example, substance classifications according to their waste types describe toxic ingredients and the necessary procedures for safe disposal. In addition to the four example phases depicted in Figure 2.5, more views are possible. For example, a sales bill of materials represents the structure of a product according to how it is shipped or which products are sold together.

2.6 Design of the CO-PLM Core Ontology

The following subsections present the formal concepts of CO-PLM in the form of separate patterns. Each of these patterns defines a set of related classes and provides a structure to follow along with the functional features of CO-PLM.

CO-PLM consists of ten ontology design patterns, which are implemented in six OWL files (see Figure 2.6). The legend depicted in Figure 2.6 also applies to the following figures. We use the `namespace:ClassName` notation for indicating the origin namespace of imported classes (gray boxes) from external ontologies such as DUL. We omit the namespace for our own classes (white boxes) defined in CO-PLM. When presenting example instantiations of our patterns as shown in Figure 2.16 and Figure 2.20, we use the `instance-name-i:ClassName` notation to highlight the type of an instance. Cardinalities are omitted if they are arbitrary (i. e., $0, \dots, *$) or if they have already been defined in a previous or imported pattern. These notations are a digest from UML and established conventions from multiple ontology papers [31, 41, 82, 69, 70, 71, 75]. Therefore, we consider them to be best practices for depicting ontology patterns in chart form.

The presentation of the patterns is organized as follows: First, the Product Part Information Entity Pattern and the Product Part Information Quality Pattern refine the Information Entity and Quality concepts from DUL in the PLM context. Next, the Product Part Information Object Pattern and the Part Entry Pattern further specify these concepts to elaborate on BOMs and their relationships to referenced parts. Finally, the Product Part Pattern and the Product Part Description Pattern utilize the previous concepts as well as DUL's Description and Situations Pattern to offer a framework for representing product information. As Figure 2.6 shows, the patterns are connected by imports and thereby reuse formerly defined classes. For example, the class `ProductPartInformationObject` defined in

2.6. DESIGN OF THE CO-PLM CORE ONTOLOGY

the Product Part Information Entity Pattern is reused in the Product Part Information Object Pattern.

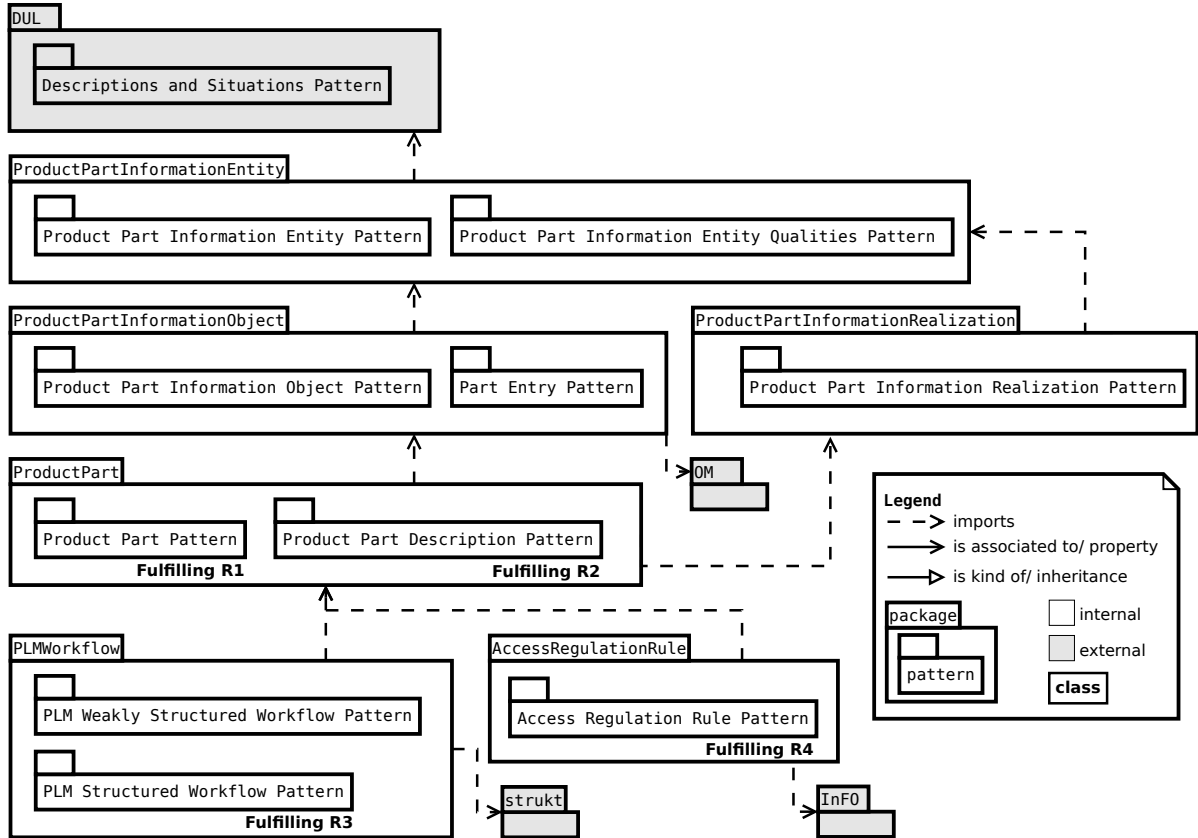


Figure 2.6: Overview of the Design Patterns of CO-PLM organized into packages. The box with the bent corner contains the legend. This notation is used in all following figures describing ontology patterns. The numbered “R”s stand for the requirements R1–R4 from Section 2.4.

The Product Part Pattern fulfills R1 (product concepts and instances), and the Product Part Description Pattern fulfills R2 (context-dependent views). R3 (workflows) is fulfilled by the PLM Structured and Weakly Structured Workflow Patterns integrating the workflow core ontology *strukt* [69], which is also based on DUL. R4 (group and access right management) is fulfilled by a combination of *InFO* [41] for access control integrated by the Access Regulation Rule Pattern and *dgFOAF* [75] for group management. Thus, CO-PLM covers all of the presented functional requirements. Below we describe each CO-PLM pattern in detail.

2.6.1 Product Part Information Entity Pattern

In DUL, an information object represents a piece of information in its abstract form, while an information realization is its physical manifestation, such as a paper document or a digital file, which consists of physical bits on a storage or transport medium. The central concepts of this pattern are `InformationEntity` and its subclasses `InformationObject` and `InformationRealization`. Based on these concepts, we introduce product-part-related counterparts, that is, `ProductPartInformationEntity`, `-Object`, and `-Realization` (see Figure 2.7). Similar to information entities, product part information objects can have one or multiple product part information realizations and vice versa; that is, one realization can realize one or multiple information objects. Example 1: One print-out (information realization) of an assembly and its subparts can depict geometrical information (information object) of the assembly as a whole, as well as geometrical information for the subparts. Example 2: One BOM (information object) can be depicted in multiple database entries (information realizations), if the BOM is entered into the database one entry per part.

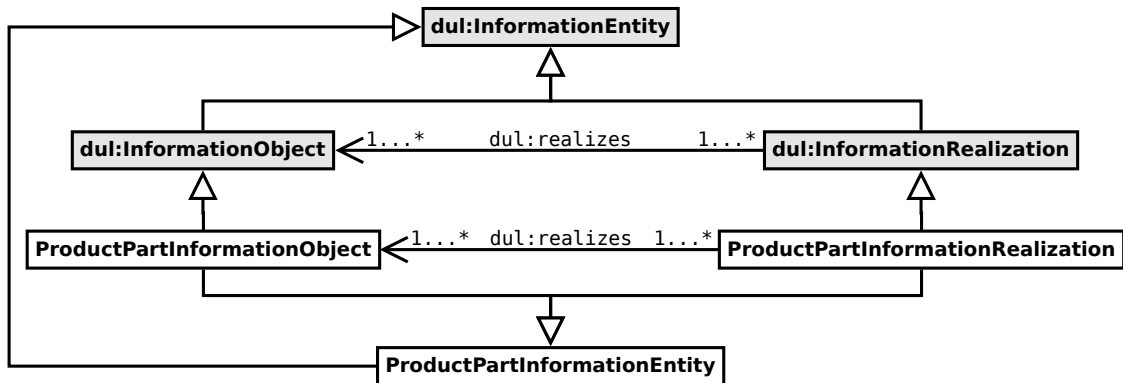


Figure 2.7: Product Part Information Entity Pattern. Entities of class `InformationEntity` are divided into `InformationObject` and `InformationRealization` realizing the former. Subsequently, `ProductPartInformationEntity` is divided into `ProductPartInformationObject` and its realization `ProductPartInformationRealization`.

2.6.2 Product Part Information Entity Qualities Pattern

ProductPartInformationEntitys may have Qualitys relevant in the PLM context (see Figure 2.8). In DUL, a Quality is defined as “any aspect of an Entity (but not a part of it), which cannot exist without that Entity” (see documentation at [18] in DUL.owl).⁵

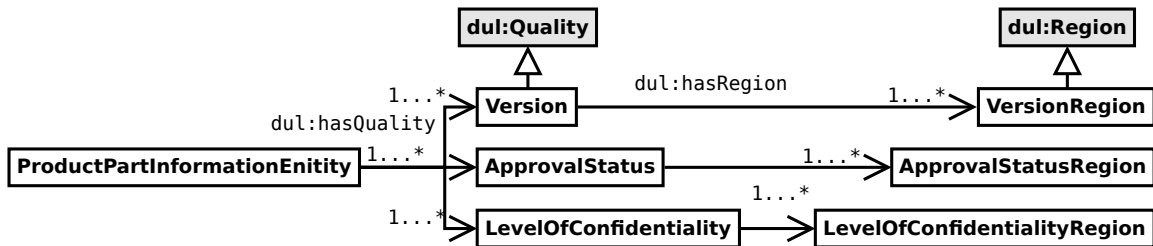


Figure 2.8: Product Part Information Entity Qualities Pattern. Entities of class ProductPartInformationEntity can have multiple PLM-related Qualitys, for example, Version, ApprovalStatus, and LevelOfConfidentiality with their respective Regions.

Qualitys of a ProductPartInformationEntity are Version, ApprovalStatus, and LevelOfConfidentiality. A Quality generally belongs to exactly one ProductPartInformationEntity, as otherwise, a change of a quality of one entity implies a change in the quality of another. In special cases, this could be desirable, such as if multiple entities can only be approved together. ProductPartInformationEntity has, in most cases, only one Version, ApprovalStatus, and LevelOfConfidentiality. However, there may be exceptions, for example, if multiple systems of confidentiality levels are used in parallel. This could be an internal company standard (“public,” “company internal,” and “company confidential”) next to governmental levels (“not classified,” “confidential,” “secret”) that can not be directly mapped to each other. Similarly, the Version could be a composition of separate versions for different countries and multiple versions for each country, for example, “Sweden” “1.0,” “Germany” “1.0,” or “Spain” “2.0.” The ApprovalStatus can have different values depending on the

⁵The definition used here is slightly different from the common definition of quality in most industries taken from ISO Standard 9000: “[...] the degree to which a set of inherent characteristics of an object fulfills requirements [...]”. The ISO definition excludes non-inherent characteristics, such as the price of a part or approval status of a document, but they can be a Quality in the context of DUL because of these differences in the definition of “quality.”

state of progress or approval by different boards. Examples of different regions for approval status are “draft,” “work in progress,” “final,” and “rework” in contrast to “not yet approved,” “approved by change board,” and “approved by customer.”

2.6.3 Product Part Information Object Pattern

The Product Part Information Object Pattern (see Figure 2.9) illustrates the relevant information objects of the CO-PLM ontology. `ProductPartInformationObjects`, as defined in the Product Part Information Entity Pattern, are all information objects related to a part (in PLM literature often called “product data” [81, 50]). The most common additional `ProductPartInformationObjects` are `ProductPartMetainformation`, such as identifiers like the `PartNumber`, `Weight`, and `Dimensions`. The `BillOfMaterials` describes the phase-dependent structure of `ProductPartInformationObject` and states its parts. `GeometricProductPartInformation` contains spatial layouts of the part and its components. `ProductPartForms` subsume information commonly represented by forms, such as `ProductPartSpecification`, `ProductPartManual`, and `ProductPartTestReport`.

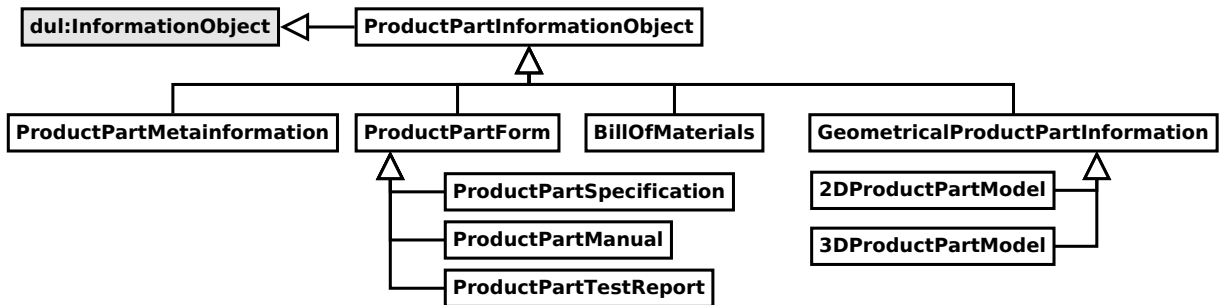


Figure 2.9: Part Information Object Pattern. Common examples of `ProductPartInformationObjects` are presented.

2.6.4 Part Entry Pattern

Figure 2.10 introduces the class `PartEntry` to represent individual entries in BOMs, using the “Ontology of units of Measure” (OM)⁶ to represent

⁶<http://www.ontology-of-units-of-measure.org/vocabularies/om-2/>, last accessed on 2021-01-16; Ontologies for units are discussed at [83].

quantities. A **Measure** is a combination of a value and a unit, such as “10 kg,” “5.2 m,” or “3 pieces.” Here, a unit is not a mere string, but an entity itself of the class **UnitOfMeasure**. Also, a part entry can refer to the more general product part information entity, as, for example, a manual could be content for a sales bill of materials that describes the deliverables. The **hasConstituent** property is a shortened way to express the relationship between the product part and the BOM or any information object describing it. Figure 2.15 extends this relationship by integrating it into a more elaborate pattern including additional parameters.

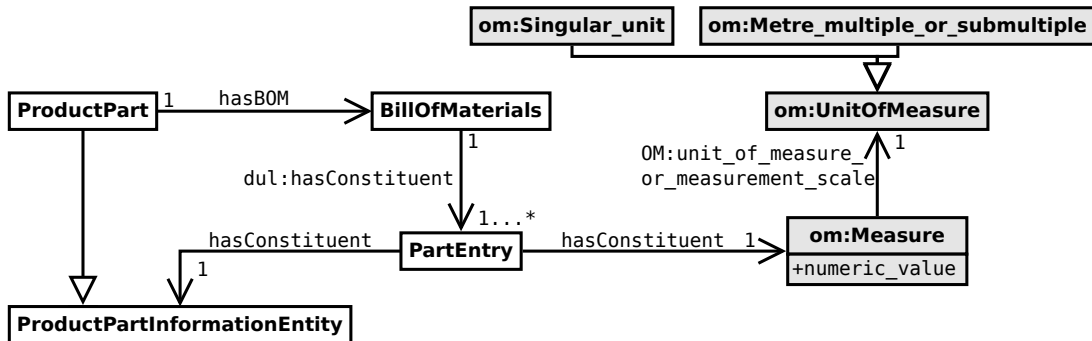


Figure 2.10: Part Entry Pattern. Each **PartEntry** consists of exactly one **ProductPartInformationEntity** providing a reference to the listed part as well as one **Measure** specifying the required quantity. A set of **PartEntry**s form a BOM describing a **ProductPart**.

2.6.5 Product Part Information Realization Pattern

The Product Part Information Realization Pattern (see Figure 2.11) describes different ways of representing the information objects mentioned above. **ProductPartInformationRealizations** are essentially **Documents** and **DatabaseObjects**. **Documents** can be either physical in the form of **PrintedDocuments** or **DigitalDocuments**. **DataRealizations** contain the classes **DataBaseRealization**, **DataTableRealization**, **DataSetRealization** and **DataFieldRealization**. In general, all **ProductPartInformationObjects** can be realized by any **ProductPartInformationRealization** (e.g., a **BillOfMaterials** can be realized by a **PrintedDocument** on paper or digitally in a **DataTableRealization** by multiple **DataSetRealizations**). An exception is **ProductPartMaster**, as its only realization is the

`PhysicalProductPart` and the latter only realizes the former. Not only `ProductPartInformationObjects` but also basically arbitrary `InformationRealizations` can be realized by `Documents` and `DataRealizations` because these classes directly inherit from `InformationRealization`.

The pattern is based on the IOLite [18]⁷ pattern and has some additional classes, as the existing ones do not cover the typical PLM data representations, that is, printed or digital documents as well as databases and their substructures (tables, sets, and fields). There are some similar, yet significantly different, concepts in IOLite. For example, a `Depiction` is “[a]n information realization consisting of depicted images/signs of any sort [...], which are inscribed [sic] on a medium [...]” (see documentation inside `IOLite.owl`). One could argue that this concept is very similar to `PrintedDocument`. But we consider `PrintedDocument` to be a significantly different subclass of `Depiction` compared to `Drawing`, `Writing`, `PlasticArt`, and `GraphicArt`. For example, `Writing` is not a suitable synonym for `PrintedDocument`, as the latter could contain texts, images, or even both. Also, `Depiction` is semantically different from `PrintedDocument`’s superclass `Document`, as `Document` includes `DigitalDocument`, which is a `DigitalResource`. There could be an argument that `DigitalResource` could be understood as a `Depiction`, as binary digits on a digital medium could be seen as some form of physical “inscription.” However, IOLite explicitly states that `Depiction` has `PhysicalArtifact` as a superclass, while `DigitalResource` does not. This definition indicates that IOLite’s `DigitalResource` is not meant to be a `PhysicalArtifact` and, therefore, is also not meant to be a `Depiction`. Furthermore, `DigitalResource` is not a synonym for `DigitalDocument`, but the first is a superclass of the latter. For example, a `WebPage` is a `DigitalResource`, while a `WebPage` cannot be considered to be a `DigitalDocument`, as a `WebPage` usually contains or embeds multiple `DigitalDocuments`. The closest IOLite concept for representing databases is `DataStructure`, but this is an `InformationObject` having the generic realization of `DigitalResource`. As we need realization classes for databases and their substructure for formalizing these essential PLM concepts, we also added classes for this purpose.

⁷<http://www.ontologydesignpatterns.org/ont/dul/IOLite.owl>, last accessed on 2021-01-16

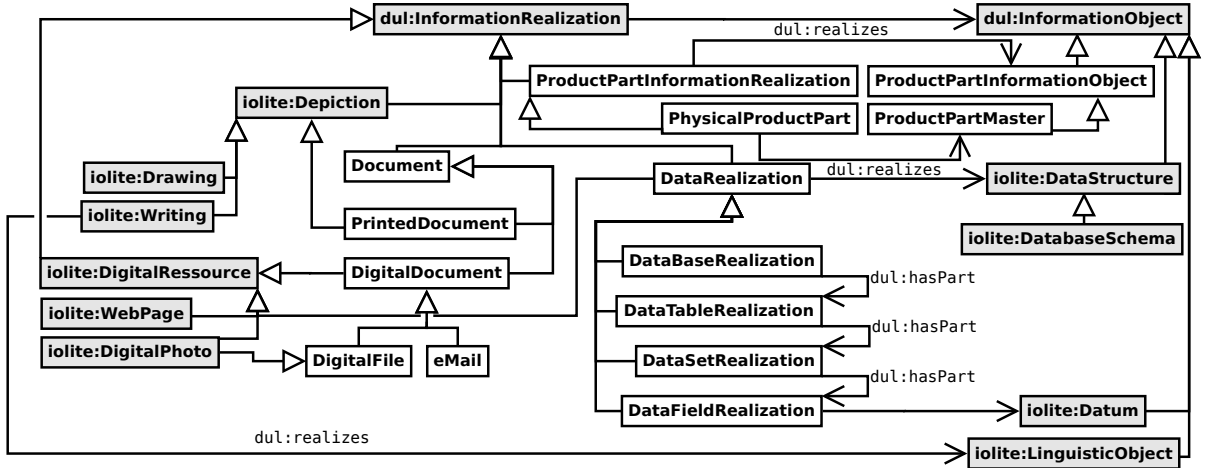


Figure 2.11: Product Part Information Realization Pattern. Realizations for the Product Part Information Objects introduced in Figure 2.9 are presented. Concepts from IOLite [18] are reused and missing concepts are added.

2.6.6 Pattern for R1: Product Part Pattern

A product part can either be a product part master or a physical product part (see Figure 2.12). While a product part master describes product parts in an abstract way, physical product parts represent the real physical objects. This pattern, therefore, fulfills R1. The physical objects are the realization of the master, which only exists for communication purposes.

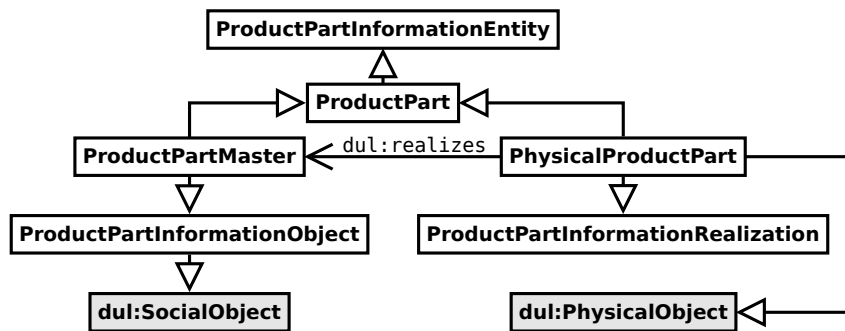


Figure 2.12: Product Part Pattern. ProductParts can be either ProductPartMasters or their realization, PhysicalProductParts.

2.6. DESIGN OF THE CO-PLM CORE ONTOLOGY

Using the example model from Section 2.3, Figure 2.13 shows different product part information entities. The product part master in the upper left represents the LEGO model 31504 in general. The photo on the bottom left depicts one specific physical build of this model. These two objects are product parts. On the right side, the top part shows a BOM and the bottom part shows its physical representation in the form of a printed document.

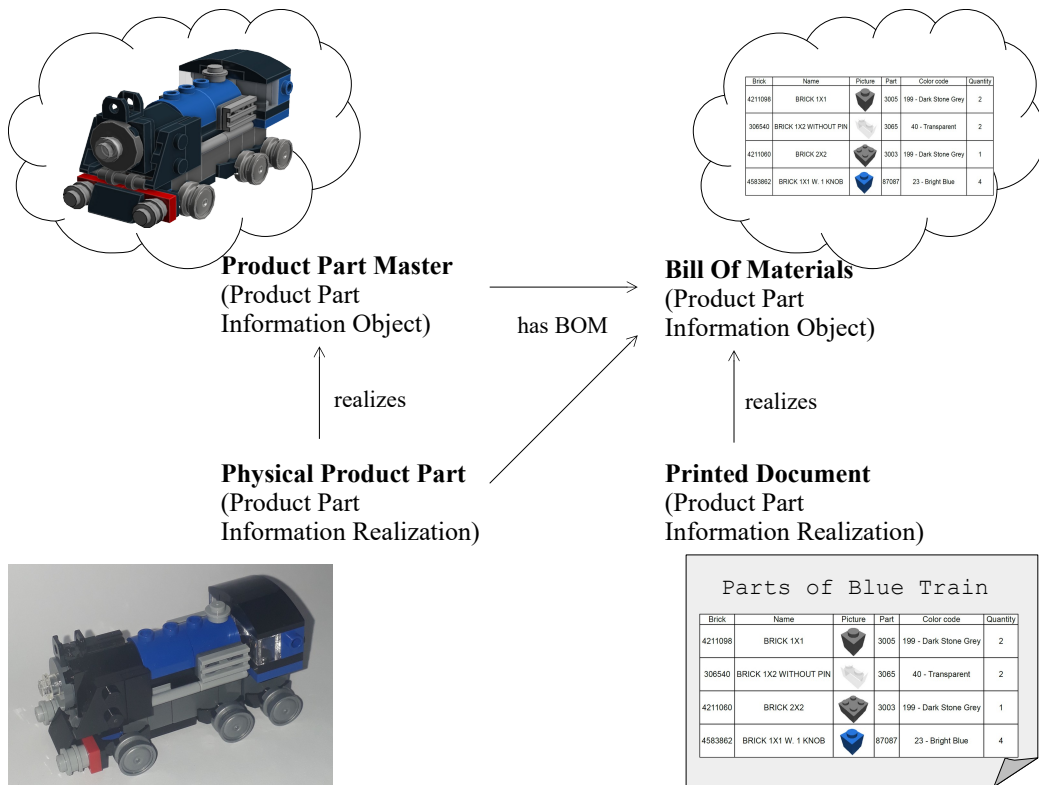


Figure 2.13: Train example to illustrate the fundamental difference between the two kinds of Product Part Information Entities and the relations between them. The entities on the top row encapsulated by clouds, ProductPartMaster and BillOfMaterials, are InformationObjects, that is, abstract objects. The clouds represent the fact that these objects are not physical, just like a thought or idea about the entities they represent. The entities on the bottom row, PhysicalProductPart and PrintedDocument are InformationRealizations, that is, tangible physical objects.

Both masters and physical product parts can have subparts, which are listed in their respective BOMs. The master can only have other masters in its BOM, but a physical product part can have both masters and

physical product parts in its BOM. As the master is the basis for multiple physical product parts, its BOM cannot contain a specific physical subpart. However, the BOM of a physical product part can either refer to a subpart master (anonymous part reference) or a physical product part (specific part reference). If individual aspects of the subpart should be traceable, such as a serial number or a production date, a specific reference is needed. Otherwise, an anonymous reference is sufficient.

2.6.7 Pattern for R2: Product Part Description Pattern.

With the Product Part Description Pattern different views of product parts can be represented, which fulfills R2. Several of the ontology patterns presented in this thesis are based on DUL's Descriptions and Situations (DnS) pattern [18] (see Figure 2.14). DnS offers "separation of states-of-affairs and their interpretation based on a non-physical context, called a description" [19]. DnS allows creation of descriptive contexts, that is, **Descriptions**, defining various **Concepts**, such as **Roles**, **Parameters**, and **EventTypes**. Therefore, **Descriptions** form a conceptual structure defining, for example, which **Roles** are relevant and how they relate to each other. These **Descriptions** may be satisfied by multiple relational contexts, called **Situations**, setting entities for the **Concepts** defined in the respective **Descriptions**. For example, a **Situation** might set an **Agent** classified by a **Role** or an **Event** classified by an **EventType**. By reusing this pattern, we are able to distinguish between descriptive as well as relational contexts and integrate our concepts into DUL.

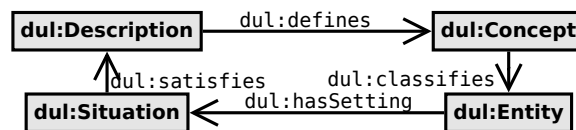


Figure 2.14: DUL DnS Pattern. Created according to the axioms in [18].

The Product Part Description Pattern uses DnS. Figure 2.15 shows how we adapted the DnS pattern to generate different views, that is, **ProductPartSituations**, to describe a **ProductPart** according to a **ProductPartDescription**.

In general, a **Description** defines various **Concepts** such as **Roles** (here: **ProductPartInformationObjectRole**) and **Parameters** (here: **ProductPartParameter**). As seen in the descriptive context

2.6. DESIGN OF THE CO-PLM CORE ONTOLOGY

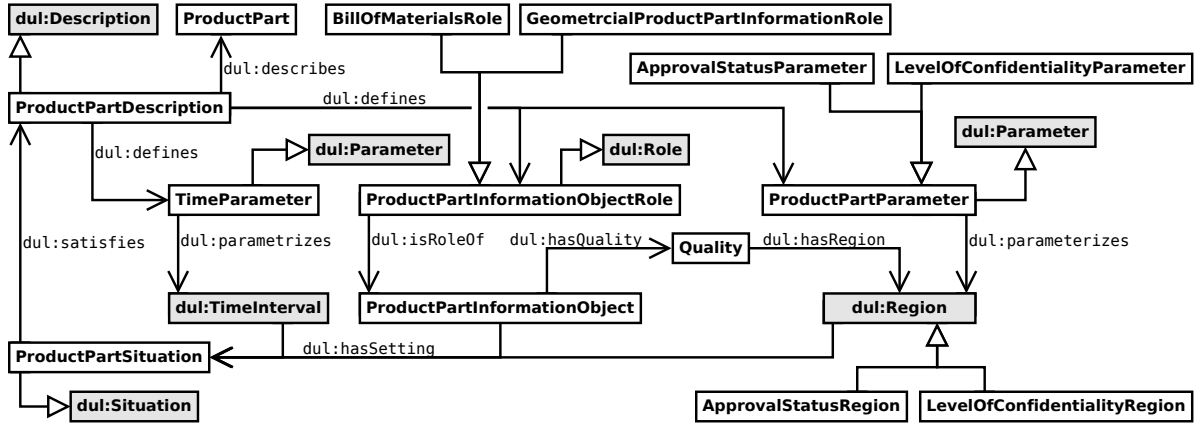


Figure 2.15: Product Part Description Pattern. PLM-related roles and parameters can be defined by a `ProductPartDescription` describing a `ProductPart`. Such a description can be satisfied by a concrete `ProductPartSituation`.

above, several `ProductPartInformationObjectRoles` can be defined in a `ProductPartDescription`, such as for geometric dimensions and structures. These `Roles` then classify specific `ProductInformationObjects`, for example, a specific MBOM has the role of giving a part description its structures, while a specific 3D CAD model fulfills the role of providing geometric dimensions. For each of these `ProductPartInformationObjects`, the `ProductPartParameters` offer the possibility to specify which approval status these individual objects are in or which level of confidence they possess.

Example: Figure 2.16 illustrates the use of the Product Part Description Pattern. `bluetrain-description` is an instance of `ProductPartDescription`, describing the product `bluetrain`. Instances of the class `ProductPartSituation` hold specific configurations that can describe the product structures of real `PhysicalProductParts`. In this case, `bluetrain-1.0-1` (not explicitly depicted in the figure) may be considered the first physical train built according to configuration `bluetrain-1.0`. The manufacturing structure is defined by `bluetrain-1.0-mbom`, and the `ApprovalStatus` of this MBOM is `approved-for-production`. A time interval is assigned to `bluetrain-1.0`, stating that it is valid for production in 2018.

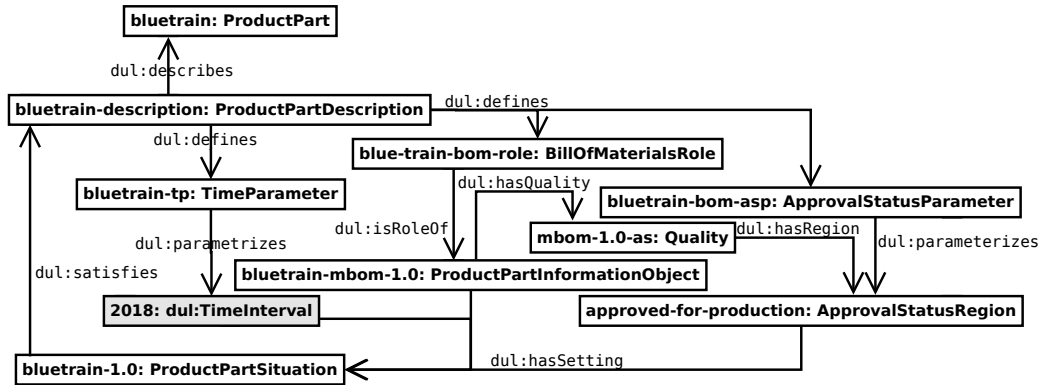


Figure 2.16: *Blue Train* Example Product Part Description Pattern. In this example instantiation of the Product Part Description Pattern, the *Blue Train* version 1.0 is described.

2.6.8 Patterns for R3: PLM Workflow Patterns

With the PLM Weakly Structured Workflow and PLM Structured Workflow Patterns, workflows using the information of all of the above patterns can be defined and executed. These workflows are based on the Weakly Structured Workflow and Structured Workflow patterns from the core ontology *strukt* [69]. The two types differ in their flexibility. While the Structured Workflow requires definition of transitions between the workflow steps, the Weakly Structured Workflow does not rely on transition definitions and thus supports more flexible workflows. Figure 2.17 shows the PLM Structured Workflow Pattern. The weakly structured counterpart is nearly identical but is missing the transition classes.

Development of a new product may require loose workflows, as especially the early design phase can require more flexibility. In later phases, like manufacturing, more rigid processes are reasonable, as, for example, a change in a product’s MBOM has a significant influence on many departments. Not only must the manufacturing department adjust its production processes, but the purchasing department must also act, as current orders need to be re-evaluated for obsolete parts and changes need to be coordinated with the supplier.

Example: To illustrate a PLM weakly structured workflow, Figure 2.18 demonstrates a change in an EBOM, such as replacing a major component that is no longer offered by the supplier with a similar part.

This change is discussed in the change board meeting to analyze whether the product still fulfills all customer requirements after the change. To keep

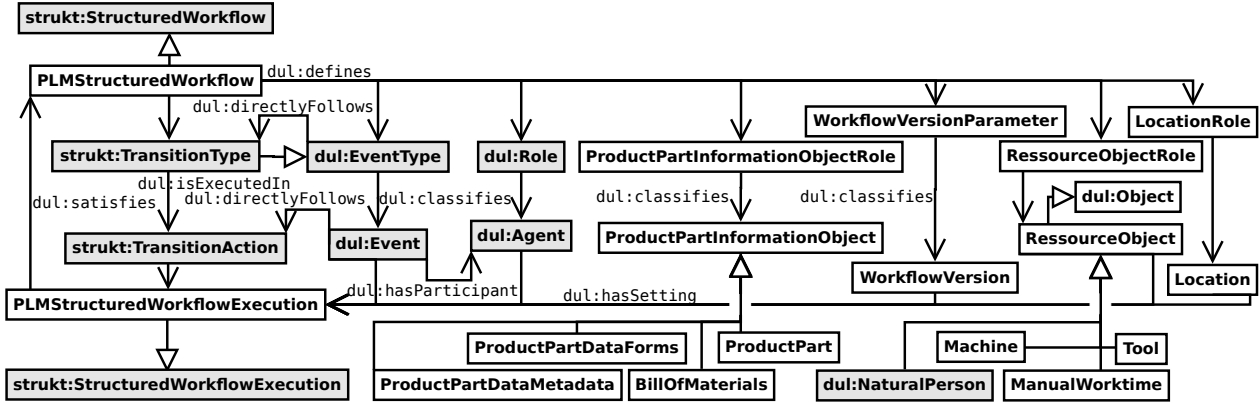


Figure 2.17: PLM Structured Workflow Pattern. A `PLMStructuredWorkflow` is a `StructuredWorkflow` extended by PLM-related concepts.

the example simple, it is designed as a Weakly Structured Workflow. The `PLMWeaklyStructuredWorkflow change-ebom-workflow-1` is applicable to any EBOM. The concrete `PLMWeaklyStructuredWorkflowExcection change-ebom-execution-1` represents the first change of *Blue Train*'s EBOM. The workflow defines `change-board-meeting-type-1` as an `EventType`. The `change-board-meeting-1` is the first board meeting deciding about the change. The role of the `configuration-manager-1` is responsible for leading the `change-board-meeting-1`. In this case, `john-miller-1` fulfills this role.

2.6.9 Pattern for R4: Access Regulation Rule Pattern

Workflows involve different parties, such as business partners, who exchange product data. However, not all parties are able to access all product data. Thus, the Access Regulation Rule Pattern depicted in Figure 2.19 allows the restriction of access to different types of data. The pattern is based on the core ontology InFO [41], an ontology for regulating information flow. InFO regulations consist of rules, policies, and meta-policies. A rule covers a specific regulation. Policies are collections of rules that share the same legal foundation or organizational motivation. A meta policy combines several policies and defines algorithms for resolving conflicts between contradicting policies.

The Access Regulation Rule Pattern of CO-PLM is designed according to InFO rules and introduces additional roles. The pattern regulates the execution of an action on product data by one active agent and several passive

2.6. DESIGN OF THE CO-PLM CORE ONTOLOGY

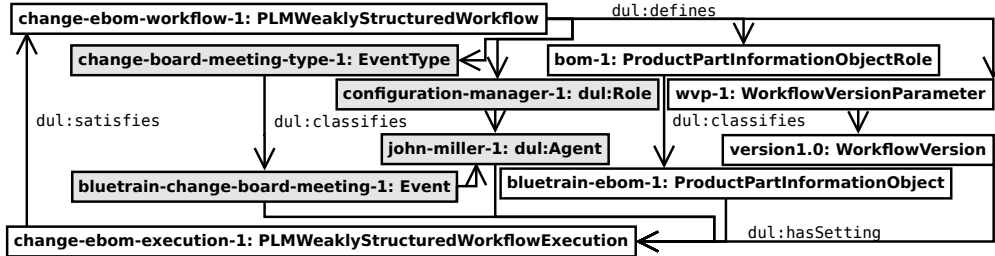


Figure 2.18: *Blue Train* Example PLM Weakly Structured Workflow Pattern. This example instantiation of the PLM Weakly Structured Workflow represents a workflow within a larger change management process. In the example, change board meetings led by the configuration manager are defined. In the first of these meetings, version 1.0 of *Blue Train*'s EBOM is defined.

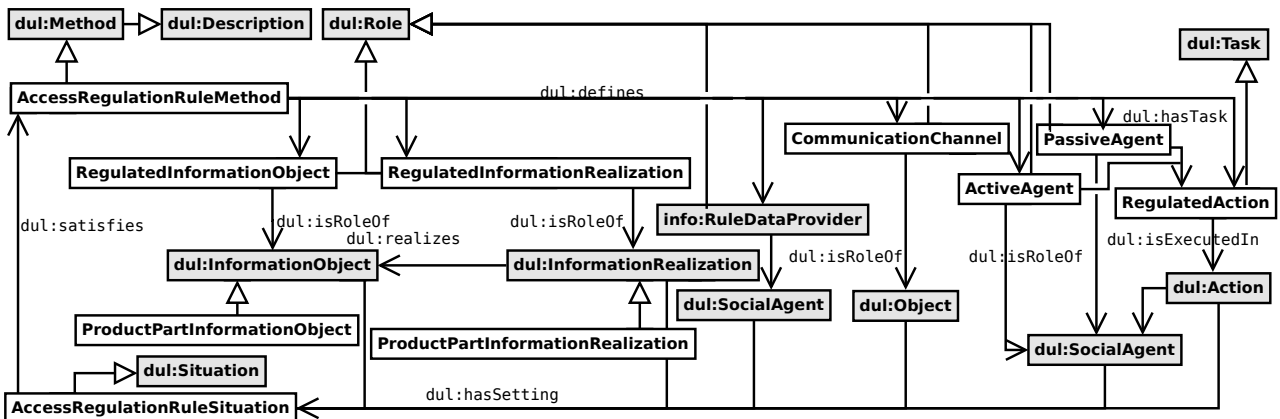


Figure 2.19: Access Regulation Rule Pattern. An `AccessRegulationRuleMethod` extends a `Method` by concepts required for regulating access.

agents. Possible actions are read, write, and transmit. An active agent initiates the action, whereas passive agents may benefit from it. Depending on the type of action, agents can be further refined by more specific roles. For example, the transmit action can define the active agent to be the sender of product data and a passive agent to be its receiver. Because not every action involves more than one party, passive agents are optional.

The main class of the pattern is `AccessRegulationRuleMethod`, which defines the context of a regulation rule. Subclasses define how the context

shall be evaluated. For example, `TrackingRuleMethod` states that the execution of the described action shall be tracked. The Access Regulation Rule Pattern represents product data as a `ProductPartInformation-Realization` and its corresponding `ProductPartInformationObjects` as described in the Product Part Information Entity Pattern. Executing an action requires a communication channel that can be, for example, a computer network for transmitting digital files or an express mail service for sending printed documents. A particular rule is created by a rule data provider, who provides all necessary details for its enforcement. All parties in the Access Regulation Rule Pattern are modeled as social agents who act within a social context. For example, employees of a company may access product data at their workplace. In this case, the workplace is the social context of the employees who act on behalf of their company. The pattern does not restrict how social agents are modeled in a particular rule. This freedom allows the pattern to define a social agent as “employee with id 4711” or by providing a rule-based approach such as “every employee working in Company Y.” Apart from the Access Regulation Rule Pattern, CO-PLM also adopts the policies and meta-policies of InFO. An access regulation policy is essentially a collection of several rules that share a common legal background or organizational motivation [41]. An access regulation meta policy combines several policies and defines various conflict resolution algorithms. Details on policies and meta policies are provided in [41].

Example: Figure 2.20 depicts an access regulation of the Product Part Specification `bluetrain-pps-1`. This rule specifies the affected resource (the digital file `df-1`) realizing the Product Part Specification of the *Blue Train* `bluetrain-pps-1`. Also, the privileged agent `sa-1` receiving read access is specified. This agent could be a person or a group. The rule states that Jane Nox is the *RuleDataProvider*. This information can be used for compliance checks. The access may only be executed via the SSL-secured channel.

2.6.10 Fulfillment of Non-Functional Requirements

CO-PLM is strongly axiomatized and reuses the foundational ontology DOLCE+DnS Ultralite [18] to address the non-functional requirement of precise semantics. Furthermore, it integrates patterns from existing core ontologies for modeling workflows [69] and policies [41]. CO-PLM features modularity, extensibility, and reusability by using a pattern-based design [70]. CO-PLM’s extensions of DUL’s Descriptions and Situations ontology pattern support reuseability and separation of concerns. Separation of concerns is also supported by establishing an ontology hierarchy, using

2.7. EVALUATION OF PRACTICAL USE

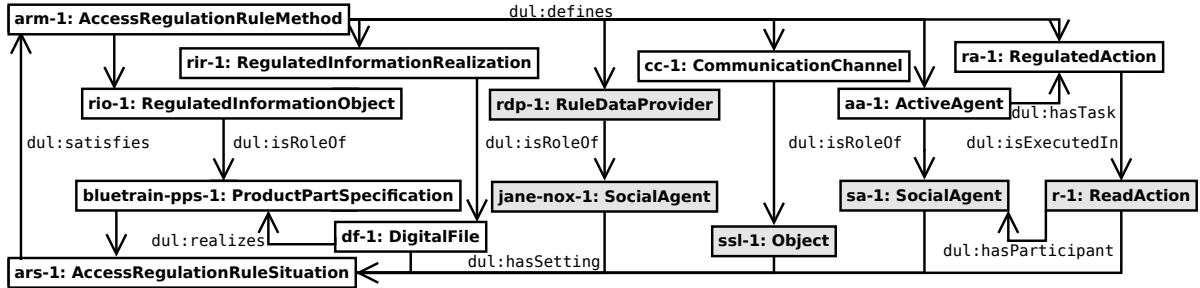


Figure 2.20: Example Application of the Access Regulation Rule Pattern. This example rule instantiation allows the agent `sa-1` to read the file `df-1` via the SSL-secured channel `ssl-1`.

a foundational ontology and multiple core ontologies, with each ontology fulfilling a particular, well-defined purpose.

2.7 Evaluation of Practical Use

The fulfillment of the functional and non-functional requirements is shown in Section 2.6. This section evaluates the practical use of CO-PLM in the context of an application in a vehicle manufacturing company. Using ontologies in practice requires an integrated engineering process that is closely aligned with system development. Therefore, we first describe our approach of combining ontologies and software design before we further discuss the technical evaluation of CO-PLM.

2.7.1 Joint Software and Semantic Engineering Process

CO-PLM's design and implementation into applications are embedded into an advanced Joint Software and Semantic Engineering Process (JoSSEP) (see Figure 2.21). As ontologies are used to create and query triples from software applications, the ontology engineering process must be aligned to the software engineering process. This alignment ensures maintainability, reusability, and extensibility of the developed software artifacts and ontologies [64]. Although ontologies and software are used in combination with each other, their development is slightly different. Ontologies are data models and follow a data-driven modeling approach. In contrast, software implements application behavior and focuses on technical functionality. Ontology design patterns

2.7. EVALUATION OF PRACTICAL USE

are predominantly relevant within ontology definitions. Not all of their details must be mapped to software source code [82]. Instead, a software's source code should provide an easy-to-use programming interface to create the individual triples for the instantiation of an ontology design pattern. Furthermore, functions related to data processing like `get`, `set`, `update`, `delete`, and `execute` are required in a software implementation but are not represented in an ontology.

Approaches to combine ontology engineering and software engineering exist, such as TwoUse [64, 91]. TwoUse is an open-source tool implementing OMG⁸ and W3C standards⁹ to develop ontology-based software models and model-based ontologies. However, additional research is needed to better integrate such approaches in modern system engineering processes, including test, configuration, build, change, and artifact management, which are not considered in current approaches.

Our solution for such an integrated engineering process, JoSSEP, is depicted in Figure 2.21. First, ontologies are created as OWL files using an ontology editor such as Protégé¹⁰. In order to combine ontologies with software artifacts, we use Apache Maven¹¹ for managing both the ontology and software projects. Maven uses a project object model (POM) to store all information of a particular project, including the project's name and its connections to other projects. Second, OWL files and POM files are stored in a version control system, which also stores the software's source code. Third, any changes to the version control system trigger a build server to download the latest version of the ontology and to perform a build process. The build process generates an API that can be used by software applications to create triples. Finally, all files of the ontology project are stored in an artifact repository that is used by both ontology engineers and software developers to retrieve any required artifact. A software developer imports the ontology's API in the source code to use an ontology. This is done by defining a Maven dependency¹² to the API using the POM file of the software artifact.

The presented process was used to implement prototypical applications using the CO-PLM core ontology. The first application is the transformation tool BOM2OWL for converting CSV-based BOMs into OWL instances of CO-PLM patterns for semantic processing. This tool was used to generate the *Blue Train* product part instances for the reasoning evaluation in the

⁸<https://www.omg.org/>, last accessed on 2021-01-16

⁹<https://www.w3.org/>, last accessed on 2021-01-16

¹⁰<https://protege.stanford.edu/>, last accessed on 2021-01-16

¹¹<https://maven.apache.org/>, last accessed on 2021-01-16

¹²<https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html>, last accessed on 2021-01-16

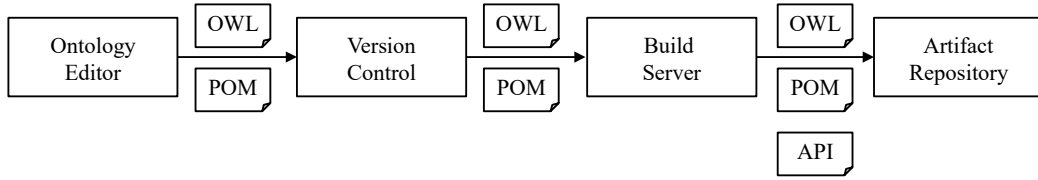


Figure 2.21: Semantic Engineering Process combining software and ontology engineering. The boxes with a bent corner represent artifacts. Regular boxes stand for information systems generating, transmitting (arrows), and processing these artifacts. Here, API means the API specification.

next section. The second tool is SyntProd, a tool for generating synthetic product parts and their subassemblies based on the CO-PLM patterns. Data generated by this tool were used for the large-scale evaluation in the next section. Also, semDIM (see Chapter 3) was developed according to the combined process.

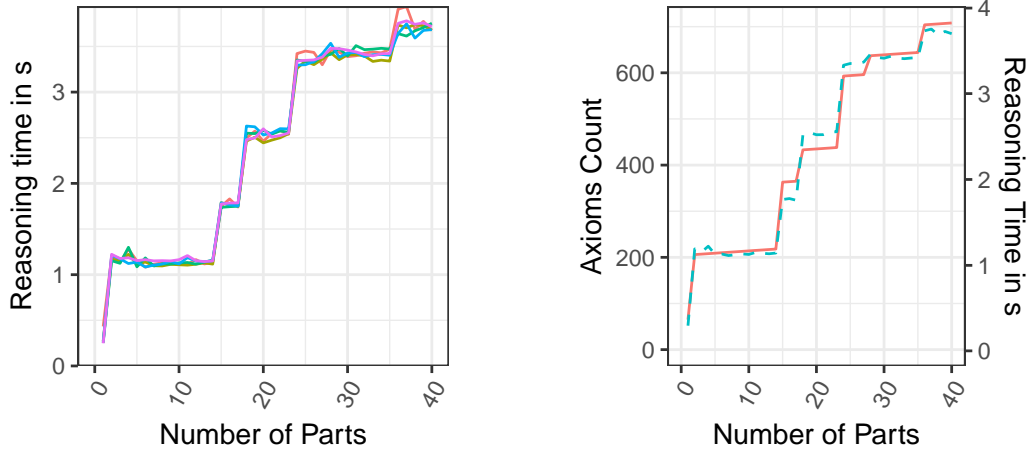
2.7.2 Reasoning Experiments on the *Blue Train* Data and Large-Scale Data

The *Blue Train* model from Section 2.3 was implemented and checked for consistency with the HermiT OWL Reasoner [77] to evaluate the practical use of CO-PLM. By implementing the model step-wise, one part at a time, we can observe how reasoning time increases with growing part counts.

***Blue Train* data:** Figure 2.22a shows five individual runs of the reasoner, each run represented by one line in the plot. The plot shows both adding parts, where there is almost no increase in reasoning time, and iterations of adding parts with sudden jumps of the reasoning time. The plateaus in Figure 2.22a appear when primitive parts without subparts are added. The jumps originate from adding subassemblies. When plotting the mean time of the five runs together with the axiom count on the y-axis (see the dotted line in Figure 2.22b), a significant correlation of these two graphs becomes apparent. A linear regression model fits with $R^2 = 0.99$, $p < .00001$ and $df = 38$. This suggests a linear correlation between the reasoning time and the number of parts, as the latter directly influences the number of axioms.

Synthetic real-world-scaled data: Beyond the *Blue Train* example, a real-life engineering product is a complex system that easily consists of ten thousands to millions of individual parts. For example, a passenger car

2.7. EVALUATION OF PRACTICAL USE



(a) Reasoning series for the *Blue Train* example. Each colored line represents a reasoning series. A series consists of multiple reasoning runs with increasing numbers of parts. As part numbers increase, the reasoning time of the run increases.

(b) Reasoning and axioms for the *Blue Train* example. The full line represents the average series of the reasoning series from a). It shows the development of the reasoning time against the number of parts. The dotted line represents the development of the axiom count against the number of parts.

Figure 2.22: Reasoning time related to number of part types added step-wise from the *Blue Train* example.

usually has around 30,000 individual parts,¹³ and an airline plane has ca. six million individual parts.¹⁴ As such real-life product data structures are not publicly available, we use synthetic data to simulate a real-life product. An important insight is that the linear correlation appears to not hold with larger part counts, as Figure 2.23 illustrates. This figure depicts five reasoning runs on automatically generated product structures. Here, the experiments start with 100 part types and 5 subassemblies. Each of the subassemblies has 20 different subpart types (covering all of the 100 part types). Each subpart type is used 3 times in a subassembly. The first point on the graph in Figure 2.23 represents a product with 300 individual part pieces, 100 different part types (à three pieces each) and five subassemblies. Then, the numbers of parts and subassemblies are

¹³<http://www.toyota.co.jp/en/kids/faq/d/01/04/>, last accessed on 2021-01-16

¹⁴<https://web.archive.org/web/20191214095113/http://magazin.lufthansa.com/xx/en/fleet/boeing-747-8-en/one-plane-six-million-parts/>, last accessed on 2021-01-16

2.7. EVALUATION OF PRACTICAL USE

increased by the same values at each step. Thus, the second experiment is run on 600 part pieces, 200 part types and 10 subassemblies, the third on 900 part pieces 300 part types and 15 subassemblies, etc. Three pieces per subpart are not relevant for the reasoning but illustrate that 100 part IDs correspond to 300 individual parts if every part ID is used, on average, three times in the product. Each “chunk” of 100 part types including five subassemblies can be thought of as an independent package, such as an assembled engine or a seat, of instantiated concepts of CO-PLM.

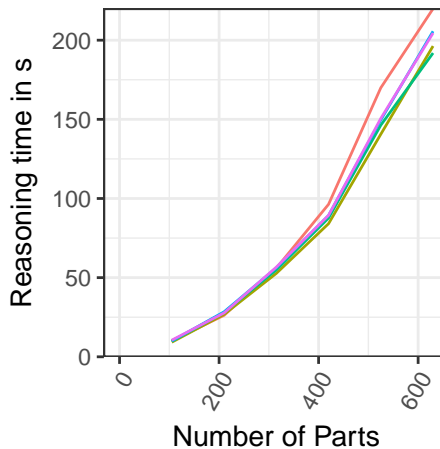


Figure 2.23: Reasoning series for synthetic data. Each colored line represents a reasoning series. A series consists of multiple reasoning runs with increasing numbers of parts. As part numbers increase, the reasoning time of the run increases.

A quadratic regression model ($R^2 = 0.9973$, $p < 0.001$ and $df = 3$) fitted to the mean of the runs in Figure 2.23 predicts that the reasoning time for a whole car will take around 16 hours. For the airplane, the estimation is 71 years. It becomes obvious that reasoning on whole products is not practical. In this example, every 100-part-chunk has the same structure. Therefore, in an optimal process, the independent reasoning of, for example, 1,000 parts (10 chunks) should take 10 times the time of 100 parts (1 chunk). In such a case, reasoning over the car triples would only take 17 minutes, and 7 days would be required for the plane. Thus, an approach for improving the reasoning time is needed. Reasoning could be accelerated by enabling reasoners to recognize instances of ontology patterns and then process these separately. As shown above, reasoning 10 chunks/ instances of ontology patterns separately is much faster than reasoning the same amount of data in one run. Another way to accelerate reasoning is selecting a more restrictive

OWL profile. As reasoning with OWL DL is most expressive but also the slowest, reasoning with one of the other OWL language profiles, EL, QL, or RL, could lead to potential compromises between performance and expressivity. We chose OWL DL in this work to include the full range of axioms presented in this work. The mentioned trade-offs of other OWL profiles in this context are possible topics for future work.

2.8 Synopsis

This work presents the core ontology CO-PLM to formalize product part information across all lifecycle phases from idea generation to disposal. CO-PLM is based on the foundational ontology DOLCE+DnS Ultralite. We evaluated the ontology with regard to its fulfillment of the functional and non-functional requirements and analyzed the reasoning times in practical use with increasingly complex products. The patterns are specified using OWL and description logic axioms. With CO-PLM, we addressed Challenge 2 of our scenario in Section 1.1. After introducing CO-PLM as the basis for product-related knowledge representations, we continue by discussing semDIM in the next chapter to address Challenge 1.

Chapter 3

Semantic Distributed Identity Management with semDIM

This chapter introduces semDIM, a semantic framework for distributed identity management. The semDIM framework was initially published in [72]. An extended journal-length article [74] introduces new discussion on consistency management and concurrency control in semDIM. This chapter is based on this work and provides additional content. In-depth views of the ontology design and the security analysis were added in this chapter. The chapter also elaborates on establishing data ownership and policy-based data validation in semDIM.

Distributed identity management refers to the ability to define distributed identities of agents and roles. This means a single agent can be represented using multiple unique identifiers managed in different namespaces and may have various roles across those namespaces. We propose semDIM, a novel framework for Semantic Distributed Identity Management based on a Semantic Web architecture. For the first time, semDIM provides a framework for a distributed definition and management of entities such as persons (who are part of an organization), groups, and roles across namespaces. It is suitable for informal, e.g., social networks, and for professional networks, such as in cross-organizational collaborations. In addition, the framework ensures authenticity, authorization, and integrity for such distributed identities by providing certificate-based graph signatures. Beyond the capabilities of existing identity management solutions, we allow distributed identifiers and management of groups (consisting of agents and sub-groups) and roles as “first-class entities.” semDIM uses `owl:sameAs` relations to represent and verify distributed identities via formal reasoning. This concept enables novel functionalities for distributed identity management; that is, entities can be identified, related to one another, and managed across namespaces. Our

semDIM approach consists of a modular software architecture, a messaging model introducing a novel approach for pattern-based concurrency control, and a set of state-of-the-art formal OWL ontology patterns. The use of formal patterns ensures semantic interoperability and extensibility for future requirements. We evaluate semDIM in the context of a real-world scenario for secure exchange of identity information across organizations.

3.1 Background and Structure of this Chapter

Existing identity management (IM) approaches typically focus on a single organization and, thus, centrally organized systems [48]. Solutions addressing distribution often focus only on managing identities, for example, see decentralized identifiers (DID) [90] or WebIDs [15, 89] (formerly FOAF+SSL). These approaches do not support the management of distributed groups nor distributed roles, both of which are needed in multi-organizational collaborations where independent organizations work on a common project. For example, identity information must be exchanged across companies to allow for managing cross-company working groups and sharing security policy based on them. (see Figure 3.1). Therefore, interoperability in distributed settings between identity management systems is the subject of current research [49].

3.1. BACKGROUND AND STRUCTURE OF THIS CHAPTER

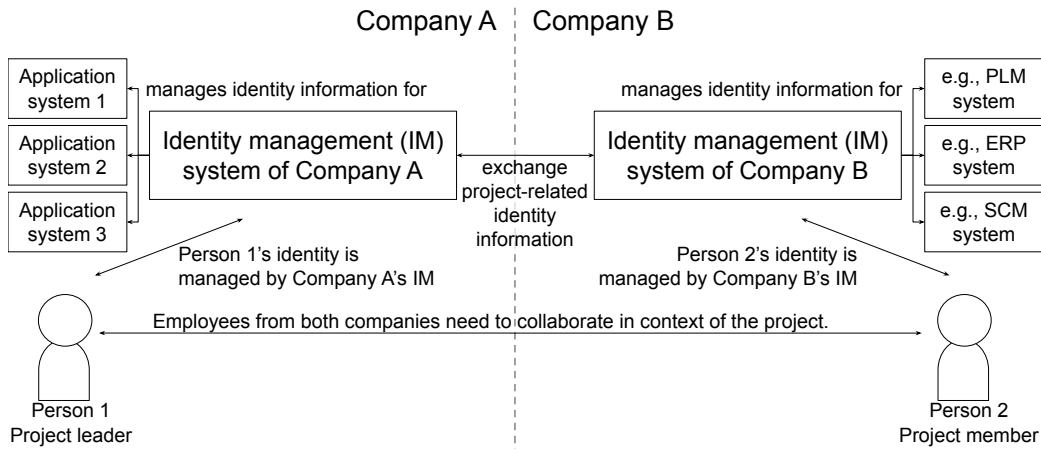


Figure 3.1: Distributed identity management. In classical identity management (IM), all identity information of one organization is handled by one system. In multi-organizational collaboration, each organization may have its own system landscape and identity management. Distributed identity management (DIM) addresses integrating such independent identity management systems.

Terminology in identity management: In the following, we will frequently use the terms person, agent, group, identity, role, and policy. Therefore, we first present definitions of these and some related terms taken from the literature. *Agent* is a collective term including natural as well as social or legal *persons*. It also includes *social agents*, such as *groups* or *organizations* [18]. Groups are sets of agents. Therefore, groups may contain other groups. Masolo et al. [55] discuss the term *role* in detail. We follow their formal definition of social concepts and, specifically, social roles. They describe roles as “[...] properties [...], which [...] have ‘dynamic’ properties [...as well as ...] a relational nature [...and ...] are linked to contexts.” The eXtensible Access Control Markup Language (XACML) [60] defines *role* as “[a] job function within the context of an organization that has associated semantics regarding the authority and responsibility conferred on the user assigned to the role.” This definition is not incompatible with the definition of Masolo et al. but is, rather, a subset of the latter. As Masolo et al.’s definition is the more general and comprehensive one, we use it. In the context of *Identity Management*, an **identity** is a “set of attributes” that are synonymous with properties “related to an entity” [36]. Examples for **attributes** are entity type, address information, telephone number, a privilege, a MAC address, and a domain

3.1. BACKGROUND AND STRUCTURE OF THIS CHAPTER

name. Attributes uniquely characterizing an identity are called an *identifier* in [36]. However, in this chapter, we use *identifier* synonymous with uniform resource identifiers (URIs), as the definition from [36] does not provide uniquely referable identifiers across namespaces. XACML defines *policy* as “[a] set of rules indicating which subjects are permitted to access which resources using which actions under which conditions.” While we mostly agree with this definition, we extend our understanding of policy to not only be applicable to access but also to flow and usage control like in [42].

Motivation for distributed identity management (DIM): The current lack of DIM causes operational costs for organizations and security risks in the sensitive topic of permission management. Without reliable DIM, changes such as removal of access rights for a group must be managed in each affected local identity management system instead of having a consistent process. Therefore, the risks related to misconfigurations, privilege creep, Man-in-the-Middle (MitM) attacks, and similar threats increases. The system landscape of most organizations is increasingly complex with diverse components from multiple providers, software developers, and hardware manufacturers each exhibiting their own functional and non-functional requirements, interfaces, and protocols. Some interfaces and protocols may even be proprietary and difficult to integrate with other frameworks. DIM must be integrated with many components of multiple organizations’ system landscapes. For this, DIM must offer a way to manage identities, groups, roles, and permissions across multiple namespaces. Existing approaches are limited to centralized solutions that are not interoperable or do not offer management of all relevant entities across namespaces. For example, current IM systems do not support associating identity **a-1** in group **g-1** with an identity **b-1** in **g-2**, where both identities refer to the same real-world person, but each identity has different group memberships and roles.

The Semantic Web approach *dg*FOAF [75] supports distributed authorities for managing groups, which is useful for scenarios like emergency response. However, the fact that users within a single organization have multiple identities due to the use of different IT systems is ignored, not to mention that users will have different identities in cross-organizational collaborations and need to access information systems outside their organization. Similarly, SOLID [54] addresses parts of DIM in the context of social web applications but does not address the management of multiple identities and their groups. Overall, there is no solution for a secure distributed definition and management of identities, groups, and roles across different namespaces, such as organizations or IT systems.

3.1. BACKGROUND AND STRUCTURE OF THIS CHAPTER

We propose Semantic Distributed Identity Management (semDIM), a semantic framework and data synchronization protocol that, for the first time, supports the distributed definition and management of identities, groups, and roles. Motivated by the foundational ontology DOLCE+DnS Ultralight (DUL) [18], we understand an entity as any kind of object. In our scenario of multiple organizations collaborating on a common project, we are concerned with distributed identities for persons, groups, and roles, as these are the entities most relevant for DIM. The semDIM framework is secure in the sense of ensuring confidentiality, integrity, and availability as well as authenticity and authorization of the distributed identity, group, and role management definitions. Relying on digital graph signatures [42] supports these security goals. In this context, a graph is a set of RDF-triples. This thesis relies on ontology patterns for knowledge representation. Therefore, in this work, graphs usually consist of instantiations of one or more of these ontology patterns. Graph signatures are comparable to digital signatures on documents. To the best of our knowledge, this is the first time graph signatures have been used to build trust networks to guarantee security properties in multi-organizational networks, such as in the scenario from Section 1.1. In summary, the contributions of semDIM are:

1. A generic, component-based client-server architecture that enables secure distributed information exchange between multiple organizations. A process-model for certificate-signed, graph-based client and server communication that models the interactions between the components of the semDIM architecture within and between different organizations.
2. A formal model of identity information based on a pattern-based core ontology. The semDIM ontology allows validating DIM requests using established reasoning engines.
3. A novel synchronization strategy to ensure consistency of DIM-specific information across distributed systems.
4. An evaluation of the semDIM framework based on functional requirements and non-functional security requirements for DIM, which are extracted from a real-world scenario.

Below, we discuss the related work. Section 3.3 presents a problem description with DIM-related details of the scenario from Section 1.1. The scenario features two variants, I and II, representing a loosely and a tightly coupled setting with different levels of required logging, respectively, which

will be referenced throughout the paper. Based on the scenario and the related work, Section 3.4 deducts requirements for a DIM solution. Section 3.5 discusses the pattern-based ontology used in our solution semDIM. In Section 3.6, we elaborate on the architecture and messaging model of semDIM. An evaluation of our solution against the requirements by applying it to the scenario from Section 1.1 follows in Section 3.7. In terms of the security requirements, we evaluate our solution in a standard-model-based security analysis in Section 3.8 before we summarize the results of this chapter's research in Section 3.9.

3.2 Related Work

We discuss the existing body of work in IM and the features offered in the current systems. We cluster the literature based on the DIM features they support. The clusters are a) Identity management *without* referencing across namespaces, that is, they do not consider uniquely referring to entities across namespaces. b) Identity management *with* unique identifiers; they uniquely refer to entities across namespaces but do not offer connecting and managing distributed entities. c) Identity management addressing distributed groups. These approaches support all features of a) and b) and additionally address the problem of group definitions being distributed over multiple sources of authority.

a) Identity management without referencing across namespaces:

Gai et al. [17] propose an attribute-based approach to group management. However, the approach does not consider connecting distributed entities. Their approach addresses security aspects such as confidentiality of transferred data as well as authorization and authentication through the encryption algorithms in the context of mobile clouds in the financial industry.

b) Identity management with unique identifiers:

Obrst et al. [62] propose a combination of OWL, Prolog, and a bit-level Java class to optimize reasoning on platform-independent access rules. As they are using a semantic basis and unique identifiers, their approach supports referring entities across namespaces. However, the connection of such entities, that is, the equality of instances and other relations between entities, is not considered. Therefore, it belongs to category b). The approach can be used to evaluate access rules for authorization. However, it does not address other security aspects, such as authentication or rules for data integrity and confidentiality. Another

3.2. RELATED WORK

approach in this category is the Semantic Access Control Policy Language by Hu et al. [29] based on XACML and OWL. XACML is a commonly used standard for attribute-based access control (ABAC). It defines an access control policy language and an architecture and a processing model for access requests. Hu et al. do not consider how the language can be used in a distributed environment to manage distributed entities. Their approach includes policy definitions but does not address other security aspects of distribution, such as validation of identity information changes. More approaches for identity and access management exist [13, 43, 66, 78, 79] that offer unique identifiers but do not consider distributed entities and distributed identifiers. Kirrane et al. give an overview of a collection of these approaches in their survey [45].

c) Identity management with support for distributed groups:

The approach closest to ours is dg FOAF [75], based on the wide-spread FOAF ontology [9]. Extending FOAF with a Datalog-based reasoning engine, dg FOAF allows defining and reasoning on group policies integrated into FOAF-profiles. However, dg FOAF does not support full distributed management of all relevant entities. First, dg FOAF does not allow connecting distributed identities, that is, related identifiers in different namespaces. Adding support for distributed group identifiers and distributed role identifiers to dg FOAF is not possible without significant changes to the approach because it is incompatible with the fundamental principles of dg FOAF: groups are equivalent if and only if they have identical group policies (including default members in the context of admin groups) and identical labels defined in each namespace they are used in (see [75] for details). This limitation of dg FOAF does not allow entities to be defined and managed across namespaces, such as remotely managing an admin group defined in a different FOAF-profile. Similarly, SOLID [54] addresses parts of DIM in the context of social web applications with a focus on social network settings. However, it does not address the management of multiple identities and their groups.

Related approaches: Some further relevant literature on identity management does not directly fit into the categories above. For example, WebID (formerly FOAF+SSL) is an authentication protocol based on FOAF-profiles [15, 89]. It offers only authentication but no other aspect of DIM, such as distributed identities, that is, connecting multiple different identities of the same person, groups, and roles. Any approach for DIM relying on certificate-based authentication and signatures could use WebID for authentication as a

3.2. RELATED WORK

subcomponent, as implemented by SOLID. However, WebID itself does not address the key challenges of DIM involving identity management, group management, and policy management. Koshutanski et al. [46] discuss a scenario similar to ours. They introduce “Digital Ecosystems.” In these, multiple organizations are connected by changing relations. Sometimes organizations collaborate, and at other times, they are rivals. In their example use case, multiple agents from different ecosystems need to negotiate with one another. Their approach is based on SAML [30] assertions and only considers authentication. Self-sovereign identities (SSI) is a concept that allows individuals to provide and manage their identities, for example, as used in the W3C standard of decentralized identifiers (DID) v1.0 [90]. Thus, SSI addresses only coining, providing, and resolving URI-based identifiers. DID explicitly encodes the parameters of identifiers into a single URI. This practice limits the stability of the URIs, as they depend on the specific methods used for coining them. Another related research area is policy languages. The literature about this area covers policy-related parts of DIM but does not address the whole spectrum of DIM. We refer to [40], [41], and [39] for a more detailed discussion on policy languages.

Table 3.1 shows an overview of the related work and compares it against the functional features of the categories a) to c). Category a) is not fit for DIM, as it does not support any distribution features. In contrast, category b) supports uniquely identifiable entities and therefore takes a large step in the direction of DIM. However, this category still misses significant functional features regarding distributed roles and groups. Category c) is very promising but falls short when applied in the context of real-life, multi-party collaboration, as not all entities can be defined, referenced, and managed in a distributed way. Only our approach, semDIM, supports all features required for a DIM system, that is, support for a distributed definition of identities, groups, and roles.

Table 3.1: Literature comparison against requirements from Section 3.4.

Identity management with ...	R1 Uniq. identifiers	R2 Dist. identities	R3 Dist. groups	R4 Dist. roles	R5 & R6 L.&Sec.
a) Standard features [17]	no	no	no	no	partial
b) Unique identifiers [62, 29, 13, 43, 66, 78, 79]	yes	no	no	no	partial
c) Distributed groups [75, 54]	yes	no	partial	no	partial
Related approaches [89, 46, 90]	yes/partial	no	no	no	partial
<i>Our approach</i> semDIM	<i>yes</i>	<i>yes</i>	<i>yes</i>	<i>yes</i>	<i>yes</i>

3.3 Problem Description

Below, we describe the general challenges for DIM that arise in a distributed collaboration, such as the one described in the scenario in Section 1.1. Subsequently, we introduce the project-specific DIM aspects of building the *Blue Train* from the scenario in Section 1.1.

3.3.1 DIM Challenges in Distributed Engineering

We describe Company A and Company B collaborating on a joint project to illustrate the challenge addressed by DIM. In this project, they are developing, manufacturing, and distributing a complex industrial product. As any larger organization, Company A and Company B are organized in different organizational units, e.g., divisions and departments. We only describe a simplified organization including key identities and group to keep the scenario simple. Each company manages its own organizational structure, identities, groups, and roles. We introduce a root identity for each company. This is a technical identity representing the root of authority in that organization. The identity does not need to belong to a real person (such as a super-administrator, a CEO, or any other person of authority), but it could also simply belong to a technical agent, for example, if only used once during the very first setup of an IM system. In a real-life setting, the chain of command may be more complex than simply having one root authority, such as multiple founders who all have equal rights. For the sake of brevity, we discuss the compact situation of one root authority per company. It is not feasible to expect the companies to agree on a shared company structure implemented by both companies. Even during the project, the company structures change.

Figure 3.2 depicts an overview of the key identities and groups available in Companies A and B in our scenario from Section 1.1. Usually, every organization has its own namespace, such as “a:” and “b:”, and may have several subnamespaces, including namespaces for departments or projects. As depicted in the figure, relationships between entities across organizational borders exist. For example, separate identities in different namespaces exist that refer to the same person (see a:a-1 and b:b-1 for Person 1 in Figure 3.2). In our context, a namespace defines a prefix of identifiers. It can relate to an organization, a part of an organization, or an information system. A similar equality exists between a:group-1 and b:group-1. Although they are technically two separate groups defined with different namespaces by their companies, they both describe the same group. This group is a project group consisting of employees of both companies. When determining who

3.3. PROBLEM DESCRIPTION

may access the shared project information systems, that is, who is a member of the project group, membership in either of the two groups is sufficient. Similarly, if a person is explicitly denied membership in one group, that person may not have membership in an equivalent group.

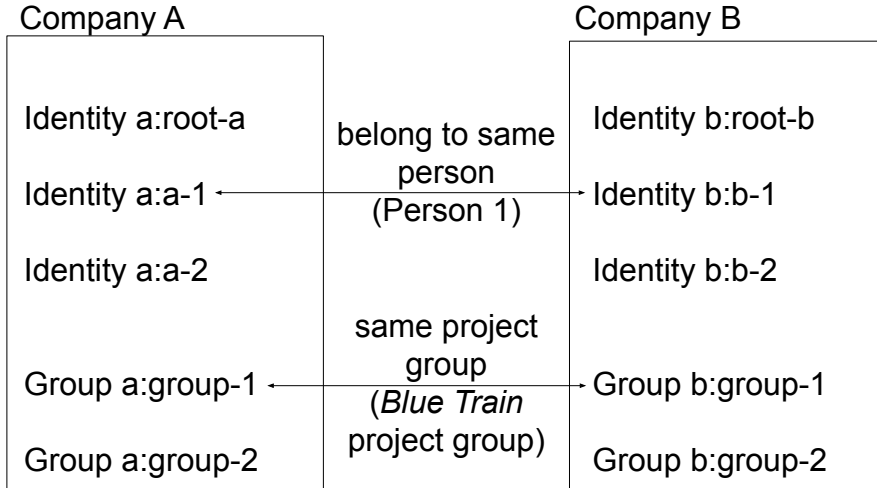


Figure 3.2: Identity management scenario with two companies A and B (with the corresponding namespaces *a:* and *b:*). Person 1 owns two identities, one in each namespace. The *Blue Train* project group is represented by two technical groups.

3.3.2 DIM for the *Blue Train* Project

To recapitulate the responsibilities of the companies regarding the *Blue Train* product from our scenario in Section 1.1, Figure 3.3 illustrates the functional components of the train. Company A is responsible for engineering technical specifications for the steam engine and the chassis, and Company B develops the cabin. In the following description, we use lower-case Roman numbers (i, ii, ...) to indicate aspects of the scenario that we will later reference for requirement deduction in Section 3.4.

Person 1, an employee of Company A, must work with IT systems of both companies (i). On the one hand, Person 1 needs to work in A's systems to design the steam engine and the chassis (*a:a-1*). On the other hand, Person 1 needs access to information from B regarding mechanical, hydraulic, and electrical assembly interfaces connecting the cabin and the chassis (*b:b-1*). In both companies, a security policy states that only identities defined in their own namespace have access to this confidential information. Thus,

3.3. PROBLEM DESCRIPTION

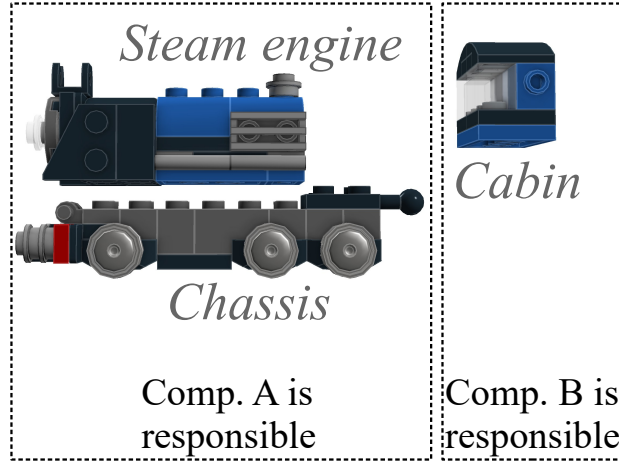


Figure 3.3: The two companies jointly designing the *Blue Train*. Company A designs the steam engine and the chassis, and Company B is responsible for the cabin.

Person 1 needs two identities, one for each company, to act in the namespaces of both of these two, which we call a **distributed identity**. Certain organizational changes involving Person 1, such as if Person 1 switched to another role, department, or even company, should influence the permissions of both identities, $a:a-1$ and $b:b-1$, for example, revoking certain system access rights (ii). Such changes related to identities are rather infrequent and small in size, compared to other processes in a project. For example, it is unlikely that one person is assigned to another department several times on the same day. Even in a week, it would be unusual for a person to switch departments more than once.

However, a more frequent process, such as reworking a 3D model for a technical part, may happen multiple times an hour and, therefore, has different requirements in the context of how often changes are processed in the information systems. Similarly, the data volume exchanged for operations in identity management is generally small. With the exception of bulk changes, such operations typically contain data in the range of kilobytes or a few megabytes, which is comparable to metadata changes in PLM. This is considered to be a small size compared to the payload of engineering operations, which often exceeds hundreds of megabytes or even gigabytes, such as when processing large 3D models. Person 1 is also the project leader for the *Blue Train* product and, as such, administrates a project group $a:group-1$ at Company A.

3.4. REQUIREMENTS

In Company B, an equivalent project group `b:group-1` exists. Person 1 (of Company A) is the project leader. Person 1 must administrate both technical groups, `a:group-1` and `b:group-1`, which both refer to the same logical **distributed group** of project members, but with different company namespaces (iii). The workflows describing the various tasks executed by the project group, such as adapting product changes to CAD data, Bill Of Materials, and manuals, require various roles, like design engineer, configuration manager, and quality assurance specialist. These **distributed roles** not only have to be able to be assigned to employees of both companies—and even external third party members, such as other partners, subcontractors, and temporary staff—but also need to be able to relate to the distributed identities and groups described above and the distributed information systems of the parties involved (iv).

The above-mentioned actions and any other relevant events must be persistently logged (v) to enable security reviews and audits on assigned and revoked roles. As most modern information systems provide logging of actions taken, this logging can be provided by build-in functionality on the database or application level (Variant I). However, if the information systems did not provide logging functionality or this logging information is not trusted (across companies), a separate IM logging is required for reliably tracing changes (Variant II). Furthermore, all collaborations and data exchanges between companies must comply with industrial information security standards (vi).

3.4 Requirements

From the discussion of the related work in Section 3.2 and its limitations, as well as the features for DIM motivated by the scenario in Section 3.3, we derive a set of functional and non-functional requirements to a DIM solution. The requirements R1 to R5 are functional with regard to DIM. They are essential to ensure that distributed identities, groups, and roles can be supported across organization borders. In addition, there are also non-functional requirements R6 to R9 that must be fulfilled by a secure DIM approach. These requirements concern information security and information consistency and are presented in the following subsections.

3.4.1 R1: Provision of Unique Identifiers for Persons, Groups, and Roles across Namespaces

For DIM, persons, groups, and roles must be *uniquely identifiable*, that is, they must be referenceable across namespaces, such as organizations (companies, universities, or social networks), via globally unambiguous identifiers. This requirement relates to (i) in the scenario. Note that each of the companies has project-related identity data that are shared within the project and data that are not related to the project, such as the identities of people not involved in the project. Therefore, we generally have to deal with a data fragmentation setting [52]. The same kind of fragmentation applies to groups and roles.

3.4.2 R2: Support for Distributed Identities

Identities uniquely refer to natural persons in namespaces, applications, or projects. *Distributed identities* extend the notion of identity by providing support for creating and connecting multiple, unique identifiers of the same person across namespaces. For example, the same natural person has identifiers on different IT systems. In the scenario, (ii) describes such a situation. Such connections between identities can be either within the same namespace or from different namespaces. In our scenario, the responsibilities of Person 1 requires identities in each namespace. The connection of two identities can be used for different purposes. The most basic is purely informational, that is, stating that the two identities belong to the same person without any technical implications. The second use is authentication; that is, the credentials (e. g., username and password) used for authenticating one of the identities can also be used for the other identity (see Single-Sign-On [38]).

In DIM, however, the connection between two identities must reach beyond such authentication, as the roles and group memberships of one identity can influence the other identity. For example, Person 1 is an employee of Company A. This group membership is explicitly stated for the identity $a:a-1$. If Person 1 is acting in the context of $b:b-a$, that is, fulfilling project-related tasks on one of Company B's information systems, certain actions may be forbidden for employees of different companies, e. g., because of legal or company policies. A naive approach would be to explicitly label $b:b-1$ as an employee of Company A by assigning it to the group of all employees of Company A. However, such redundant assignments result in additional administrative overhead and are error-prone. Another example is Person 1 potentially switching departments and thereby changing the

organizational structure, as mentioned in the scenario. This switch will only be explicitly applied to $\mathbf{a}:\mathbf{a}-1$ in the namespace of Company A. However, this switch also affects $\mathbf{b}:\mathbf{b}-\mathbf{a}$ and must be considered when evaluating which actions Person 1 is allowed to perform in Company B's namespace. Therefore, a DIM solution must support evaluating role and group assignments for distributed identities across namespaces.

3.4.3 R3: Support for Distributed Groups

Groups consist of one or more persons or other groups [75]. This includes persons and groups from different namespaces, e.g., persons and groups can be members of different organizations. Like distributed identities for persons (see R2 above), *distributed groups* must support multiple identifiers to refer to the same group across namespaces. For example, as discussed in the scenario, shared information systems must process requests regardless of namespace, for example, when querying information about a distributed group. Therefore, they need to work with distributed identifiers for the same (logical) group. This is described at (iii) in the scenario. Again, multiple identifiers referring to the same group may exist in either the same or in different namespaces.

In combination with R2, this can lead to different constellations regarding the namespaces when considering the administration of distributed groups. For example, a distributed group could have only one identifier in one namespace but multiple administrators from the same namespace and different namespaces. Alternatively, a distributed group could have one identifier in one namespace and another identifier in a different namespace, while one administrator is responsible for the whole group, that is, in the context of both identifiers of the group. The latter case describes the situation in our scenario (iii).

The reasons for such different constellations may be technical, such as security aspects, or organizational, such as avoiding unintentional changes on company structures outside of the project's context. Here, the actions which Person 1 is required to fulfill in Company B's namespace require an identity in the same namespace as stated by a security policy. A DIM solution must support any of such constellations.

3.4.4 R4: Support for Distributed Roles

Roles describe a set of tasks or responsibilities executed or held by a person or group this role is assigned to. A role, like persons and groups, must be uniquely identified within a system. *Distributed roles* must be connectable

and manageable from different namespaces. It must be possible to have different role identifiers in different namespaces. These different roles must be interchangeable in authorization checks because they refer to the same logical role. For example, two roles for the membership in the project group `group-1` exist in our scenario (see Figure 3.2). One role is defined in the namespace of Company A (`a:group-1`) and another in the namespace of Company B (`b:group-1`). All agents holding either of these two membership roles belong to the joint *Blue Train* project group. Thus, there are two identifiers (one per company) for the project member role, which are semantically equivalent with regard to the project (see (iv) in the scenario).

3.4.5 R5: Logging of DIM Activities

As data are exchanged across company borders, logging becomes essential for tracing who transferred which data and how the data were handled. As mentioned in (v) in the scenario, identity information also needs to be logged. This need for logging may arise from different requirements, such as personal, business, and legal [68]. This requirement is optional for a DIM solution, as most modern information systems offer logging capabilities. As discussed in the scenario in Section 3.3.2 (v), the standard system logging may be sufficient (Variant I) or, otherwise, the DIM solution must offer additional logging (Variant II). Ideally, the transferred identity data itself carries evidence about authorship, legitimacy, and time of processing, for example, as implemented in sticky logging [68].

3.4.6 R6: Guarantee of Information Security

The primary goal of information security is ensuring information is “not made available or disclosed to unauthorized individuals, entities, or processes” (**confidentiality**), provides “accuracy and completeness” (**integrity**), and is “accessible and usable on demand by an authorized entity” (**availability**) [34]. Furthermore, **authentication** is the “provision of assurance that a claimed characteristic of an entity is correct” [34]. Transferring these principles to DIM, as indicated by (vi), has the following information security requirements:

- (a) Identity, group, and role information may only be readable for authorized agents (confidentiality).
- (b) This information may only be changed by authorized agents; for example, only group admins may add members to an administrated group (integrity).

3.4. REQUIREMENTS

- (c) This information needs to be available to the authorized agents in a distributed environment (availability).
- (d) Persons claiming to own identities, group memberships, or role assignments are required to provide sufficient proof (authentication).

As discussed in Sections 3.3.2, Scenario Variant II requires additional security measurements to enable persistent logging, monitoring, and validation of events.

3.4.7 R7: Guarantee for Consistent Distributed Identity Information

The balance between information consistency and availability in distributed systems is a regularly discussed trade-off and commonly referred to as the CAP-Theorem [8, 21]. The CAP-Theorem states that only two of the following three properties can be guaranteed in distributed systems: **Consistency**: reading data returns the same values at each node, **Availability**: requests are always processed, although not necessarily with the most current data, and **Partition tolerance**: even with network delays or failures, the system keeps operating. In our context, nodes are synonymous with servers holding and processing identity information.

As stated in the scenario, identity information changes rather infrequently (in general, not multiple times per hour) and is of low data volume. Therefore, high performance, which is closely related to availability in the context of transactional throughput, is not required for changes in DIM. The consistency of the distributed identity information, however, is of major significance. This means querying identity information, e. g., if an identity is assigned a certain role, must deliver correct responses, regardless of which node is queried in the DIM system. If queries rely on stale data caused by insufficient concurrency management, unintended system behavior could occur. A DIM solution must ensure proper identity information consistency to avoid this.

3.4.8 R8: Support for Different Degrees of Technical Coupling

Information systems relying on web technologies can generally be considered loosely coupled [12]. Our scenario illustrates that collaboration between organizations may require different degrees of coupling. This manifests in the two variants described in our scenario. Variant I features a tighter

coupling than Variant II. In tightly coupled systems (Variant I), messages sent from one system are considered to be reliably received, trusted, and timely processed by the receiver. No separate logging of the DIM processes is necessary. In loosely coupled systems (Variant II), messages might need to be tracked and inspected at a later point in time even by external parties, such as auditors or regulatory forces. A DIM solution must support tight as well as loose coupling and be flexible enough for different collaboration settings and existing IT landscapes.

3.4.9 R9: Concurrent Operations

Although the frequency of changes is low (see R7), multi-user systems—such as DIM—may become extremely slow if non-conflicting operations cannot be processed concurrently [44]. Therefore, a DIM solution must provide a strategy to efficiently, yet consistently, exchange information between the nodes and prevent conflicting operations. This strategy must be specifically adapted to DIM, as it is unfeasible to exchange the entire set of all identity information for every read or write operation. In contrast, synchronizing only on the data object or row level could lead to inconsistencies, for example, if a role assignment is not synchronized as a whole but only parts of the assignment. Therefore, a change-based synchronization mechanism tailored to DIM must be in place. This mechanism must ensure that relevant entities and their relations are exchanged when changes are processed.

Furthermore, appropriate locking decisions must be conducted based on such relationships for both read and write operations. The relevant relationships between entities are not inherent or obvious. For example, if a role assignment is proposed by an assigner, the relevant administrative role that authorizes the assigner is also related to this process. Some inherent relations may exist in the context of IM, such as hierarchical relations. In IM, groups and roles often have hierarchical relations similar to part-of relations. Roles may have subroles, and groups may have subgroups. In general, a sub entity inherits properties from the entity higher up in the hierarchy. For example, if the role *MemberOfDepartmentHR_role-1* is a subgroup of *EmployeeOfCompanyA_role-1*, *MemberOfDepartmentHR_role-1* inherits the roles from *EmployeeOfCompanyA_role-1*. However, the relations between data objects in DIM are not only hierarchical. This means that if a change is processed, the relevant data objects are blocked and synchronized across the nodes. An example of such a change is a role assignment in which the relevant data objects are the ones affecting the assignee, the assigned role, and the assigner that do not have direct hierarchical connections to each other. Ideally, as few other, non-relevant data objects are blocked as possible.

Therefore, a DIM solution must offer a synchronization strategy supporting concurrent operations with a balanced, fine granularity specifically adapted to the DIM use case.

After defining the requirements for a DIM solution, we present our solution `semDIM` in the following sections. For this, Section 3.5 elaborates the ontology patterns developed to represent IM information. Section 3.6 follows by presenting the architecture and the message model of `semDIM`. Section 3.7 demonstrates the use of the patterns in the context of the message passing defined in `semDIM` for the scenario from Section 1.1.

3.5 Design of the `semDIM` Core Ontology

The triples created and communicated between clients and servers within a company, and more importantly, between servers of different companies, are created based on a pattern-based core ontology for representing DIM information. Below, we provide an overview of this core ontology, depicted in Figure 3.4.

The design of `semDIM`'s core ontology is developed following a pattern-based approach based on foundational and other core ontologies, as described in Scherp et al. [70]. This ensures modularity, extensibility, and reuse of established concepts. We base our ontology on the foundational ontology DOLCE+DnS Ultralight (DUL) [18], as it provides strong semantics and supports pattern-based extensions. Thereby, our solution ensures compatibility with other frameworks relying on DUL-based ontologies. The common use of shared concepts guarantees semantic integrateability. We reuse the core ontology `strukt` for distributed workflows [69], `InFO` for policies [41], and certificate and signature patterns from [40]. This enables associating policies with group and role management. Furthermore, the DIM ontology can be combined with other core and domain-specific ontologies. For example, as our scenario is set in the PLM context, we integrate `CO-PLM` [71] for phase-dependent semantic PLM. This integration allows connecting general DIM entities of identities, groups, roles, and DIM policies and workflows with PLM-specific concepts. For example, specific files on a file share that belong to a certain project and are of a specific document type (e. g., design plans, manuals, etc.) may be regulated by a policy to only be accessible by project members of certain organizations with a designated role. The following section discusses the ontology patterns of `semDIM` in detail.

We use the following symbols in the overview diagram (see Figure 3.4) and in the following pattern figures: A gray box indicates a class imported from an

3.5. DESIGN OF THE SEMDIM CORE ONTOLOGY

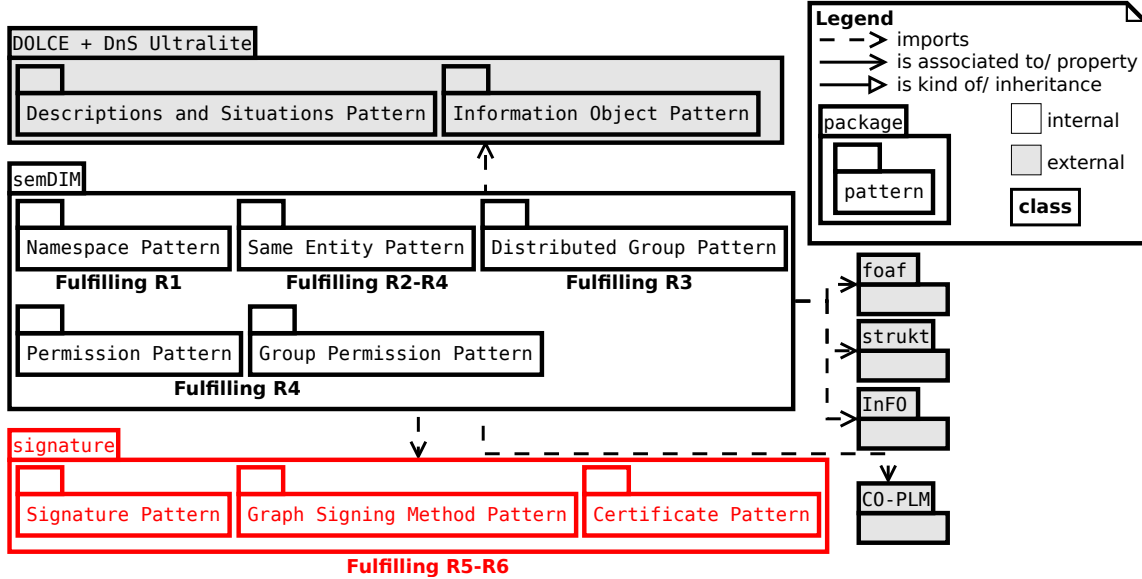


Figure 3.4: Overview of the pattern of the semDIM core ontology, the imported ontologies, and how they match to functional requirements (see Section 3.4). The box with the bent corner contains the legend. This notation is used in all following figures describing ontology patterns. Red patterns are only needed for Scenario Variant II. The numbered “R”s stand for the requirements R1–R6 from Section 3.4.

external ontology (such as DUL). White boxes describe classes introduced by semDIM. Open arrowheads represent property relations. A closed triangular arrowhead points to a superclass of the class the arrow starts from.

3.5.1 Pattern for R1: Namespace Pattern for Unique Identification of Entities and Namespace Ownership

Any solution relying on URIs, such as $_{dg}$ FOAF, fulfills R1, as these identifiers can be used across namespaces. Implicitly, the relation between an entity and its namespace (including any top or subnamespaces) can be extracted from the entity’s URI. As semDIM relies on URIs for referencing entities, it fulfills R1. In addition, we explicitly introduce the Namespace Pattern (see Figure 3.5) for relations between namespaces, their entities, and their owners, as semDIM allows reasoning upon these relations. For example, an

agent may only add triples to a namespace if that agent has the required ownership or, alternatively, a delegated permission from the owner.

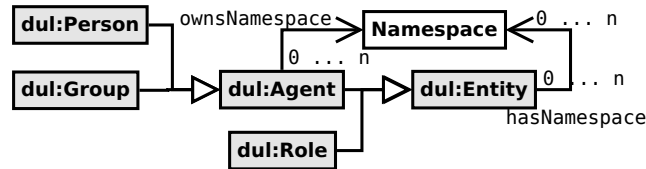


Figure 3.5: Namespace Pattern. Entities can belong to multiple namespaces, including subnamespaces. Namespaces may have multiple owners, that is, agents having full control over defining entities in the respective namespace.

3.5.2 Pattern for R2–R4: Same Entity Pattern

Persons, groups, and roles can be connected with the Same Entity Pattern (see Figure 3.6). For example, identities referring to the same person can be connected by stating their semantic equivalence. The properties introduced in Figure 3.6 are sub-properties of `owl:sameAs`. The pattern can be applied to identities and also to groups and roles in the same way. Therefore, this pattern also contributes to R3 and R4. The major part of distributed groups and roles is their management, which is addressed by the next patterns. The `<instance>` annotation indicates that these `same...As` subproperties relate to the instances of the respective classes, `Person`, `Group`, and `Role`.

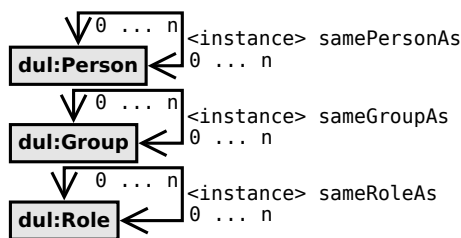


Figure 3.6: Same Entity Pattern. Subproperties to `owl:sameAs` are introduced for explicitly connecting related instances of persons, groups, and roles even across namespaces. The `<instance>` annotation indicates that these properties are used in the context of instances and does not mean the classes themselves are `owl:sameAs` or `owl:equivalentClass`. The latter is inherently true, however, as classes are always equivalent to themselves.

3.5.3 Pattern for R3: Distributed Groups Pattern

Distributed groups must be able to contain agents from different namespaces as members (see Figure 3.7). For this, we reuse the `dul:hasMember` property from DUL, which has `Group`'s superclass `Collection` as its domain and `Agent`'s superclass `Entity` as its range. Also, groups need to be administrated by agents, which we represent with the `administratedBy` property. Again, agents can be persons or groups in this context. Both `Groups` and `Agents` may have different origin namespaces in this pattern. Therefore, using this pattern allows the management of distributed groups across companies and addresses R3.

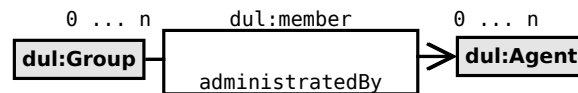


Figure 3.7: Distributed Groups Pattern. Distributed groups have members and administrating agents from potentially multiple namespaces.

3.5.4 Pattern for R4: Permission Patterns (for Distributed Roles)

DUL's Descriptions and Situations (DnS) pattern [18] is essential for the design of ontology patterns in both CO-PLM and semDIM. The DnS pattern was thoroughly discussed in Section 2.15. The Permission Patterns introduced in this section are based on DnS. They reuse concepts from `strukt` [69], which defines `StructuredWorkflow`, a subclass of `Description`. A `StructuredWorkflow` defines `Roles` and their `Tasks` as well as their transitions. It is satisfied by `StructuredWorkflowExecutions`, a `Situation` subclass. Figure 3.8 shows the Permission Assignment Pattern.

A `PermissionAssignmentWorkflow` defines the `Role` to be assigned as well as the `Task` related to this `Role`. Also, an `AssignerRole` is defined. It describes which `Role` is needed for the `Assigner` to legitimately propose the permission pattern. Which `AssignerRole` is necessary for an `Assigner` to be allowed to propose a permission pattern depends on the IM policies. The standard policy for DIM states that only the namespace owner, as defined by the `Namespace` pattern from Figure 3.5 in Section 3.5.1, of a specific role may assign it. The namespace owner, in return, may delegate parts of its namespace to a different `Agent`. For example, the root identity of a company might define departments and delegate the namespace "a:HR" as part of

3.5. DESIGN OF THE SEMDIM CORE ONTOLOGY

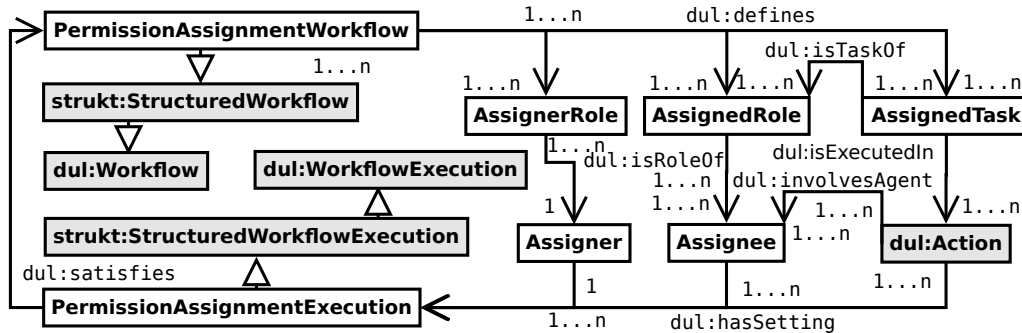


Figure 3.8: Permission Pattern. Based on the DnS, all concepts required to define permission assignment workflows and their executions are introduced.

the top namespace “a:” to the head of the HR department. Furthermore, the permission to assign a Role can explicitly be transferred. For example, assigning the role `group_admin-1` to `a-1` then lets `a-1` use this Role to assign the Role `group_member-1` to any other Agent (this example is the same as the example later shown in Figure 3.10).

A `PermissionAssignmentExecution` is the concrete Situation for a `PermissionAssignmentWorkflow`. It sets the `Assignee` (receiving the `AssignedRole`) and the `Assigner` (assigning the `AssignedRole` to the `Assignee`). The `Action` in a `PermissionAssignmentExecution` is a placeholder for activities attempted by the `Assignee`. It can be used to reason whether an action is valid in the context of the assignment; that is, the `Action` executes (`dul:executesTask`), one of the `Tasks` defined in the `PermissionAssignmentWorkflow`.

A `PermissionAssignmentOnGroupWorkflow` is a special type of `PermissionAssignmentWorkflow` related to a certain `Group`. Figure 3.9 depicts the Group Permission Pattern. It extends a generic permission pattern by adding `AffectedGroupRole` and `AffectedGroup`.

A common example of a group permission is adding or deleting a group member, group administrator, or group owner. `semDIM` allows custom definition of such permissions, meaning that `Groups` do not necessarily need dedicated admins but could be managed by group owners, a namespace owner, or a combination of any such. `PermissionAssignmentOnGroupWorkflows` define the `AffectedGroupRole` to which the `AssignedRole` relates, for example, `administrated_group_role-1`. Likewise, `PermissionAssignmentOnGroupExecutions` set the concrete `Group` for which that assignment is valid.

3.5. DESIGN OF THE SEMDIM CORE ONTOLOGY

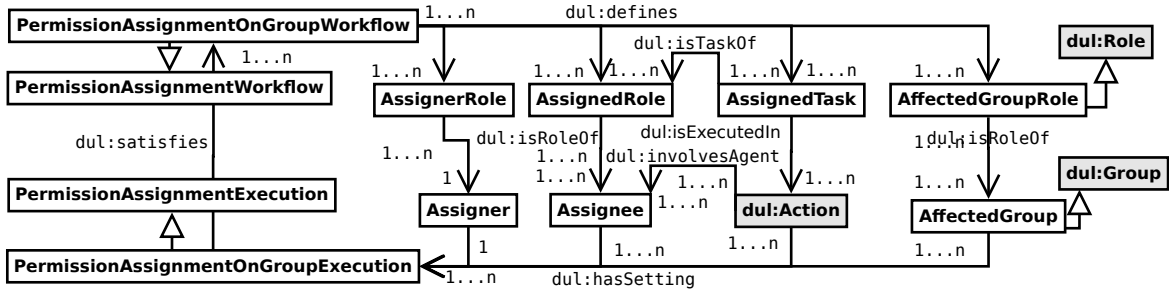


Figure 3.9: Group Permission Pattern. The Permission Pattern is extended by group-related concepts.

Group Permission Admin Example Instantiation: Figure 3.10 shows a group permission example with an admin permission assignment workflow `admin_perm-1` and its execution `admin_assign-1`. `admin_perm-1` defines that the permission assigner role `namespace_owner-1` is allowed to assign the role `group_admin-1`, which has the task `group_administration-1` for the group role `administrated_group_role-1`. With assignment `admin_assign`, the assigner `root-a-1` assigns `a-1` the role `group_admin-1` for `group-1`. This can be aggregated to: `a-1 semDIM:administrates group-1`, which is signed by `root-a-1`.

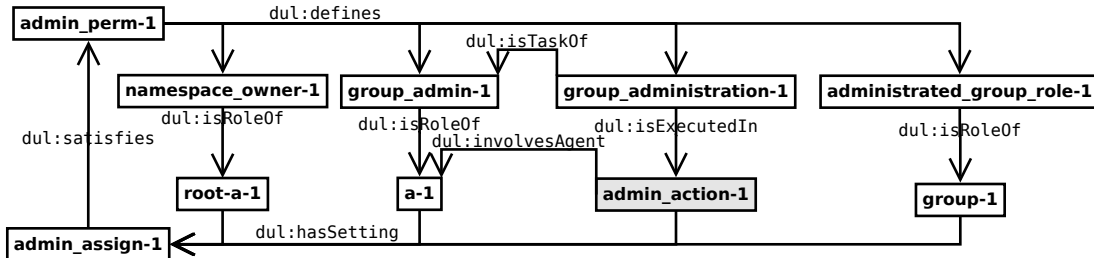


Figure 3.10: Group Permission Admin Example Instantiation. In the *Blue Train* scenario, the root identity `root-a-1` assigns `a-1` admin permissions for `group-1`.

3.5.5 Patterns for R5–R6: Signature Pattern, Certificate Pattern, and Graph Signing Pattern

To ensure integrity (R6) of transferred identity information in loosely coupled settings (see Variant II described in Section 3.3), semDIM uses signed

graphs. Furthermore, signed graphs can be persistently stored for logging (R5) exchanged identity information. In contrast to unsigned graphs, signed graphs contain proof of authorship. Only the owner of the private key of the certificate used for the graph’s signature could have authored the graph and its signature. By adding a timestamp, this graph signing approach produces identity data carrying proof of authorship, legitimacy, and time of processing (R5). Therefore, these signed graphs can later be reviewed in security assessments or audits (Variant II).

For signing graph data, we rely on the graph signing framework from Kasten et al. [42, 40]. For integration into semDIM, we reuse the Signature Pattern, Graph Signing Method Pattern, and Certificate Pattern Agents. The Signature Pattern (see Figure 3.11) is used to represent a signature. In principle, the pattern can be used for the signature of arbitrary `dul:Things`, but we use it specifically for signing graphs containing identity information here.

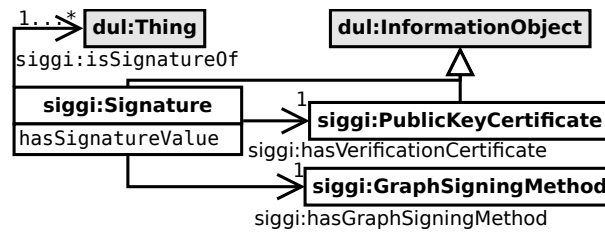


Figure 3.11: Signature Pattern. Concepts for signing arbitrary `dul:Things`, such as graphs, are presented. Adapted from [40].

We use the Graph Signing Method Pattern to represent the various sub-methods required for graph signing, such as serialization or hashing.

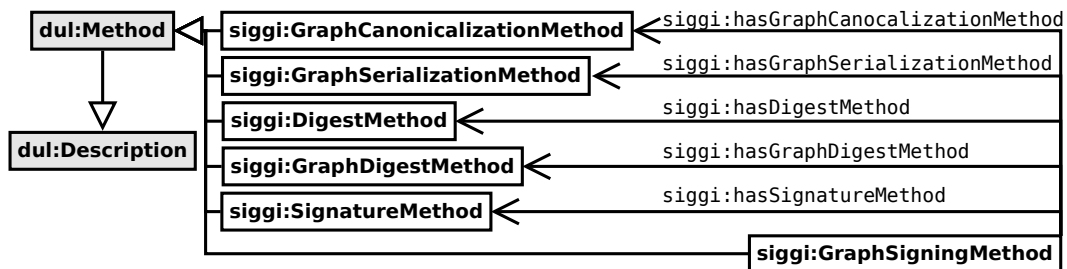


Figure 3.12: Graph Signing Method Pattern. Different methods required for graph signing are presented. Adapted from [40].

3.6. SEMDIM'S ARCHITECTURE AND SECURE IDENTITY INFORMATION EXCHANGE

The pattern illustrates how a signing method consists of various sub-methods, such as serialization or digesting. The Certificate Pattern (see Figure 3.13) allows agents to be assigned PGP¹ or X509² certificates, which can be processed by the signing framework. Figure 3.12 shows the Graph Signing Method Pattern.

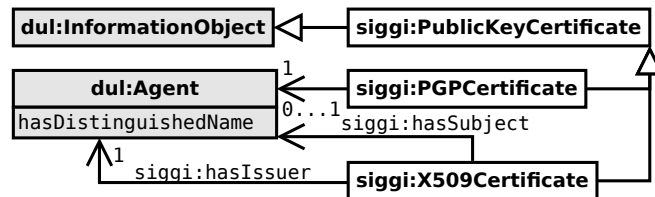


Figure 3.13: Certificate Pattern. PGP and X509 certificates are supported. Adapted from [40].

3.6 semDIM's Architecture and Secure Identity Information Exchange

In the last section, we developed ontology patterns for the knowledge representation in semDIM. This section presents semDIM's architecture and how it securely exchanges identity information across namespaces. First, we present the component-based architecture in Section 3.6.1. Next, Subsection 3.6.2 discusses the steps in semDIM to exchange identity information based on the above patterns across organizations. Section 3.6.3 introduces our novel Pattern-based Concurrency Control (PbCC) mechanism. This mechanism is developed for and used in semDIM to ensure the consistency of pattern-based data in the presence of conflicting concurrent read and write attempts. Lastly, we discuss the role of data ownership in semDIM's information exchange and how it can be used to validate proposed changes of identity information across different namespaces in Section 3.6.4.

¹<https://web.pa.msu.edu/reference/pgpdoc2.html>, last accessed on 2021-01-16

²<https://www.itu.int/rec/T-REC-X.509>, last accessed on 2021-01-16

3.6.1 Component-based Architecture of semDIM

We propose a component-based client-server architecture [84] on the Semantic Web, depicted in Figure 3.14 as an UML 2 Deployment Diagram³. The architecture supports the two scenario variants from Section 3.3. For Variant I, it offers data transport protection and authentication with standard Transport Layer Security (TLS)⁴. For Variant II, TLS is used in addition to signing the graph-based messages between servers to further increase data integrity. Figure 3.14 illustrates all semDIM components and their connections for both variants. Any activity in semDIM, such as creating new groups or adding group members, is represented as a semantic graph. These graphs are exchanged between client and server through a safe communication channel.

Without a loss of generality, we assume in the following a basic setup consisting of several clients (**Client-a-1**, **Client-b-1**, etc.) and one server per organization (**Server-a** and **Server-b**) running on separate logical systems. If necessary, this model can be expanded by multiple servers per organization. One option is to determine one server per company as the master server. Alternatively, additional servers could be handled as if they were servers of additional companies.

In the case of Variant II, each server runs a Sign Engine such as by Kasten et al. [42]. A triple store on the server is used as persistent storage for valid triples. Triples are valid if their requirements are met, such as when a user who is attempting to add a group member actually has the group admin role of that group. The connection between client and server is secured with TLS to ensure confidentiality of the transmitted information. In a more advanced setting, the client may have its own triple store for local reference when creating or updating triples before signing graphs. There also may be multiple interconnected servers within one company.

The **Client Agent** is the software program running on the user's client. Theoretically, the user could read and write RDF graph requests manually, but this would be quite impractical. Therefore, the Client Agent offers data input through guided user interaction. It processes data into graph form and handles the messaging and the responses from the server. Furthermore, the Client Agent could be connected to other applications running on the client requiring or affecting IM information, such as collaboration tools. The Client Agent is the only client-side component. Alternatively, the Client Agent could also run directly on the server and be exposed to the user by a web interface. All other components must run on the server. The **Request**

³<https://www.omg.org/spec/UML/2.5.1/PDF>, last accessed on 2021-01-16

⁴<https://tools.ietf.org/html/rfc8446>, last accessed on 2021-01-16

3.6. SEMDIM'S ARCHITECTURE AND SECURE IDENTITY INFORMATION EXCHANGE

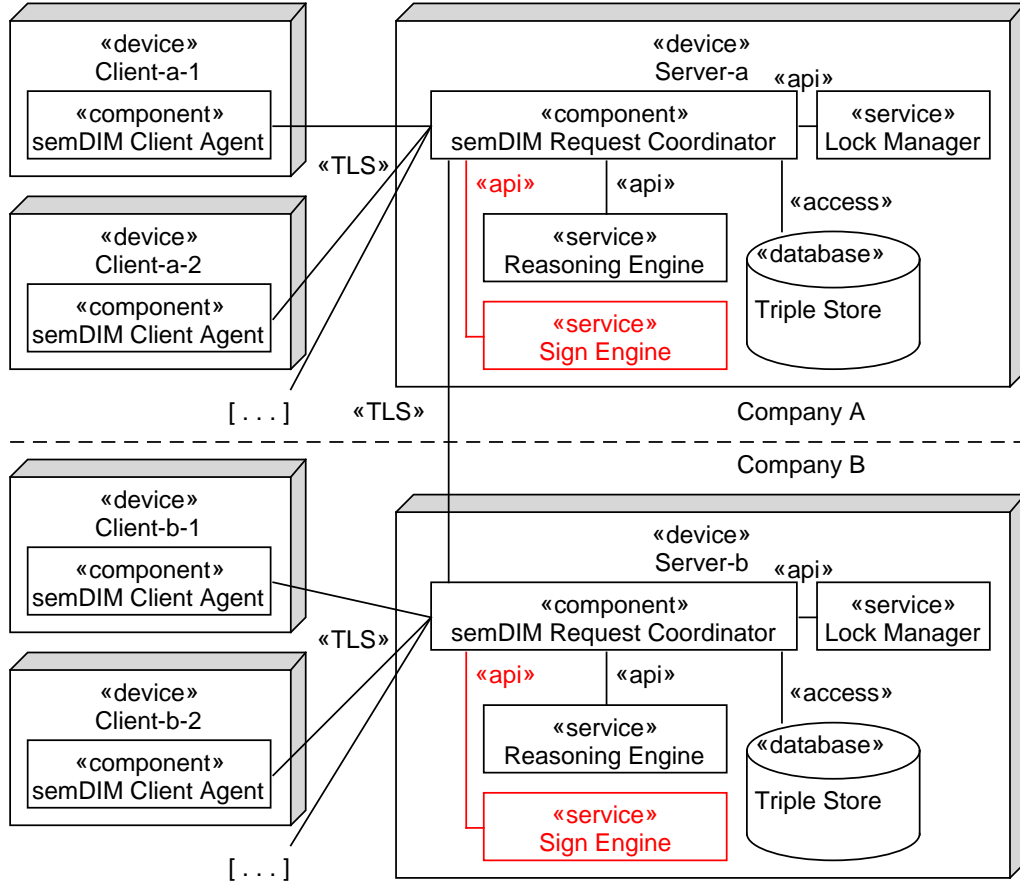


Figure 3.14: semDIM client-server architecture featuring two companies, each owning one DIM Server and an arbitrary number of clients. Components in red are only required for the Scenario Variant II.

Coordinator acts as a central component of semDIM. It offers an interface for incoming requests coming from the Client Agents or other servers' Request Coordinators as well as the Sign Engine in the case of Variant II. The Request Coordinator communicates internally with the Reasoning Engine and Triple Store. The **Sign Engine** is only used in Variant II. It signs graphs before they are sent to other servers, and it verifies the signatures of graphs received by other servers. The **Reasoning Engine** checks for any inconsistencies in the requests received from the Request Coordinator, such as when a user not authorized to add a group member tries to do so. The reasoning consists of

formal consistency checks regarding the ontology patterns. Here, different OWL reasoning engines can be integrated, such as Pellet [80] or Hermit [77]. In addition, the Reasoning Engine verifies if the requests are valid in terms of the agreed identity management policies, e.g., the policy “only group administrators can add new group members.” The **Triple Store** is used as persistent storage of identity information. The **Lock Manager** is responsible for keeping track of set locks on the data objects held in the Triple Store. It is essential for avoiding concurrent changes on the same or on different Triple Stores. This feature is commonly called concurrency control and is discussed in detail in Section 3.6.3.

3.6.2 Message Passing between the semDIM Components

We introduced the components of semDIM's architecture in the previous section. Based on these, we present the secure, ontology-based message passing between clients and server and between server and server. Figure 3.15 shows the corresponding UML sequence diagram. The semDIM server-to-server message passing for distributed collaboration is based on the popular two-phase commit protocol (2PC) [52], shown in steps (6) and (10–14). The Sign Engine is only needed in steps (2) and (4) to log activities, that is, in Variant II of the scenario (the components drawn in red in Figure 3.15). We first describe the full sequence diagram based on 2PC before we discuss possible alternative protocols.

Message passing in semDIM using 2PC: For DIM, the first step is that (1) a user client creates new triples locally, for example, adding somebody to a group that a user administrates via the Client Agent. (2) The Client Agent sends these new triples to Server A via a secured channel to trigger the intended action, which is, in this case, assigning group membership to somebody. In this step, the process switches from Client A to Server A, as indicated by the dotted line in Figure 3.15. The Request Coordinator receives the new triples and starts the evaluation process. (3) The Request Coordinator sends a query to validate the prerequisites for accepting the triples in the graph, such as whether the signer has a necessary role, such as group administrator or namespace owner. (4) The Reasoning Engine processes the triples stored in the Triple Store together with the client graph and decides if the identity information transmitted in the graph, such as a permission assignment, is valid. In the case of Variant II, one further step applies: (5) Server A signs the client graph with its private key using the

3.6. SEMDIM'S ARCHITECTURE AND SECURE IDENTITY INFORMATION EXCHANGE

Sign Engine and receives a signed graph of the new triples. (6) Server A sends this graph (for Variant II: plus the digital graph signature) via the Request Coordinators to Server B. Here, the process switches from Server A to Server B, again, indicated by a dotted line in Figure 3.15. For Variant II only: (7) Server B validates the signature of the sent graph with the public key of Server A via the Sign Engine. (8 and 9) Server B evaluates the request like Server A did before. These two steps are like (3) and (4). (10) Server B confirms the validity of the action to Server A. If more than two companies are involved, Server A will wait for all confirmations before continuing. (11) After receiving all confirmations, Server A applies the changes on the Triple Store to make it persistent. (12) Server A then sends the commit command to all other servers (in our scenario, Server B is the only other server) so they can also make the changes persistent. (13) Server B applies the changes to its Triple Store like Server A did in (11). (14) By sending an acknowledgment, Server B lets Server A know that the changes were made persistent on Company B's side. In another setting with more than two companies, each additional server would send an acknowledgment. Finally, (15) after all sides applied the changes, Server A informs the client about the success of the process.

Regarding the server-to-server communication, the two phases, *commit-request* and *commit*, of the 2PC protocol are supported in the following way: For the commit-request phase, Server A sends a commit request in the form of the transmitted graph in step (6) and waits for confirmation, that is, Server B voting "yes" for the change, in step (10). Based on the voting, in the commit phase, Server A either sends an abort command or submits the commit command (12), which is processed (13) and acknowledged (14) by Server B. In semDIM, the Request Coordinator of that server takes over the 2PC coordinator role, where the change is triggered (Server A in our example).

Discussion of alternative commit protocols: While 2PC is a de facto standard for a distributed commit process [44], there are alternative variations, including tree 2PC, dynamic 2PC, and three-phase commit (3PC) [44]. Tree 2PC is applied on networks with hierarchically ordered computing nodes, where voting decisions are propagated transitively from the leaf nodes to the root node. Dynamic 2PC is basically tree 2PC without predetermining which component is fulfilling the coordinator role, which, in 2PC and tree PC, is always the server triggering the change. Both tree PC and dynamic 2PC could be reasonably used and extended to the semDIM framework if the scenario in Section 3.3 modeled a hierarchical,

3.6. SEMDIM'S ARCHITECTURE AND SECURE IDENTITY INFORMATION EXCHANGE

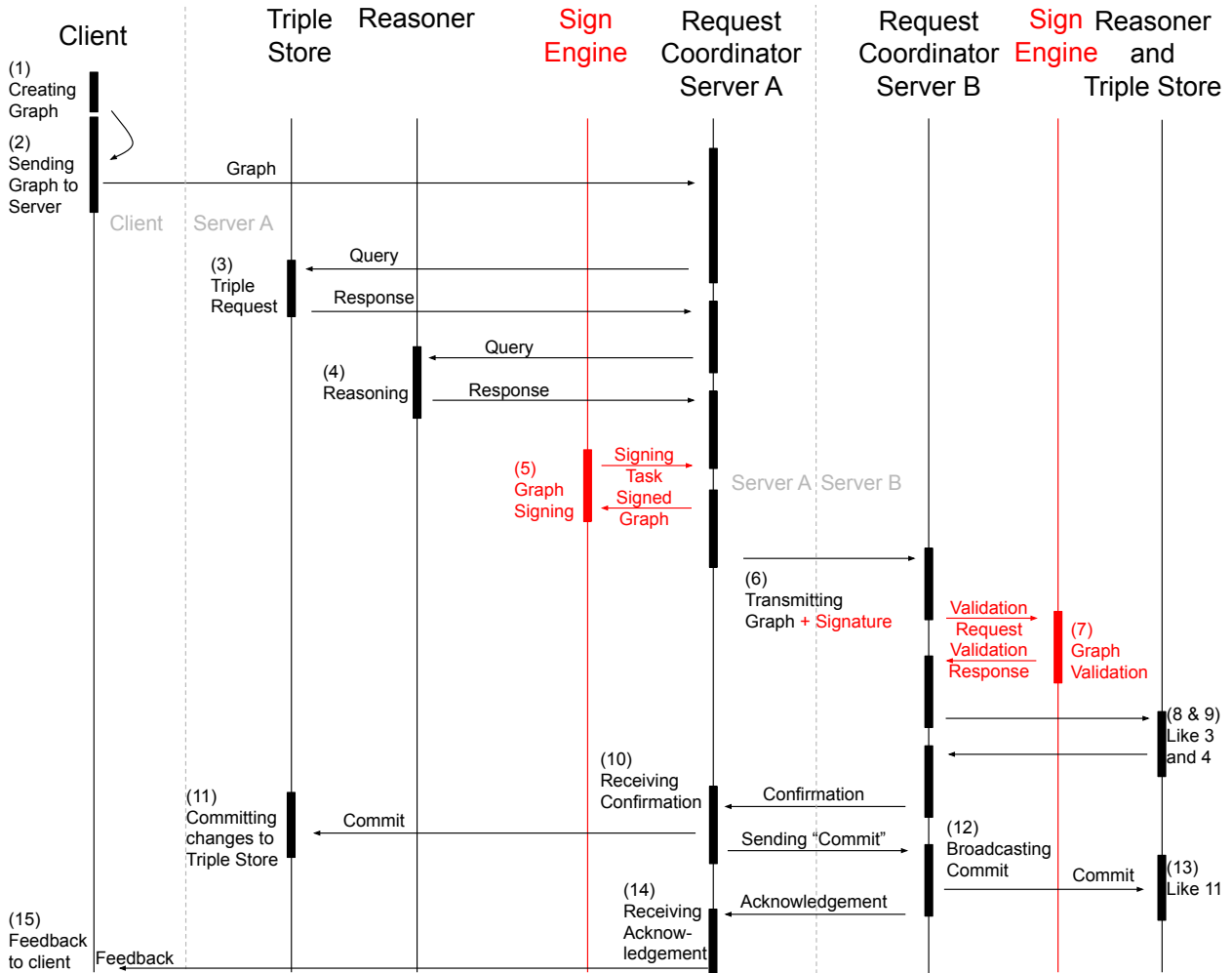


Figure 3.15: semDIM Message Model presented as a UML sequence diagram showing the client and server communication within and across company borders. The red parts are only required for Scenario Variant II.

ordered network. For example, tree 2PC could be added to semDIM if company subsidiaries or multiple servers per company are used. Finally, 3PC extends 2PC by an additional preparation phase, enhancing the robustness against possible system failures, thereby conferring higher availability. As stated in requirement R7, we do not require extensive availability features for DIM as the change frequency is considerably low. However, like Tree 2PC and Dynamic 2PC, 3PC could be used in semDIM and would not substantially change the communication flow of Figure 3.15. Only an additional preparation phase would be added. In summary, without loss

of generality, we use the simple 2PC protocol, which is sufficient for the scenario.

3.6.3 Pattern-based Concurrency Control in semDIM

By using 2PC as the commit mechanisms, we ensure synchronicity across all servers; that is, changes can be only either committed or rejected by all servers unanimously. However, in multi-user transactional systems, concurrency control must be considered [22] to ensure consistency while multiple agents try to propose potentially conflicting changes *simultaneously*. In general, concurrency control refers to the problem of how to schedule the sequences of basic operations, such as read and write actions, potentially overlapping in time. Concurrency control “regulates the access of multiple processes or transactions to shared data” [22]. For semDIM, this could be overlapping actions such as adding users or assigning a role while deleting such an entity in DIM. We first describe the solution implemented by semDIM, which is based on distributed two-phase locking and multiple granularity locking, before we discuss possible alternatives.

Proposed pattern-based concurrency control in semDIM: Based on established principles in database management [52], we propose for DIM a novel combination of distributed two-phase locking and multiple granularity locking adapted to ontology patterns. Two-phase locking (2PL) is the de-facto standard for concurrency control in multi-user databases [52]. We need distributed 2PL [52] in particular, as every company requires its own Lock Manager. Multiple granularity locking (MGL) [52] allows setting locks on groups of data objects based on hierarchies, but we need a grouping adjusted to DIM. We adapt the key feature of MGL, fine-grained locking, to ontology-pattern-based graphs to provide a mechanism tailored to pattern-based data (R9), such as the one used in semDIM.

The Lock Manager introduced in Section 3.6.1 sets or releases locks on request and keeps track of current locks, as in standard 2PL [52]. For this, we look into more detail in the sequence diagram from Section 3.6.2 and refine it with additional steps. We insert the new steps by appending an “a” to the previous step, that is, (5a) is a step between (5) and (6). Figure 3.16 illustrate the new steps. Note that only the new steps are depicted. Please see Figure 3.15 for the set of original steps. As 2PL features two phases, the following steps have been added. *Phase 1:* (5a) Server A sets locks on the affected patterns before sending the graph to Server B. (9a) Server B sets equivalent locks on Company B’s side. *Phase 2:* (11a) After receiving the confirmation from Server B and committing the changes on Company A’s

3.6. SEMDIM'S ARCHITECTURE AND SECURE IDENTITY INFORMATION EXCHANGE

side, Server A releases its locks. (13a) After receiving the “commit” from Server A and committing the changes, Server B also releases its locks. With that, we integrated distributed 2PL into semDIM.

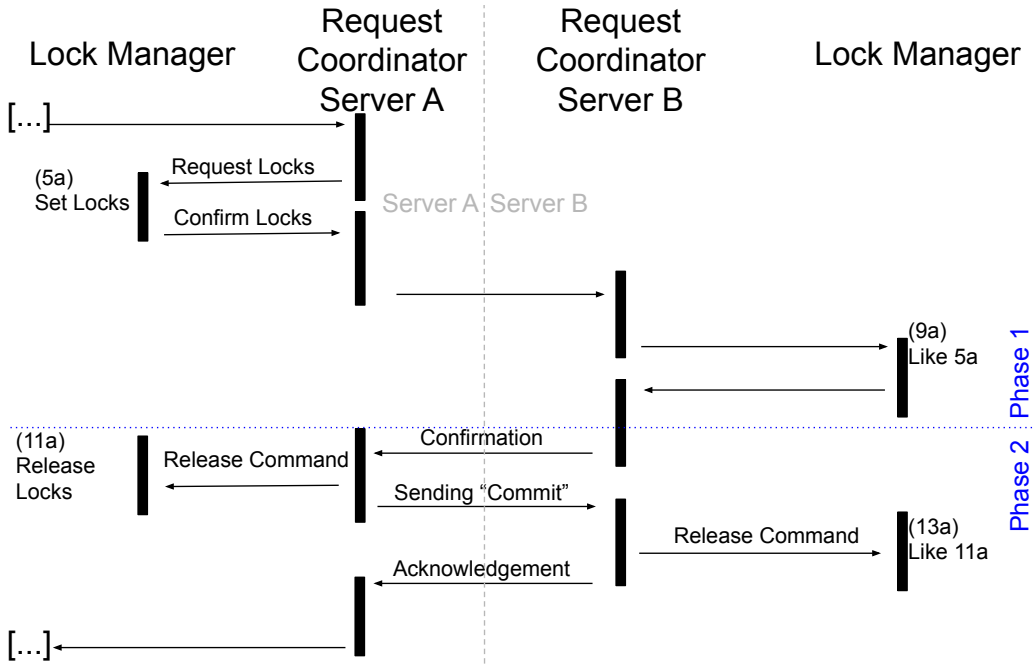


Figure 3.16: Additional steps in the semDIM Sequence Diagram to allow locking.

MGL requires a hierarchical ordering of the data objects, which is not available for all DIM entities. Therefore, we propose using the ontology patterns introduced in Section 3.5 to determine the locking granularity. We extend the semDIM message-passing model by introducing pattern-based concurrency control (PbCC). As discussed in the previous sections, granular locking is needed for concurrency control in DIM. PbCC can be summarized by the following postulates:

1. **Current state as one graph:** The current state of the system is considered to be one connected graph, which is the union of all distributively managed project identity information.
2. **Pattern-based graphs as atomic units for transactions:** Changes to the state of the system (transactions) are represented as pattern-based graphs. This means that changes consist of (usually) multiple

3.6. SEMDIM'S ARCHITECTURE AND SECURE IDENTITY INFORMATION EXCHANGE

triples conforming to an ontology-pattern, such as a role assignment conforming to the Permission Pattern.

3. **Nested locking granularity:** Locking granularity of a transaction is determined by the patterns used in the transmitted graphs. Depending on these patterns, additional patterns are relevant. These dependencies are determined by the policies the collaboration partners agreed upon, such as for a role assignment (Permission Pattern), the assigner must possess namespace ownership (Namespace Pattern). Based on such dependencies among the patterns, the respective graphs also must be considered regarding the original transaction's lock management.

Postulate 1 is the conceptual basis of PbCC. It determines that data are not considered and handled as single database entries, rows, or tables but as interconnected information pattern instances. The sum of all these pattern instances forms a global graph that represents the current state. PbCC does not necessarily require data to be represented as triples, but it inherently matches with PbCC by supporting relationships between any kind of resource. Postulate 1 and 2 restate the fundamental pattern-based approach for knowledge representation of semDIM as illustrated by the discussion in Section 3.5. Postulate 3 now adds the novel concept of fitting the locking granularity depending on both the patterns used in the transaction and related existing pattern instantiations to semDIM. Note that while PbCC is an essential component of semDIM, it is not specifically designed for DIM. PbCC could also be applied in other domains of distributed systems operating on ontology patterns. With that, we presented our PbCC approach. Next, we discuss alternative concurrency control mechanisms. In the Section 3.6.4 following this comparison of alternatives, we discuss the implications of the above postulates on semDIM in the context of data ownership.

Discussion of alternative concurrency control mechanisms: Concurrency control can be categorized into optimistic and pessimistic algorithms [52, 44, 87]. While optimistic concurrency control performs checking for data integrity at the end of an operation, the pessimistic approach blocks an operation until a violation of system integrity can be ruled out. Optimistic algorithms are based on the assumption that concurrent operations rarely lead to a conflict, while pessimistic algorithms anticipate frequent conflicts [87]. Figure 3.17 shows an overview of the most common concurrency control mechanisms. Thus, generally, if the probability of conflicts is low, optimistic algorithms have increased performance due to less procedural overhead.

3.6. SEMDIM'S ARCHITECTURE AND SECURE IDENTITY INFORMATION EXCHANGE

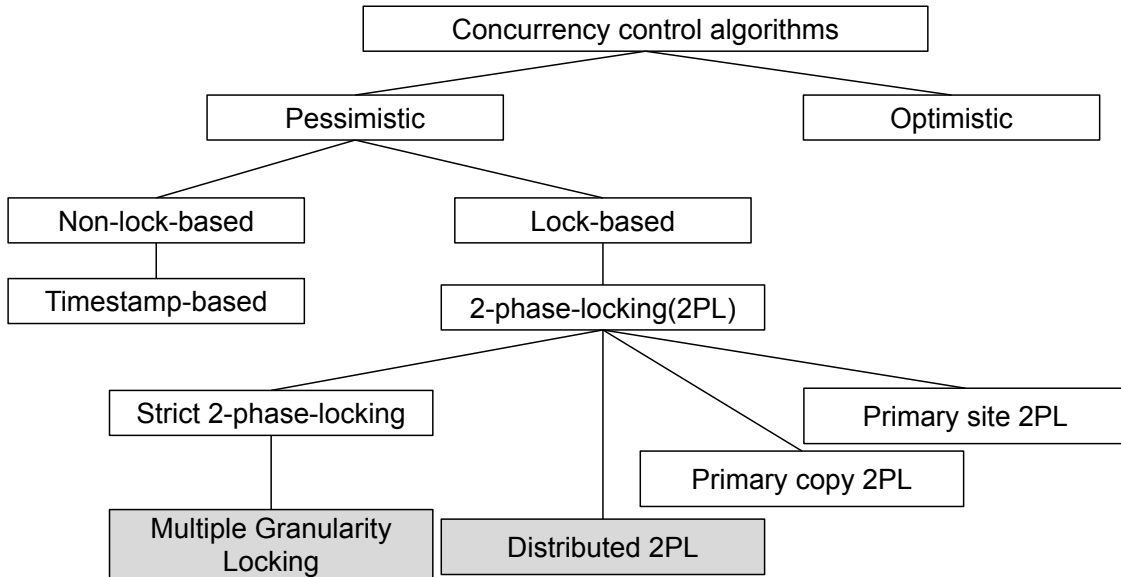


Figure 3.17: Taxonomy of concurrency control algorithms, adapted from [44] and [87]. The algorithms combined in our approach are highlighted in gray.

As stated in requirement R7, due to the slow change rate of DIM, that is, group memberships and their roles change rather rarely, it is acceptable to wait on a lock to avoid data inconsistencies or stale data. Thus, we prefer pessimistic concurrency control for semDIM. One finds non-locking and locking algorithms within pessimistic concurrency control. An example of non-locking concurrency control is timestamped-based transactions [52], where transactions are marked with their creation time and objects are marked with a last-read last-write timestamp. In a distributed setting (see R8), this could lead to complex rollbacks if conflicts arise, as multiple changes might have been propagated across multiple partner systems. Such situations are avoided with lock-based approaches, such as 2PL [52, 44, 87] and its variants.

The two phases of 2PL are the growth phase for acquiring locks and the shrink phase for releasing locks. In 2PL, transactions can acquire two types of locks: shared/read lock and exclusive/write lock. Shared locks are for reading and can be acquired from multiple transactions at the same time. Exclusive locks are for writing and can only be held by one transaction at a time. Classic 2PL is designed to address concurrency control on a single, multi-user database. However, 2PL is often too restrictive for distributed systems [44]. It requires operations to be blocked according to the locks

3.6. SEMDIM'S ARCHITECTURE AND SECURE IDENTITY INFORMATION EXCHANGE

acquired on multiple systems. In a distributed setting, this can lead to low system performance when systems wait for remote operations to finish and release locks needed for other operations. Thus, special variants of 2PL for distributed systems have been developed, such as Primary site 2PL (PS2PL) offers one central lock manager managing all connected systems [52]. All requests for acquiring and releasing locks must be directed to this central lock manager to avoid global deadlocks. The central lock manager as a potential bottleneck is an “obvious disadvantage” [52] of PS2PL. Not only is the central lock manager a single point of failure in both operations and security, but the performance of the system is also dependent on this single component. In a tightly coupled setting, partners could agree on a common, single lock manager. This central lock manager could be managed by a trusted third party. However, such an agreement is not always intended, and the single lock manager is a risk to attacks and failures and may result in the collaboration coming to a halt. Therefore, a different solution is required for DIM. Another 2PL mechanism addressing a distributed system is primary copy 2PL (PC2PL). PC2PL consists of multiple central lock managers [52], each responsible for a certain data subset, which lowers the failure probability. However, PC2PL requires consensus about which servers will host such central lock managers. Agreeing on a central lock manager is contrary to a loosely coupled, distributed setting (R8). In contrast, distributed 2PL introduces individual lock managers at each partner’s site, making the concurrency control truly distributed across organizations (which fulfills R8). However, distributed 2PL does not address the requirement of lock granularity (R9). Multiple granularity locking (MGL) addresses this requirement by allowing locks to be set on groups of data objects based on hierarchies. Such a hierarchical order may exist within an entity type—generally for groups and roles—but that order lacks connections between entities of different types.

For example, as discussed in Section 3.5.4, the assignment of a role involves not only the assignee (an **Agent**) and the assigned role (a **Role**) but also the assigner (another **Agent**, not hierarchically connected to the assignee) and the assigner role (another **Role**, not hierarchically connected to the assigned role) and the affected group and its type, in the case of a group permission. The connections between all these entities are relevant for role assignment and are captured by the corresponding ontology pattern. However, the ontology patterns cannot be represented by MGL. Thus, we developed pattern-based concurrency control (PbCC) (see the beginning of Section 3.6.3) as an adaptation of MGL. PbCC extends MGL by supporting pattern-based relationships and combines this adaptation with distributed lock managers from distributed 2PL. In summary, this novel combination of

3.6. SEMDIM’S ARCHITECTURE AND SECURE IDENTITY INFORMATION EXCHANGE

MGL and distributed 2PL addresses the requirements R7, R8, and R9 from Section 3.4.

We summarize the related work for concurrency control in DIM in Table 3.2. Further discussion on standard database terms and techniques used in this section can be found in [26], [22], and [52].

Table 3.2: Concurrency control approaches compared against requirements.

Approach	R7: Consistency	R8: Distributed	R9: Granular
Optimistic	no	n/a	n/a
Timestamp	yes	no	no
2PL	yes	no	no
Strict 2PL	yes	no	no
Dist. 2PL	yes	yes	no
MGL	yes	no	partial
<i>PbCC (Ours)</i>	yes	yes	yes

3.6.4 Data Ownership in semDIM

We have discussed how the DIM information represented by the semDIM ontology patterns (see Section 3.5) can be exchanged across companies using the semDIM architecture and messaging model in Sections 3.6.1–3.6.3.

In this section, we discuss data ownership in the context of semDIM. Data ownership is crucial for verifying the exchanged pattern instantiations, that is, the graphs. In a setting such as the scenario in Section 3.3, any partner may read all of the project-relevant IM information, including the information owned by another partner, for example, identifiers for employees involved in the project. Furthermore, any partner may write or change IM information, including information *originally owned* by another partner, if the appropriate permissions are assigned. For example, in our scenario, Person 1 is an employee of Company A but may add employees of Company B to the joint project group (see Section 3.3.2).

Replication of IM information in semDIM: To ensure data synchronization across all nodes, we introduced semDIM’s messaging passing process model in Section 3.6.2, including 2PC. Now, we need to determine which data is replicated on which node, that is, how the IM information is fragmented.

The simplest approach for distribution is full project-related replication; that is, all servers hold all project-related identity information. The IM information is still fragmented within the companies because they most likely

3.6. SEMDIM'S ARCHITECTURE AND SECURE IDENTITY INFORMATION EXCHANGE

hold project-related and non-project-related (company-internal or related only to different projects) information. Data fragmentation is thus avoided within the scope of one project. Full replication implies redundancy and higher data volumes for each server. However, we rely on it in our example semDIM implementation, as the volume of DIM data is very low compared to other data types (see Section 3.3.2). In addition, the change frequency of the DIM data is low, such that waiting for all nodes to commit a change, that is, the 2PC protocol, with a full replication can be justified here.

Ownership-based fragmentation as an alternative to project-related replication: An alternative to full project-related replication is ownership-based fragmentation; that is, each company holds only the data it owns. Fragmentation would reduce stored data and could potentially simplify synchronization processes if a successful commit of a change only depends on the decision of the nodes holding related data. However, fragmentation has a major drawback in the context of DIM: Nodes cannot reliably track changes to data they do not hold if the node(s) holding them execute changes without notifying all other nodes. Furthermore, if only nodes holding related data are involved in the commit decision, a mechanism must be implemented to determine all affected nodes before the commit. Fragmentation appears to only introduce challenges without offering significant benefits, as the frequency and size of exchanged data in DIM are low. Therefore, semDIM uses a project-related full-replication of IM information by default.

Determining data ownership and the legitimacy of changes in semDIM: Although all companies hold a full replication of the project-related DIM information, there are differences regarding the data ownership, such as who holds the authority over and has the highest privileges regarding creating, changing, and deleting DIM entities, for example, the identities of employees of one company. Data ownership is an important indicator to determine if a proposed change is legitimate, that is, if the proposing company has the right to change the data of a particular entity. In the simple case, the proposer owns all of the affected entity data. For example, Company A changes the assigned department of one of its employees. A more complex case arises if the proposer does not hold ownership of all data but received special permission to change data in a foreign namespace under certain conditions. An example of the latter case is the project leader from Company A adding an employee of Company B to the joint project group.

3.6. SEMDIM'S ARCHITECTURE AND SECURE IDENTITY INFORMATION EXCHANGE

Basic data ownership policies in semDIM: In semDIM, the ownership relationships generally are intuitive; for example, the identity of an employee of Company A defined in the namespace of Company A is owned by the namespace owner of Company A `a:root-a`. However, for some entities, ownership is not that easily determinable. For example, to whom does a permission assignment belong if using entities of multiple namespaces? We formalized this approach in our discussion addressing namespaces (see the specific scenario setting in Section 3.3.2 and the Namespace Pattern in Section 3.5.1 and the Permissions Patterns in Section 3.5.4). We summarize the above discussion on data ownership in semDIM with the following simple, standard policies regarding ownership: (1) The owner of a namespace also owns all entities defined in this namespace (see Section 3.5.1). (2) The owner of a namespace may delegate ownership of this namespace or parts of it (see Section 3.5.1). (3) The owner of a namespace may delegate permissions on entities in the form of roles to other agents (see Section 3.5.4). (4) Agents with the appropriate roles may further assign roles to agents and, by that, further delegate permissions (see Section 3.5.4).

Special case—ownership of Permission Pattern instantiations: So far, we determined the ownership of DIM entities but not yet of Permission Pattern instantiations, which specify who in the distributed project is allowed to define which change. The ontology patterns of semDIM are based on the popular Descriptions-and-Situations (DnS) approach of DUL (see Section 3.5). As DnS-based pattern instantiations are themselves entities—the **Descriptions** and **Situations**—the general DIM entity policies from above can also be applied to permission assignments.

However, there is also a need for regulation about how to handle pattern instantiations in which entities from multiple namespaces appear, for example, an agent from Company A's namespace assigning a role to an agent from Company B's namespace. The simplest solution is the following policies: (5) A `PermissionAssignmentExecution` `exec-x` must have the same namespace as the `AssignerRole` defined in `PermissionAssignmentWorkflow`, which is satisfied by `exec-x`. (6.) The holder of an `AssignerRole` `role-x` is automatically allowed to propose `PermissionAssignmentExecutions` satisfying the `PermissionAssignmentWorkflow` defining `role-x` in the namespace of `role-x`. (7.) Only holders of an `AssignerRole` `role-x` may propose `PermissionAssignmentExecutions` satisfying the `PermissionAssignmentWorkflow` defining `role-x` in the namespace of `role-x`.

3.6. SEMDIM'S ARCHITECTURE AND SECURE IDENTITY INFORMATION EXCHANGE

These policies determine that a permission assignment is owned by the namespace holder of the assigner role, even if the assigner is from a different namespace (see the scenario in Section 3.3 for an explanation of the role of company namespaces). These policies supports requirements R1-4 because they allow distributed management of identities, roles, and groups across namespaces by using `owl:sameAs` properties to connect entities across namespaces. For example, Person 1 in our scenario from Section 3.3 can add employees from both companies' namespaces to the joint project group.

Alternative policies to adjust namespace limitations: There are plausible use cases for allowing instantiations of additional patterns with entities of different namespaces, for example, if companies do not want to rely on `owl:sameAs` for certain entities. Such a situation may arise when existing domain ontologies already provide roles and groups that should directly be reused without using `owl:sameAs`. Administrative overhead is reduced by avoiding managing multiple synonymous identifiers in each of the companies' namespaces. However, this option could result in security problems if not handled with care. Not limiting the valid namespaces for entities such as roles removes a mechanism for detecting misconfiguration and preventing unintended effects.

Such scenarios that used less limitation of allowed namespaces are different from the one presented in Section 3.3, in which rather strict regulations are agreed on. Nonetheless, semDIM is adaptable to such scenarios by replacing the above policies 5 and 6 with policies tailored to the patterns in which different namespaces are allowed. For example, one such replacement policy might be: for a `PermissionAssignmentExecution`, the namespaces of the `PermissionAssignmentExecution` itself, the related `PermissionAssignmentExecution`, the `AssignerRole`, the `Assigner`, and the `AssignedRole` all must be the same, while all other entities may have the same or different namespaces.

Such a policy would ensure that role assignments could only be executed by assigners from the role's namespace, but the assignee receiving that role could actually be from a different namespace. Furthermore, this policy would determine that the ownership of a permission assignment depends on the namespace of the `Assigner` and the `Role`, not the `Assignee`. Also, policies for shared ownership are possible.

Proposing changes to pattern instantiations owned by the other company: The company that owns the server that proposes a change (and signs the graph in Variant II) is not necessarily the data owner. For example,

3.6. SEMDIM'S ARCHITECTURE AND SECURE IDENTITY INFORMATION EXCHANGE

as discussed above, the `Assigner` of a `PermissionAssignmentExecution` may trigger its propagation on their company's server, which then signs the graph. However, the actual owner of the `PermissionAssignmentExecution` may be a different company if the `AssignerRole` is from a different namespace. Supporting such ownership and authorship constellations is intentional and an essential feature of semDIM.

For example, in the scenario in Section 3.2, Person 1 is an employee of Company A. When adding an employee of Company B to the joint project group, this is to be proposed (and signed in case of Variant II) by Company A. However, the assignment is then owned by Company B because it affects one of its employees. Because Company B explicitly allowed Person 1 to execute such assignments, that is, Company B delegated some of its namespace rights to Person 1 (see Section 3.3.2), the above procedure is valid. However, if certain agreements between companies require different regulations, the procedure should be adjusted by altering the policies above.

Collaboration termination: Multiple procedures for ending collaboration across namespaces are offered by semDIM. The simplest procedure is to terminate the communication between the information systems between the companies, that is, blocking previously allowed ports and IPs on the infrastructure level. This procedure is the easiest but might not be practical when not all communication between partners should be terminated. For example, there could be other collaboration projects between the companies that should continue, or the companies may consider collaborating again in the near future.

Therefore, the next approach is to archive or delete permission assignments and `owl:samAs` connections benefiting all entities (or only selected entities, such as project group leaders) in the namespace of the previous partner. Thereby, all (or only selected) identities of the previous partner lose all permissions, and connections to privileged identities within the company's namespace are cut. This procedure is also straight-forward but needs alteration to the triple stores. If both continuing system connections between the partners and minimizing triple changes are required, semDIM offers a third termination procedure.

With graph-signing (Variant II), both companies can stop accepting graphs signed by the previous partner. Thereby, all triples (except the ones legitimizing the partners' certificates) may stay the same, and no adjustments to permissions are necessary. Furthermore, the connection between the partners may stay intact. This is a major benefit of Variant II because allowing or disallowing certain certificates for changes to specific namespaces

can be used as “switches.” While a certificate is accepted, the connecting switch is on, and revoking this permission turns it off. In the scenario from Section 3.3, there is only full collaboration between the companies during the project or absolute disconnection after the project ended. In real-life settings, states between these extremes exist, where the main collaboration ended, but certain partial operations continue. By offering a fine-grained adjustment of which certificates are allowed to perform changes in which namespaces, semDIM also supports such hybrid-states.

3.6.5 Summary

We introduced a client-server architecture of our semDIM framework for Semantic Distributed Identity Management in Section 3.6.1. In Section 3.6.2, we introduced the message flow between the different components, which we refined with our PbCC in Section 3.6.3. Section 3.6.4 discussed data ownership and how it affects graph validation in semDIM. Below, we demonstrate how the semDIM framework is applied in practice by demonstrating its use in our scenario.

3.7 Application of semDIM to the Scenario

We evaluate our solution against the requirements stated in Section 3.4. For this, we demonstrate the application of a prototype implementation of semDIM in the context of the sequence diagram from Section 3.6.2 adapted to the scenario from Section 3.3. The two Sign Engines were realized by reusing the signing libraries from Kasten et al. [42]. For the prototypical implementation of the architecture, we use RDF4J⁵ for SPARQL-Endpoint and triple store management. The following nomenclature highlights which type of entity or property a name refers to. *Agents* are named by a single letter or a short description followed by a hyphen and a number, for example, `a-1`, `b-1`, `root-a-1`. *Groups* are named similar to agents but have `group` or `_g` within their name, for example, `group-1`, `group-2`, `admin-group-1`, `admin-group-2`, `project_g-1`. *Roles* are identified by having `_role` or `_r` in their name, for example, `admin.role-1`. These conventions only serve readability and are not normative. The following triples are described primarily from Company A’s perspective. As the following statements are representative of actions at different points in time of the project, we introduce the following four project phases: *Initiation*, *Setup*, *Operations*, and *Closure*. These four phases are inspired by traditional

⁵<https://rdf4j.org/>, last accessed on 2021-01-16

3.7. APPLICATION OF SEMDIM TO THE SCENARIO

project management [92]. The exact number and names of project phases vary in the literature; for example, Wysocki [92] divides the Operations phase further into Launch and Execution.

For the sake of the following discussion, it is sufficient to consider the above introduced technical phases to be only loosely coupled to project phases. For example, the technical phase *Initiation* typically also starts at the very beginning of a project. DIM can exist outside of a project context. However, we refer to these project-related technical phases to demonstrate that semDIM is applicable in a real-life project as described in our scenario in Section 3.3. It is important to note that these four technical phases are not the same as the four product lifecycle phases used in PLM [71]. The product lifecycle can be iterated multiple times within the *Operations* and possibly even after the *Closure* of the initial project, such as when the partnership between Company A and B is later terminated, and the product is continued without DIM. One iteration of a product lifecycle can be connected to one project, but the life of a product may or may not extend over one or several projects. We discussed how semantic information changes across product lifecycle phases can be modeled and handled in Chapter 2. The following summarizes the technical phases:

Initiation: The companies prepare for collaboration in the joint project. They define the top namespaces and their ownership that they want to exchange with one another to relate entities across company borders. After this phase, the companies can start to define relations across namespaces. In *Setup*, the companies begin to establish these relations between the entities across namespaces, for example, by connecting the identifiers of the joint project group from each namespace. The distributed identities, groups, and roles required to start the actual project are created and exchanged. Up to this phase, no concurrency control is needed, as only individually triggered information exchanges happen. Concurrency control becomes necessary when multiple users start to simultaneously execute transactions, which happens in the next phase. *Operations* is the phase wherein the actual project is executed. Now, DIM is used to create and change identity information, such as adding members to groups or changing agents' roles. In the final *Closure* phase, DIM between the partners gets separated again to prevent any further cross-company collaboration based on distributed identities, roles, or groups. These technical phases, like traditional project phases, may overlap in real-life settings. The following paragraphs present the execution of the DIM steps (from our scenario in Section 3.3) structured along the requirements from Section 3.4.

3.7.1 Example for R1: Namespaces and Unique Identifiers

The two companies, A and B, have unique namespaces, for example, their publicly registered domain names. Therefore, they own their respective namespace, giving them full read and write access. Every set of triples using identifiers in these namespaces either explicitly states the full identifier to the respective namespace or reuses the prefixes of the Listings 3.1 and 3.2, line 1. The companies coin identifiers and subnamespaces, that is, creating a new namespace by extending an existing namespace by a slash followed by a string, for their namespace. For example, the URI `https://companyA.example/HR#a-1` defines the employee `a-1`, while the URI `https://companyA.example/HR` models the HR department. In a real-life setting, such subnamespaces could be used to define department-specific namespaces for which fine-grained namespace ownerships could be defined, such as the head of HR owning the HR namespace. The simplified scenario in Section 3.3 does not include any company department structures except the project team. Therefore, we only use the top namespace for each company, that is, the namespace `a:` resolving to `https://companyA.example/` and `b:` resolving to `https://companyB.example/`. At the Project Initiation phase, both companies create their root namespaces (if not yet existent) and define the initial ownerships (see Listings 3.1 and 3.2). Subsequently, they exchange these graphs with each other. The very first triples exchanged by the companies at Project Initiation are the root identities, `root-a-1` and `root-b-1`. These triples instantiate the Namespace Pattern in Figure 3.5.

```

1 @prefix a: <https://companyA.example/> .
2 a:root-a-1 a foaf:Person ;
3   semDIM:ownsNamespace a: .

```

Listing 3.1: Namespace definitions defined by `a:root-a-1` for Company A.

```

1 @prefix b: <https://companyB.example/> .
2 b:root-b-1 a foaf:Person ;
3   semDIM:ownsNamespace b: .

```

Listing 3.2: Namespace definitions defined by `b:root-b-1` for Company B.

After this exchange, each company holds definitions for its own and the other company's namespaces and root identities. At this point, each company is only allowed to propose changes to its own namespace because

3.7. APPLICATION OF SEMDIM TO THE SCENARIO

no delegations of rights across namespaces have been declared yet. These root identities are used to define initial identities and groups. Listing 3.3 depicts those first identities and groups for Company A defined by `root-a-1`. As they use the namespace `a:`, they are uniquely identifiable even outside of Company A. Similar triples also exist on B's side using the namespace `b:` (listing not depicted). As shown in the semDIM architecture (see Figure 3.14), the triples exchanged by the companies are verified through the Reasoning Engine and incorporated in the triple stores. These triples are also exchanged during the Initiation phase.

```
1 a:a-1 a foaf:Person .
2 a:a-2 a foaf:Person .
3
4 a:group-1 a foaf:Group .
5 a:group-2 a foaf:Group .
6
7 a:groupAdmins_g-a-1 a foaf:Group ;
8   foaf:member a:a-1 .
9 a:groupAdmins_g-a-2 a foaf:Group .
10
11 a:group-1 semDIM:administratedBy a:groupAdmins_g-a-1 .
12 a:group-2 semDIM:administratedBy a:groupAdmins_g-a-2 .
```

Listing 3.3: First identities and groups defined by `a:root-a-1`.

At the end of the Initiation phase, both companies' servers hold the initial namespaces and entities relevant to the project, but there are no connections across companies yet. These are created in the Setup phase, which is described below.

3.7.2 Example for R2: Distributed Identifiers for Agents

The identities can be referenced across namespaces, for example, by connecting two separate identities of the same person in both companies A and B. This is done by using the `owl:sameAs` property, as shown in Listing 3.4. In our scenario, Person 1 belongs only to Company A but needs a separate account and, therefore, an additional identifier in Company B. This triple is exchanged during the Setup phase.

```
1 a:a-1 owl:sameAs b:b-1 .
```

Listing 3.4: Identity connection defined by `a:root-1`. This is an instantiation of the Same Entity Pattern illustrated in Figure 3.6.

3.7.3 Example for R3: Distributed Identifiers for Groups

In Listing 3.3, line 8 we show a group membership definition. Lines 11-12 define the administration of `group-1` and `group-2`. All of these triples are created by `a:root-a-1` and are also transmitted to B’s server in the Setup phase. Groups can also be defined across namespaces, that is, as distributed groups, as Listing 3.5 demonstrates.

```
1 a:group-1 owl:sameAs b:group-1 .
```

Listing 3.5: Group connection defined by `a:root-a-1`. This is an instantiation of the Same Entity Pattern illustrated in Figure 3.6.

3.7.4 Example for R4: Distributed Identifiers for Roles

Just like groups, roles are also defined across namespaces. The two companies agree on a common understanding about which roles are equivalent across namespaces. Listing 3.6 shows the connections between roles from both namespaces. We shortened this listing with `[...]`, leaving out some of these “`a:role-x owl:sameAs b:role-x`” statements. Which company declares these statements is irrelevant as long as both companies accept them.

For the sake of this example, `a:root-a-1` on Company A’s system defined them. After exchanging these triples, the Setup phase is completed. The actions in the Initiation and Setup phases are a fundamental basis for the core actions of semDIM in the Operations phase.

```
1 a:group_admin-1 owl:sameAs b:group_admin-1
2 [...]
3 a:group_member-1 owl:sameAs b:group_admin-1
```

Listing 3.6: Role connections defined by `a:root-a-1` declaring `a:a-1` to be the group administrator of `group-1`. These are instantiations of the Same Entity Pattern illustrated in Figure 3.6.

3.7. APPLICATION OF SEMDIM TO THE SCENARIO

In the Operations phase, identity information frequently changes; for example, new accounts are created and added to groups or roles are assigned. Also, project data not related to IM is exchanged now, such as *Blue Train* product information in our scenario. Listing 3.7 shows the RDF data equivalent to a group admin permission assignment. Here, the Group Permission Pattern from Figure 3.9 and the example from Figure 3.10 are used. Therefore, this Listing shows Figure 3.10 in triple form. In this example, the administrator role for group `a:group-1` is assigned to `a:a-1` by `root-a-1`. With this role, `a:a-1` can add or remove members to the project group `group-1`, which is one of `a:a-1`'s essential tasks as project leader.

```
1 a:admin_perm-1 a semDIM:GroupPermission ;
2   dul:defines a:namespace_owner-1 ,
3   a:group_admin-1 ,
4   a:group_administration-1 ,
5   a:administrated_group_role-1 .
6
7 a:namespace_owner-1 a semDIM:AssignerRole ;
8   dul:isRoleOf a:root-a-1 .
9
10 a:group_admin-1 a semDIM:AssignedRole ;
11   dul:isRoleOf a:a-1 .
12
13 a:group_administration-1 a semDIM:AssignedTask ;
14   dul:isExecutedIn a:admin_action-1 .
15
16 a:administrated_group_role-1 a semDIM:AffectedGroupRole ;
17   dul:isRoleOf a:group-1 .
18
19 a:admin_assign-1 a semDIM:GroupPermissionExecution ;
20   dul:isSettingFor a:root-a-1 ,
21   a:a-1 ,
22   a:admin_action-1 ,
23   a:group-1 .
```

Listing 3.7: Group Permission Pattern instantiation showing Role connections defined by `a:root-a-1` declaring `a:a-1` to group administrator of `group-1`.

Listing 3.8 shows how `a:a-1` uses the group administrator Role to assign the group member Role to the project member `b:b-2`. In this context,

3.7. APPLICATION OF SEMDIM TO THE SCENARIO

Person 1—the owner of the identity `a:a-1`—uses the other identity, `b:b-2`. Person 1 is allowed to do so, as both companies accepted the `sameAs` relationship between these two identities (see Listing 3.4). Person 1 is required to use this other identity because only identities from the same namespace as the assignee may assign roles (see Section 3.3 and the policy from Section 3.6.4).

```
1 b:admin_perm-2 a semDIM:GroupPermission ;
2   dul:defines b:group_admin-1 ,
3   b:group_member-1 ,
4   b:group_task-1 ,
5   b:administrated_group_role-1 .
6
7 b:group_admin-1 a semDIM:AssignerRole ;
8   dul:isRoleOf b:b-1 .
9
10 b:group_member-1 a semDIM:AssignedRole ;
11   dul:isRoleOf b:b-2 .
12
13 b:group_administration-1 a semDIM:AssignedTask ;
14   dul:isExecutedIn b:group_action-1 .
15
16 b:administrated_group_role-1 a semDIM:AffectedGroupRole ;
17   dul:isRoleOf b:group-1 .
18
19 b:admin_assign-2 a semDIM:GroupPermissionExecution ;
20   dul:isSettingFor b:b-1 ,
21   b:b-2 ,
22   b:group_action-1 ,
23   b:group-1 .
```

Listing 3.8: Role assignment defined by `b:b-1`, which is Person 1 acting in Company B’s namespace, declaring `b:b-2` to be a group member of `group-1`.

Finally, in the Closure phase, the project ends, and no further communication between the systems of Company A and Company B is expected. Now, each company deletes or archives the triples containing the namespace of the other company to revoke all entity assignments and connections. With that, we discussed `semDIM`’s application across the *Blue Train* project’s phases in the context of all functional requirements from Section 3.4. In the following, we revisit some aspects of our example to address the non-functional requirements.

3.7.5 Example for R5–R6: Signed Triples Using Graph Signatures

Support for graph signing is needed in Variant II of the scenario in Section 3.3. User-defined triples need to be signed by the server distributing them to the other server(s). Therefore, the integrity and authenticity of the signed graphs must be ensured (R6).

We use the signing framework from Kasten et al. [42] to sign graphs. The signed graph with the triples mentioned before is depicted in Listing 3.9 using the patterns from Figures 3.11, 3.13, and 3.12. The graph is signed on Server A (step 5 in the sequence diagram of Figure 3.15), as the initiator of this action. It is then sent to other servers (step 6), which validate the graph's signature (step 7) and then vote on the commit (step 10).

```

1  _:sigGraph {
2    _:gsm-1 signature:hasDigestMethod signature:dm-sha-256.
3    ...
4    _:gsm-1 a signature:GraphSigningMethod .
5    _:sig-1 a signature:Signature ;
6      signature:hasGraphSigningMethod _:gsm-1 ;
7      signature:hasSignatureValue "AbCd1..." ;
8      signature:hasVerificationCertificate "cert" .
9    a:a-1 owl:sameAs b:b-1 .
10   a:group-1 dul:hasMember a:a-2 ; owl:sameAs b:group-1 .
11   a:groupAdmins_g-a-1 foaf:member a:a-1 , a:a-2 .
12   a:groupAdmins_g-a-1 owl:sameAs b:groupAdmins_g-b-1 .
13 }
```

Listing 3.9: Example of a signed graph applied on user data defined by a:a-1.

3.7.6 Example for R7–R9: Concurrency Control

We introduced PbCC in Section 3.6.3. To illustrate the application of PbCC, we reuse the example from Section 3.5.4 depicted in Figure 3.10 and used in Listing 3.7. In this example, the agent `root-a-1` assigned the role `group_admin-1` for the project group `group-1` to the agent `a-1`. This role assignment is an instantiation of the Group Permission Pattern (see Figure 3.9 in Section 3.5.4). Therefore, this transaction is represented as a pattern instantiation, which conforms with Postulate 2 (Patterns as atomic transaction units). When this transaction is now propagated across systems, each reasoner must check for the prerequisites of this transaction. These steps are described in the message passing model in Section 3.6.2. In our

example, the standard policy for assigning group admin privileges of a group is “the assigner must be namespace owner of the group.” This means that, in the current state of the DIM system (the system graph), a pattern must exist that declares `root-a-1` as namespace owner of `group-1`. This is an instantiation of the Namespace Pattern (see Figure 3.5 in Section 3.5.1).

Postulate 3 (nested locking granularity) declares that this namespace pattern is also to be considered in concurrency control when the above role assignment is considered. This means that all triples of this pattern must be locked during the processing of the role assignment. We demonstrated the specific use of semDIM in the context of our scenario from Section 3.3. Furthermore, we checked the consistency of the statements in the listings above with the patterns by applying the Hermit reasoner [77].

3.7.7 Summary

In this section, we have presented in detail the knowledge representation for semDIM. We have implemented all of semDIM’s ontology patterns from Section 3.5 in the context of our scenario in Section 3.3. In the next section, we evaluate the presented semDIM approach regarding the security requirements (see R6 in Section 3.4).

3.8 Security Analysis of semDIM

We assume an underlying technical IT infrastructure managed in compliance with modern security standards, as described by [34, 59] or similar standards. This assumption means we will not discuss threats addressing, for example, unpatched operating systems, faulty configured communication channel encryption, or clients pre-infected by remote-access-kits, or similar attack scenarios. These are generic to any kind of IT system and are addressed by patch, configuration, and vulnerability management, business continuity management, or endpoint protection. semDIM provides defense mechanisms even to such fundamental technical threats by offering a second line of defense with graph signatures that potentially compensate for faulty transport encryption. However, we concentrate on the following threats specific to distributed settings. Below, we discuss the specific security aspects of semDIM along the STRIDE model [57].

3.8.1 Spoofing

Spoofing describes the impersonation of an attacker as another person or program by falsifying data. In our context, spoofing is prevented with safe transport encryption and robust authentication of the sender and receiver. Both client-server and server-server communication is thereby protected. Furthermore, the servers exchange certificates before operations on a secure channel (mutual or two-way TLS⁶). The validity of the certificates is checked by inspecting the hashes of the exchanged certificates by phone, personally, or over another, already-established encrypted channel. This rules out the possibility that certificates were manipulated during transmission.

Because the companies have representatives who know each other personally, an attacker could not inject an illegitimate certificate, even by bribing or compromising a Certificate Authority to issue a certificate in one of the companies' names. Such a forged certificate—although technically appearing completely valid—would be regarded as illegitimate by the companies, as it is not the one they explicitly exchanged during their mutual TLS certificate exchange. Therefore, none of the companies relies on a third party for validating the other partner's certificate. That is, they do not have to trust the Certificate Authority to determine that the partner's certificate is not compromised, malicious, or acquired by bribing. By that, a Man-in-the-Middle (MitM) attack, which is a malicious agent intercepting the communication between the partners faking and acting on behalf of both partners' identities, is prevented. To make a MitM attack possible, the attacker would need to exchange the pre-exchanged certificates inside the secure certificate store on the companies' servers. This exchange would require the equivalent of full-access infection of both servers. This scenario is no longer a MitM but instead is a total-compromise security breach.

Graph signing also limits the potential damages of spoofing. Even if an attacker could impersonate a client or server, the attacker could only transmit signed (valid) messages from the original sender. Replay attacks are prevented by timestamps within the signed graphs. In this thesis, we demonstrated server-side signatures for the sake of simplicity, but semDIM also supports client-side signatures. The connection between client and server also can be further protected against spoofing by establishing client-side graph signatures and combining these with mutual TLS between client and server. A certificate used for this signing could also be used for client authentication at the server. Ideally, the certificate of the respective server is also pre-installed on the client to prevent an attacker from pretending to be the server towards the client. Mutual TLS between client and

⁶<https://tools.ietf.org/html/rfc8446>, last accessed on 2021-01-16

server is especially beneficial if the client component is moved to the server, as suggested as a possible modification of the architecture from Chapter 3.14. Using mutual TLS prevents a client from assuming a client component offered by a compromised server to be legitimate and potentially compromising confidential information or even enabling the attacker to forge signatures for illegitimate graphs in the client's name. Using certificates for client-side graph signatures and additionally using the same certificates for (mutual) TLS could strengthen or even replace the existing organization's authentication method for DIM. However, there is no need for a separate DIM-specific authentication if reliable authentication is established in the organization. We do not consider spoofing between components on the server as a plausible MitM attack scenario. For this, an attacker would need code execution privilege on the server and to fake one of the components on the server. As a code execution privilege usually can be escalated by a skilled attacker to full system access, this is not a MitM but a total compromise. Furthermore, it is not a reasonable scenario for the server to impersonate a client, as the clients only communicate with their respective servers. Lastly, a server cannot spoof another server for being a third server (for the same reason an external attacker cannot do this).

3.8.2 Tampering

Tampering is the intentional manipulation of systems or messages. The protection mechanisms mentioned for spoofing also apply to external attackers. Because all messages are encrypted, an attacker could neither read nor manipulate the messages. Signing the graphs further protects against tampering because the signatures would no longer be valid. Besides network communication, tampering is, in our context, also relevant regarding false claims about, for example, group membership, assigned privileges, or data classification.

The only resting data in our architecture are found on the triple store, which is not directly exposed to the network. Therefore, an attacker must either compromise the server operating system or the Request Coordinator. Similarly, for manipulating messages to or from the Reasoning or the Sign Engine, the attacker must be able to compromise either the server as a whole or the Request Coordinator. As mentioned before, we do not discuss the weaknesses of the operating system (OS) in detail here. The request coordinator, as the only server component exposed to the network, must be hardened for unauthorized manipulation. Ideally, web application firewalls restrict any communication to and from the Request Coordinator to only allow the exchange of graph data and the necessary commit messages. It

is noteworthy that even in the case of a successful tampering attack on resting or transmitted data, the potential damage is greatly reduced by the distributed nature of semDIM. This means that in case of tampering, a single node's messages or persistent data sets become inconsistent with the global state. Such an inconsistency will likely lead to an increase in aborted transactions because other nodes deny transactions that are not compatible with their intact state. Even if only two nodes exist, for example, as in our scenario setting from Section 3.3, an inconsistency would lead to aborted transactions alarming administrators to investigate which of the two nodes is in an inconsistent, compromised state. Tampering would only be non-detectable in the event that all nodes were successfully manipulated.

3.8.3 Repudiation

Repudiation is the situation where an author can deny authorship of a statement or a message. In our context, repudiation is relevant as it is important to prove that, for example, role assignments were really done by the referred author and not somebody else when such an assignment violates, for example, compliance or legal regulations. As mentioned before, we only discussed server signatures in the above scenario, thereby preventing companies from repudiating messages. Individual user repudiation could also be prevented by using client signatures.

3.8.4 Information disclosure

Information disclosure happens if confidential information, most often personal information or intellectual property, can be read by non-legitimate agents. Again, external attackers can neither read nor manipulate data encrypted by the mutual TLS connection. We address this threat by having our Reasoning Engine ensure that classified information, such as confidential product information in our scenario, can only be accessed by legitimate agents. As any server may read the whole project-related IM data set, servers can only expose information unintentionally by accidentally or maliciously transmitting non-project-related IM. This might be information related to other projects the partner is not involved in or purely internal IM data. To prevent this, for example, in the case of misconfiguration or human error, separation mechanisms may help, such as having separate triple stores per project.

3.8.5 Denial of Service

Denial of Service (DoS) describes an attack pattern in which the functionality of systems or components, such as network resources or servers, is disrupted in such a way that it is unavailable or unresponsive for significant amounts of time. An often-used type of such attacks is the distributed denial of service (DDoS) attack, which uses multiple attack sources to flood systems with overwhelming amounts of traffic. DDoS attacks are of lesser relevance in our context, as we do not rely on publicly reachable systems, in contrast to *dg*FOAF, which requires publicly available FOAF files on web servers. However, a malicious actor might like to disrupt the availability of the components required for our solution. A plausible attack method would be a massive amount of validation requests, such as for group membership or legitimate privilege assignment. We address this issue by only allowing authenticated agents to query such requests.

A more generic DoS attack approach is SYN flooding, that is, trying to disturb the operation of a server by opening huge amounts of connections by sending a SYN package without responding with ACK to complete the TCP three-way handshake. This attack is not specific to semDIM and can be addressed with well-known countermeasures such as those listed in RFC 4987⁷, including network filtering, reduced SYN-RECEIVED timer, and recycling half-open TCP connections oldest first. In any case, DoS or DDoS attacks are not really an issue in protected environments, such as company or project networks, as malicious agents can be identified quite rapidly and, in the case of DDoS attacks, the number of potential attack sources is very limited.

3.8.6 Elevation of Privilege

Elevation of Privilege happens if an actor can reach a higher level of privileges than that actor is authorized for. Most often, such attacks are associated with exploiting weaknesses on the system level, usually in the OS or applications. This attack depends on the robustness of the implementation and the privilege management for processes on the OS level. These predominantly technical issues are addressed by establishing security measurements during software development as well as a restrictive privilege configuration, such as not running application tasks with SYSTEM rights. A different elevation of privilege approach is trying to illegitimately assign group membership or other roles to reach higher privileges. This approach

⁷<https://tools.ietf.org/html/rfc4987>, last accessed on 2021-01-16

is explicitly prevented by semDIM's reasoning, controlling each assignment for legitimacy.

3.8.7 Summary

In summary, we analyzed the security threats and their potential effects on our proposed semDIM framework following the STRIDE model. Many threats exist independently from semDIM as they try to exploit underlying infrastructural weaknesses, such as tampering with data on the OS or database level. These infrastructural weaknesses need to be addressed on a lower technical level, including hardening the OS and network components. However, semDIM's security mechanisms also offer increased infrastructural protection by detecting inconsistencies through the 2PC message exchange in the case of successful tampering and offering checking the integrity of messages with graph signatures. The analysis shows that a network's security level is increased by using semDIM. Processes specifically introduced by semDIM, that is, creating, exchanging, and processing graph-based representations of IM information, are sufficiently protected. For this, we reuse state-of-the-art security measurements, such as two-way TLS and client authentication, and apply our certificate-based graph signing and validation framework specifically developed for graph-based information exchange.

3.9 Synopsis

We presented semDIM, a solution for Semantic Distributed Identity Management. We discussed the shortcomings of existing solutions and proposed an approach based on Semantic Web technologies to define and manage not only distributed identities of persons but also distributed groups and roles. Our approach features a client-server-based architecture providing DIM for a multi-organizational collaboration. We presented an ontology-pattern-based knowledge representation based on the foundational ontology DUL and integrated existing ontologies. The main benefit of this approach is its modularity and extendability. Thus, semDIM can easily be integrated into use-case- or domain-specific knowledge bases, for example, conforming to CO-PLM for industrial collaboration projects. The semDIM information-flow model for message processing features graph-based changes that can be triggered by a client on a server, certificate-based signing of the graph to ensure information integrity, 2PC protocol for distributed management of changes, and the new PbCC protocol combining features from distributed 2PL and MGL adapted to DIM. We discussed and presented the application

3.9. SYNOPSIS

of semDIM in a scenario of a cross-organizational collaboration project. Thus, we demonstrated the feasibility of our approach. Furthermore, we conducted a security analysis by applying the STRIDE standardized threat model.

Chapter 4

Conclusion

This thesis presented semDIM, a semantic framework to support secure distributed identity management based on Semantic Web technologies (see Chapter 3). The framework is the first to cover all requirements of secure semantic information management for multi-organizational collaboration. The key novel feature of semDIM is distributed management of semantic identities, groups, and roles. For each of those entities, semDIM offers ontology patterns based on the foundational ontology DUL to provide a formal, modular, and extendable knowledge representation of identity information. Its message exchange protocol was tailored to the requirements for secure, pattern-based identity information exchange across organizations. Its pattern-based concurrency control mechanism is the first to ensure consistency in pattern-based triple stores distributed and managed by multiple organizations. Furthermore, semDIM is the first framework to feature graph-based signatures based on an existing framework to check the integrity of exchanged identity information. In this thesis, the application of semDIM was demonstrated in the context of a scenario (see Section 1.1) inspired by a real-life industrial engineering and manufacturing project. By successfully evaluating semDIM against requirements regarding secure multi-organisational collaboration, we validated semDIM as the answer to **Research Question 2** of this thesis: How can Semantic Web technologies be implemented to support secure management of DIM in protected environments?

Focusing on the specific industrial scenario, we also demonstrated how core ontologies should be developed to fit the requirements of a knowledge representation suitable for cross-organizational information exchange. The developed core ontology, CO-PLM (see Chapter 2), is the first to cover all PLM phases and integrate workflows and security regulations. Therefore, CO-PLM is the answer to **Research Question 1** of this thesis: How

can Semantic Web technologies be implemented to support secure creation, exchange, and management of product information across organizations and lifecycle phases? Both CO-PLM and semDIM use modular ontology patterns based on the foundational ontology DUL and reuse other core ontologies. Therefore, they can not only be used in the addressed scenario domain but also in other use cases by extending them with additional ontologies. All CO-PLM patterns can be obtained from <https://github.com/FSCHOEN/CO-PLM>. A prototypical implementation of semDIM focusing on triple validation and verification as well as .owl-files of the ontology patterns can be accessed under <https://github.com/FSCHOEN/semDIM>.

4.1 Lessons Learned

Besides the above-mentioned contributions directly answering the research questions, this thesis includes first-hand experience of ontology engineering and building ontology-based software systems in the context of multi-organizational collaboration. Section 2.7.1 presented a Joint Software and Semantic Engineering Process (JoSSEP). JoSSEP is based on analyzing existing approaches and is enriched by experiences of working with ontologies. This process was developed and executed in this thesis to design and implement ontologies as well as prototypical applications based on them (see Section 2.7.1 and Sections 3.6–3.7). Practically executing JoSSEP in this thesis produced the following lessons learned regarding combining ontology and software engineering:

Using ontologies for validating data structure specifications in the design and operation of software We demonstrated the benefit of building data structures for applications on ontologies (see Section 2.7). In the design of software, test data can be created and checked according to the ontologies to ensure compliance with the specified data model. Ontology-based data models can be automatically evaluated in contrast to prose specifications. However, using ontologies for validating data in the operation of software can yield disadvantages in performance if applied without caution. Section 2.7 observed quadratic reasoning times when adding CO-PLM product part patterns and discussed chunking as a possible remediation.

Difficulties in interpreting concept meanings when reading ontologies Implementing ontologies correctly requires developers to understand the semantics of concepts. Interpreting ontologies, for example, those written in OWL, is a technical skill that most software developers

do not acquire. Furthermore, concepts may be ambiguously described, for example, by annotations, or developers may disagree about the semantic of a concept. In Section 2.6.5, such difficulties in reading and interpreting concept definitions in ontologies became apparent. We explained why we interpreted the existing axioms from IOLite to imply that `DigitalResource` is not a `Depiction`. The interpretation leading to adding such axioms that are not explicitly stated may be wrong. This could lead to inconsistent use of ontologies. Therefore, ontologies must not only be published and exchanged between practitioners but also regularly discussed to reevaluate them, such as when new use cases arise.

Anticipating changes when importing external ontologies Even the smallest inconsistencies when using ontologies can lead to reasoning errors. We experienced such reasoning errors when updating the foundational ontology DUL (from version 3.31 to 3.32) used for CO-PLM.¹ After years of being consistent when reasoning, some of our test triples became inconsistent. DUL is only provided in its latest version.² Furthermore, hosting systems must provide long-term stability of URLs when they store ontologies. However, we experienced changing URLs and, even worse, ontologies disappearing from their original locations. In particular, ontologies hosted on volatile project web sites, such as the ones of temporary university working groups, have proven to be occasionally unreliable. A possible solution is using online services that offer long-term URL resolution, such as PURL³ [88]. This approach could solve the problem of link stability for ontologies. However, using a third-party domain in an ontology's URIs is not always desired. Ontology libraries exist [14], but they also do not address the dependency management that is necessary for robust software development.

In a real-life setting, such risks from externally hosted ontologies must be anticipated by including them in the organization's version control and dependency management, as in JoSSEP (see Section 2.21). Version control can ensure consistency for the organization's existing data structures and applications, but the long-term consequences of managing and using different versions of ontologies in productive settings are yet to be researched.

¹The error involved social attributes being allowed exclusively for describing social objects in DUL 3.32. The inconsistent triples could be fixed using the social attributes superclass `Region` for CO-PLM's product attributes.

²<http://www.ontologydesignpatterns.org/ont/dul/DUL.owl>, last accessed on 2021-01-16

³<https://archive.org/services/purl/>, last accessed on 2021-01-16

4.2 Future Work

Both CO-PLM and semDIM were developed with a focus on multi-organizational collaboration represented by an industrial scenario. They provide extensive fundamental features that can benefit many different applications. Further research can investigate the applicability of these two solutions as well as possible extensions. For example, their use in other economic sectors could be explored, such as in chemical/pharmaceutical engineering, the agriculture sector, or in online learning.

The presented semDIM framework may be enhanced by additional features. For example, semDIM could be applied to (temporary) offline applications, such as when an employee of Company A and an employee of Company B exchange data ad-hoc without Internet access or other connections to their organization networks. Such an offline support could be achieved by storing and managing identity information on the clients and enabling them to reason on certain processing requests. This feature addresses a niche application and may offer benefits, for example, in emergency scenarios (such as widespread network or power outage), military settings (such as different security classifications of mobile devices and equipment or operations in networks under foreign control) or in a context of working in areas with unreliable network connections (such as maintenance procedures in remote places). This feature was not needed for the scenario from Section 1.1 and, therefore, is not a focus of this thesis. However, semDIM already offers a strong semantic basis, and its architecture and messaging process model may be adapted for such settings. For example, certificate-based temporary tokens can be used for authentication and authorization when servers are unreachable. Iterative signed graphs could enable clients to reconstruct, for example, multiple role assignments based on one another, without the help of a server. Iterative signing is already implemented in the graph signing framework used by semDIM. Furthermore, the certificates, the tokens, or both could be placed on distributed ledgers [33], such as blockchains, in the future. These ledgers could be public or private depending on, for example, the required confidentiality or the number and relationships of the involved partners. Since both Semantic Web technologies and distributed ledgers are currently evolving technologies, they can benefit from one another to form a use case relevant within professional applications.

The presented process for combining ontologies with software engineering, JoSSEP, has proven effective for developing prototypes in the context of the discussed scenario. Further practical employment and evaluation in other applications may help to improve this process for future projects.

Bibliography

- [1] Reiner Anderl and Dietmar Trippner, editors. *STEP Standard for the Exchange of Product Model Data: Eine Einführung in die Entwicklung, Implementierung und industrielle Nutzung der Normenreihe ISO 10303 (STEP)*. Vieweg+Teubner Verlag, Wiesbaden, 2000.
- [2] Jason Andress. *The basics of information security: Understanding the fundamentals of InfoSec in theory and practice*. Basics. Syngress, Amsterdam and Boston, 2011.
- [3] Renzo Angles and Claudio Gutierrez. An Introduction to Graph Data Management. In George Fletcher, Jan Hidders, and Josep Lluís Larriba-Pey, editors, *Graph Data Management, Data-Centric Systems and Applications*, pages 1–32. Springer, 2018.
- [4] Sören Auer, Rene Pietzsch, and Jörg Unbehauen. Datenintegration im Unternehmen mit Linked Enterprise Data. In Tassilo Pellegrini, Harald Sack, and Sören Auer, editors, *Linked Enterprise Data*, X.media.press. Springer, Berlin, 2014.
- [5] Tim Berners-Lee. *Linked Data*. Design Issues. W3.org, 2006. <https://www.w3.org/DesignIssues/LinkedData.html>, last accessed on 2021-01-16.
- [6] Andreas Blumauer. Linked Data in Unternehmen. Methodische Grundlagen und Einsatzszenarien. In Tassilo Pellegrini, Harald Sack, and Sören Auer, editors, *Linked Enterprise Data*, X.media.press. Springer, Berlin, 2014.
- [7] Steve Bratt. Semantic Web, and other technologies to watch: Presentation at the INCOSE International Workshop, 2007. <http://www.w3.org/2007/Talks/>, last accessed on 2021-01-16.

BIBLIOGRAPHY

- [8] Eric A. Brewer. Towards Robust Distributed Systems. In *Symposium on Principles of Distributed Computing (PODC)*, New York, NY, 2000. ACM.
- [9] Dan Brickley and Libby Miller. FOAF Vocabulary Specification 0.99, 2014. <http://xmlns.com/foaf/spec/20140114.html>, last accessed on 2021-01-16.
- [10] Giulia Bruno, Dario Antonelli, and Agostino Villa. A Reference Ontology to Support Product Lifecycle Management. *Procedia CIRP*, 33:41–46, 2015.
- [11] Jacopo Cassina, Maurizio Tomasella, Marco Taisch, and Andrea Matta. A new closed-loop PLM Standard for mass products. *International Journal of Product Development (IJPD)*, 8(2):141, 2009.
- [12] Cesare Pautasso and Erik Wilde. Why is the Web Loosely Coupled? A Multi-Faceted Metric for Service Design. In *International Conference on World Wide Web*. ACM, 2009.
- [13] Lorenzo Cirio, Isabel F. Cruz, and Roberto Tamassia. A Role and Attribute Based Access Control System Using Semantic Web Technologies. In *On the Move to Meaningful Internet Systems: OTM Workshops*, volume 4806 of *Lecture Notes in Computer Science*, pages 1256–1266, Berlin, 2007. Springer.
- [14] Mathieu d’Aquin and Natalya F. Noy. Where to Publish and Find Ontologies? A Survey of Ontology Libraries. *Journal of Web Semantics*, 11:96–111, 2012.
- [15] José G. Faísca and José Q. Rogado. Decentralized Semantic Identity. In *International Conference on Semantic Systems (SEMANTiCS)*, pages 177–180, New York, NY, 2016. ACM Press.
- [16] Sebti Foufou, Steven J. Fenves, Conrad Bock, Sudarsan Rachuri, and Ram D. Sriram. A core product model for PLM with an illustrative XML implementation. In *International Conference on Product Lifecycle Management*, Geneve, Switzerland, 2005. Inderscience Enterprises Limited.
- [17] Keke Gai, Meikang Qiu, Bhavani Thuraisingham, and Lixin Tao. Proactive Attribute-based Secure Data Schema for Mobile Cloud in Financial Industry. In *IEEE HPCC, CSS, and ICSS*, 2015.

BIBLIOGRAPHY

- [18] Aldo Gangemi. *DOLCE+DnS Ultralite (DUL)*. ontologydesignpatterns.org, 2009. http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite, last accessed on 2021-01-16.
- [19] Aldo Gangemi and Peter Mika. Understanding the Semantic Web through Descriptions and Situations. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Robert Meersman, Zahir Tari, and Douglas C. Schmidt, editors, *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888 of *Lecture notes in computer science*, pages 689–706. Springer, Berlin, Heidelberg, 2003.
- [20] Aldo Gangemi and Valentina Presutti. Ontology Design Patterns. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, pages 221–243. Springer, Berlin, Heidelberg, 2009.
- [21] Seth Gilbert and Nancy Lynch. Perspectives on the CAP Theorem. *Computer*, 45(2), 2012.
- [22] Jim Gray and Andreas Reuter. *Transaction processing: Concepts and techniques*. Data Management Systems. Morgan Kaufmann, San Francisco, CA, 1992.
- [23] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.
- [24] Hariolf Grupp, Andre Jungmittag, Ulrich Schmoch, and Harald Legler. *Hochtechnologie 2000: Neudefinition der Hochtechnologie für die Berichterstattung zur technologischen Leistungsfähigkeit Deutschlands*. Fraunhofer-Institut für Systemtechnik und Innovationsforschung (ISI), Karlsruhe, Hannover, 2000. http://publica.fraunhofer.de/eprints/urn_nbn_de_0011-n-36794.pdf, last accessed 2021-01-16.
- [25] Nicola Guarino, Simone Pribbenow, and Laure Vieu. Modeling Parts and Wholes. *Data Knowledge Engineering*, 3(20):257–258, 1996.
- [26] Joseph M. Hellerstein, Michael Stonebraker, and James Hamilton. Architecture of a Database System. *Foundations and Trends in Databases*, 1(2):141–259, 2007.
- [27] Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. *Foundations of semantic web technologies*. Chapman & Hall, Boca Raton, FL, 2009.

BIBLIOGRAPHY

- [28] Bo Hu and Glenn Svensson. A Case Study of Linked Enterprise Data. In *International Semantic Web Conference (ISWC)*, Lecture Notes in Computer Science, pages 129–144, Berlin, 2010. Springer.
- [29] Luokai Hu, Shi Ying, Xiangyang Jia, and Kai Zhao. Towards an Approach of Semantic Access Control for Cloud Computing. In *Cloud Computing*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.
- [30] John Hughes and Eve Maler. *Security Assertion Markup Language(SAML) 2.0 Technical Overview - Working Draft 03, 20 February 2005*. OASIS SSTC Working Draft, 2005. <https://www.oasis-open.org/committees/download.php/11511/sstc-saml-tech-overview-2.0-draft-03.pdf>, last accessed on 2021-01-16.
- [31] Muhammad Imran. *Towards an Assembly Reference Ontology*. Loughborough University Institutional Repository, 2013. <https://dspace.lboro.ac.uk/2134/13995>, last accessed on 2021-01-16.
- [32] Muhammad Imran and R.I.M. Young. Reference ontologies for interoperability across multiple assembly systems. *International Journal of Production Research (IJPR)*, 54(18):5381–5403, 2015.
- [33] Nabil El Ioini and Claus Pahl. A Review of Distributed Ledger Technologies. In *On the Move to Meaningful Internet Systems: OTM Conferences*, pages 277–288, Cham, 2018. Springer.
- [34] ISO/IEC. *ISO/IEC 27001:2013 Information technology — Security techniques — Information security management systems*. ISO/IEC, 2013.
- [35] ISO/IEC. *ISO/IEC 62264-1:2013 Enterprise-control system integration — Part 1: Models and terminology*. ISO/IEC, 2013.
- [36] ISO/IEC. *ISO/IEC 24760-1:2019 IT Security and Privacy — A framework for identity management*. ISO/IEC, 2019.
- [37] ISO/IEC. *ISO/DIS 10303-1:2020-01 Draft Industrial automation systems and integration - Product data representation and exchange - Part 1: Overview and fundamental principles*. ISO/IEC, 2020.
- [38] Asya I. Ivanova and Shahper Vodanovich. Single sign-on taxonomy. In *International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 151–155. IEEE, 2017.

BIBLIOGRAPHY

- [39] Saffija Kasem-Madani and Michael Meier. Security and Privacy Policy Languages: A Survey, Categorization and Gap Identification, 2015. <https://arxiv.org/pdf/1512.00201>, last accessed on 2021-01-16.
- [40] Andreas Kasten. *Secure Semantic Web Data Management: Confidentiality, Integrity, and Compliant Availability in Open and Distributed Networks*. University Koblenz-Landau, 2016.
- [41] Andreas Kasten and Ansgar Scherp. Ontology-Based Information Flow Control of Network-Level Internet Communication. *International Journal of Semantic Computing (IJSC)*, 9(01):1–45, 2015.
- [42] Andreas Kasten, Ansgar Scherp, and Peter Schauß. A Framework for Iterative Signing of Graph Data on the Web. In *ESWC 2014 Proceedings*. Springer, 2014.
- [43] A. S. M. Kayes, Jun Han, and Alan Colman. An ontological framework for situation-aware access control of software services. *Information Systems*, 53, 2015.
- [44] Alfons Kemper and André Eickler. *Datenbanksysteme: Eine Einführung*. De Gruyter Oldenbourg Studium. De Gruyter Oldenbourg, Berlin and Boston, 10 edition, 2015.
- [45] Sabrina Kirrane, Alessandra Mileo, and Stefan Decker. Access control and the Resource Description Framework: A survey. *Semantic Web*, 8(2), 2017.
- [46] Hristo Koshutanski, Mihaela Ion, and Luigi Telesca. Distributed Identity Management Model for Digital Ecosystems. In *International Conference on Emerging Security Information, Systems and Technologies*, pages 132–138, Los Alamitos, CA, 2007. IEEE Computer Society.
- [47] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, D. Sivakumar, Andrew Tompkins, and Eli Upfal. The Web as a graph. In *SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–10, New York, NY, 2000. ACM.
- [48] Vikas Kumar and Aashish Bhardwaj. Identity Management Systems. *International Journal of Strategic Decision Sciences*, 9(1):63–78, 2018.
- [49] Hasnae L’Amrani, Younès EL Bouzekri EL Idrissi, and Rachida Ajhoun. Identity Management Systems: Techno-Semantic Interoperability for Heterogeneous Federated Systems. *Computer and Information Science (CIS)*, 11(3):102, 2018.

BIBLIOGRAPHY

- [50] S. G. Lee, Y.-S. Ma, G. L. Thimm, and J. Verstraeten. Product lifecycle management in aviation maintenance, repair and overhaul. *Computers in Industry*, 59(2-3):296–303, 2008.
- [51] Paulo Leitão and Francisco Restivo. ADACOR: A holonic architecture for agile and adaptive manufacturing control. *Computers in Industry*, 57(2):121–130, 2006.
- [52] Wilfried Lemahieu, Seppe vanden Broucke, and Bart Baesens. *Principles of Database Management*. Cambridge University Press, 2018.
- [53] Wei Liu, Yong Zeng, Michael Maletz, and Dan Brisson. Product Lifecycle Management: A Survey. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 1213–1225, New York, NY, 2009. ASME.
- [54] Essam Mansour, Andrei Vlad Samba, Sandro Hawke, Maged Zereba, Sarven Capadisli, Abdurrahman Ghanem, Ashraf Abounaga, and Tim Berners-Lee. A Demonstration of the Solid Platform for Social Web Applications. In *Proceedings of the 25th International Conference Companion on World Wide Web - WWW 2016 Companion*. ACM Press, New York, 2016.
- [55] Cladio Masolo, Laure Vieu, Emanuele Bottazzi, Carola Catenacci, and Nicola Guarino. Social Roles and their Descriptions. In *Principles of Knowledge Representation and Reasoning*, Menlo Park, CA, 2004. AAAI Press.
- [56] Aristeidis Matsokis and Dimitris Kiritsis. An ontology-based approach for Product Lifecycle Management. *Computers in Industry*, 61(8):787–797, 2010.
- [57] Microsoft. The STRIDE Threat Model, 2009. [https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878\(v=cs.20\)?redirectedfrom=MSDN](https://docs.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)?redirectedfrom=MSDN), last accessed on 2021-01-16.
- [58] X. G. Ming, J. Q. Yan, X. H. Wang, S. N. Li, W. F. Lu, Q. J. Peng, and Y. S. Ma. Collaborative process planning and manufacturing in product lifecycle management. *Computers in Industry*, 59(2-3):154–166, 2008.
- [59] NIST. *NIST SP 800-53, Security and Privacy Controls for Federal Information Systems and Organizations*. NIST, 2014. <https://>

BIBLIOGRAPHY

- `csrc.nist.gov/publications/detail/sp/800-53/rev-5/final`, last accessed on 2021-01-16.
- [60] OASIS. *eXtensible Access Control Markup Language Version 3.0*. OASIS, 2010. <http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html>, last accessed on 2021-01-16.
- [61] Daniel Oberle. *Semantic Management of Middleware*, volume 1 of *Semantic Web and Beyond*. Springer Science+Business Media Inc, Boston, MA, 2006.
- [62] Leo Obrst, Dru McCandless, and David Ferrell. Fast Semantic Attribute-Role-Based Access Control (ARBAC) in a Collaborative Environment. In *Proceedings of the 8th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*. IEEE, 2012.
- [63] Hervé Panetto, Michele Dassisti, and Angela Tursi. ONTO-PDM: Product-driven ONTOlogy for Product Data Management interoperability within manufacturing process environment. *Advanced Engineering Informatics*, 26(2):334–348, 2012.
- [64] Fernando Silva Parreiras. *Marrying model-driven engineering and ontology technologies: The TwoUse approach*. PhD thesis, University of Koblenz-Landau, 2011.
- [65] Tassilo Pellegrini, Harald Sack, and Sören Auer, editors. *Linked Enterprise Data: Management und Bewirtschaftung vernetzter Unternehmensdaten mit Semantic Web Technologien*. X.media.press. Springer, Berlin, 2014.
- [66] Torsten Priebe, Wolfgang Dobmeier, and Nora Kamprath. Supporting attribute-based access control with ontologies. In *International Conference on Availability, Reliability and Security (ARES)*, 2006.
- [67] Robert D. Reid and Nada R. Sanders. *Operations management: An integrated approach*. John Wiley & Sons Inc, Hoboken, NJ, 4 edition, 2010.
- [68] Christoph Ringelstein and Steffen Staab. DIALOG: Distributed Auditing Logs. In *2009 IEEE International Conference on Web Services*, pages 429–436. IEEE, 06.07.2009 - 10.07.2009.

BIBLIOGRAPHY

- [69] Ansgar Scherp, Daniel Eißing, and Steffen Staab. strukt—A Pattern System for Integrating Individual and Organizational Knowledge Work. In *International Semantic Web Conference (ISWC)*, pages 569–584, Berlin, Heidelberg, 2011. Springer.
- [70] Ansgar Scherp, Carsten Saathoff, Thomas Franz, and Steffen Staab. Designing Core Ontologies. *Applied Ontology*, 6(3):177–221, 2011.
- [71] Falko Schönreich, Andreas Kasten, and Ansgar Scherp. A Pattern-Based Core Ontology for Product Lifecycle Management based on DUL. In *Workshop on Ontology Design and Patterns (WOP) colocated at International Semantic Web Conference (ISWC)*, CEUR Workshop Proceedings. CEUR-WS.org, 2018.
- [72] Falko Schönreich, Andreas Kasten, and Ansgar Scherp. Distributed Identity Management for Semantic Entities. In *International Conference on Information Management and Big Data (SIMBIG)*. Springer, 2020.
- [73] Falko Schönreich, Andreas Kasten, and Ansgar Scherp. Secure Product Lifecycle Management with CO-PLM. In *Advances in Pattern-based Ontology Engineering*. IOS Press, 2021.
- [74] Falko Schönreich, Ansgar Scherp, and Andreas Kasten. Distributed Identity Management for Semantic Entities based on Graph Signatures and owl:sameAs. *International Journal of Semantic Computing (IJSC)*, 2021.
- [75] Felix Schwagereit, Ansgar Scherp, and Steffen Staab. Representing Distributed Groups with dgFOAF. In *Extended Semantic Web Conference (ESWC)*, volume 6089 of *Lecture notes in computer science*, pages 181–195. Springer, Berlin, 2010.
- [76] Toby Segaran, Colin Evans, and Jamie Taylor. *Programming the Semantic Web*. Safari Books Online. O’Reilly Media Inc, Sebastopol, 2009.
- [77] Rob Shearer, Boris Motik, and Ian Horrocks. HerMiT: A highly-efficient OWL reasoner. In *OWLED Workshop on OWL: Experiences and Directions, colocated with the International Semantic Web Conference (ISWC)*, volume 432 of *CEUR Workshop Proceedings*, 2008.
- [78] Haibo Shen. A Semantic-Aware Attribute-Based Access Control Model for Web Services. In *International Conference on Algorithms and*

BIBLIOGRAPHY

- Architectures for Parallel Processing (ICA3PP)*, Berlin, Heidelberg, 2009. Springer-Verlag.
- [79] Edelberto F. Silva. ACROSS-FI: Attribute-Based Access Control with Distributed Policies for Future Internet. In *ICN*. IARIA XPS Press, 2015.
- [80] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2), 2007.
- [81] John Stark. *Product Lifecycle Management: 21st Century Paradigm for Product Realisation*. Decision engineering. Springer, London and New York, 2 edition, 2011.
- [82] Stefan Scheglmann, Ansgar Scherp, and Steffen Staab. Declarative Representation of Programming Access to Ontologies. In *Extended Semantic Web Conference (ESWC)*, Berlin, Heidelberg, 2012. Springer.
- [83] Markus D. Steinberg, Sirko Schindler, and Jan M. Keil. Use Cases and Suitability Metrics for Unit Ontologies. In Mauro Dragoni, María Poveda-Villalón, and Ernesto Jimenez-Ruiz, editors, *OWL: experiences and directions - reasoner evaluation*, LNCS. Springer, 2017.
- [84] Clemens Szyperski, Dominik Gruntz, and Stephan Murer. *Component software: Beyond object-oriented programming*. Component Software Series. Addison-Wesley, London, 2 edition, 2003.
- [85] Achille C. Varzi. Parts, Wholes, and Part-Whole Relations: The Prospects of Mereotopology. *Data and Knowledge Engineering*, 3(20):259–286, 1996.
- [86] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. A comparison of a graph database and a relational database. In H. Conrad Cunningham, editor, *Annual Southeast Regional Conference*, page 1, New York, NY, 2010. ACM.
- [87] Gottfried Vossen. Concurrency Control: Traditional Approaches. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of database systems*, pages 575–578. Springer, New York, NY, 2018.
- [88] W3C. *Best Practice Recipes for Publishing RDF Vocabularies - W3C Working Group Note 28 August 2008*. W3C, 2008. <https://www.w3.org/TR/swbp-vocab-pub/>, last accessed on 2021-01-16.

BIBLIOGRAPHY

- [89] W3C. *WebID 1.0 - Web Identity and Discovery Editor's Draft 05 March 2014*. W3C, 2014. <https://www.w3.org/2005/Incubator/webid/spec/identity/>, last accessed on 2021-01-16.
- [90] W3C. *Decentralized Identifiers (DIDs) v1.0 - Core architecture, data model, and representations - W3C Working Draft 07 September 2020*. W3C, 2020. <https://www.w3.org/TR/did-core/>, last accessed on 2021-01-16.
- [91] Tobias Walter. *Bridging Technological Spaces: Towards the Combination of Model-Driven Engineering and Ontology Technologies*. PhD thesis, University of Koblenz-Landau, 2011.
- [92] Robert K. Wysocki. *Effective Project Management: Traditional, Agile, Extreme, Hybrid*. Wiley, 8. edition, 2019.