INSTITUT FÜR INFORMATIK

Kohärente Softwareentwicklung – Grundlagen, Arbeitsumgebungen, Vorgehensweisen

Michael Skusa, Bernhard Thalheim

Bericht Nr. 1018
Dezember 2010

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Institut für Informatik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

Kohärente Softwareentwicklung – Grundlagen, Arbeitsumgebungen, Vorgehensweisen

Michael Skusa, Bernhard Thalheim

Bericht Nr. 1018
Dezember 2010

e-mail: michael.skusa@gmx.net,
thalheim@is.informatik.uni-kiel.de

Zusammenfassung

Bei der Entwicklung von Softwaresystemen werden die unterschiedlichen Aspekte und Facetten des zu entwickelnden Systems getrennt betrachtet und entsprechend getrennt voneinander modelliert. Eine wiederkehrende Herausforderung in der Softwareentwicklung ist die Zusammenführung der so entstandenen Teilmodelle zu einem konsistenten Gesamtmodell des Softwaresystems. Wir stellen hierzu mit den Modell-Suiten einen Ansatz vor, mit dessen Hilfe sich solche Modelle integrieren, auf Konsistenz prüfen und konsistent weiterentwickeln lassen. Die Beiträge bestehender Entwicklungswerkzeuge und Arbeitsumgebungen zur Unterstützung dieses Ansatzes werden dazu untersucht. Die sich daraus ergebenden Ergänzungen bestehender Arbeitsumgebungen werden diskutiert und an Beispielen demonstriert. Die Arbeit zeigt, wie Risiken, Mehrdeutigkeiten und Widersprüche, die bei Verwendung unterschiedlicher Modellierungssprachen und Modellierungswerkzeuge entstehen, durch den Einsatz von Modell-Suiten reduziert und vermieden werden.
Schlüsselwörter: Kohärenz, Modell-Suiten, modellübergreifende Konsistenz.

Inhaltsverzeichnis

1 Überblick

Bei der Entwicklung von Softwaresystemen entstehen Spezifikationen unter anderem von Datenstrukturen, Funktionalität, Gliederung der Funktionalität in Komponenten, Kommunikation zwischen Komponenten, Schichtung, Verteilung, Benutzerinteraktion usw. Jede dieser Spezifikationen repäsentiert ein Modell für einen Teilaspekt der zu entwickelnden Software. Würden alle Spezifikationen in einer einzigen, einheitlichen Modellierungssprache entwickelt, ließen sie sich mit den Mitteln dieser Sprache zu einem einzigen zusammenhängenden Modell zusammenfügen. Dieselben Mittel, mit denen sich die Konsistenz der Einzelmodelle prüfen und erreichen ließe, ständen auch für die Konsistenzprüfung des Gesamtmodells zur Verfügung.

Die Verwendung eines einzigen, homogenen Modellierungsansatzes ist in realen Entwicklungsprojekten nur selten anzutreffen. Dort werden unterschiedliche Modellierungsmethoden nebeneinander eingesetzt. Diese sind einerseits optimal auf spezifische Teilaspekte der Modellierung abgestimmt, führen andererseits aber zur Entstehung von Teilmodellen, für die kein Zusammenhang spezifiziert ist. Widersprüche zwischen Teilmodellen werden dadurch nicht oder zu spät erkannt und eskalieren schlimmstenfalls erst bei der Implementierung des Informationssystems, da die Implementierung eine implizite Zusammenführung von Teilmodellen erzwingt. Eine modellübergreifende Konsistenzsicherung oder gar automatisierte Unterstützung des Entwicklers bei der Änderung und Entwicklung von Teilmodellen stellt eine bisher nur unzureichend bewältigte Herausforderung dar.

1.1 Modellvielfalt in der Softwareentwicklung

Als Beispiel für die in praktischen Entwicklungsprojekten anzutreffende Vielfalt von Modellierungsansätzen mag die UML dienen. Obwohl diese Sprache den Begriff *unified* im Namen führt, ist sie keine homogene Sprache mit einheitlicher Syntax und Semantik. Vielmehr besteht sie aus unterschiedlichen Sprachen, die ursprünglich unabhängig voneinander entstanden sind und die untereinander nur vergleichsweise lose gekoppelt sind. Abbildung 1 zeigt diese Variationsbreite anhand der in der UML definierten Diagrammtypen. Die unterschiedliche formale Fundierung beeinflusst zudem die Entscheidung des Entwicklers für einen bestimmten Diagrammtyp und seine Interpretation des gewählten Diagrammtyps. In [16] wird gezeigt, wie bestimmte Eigenschaften der UML Entscheidungen des Entwicklers beeinflussen und ihn auch fehlleiten können. Selbst bei Beschränkung auf einen einzigen Diagrammtyp besteht für den Entwickler noch erheblicher Interpretationsspielraum. [3] verweist auf allein 48 mögliche Interpretationen für UML-Zustandsdiagramme.

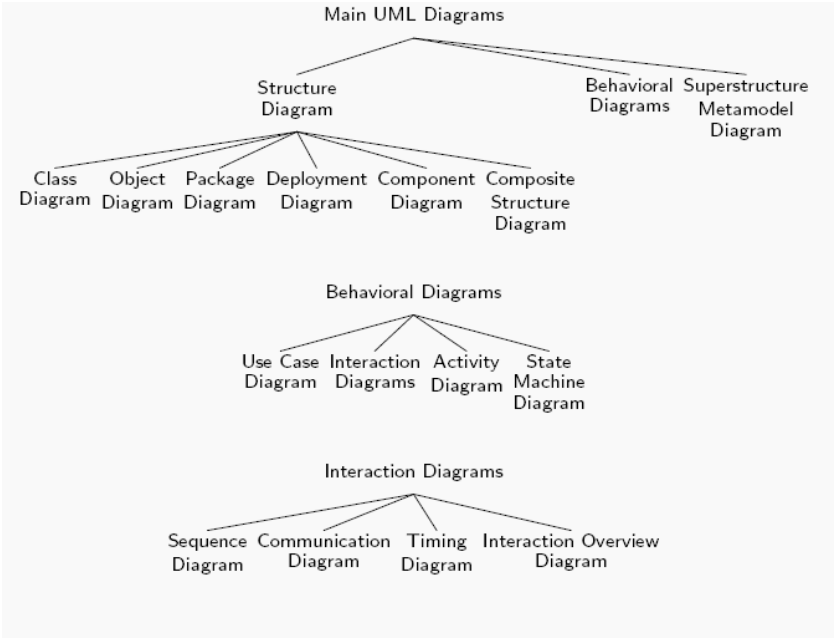

Abbildung 1: UML-Diagrammtypen

Die Semantik diagrammübergreifender Beziehungen ist in der UML teilweise formal, teilweise informal, teilweise überhaupt nicht spezifiziert. Eine Entwicklung von zusammenhängenden UML-Artefakten unterschiedlicher UML-Teilsprachen gelingt dadurch nur partiell [11, 12]. Abbildung 2 zeigt einen Satz von UML-Diagrammen, die bei der Entwicklung eines Informationssystems entstanden sein könnten. Ein Anwendungsfalldiagramm benennt die vom Informationssystem abzudeckenden und für den Nutzer des Systems sichtbaren Anwendungsfälle und ggf. deren Abhängigkeiten. Ein Klassendiagramm dient der Spezifikation der beteiligten Entitäten, ihres Zustandes, ihrer Beziehungen und Teilaspekten ihres Verhaltens. Ein Zustandsdiagramm modelliert hingegen ausschließlich das Verhalten einer Entität. Ändert der Entwickler beispielsweise ein Element des Zustandsdiagramms, das aus einer Eigenschaft des Klassendiagramms abgeleitet wurde, so muss diese Änderung auch im Klassendiagramm nachvollzogen werden. Sind derartige Beziehungen formal spezifiziert und werden sie von den beteiligten Teilmodellen eingehalten, dann bezeichnen wir die Teilmodelle im Folgenden als kohärent. Werden die Teilmodelle eines Informationssystems unter Verwendung unterschiedlicher Modellierungssprachen entwickelt, dann müssen Beziehungen zwischen den Modellen explizit spezifiziert werden, bevor ihre Einhaltung mit entsprechenden Entwicklungswerkzeugen überwacht und sichergestellt werden kann. Das Beispiel UML zeigt, dass dies selbst innerhalb einer vermeintlich einheitlichen Sprache notwendig ist.

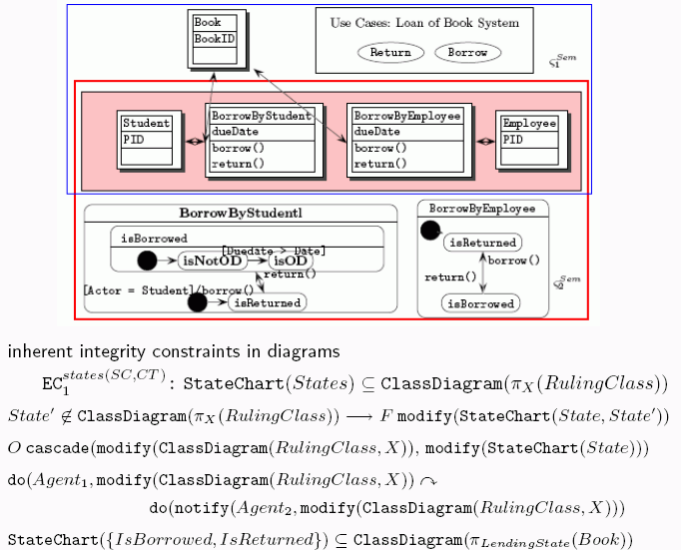

Abbildung 2: Zusammengehörige UML-Diagramme

1.2 Kohärenz in der Modellierung

Bisher existieren Modelle eines Entwicklungsprozesses vorwiegend nebeneinander, ohne dass zwingend ein formal spezifizierter Zusammenhang zwischen den Modellen besteht. Um einen Entwicklungsprozess zu erreichen, bei dem Modelle nicht nur zusammen existieren (Co-Existence), sondern auch zusammen konsistent weiterentwickelt werden (Co-Evolution), sind laut [3] unter anderem folgende Aufgaben zu lösen:

1. Explizite Spezifikation des Zusammenwirkens von Modellen.
2. Integrierte Entwicklung unterschiedlicher Modelle.
3. Datenaustausch zwischen Modellen, insbesondere zur Propagierung von Änderungen.
4. Werkzeugunterstützung für die Spezifikation, Speicherung und Propagierung von Modelländerungen.

Ziel der oben skizzierten Entwicklung ist die Erzielung von Kohärenz. Diese wird in [3] als feste Beziehung zwischen Modellen definiert: Zwei Modelle sind genau dann kohärent, wenn jede Änderung in einem der Modelle zum jeweils anderen Modell propagiert wird. Durch die Änderung dürfen die jeweils für die Einzelmodelle definierten Integritätsbedingungen nicht verletzt werden.

1.3 Gliederung

Im Abschnitt 2 wird der Begriff der semantischen Kohärenz aus bestehenden Definitionen von Konsistenz und Kohärenz hergeleitet, an Beispielen diskutiert und auf den Anwendungsbereich der Softwareentwicklung übertragen. Mit den Modell-Suiten stellen wir einen Ansatz vor, mit dessen Hilfe sich die Anforderungen an semantisch kohärente Softwareentwicklung erfüllen lassen (Abschnitt 3). Wie eine bestehende Arbeitsumgebung zur Softwareentwicklung inkrementell für die modellübergeifende Konsistenzsicherung angepasst wird, zeigt Abschnitt 4.

2 Semantische Kohärenz

Die Begriffe *Kohärenz* und *Semantik* kommen sowohl in der Wissenschaftstheorie und Philosophie als auch in unterschiedlichen Einzelwissenschaften zur Anwendung, unter anderem in der Linguistik, der Mathematik und der Informatik. Je nach Anwendungsgebiet sind diese beiden Begriffe unterschiedlich definiert. Im vorigen Abschnitt wurde Kohärenz über die konsistente Propagierung von Änderungen in zusammengehörigen Modellen definiert. In diesem Abschnitt wird der Begriff der semantischen Kohärenz präzisiert, indem auf bestehende Definitionen von Semantik und Kohärenz zurückgegriffen wird.

2.1 Konsistenz in der mathematischen Logik

In der mathematischen Logik ist der Kohärenzbegriff eng mit dem Begriff der Konsistenz beziehungsweise der Widerspruchsfreiheit verknüpft. In [9, S. 689] wird Konsistenz wie folgt definiert:

> "widerspruchsfrei/Widerspruchsfreiheit (engl. consistent/consistency), auch*: konsistent/Konsistenz*, in der mathematischen Logik Bezeichnung für eine Eigenschaft von Formelmengen. Man unterscheidet zwischen *semantischer* und *syntaktischer* Widerspruchsfreiheit. Eine Formelmenge Γ heißt semantisch widerspruchsfrei, wenn sie ein Modell hat, wenn es also eine Interpretation gibt, unter der alle Formeln in Γ gelten. Von syntaktischer Widerspruchsfreiheit spricht man in Bezug auf einen vorausgesetzten Ableitbarkeitsbegriff und damit auf ein formales System S. ..."

Die syntaktische Widerspruchsfreiheit wird dort wie folgt präzisiert:

> "(1) Γ heißt syntaktisch widerspruchsfrei, falls aus Γ in S kein Widerspruch herleitbar ist, d.h. für keine Formel A die Behauptung $\Gamma \vdash_S A \wedge \neg A$ gilt; (2) Γ heißt syntaktisch widerspruchsfrei, falls nicht jede Formel der betrachteten Sprache aus Γ in S herleitbar ist, d.h. nicht für jede Formel A die Behauptung $\Gamma \vdash_S A$ gilt. Die beiden syntaktischen Widerspruchsfreiheitsbegriffe sind in der Regel unter der Annahme des ex falso quodlibet, äquivalent; ..."

Auf Artefakte der Softwareentwicklung ließe sich diese Definition wie folgt anwenden:

- Jeder Artefakttyp AT sei beschrieben durch eine Menge von Formeln Γ_{AT} in einem formalen System S.
- Jedes Artefakt entspreche einer Interpretation solcher Formeln.
- Genau dann, wenn die dem Artefakttyp entsprechenden Formeln unter der Interpretation des konkreten Artefaktes gelten, ist das Artefakt ein Modell für den Artefakttyp.
- Eine Menge $\mathfrak{AT} = \{AT_1, \ldots, AT_n\}$ von Artefakttypen ist genau dann semantisch konsistent, wenn die zugehörige Formelmenge $\Gamma_{\mathfrak{AT}} = \bigcup_{i=1}^{n} \Gamma_{AT_i}$ ein Modell hat, also eine Interpretation, unter der alle den Artefakttypen entsprechenden Formeln gelten.
- Eine Menge von Artefakttypen $\mathfrak{AT}$ ist genau dann syntaktisch konsistent, falls für keine der den Artefakttypen entsprechenden Formeln A gilt $\Gamma_{\mathfrak{AT}} \vdash_S A \wedge \neg A$.

Um überhaupt Aussagen über den Zusammenhang von Inhalten treffen zu können, wird hier die Verwendung eines einheitlichen formalen Systems S unterstellt. Die Ableitbarkeit von Formeln ist nur innerhalb desselben Systems beziehungsweise eines zugehörigen Kalküls definiert. Um also Aussagen über die Konsistenz von Artefaktmengen im Sinne der mathematischen Logik treffen zu können, müssen die Artefakttypen auf Formeln eines einheitlichen formalen Systems und die Artefakte auf Interpretationen dieser Formeln abgebildet werden. Als Beispiel für ein solches formales System betrachten wir im Folgenden die Prädikatenlogik erster Stufe. Entsprechend beschränken sich dann Konsistenzbetrachtungen auf solche Artefakteigenschaften, die sich als Formeln in der Prädikatenlogik erster Stufe darstellen lassen.

Angenommen ein Artefakttyp ließe sich als Menge prädikatenlogischer Formeln repräsentieren. Dann könnte man auch die Formelmengen unterschiedlicher Artefakttypen zu einer einzigen Menge von prädikatenlogischen Formeln vereinigen. Die

syntaktische Konsistenz dieser Formelmenge ließe sich mit Hilfe der Regeln des Prädikatenkalküls feststellen. Auch die semantische Konsistenz der Formelmenge ließe sich feststellen, wenn sich aus den Modellen der einzelnen Artefakttypen eine Interpretation konstruieren lässt, die ein Modell für die Vereinigung der Artefakttypen ist.

2.2 Kohärenz in der mathematischen Logik

Konsistenz beschreibt die Widerspruchsfreiheit einer Formelmenge, aber keine darüber hinausgehenden Zusammenhänge zwischen den Elementen der Formelmenge. Angenommen, zwei Artefakttypen AT_i, AT_j seien insofern "disjunkt", als dass die Schnittmenge ihrer Formelmengen leer ist $\Gamma_{AT_i} \cap \Gamma_{AT_j} = \emptyset$, inbesondere auch keine Teilformeln existieren, die sowohl in Γ_{AT_i} als auch in Γ_{AT_j} auftreten. Dann ist die Vereinigung $\Gamma_{AT_i} \cup \Gamma_{AT_j}$ genau dann syntaktisch konsistent, wenn auch Γ_{AT_i} und Γ_{AT_j} syntaktisch konsistent sind. Jede in Γ_{AT_i} oder Γ_{AT_j} ableitbare Formel ist auch in $\Gamma_{AT_i} \cup \Gamma_{AT_j}$ ableitbar. Aus Γ_{AT_i} sind keine Widersprüche zu Γ_{AT_j} herleitbar, da Γ_{AT_i} keine Formeln oder Prädikate enthält, die auch in Γ_{AT_j} vorkommen. Entprechend sind aus Γ_{AT_j} keine Widersprüche zu Γ_{AT_i} herleitbar. $\Gamma_{AT_i} \cup \Gamma_{AT_j}$wäre also konsistent, es ist aber keine Formel ableitbar, die einen Zusammenhang zwischen den beiden Artefakttypen beschreibt, in der also sowohl Teilformeln aus Γ_{AT_i} als auch Teilformeln aus Γ_{AT_j} vorkommen. Syntaktisch ist die Vereinigung der beiden Artefakttypen konsistent, aber nicht zusammenhängend.

Eine analoge Überlegung lässt sich für die semantische Konsistenz anstellen. Wenn in einer Artefaktmenge jedes Artefakt ein Modell für die Formeln seines zugehörigen Artefakttyps ist und die Vereinigung der zugehörigen Artefakttypen syntaktisch konsistent, aber nicht zusammenhängend ist, dann lässt sich zur Vereinigung der Artefakttypen eine Interpretation angeben, in der jedes Artefakt immer noch ein Modell für die Formeln seines zugehörigen Artefakttyps ist, in der aber keinerlei Relationen zwischen Artefakten unterschiedlicher Typen existieren.

Beschreibungen solcher artefaktübergreifenden Zusammenhänge sind aber erforderlich, um einen Softwareentwickler bei der Erstellung neuer Artefakte auf Basis von Daten anderer Artefakte zu unterstützen. Das Vorhandensein solcher Zusammenhänge beschreibt also eine Eigenschaft einer Formelmenge, die über bloße Konsistenz hinausgeht. Um diese begrifflich zu fassen, ist folgende Definition von Kohärenz hilfreich [8]:

> "kohärent/Kohärenz (von lat. cohaerere, zusammenhängen), Terminus zur allgemeinen Kennzeichnung des Zusammenhangs von (1) physischen,

> (2) psychischen und (3) sprachlichen bzw. gedanklichen Mannigfaltigkeiten; ... in letzterer Bedeutung oft synonym mit 'Konsistenz von Sätzen' im Sinne ihrer Widerspruchsfreiheit gebraucht. Es empfiehlt sich jedoch, 'Kohärenz' terminologisch der Kennzeichnung einer Bezeichung vorzubehalten, die sich nicht in der Kontinuität oder Konsistenz erschöpft. ..."

Der letzte Satz wird dort im Weiteren wie folgt präzisiert:

> "... (3) Die über bloße Widerspruchsfreiheit hinausgehende Kohärenz von Sätzen kann in ihrer Verknüpfung nach solchen Regeln gesehen werden, die dem jeweils behandelten Gegenstandsbereich methodisch angemessen sind, insbesondere den Regeln zur Verwendung bereichs- und methodenspezifischer Termini und der methodenspezifischen Begründung eines Satzes durch einen anderen."

Der erste Teil dieser Definition ergänzt den Konsistenzbegriff. Alle kohärenten Mannigfaltigkeiten sind auch konsistent. Aber nicht jede konsistente Mannigfaltigkeit ist kohärent. Kohärent wird sie erst dadurch, dass sie eine über die Widerspruchsfreiheit hinausgehende Form von Zusammenhang besitzt. Der zweite Teil der Definition gibt einen Hinweis darauf, worin solche Zusammenhänge bestehen können und wie sie sich repräsentieren lassen. Sätze können nach Regeln verknüpft werden, die einem bestimmten Gegenstandsbereich methodisch angemessen sind, so dass ein Satz als Begründung eines anderen Satzes fungieren kann. In der Softwareentwicklung könnte ein Algorithmus zur Erzeugung eines Artefakts aus den Daten eines anderen Artefakts eine solche Regel sein, nach der Sätze, hier: Artefakte, miteinander verknüpft werden. Das Ausgangsartefakt fungiert dann zusammen mit dieser Regel als Begründung für das Zielartefakt.

Solange solche über Widerspruchsfreiheit hinausgehenden Zusammenhänge nicht formal repräsentiert sind, beschreibt Kohärenz eine pragmatische Eigenschaft. Angewandt auf die Softwareentwicklung bedeutet das, ein menschlicher Nutzer kann eine Menge von Artefakten als zusammengehörig erkennen, obwohl sich diese Zusammengehörigkeit weder aus der Syntax noch aus der Semantik der Sprache ableiten lässt, in der die Artefakte formuliert sind. Er nutzt dazu (pragmatisch) sein Wissen. Beispiele für solches Wissen:

- Aggregationen: Wenn beide Artefakte Teil derselben Projektdatei sind, beschreiben sie vermutlich Eigenschaften derselben Software.
- Identität: Tauchen in unterschiedlichen Artefakten desselben Projekts identische Bezeichner auf, dann benennen diese dieselbe Entität.

- Ähnlichkeit: Sind zwei Bezeichner identisch, nachdem man alle Sonderzeichen entfernt und Groß-/Kleinschreibung ignoriert hat, dann bezeichnen sie vermutlich dieselbe Entität.

Kohärenz kann ein Nutzer also nur feststellen, wenn er solche Regeln, die artefaktübergreifende Beziehungen beschreiben, kennt und auf eine gegebene Artefaktmenge erfolgreich anwenden kann. Sollen diese Regeln algorithmisch angewendet oder überprüft werden, müssen sie im selben formalen System repräsentiert werden, wie die zu prüfenden Artefakte.

Gegeben seien beispielsweise die Formelmengen $\Gamma_1 = \{A, B, A \to B\}$ und $\Gamma_2 = \{C, D, C \to D\}$. Dann ist $\Gamma_1 \cup \Gamma_2 = \{A, B, A \to B, C, D, C \to D\}$ konsistent, aber nicht zusammenhängend. Fügt man nun die Formel $B \to C$ hinzu, erhält man die neue Menge $\Gamma_1 \cup \Gamma_2 \cup \{B \to C\} = \{A, B, A \to B, C, D, C \to D, B \to C\}$. Diese ist konsistent und zusammenhängend, also kohärent. Man kann in dieser neuen Menge aus der Formel B aus Γ_1 eine Formel C aus Γ_2 ableiten.

Dass nicht jede Ergänzung zu einer Kohärenz führt, zeigt sich, wenn man im obigen Beispiel anstelle von $B \to C$ die Formel $B \to \neg C$ ergänzt. Dann ergibt sich $\Gamma_1 \cup \Gamma_2 \cup \{B \to \neg C\} = \{A, B, A \to B, C, D, C \to D, B \to \neg C\}$. Diese neue Formelmenge ist zwar zusammenhängend, weil sie immer noch Formeln unterschiedlicher Ausgangsmengen verknüpft, aber sie ist nicht mehr konsistent. Aus ihr läßt sich jetzt ein Widerspruch herleiten: $\{B, C, B \to \neg C\} \vdash \{C, \neg C\}$.

Der Begriff des Zusammenhangs wird im folgenden auf Basis der Teilformelrelation auf der Menge der prädikatenlogischen Formeln präzisiert.

Definition Sei $\mathcal{F}$ die Menge aller prädikatenlogischen Formeln und seien F und G prädikatenlogische Formeln, also $F, G \in \mathcal{F}$. Dann bezeichne das Prädikat $\Subset$ die *Teilformelrelation*, also

$$F \Subset G \longleftrightarrow (F, G) \in \{(x, y) | x, y \in \mathcal{F} \wedge x \, ist \, Teil formel \, von \, y\}$$

Über die Teilformelrelation lässt sich nun die Verknüpfung zweier Formeln mit Hilfe einer dritten wie folgt definieren:

Definition Seien F, G und H prädikatenlogische Formeln, also $F, G, H \in \mathcal{F}$. F und G heißen genau dann *verknüpft durch* H oder $F \underset{H}{\boxtimes} G$, wenn H sowohl F als auch G als Teilformel enthält, also

$$F \underset{H}{\boxtimes} G \longleftrightarrow (F, H, G) \in \{(x, z, y) | x, y, z \in \mathcal{F} \wedge x \Subset z \wedge y \Subset z\}$$

Definition Seien $\mathcal{M}, \mathcal{N}$ und $\mathcal{O}$ Mengen prädikatenlogischer Formeln, also $\mathcal{M}$,$\mathcal{N}$,$\mathcal{O} \subseteq \mathcal{F}$. $\mathcal{M}$ und $\mathcal{N}$ heißen genau dann *verknüpft durch* $\mathcal{O}$, oder $\mathcal{M} \underset{\mathcal{O}}{\otimes} \mathcal{N}$ wenn $\mathcal{O}$ mindestens eine Formel enthält, die eine Formel aus $\mathcal{M}$ mit einer Formel aus $\mathcal{N}$ verknüpft, also

$$\begin{aligned} \mathcal{M} \underset{\mathcal{O}}{\otimes} \mathcal{N} \longleftrightarrow \quad & (\mathcal{M}, \mathcal{O}, \mathcal{N}) \in \{(x, z, y) | \\ & x, y, z \subseteq \mathcal{F} \wedge \\ & \exists F, G, H \in \mathcal{F} : (F \in x) \wedge (G \in y) \wedge (H \in z) \wedge (F \underset{H}{\boxtimes} G)\} \end{aligned}$$

Die transitive Verknüpfung $\mathcal{M} \underset{\mathcal{O}}{\circledast} \mathcal{N}$, wird rekursiv definiert als

$$\mathcal{M} \underset{\mathcal{O}}{\circledast} \mathcal{N} \longleftrightarrow \mathcal{M} \underset{\mathcal{O}}{\otimes} \mathcal{N} \vee \exists \mathcal{P} \subseteq \mathcal{F} : \mathcal{M} \underset{\mathcal{O}}{\otimes} \mathcal{P} \wedge \mathcal{P} \underset{\mathcal{O}}{\circledast} \mathcal{N}\}$$

Ausgehend von den bisherigen Definitionen der Konsistenz, der Verknüpfungsrelationen und der Kohärenz lassen sich nun folgende Begriffe definieren:

Definition Gegeben seien Formelmengen $\Gamma_1, \ldots, \Gamma_n, \mathcal{O} \subseteq \mathcal{F}$.

- $\mathcal{O}$ heißt *zusammenhängende Obermenge* von $\Gamma_1, \ldots, \Gamma_n$ genau dann, wenn $\bigcup_{i=1}^{n} \Gamma_i \subseteq \mathcal{O}$ und jedes Γ_i direkt oder transitiv durch $\mathcal{O}$ mit jedem Γ_j mit $i \neq j$ verknüpft ist, also folgende Bedingung erfüllt ist

  $$\forall \Gamma_i, \Gamma_j \in \{\Gamma_1, \ldots, \Gamma_n\} : (i \neq j) \rightarrow \Gamma_i \underset{\mathcal{O}}{\circledast} \Gamma_j$$

  Interpretiert man die Γ_i als Knoten eines Graphen und die Paare (Γ_i, Γ_j) aus $\Gamma_i \underset{\mathcal{O}}{\otimes} \Gamma_j$ als Kanten des Graphen, dann ist $\mathcal{O}$ eine zusammenhängende Obermenge von $\Gamma_1, \ldots, \Gamma_n$, wenn der sich aus $\underset{\mathcal{O}}{\otimes}$ ergebende Graph zusammenhängend ist.

- $\mathcal{O}$ heißt *kohärente Obermenge* von $\Gamma_1, \ldots, \Gamma_n$ genau dann, wenn $\mathcal{O}$ zusammenhängende Obermenge von$\Gamma_1, \ldots, \Gamma_n$ ist und $\mathcal{O}$ konsistent ist.
  Genau dann, wenn $\mathcal{O}$ eine zusammenhängende Obermenge von $\Gamma_1, \ldots, \Gamma_n$ ist und syntaktisch konsistent ist, heißt $\mathcal{O}$ *syntaktisch kohärente Obermenge* von $\Gamma_1, \ldots, \Gamma_n$. Genau dann, wenn $\mathcal{O}$ eine zusammenhängende Obermenge von $\Gamma_1, \ldots, \Gamma_n$ ist und semantisch konsistent ist, heißt $\mathcal{O}$ *semantisch kohärente Obermenge* von $\Gamma_1, \ldots, \Gamma_n$.

Eine kohärente Obermenge kann auf zwei Wegen entstehen. Ist die Schnittmenge zweier Γ_i nicht leer, dann ist bereits deren Vereinigung eine kohärente Obermenge,

sofern diese Vereinigungsmenge konsistent ist. Ist die Schnittmenge zweier Γ_i leer, dann muss ihrer Vereinigung mindestens eine Formel explizit hinzugefügt werden, welche Formeln der beiden Ausgangsmengen verknüpft, um eine zusammenhängende Obermenge zu erhalten. Ist die so entstandene Menge konsistent, dann ist sie auch kohärent. Aus der Definition der semantisch kohärenten Obermenge folgt, dass diese ein Modell haben muss, also eine Interpretation, in der alle ihre Formeln gelten. Wie eine solche Interpretation aus den Interpretationen der einzelnen Γ_i gewonnen werden kann, ist Gegenstand des folgenden Abschnitts.

2.3 Integration semantischer Strukturen

In Abschnitt 2.1 wurde als Beispiel für ein formales System die Prädikatenlogik erster Stufe gewählt. Die Interpretation von Formeln solcher Systeme und die Fragestellungen bezüglich der Integration solcher Interpretationen werden daher im Folgenden auf Basis der Semantik der Prädikatenlogik diskutiert. Bei Schöning [13, S. 52 ff.] wird die Semantik der Prädikatenlogik über den folgenden Begriff der Struktur definiert:

> "Eine *Struktur* ist ein Paar $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$ wobei $U_\mathcal{A}$ eine beliebige aber nicht leere Menge ist, die die Grundmenge von $\mathcal{A}$ (oder der *Grundbereich*, der *Individuenbereich*, das *Universum*) genannt wird. Ferner ist $I_\mathcal{A}$ eine Abbildung, die
>
> - jedem k-stelligen Prädikatsymbol P (das im Definitionsbereich von $I_\mathcal{A}$ liegt) ein k-stelliges Prädikat über $U_\mathcal{A}$ zuordnet,
> - jedem k-stelligen Funktionssymbol f (das im Definitionsbereich von $I_\mathcal{A}$ liegt) eine k-stellige Funktion auf $U_\mathcal{A}$ zuordnet,
> - jeder Variablen x (sofern $I_\mathcal{A}$ auf x definiert ist) ein Element der Grundmenge $U_\mathcal{A}$ zuordnet.
>
> ... Sei F eine Formel und $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$ eine Struktur. $\mathcal{A}$ heißt zu F *passend*, falls $I_\mathcal{A}$ für alle in F vorkommenden Prädikatsymbole, Funktionssymbole und freien Variablen definiert ist.
>
> ... Falls für F und eine zu F passende Struktur $\mathcal{A}$ gilt $\mathcal{A}(F) = 1$, so schreiben wir wieder $\mathcal{A} \models F$ (Sprechweise: F *gilt* in $\mathcal{A}$ oder $\mathcal{A}$ ist *Modell* für F)"

Auch die Semantik von Aussagen einer Logik höherer Ordnung wird in der Literatur über vergleichbare Strukturen definiert [15, S. 757 ff.]. Sollen zwischen Modellen

zweier Formelmengen $\mathcal{M}$ und $\mathcal{N}$ übergreifende Beziehungen beschrieben werden, ist zunächst eine kohärente Obermenge $\mathcal{O}$ zu bilden, für die also $\mathcal{M} \cup \mathcal{N} \subseteq \mathcal{O}$ und $\mathcal{M} \underset{\mathcal{O}}{\otimes} \mathcal{N}$ gilt. Sei $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$ eine zu $\mathcal{M}$ passende Struktur und $\mathcal{B} = (U_\mathcal{B}, I_\mathcal{B})$ eine zu $\mathcal{N}$ passende Struktur. Dann wird für $\mathcal{O}$ eine Struktur $\mathcal{C} = (U_\mathcal{C}, I_\mathcal{C})$ benötigt, die mindestens folgende Anforderungen erfüllt:

- Individuen von $\mathcal{A}$ und $\mathcal{B}$ sind auch Individuen von $\mathcal{C}$,
- für jedes k-stellige Prädikatsymbol P, für das eine Interpretation in $\mathcal{A}$ oder $\mathcal{B}$ existiert, existiert eine kompatible Interpretation in $\mathcal{C}$,
- für jedes k-stellige Funktionssymbol f, für das eine Interpretation in $\mathcal{A}$ oder $\mathcal{B}$ existiert, existiert eine kompatible Interpretation in $\mathcal{C}$.

Der Begriff der *kompatiblen Interpretation* ist hierbei erklärungsbedürftig. Da sowohl die Individuen aus $\mathcal{A}$ als auch die Individuen aus $\mathcal{B}$ Teil des Universums von $\mathcal{C}$ werden sollen, gilt mindestens $U_\mathcal{C} \supseteq U_\mathcal{A} \cup U_\mathcal{B}$. Dadurch vergrößert sich potentiell der Definitionsbereich von Funktionen und Prädikaten in $I_\mathcal{C}$. Dieselben Elemente aus $U_\mathcal{A}$, die ein Prädikat P in $I_\mathcal{A}(P)$ erfüllen, sollen dieses auch in $I_\mathcal{C}(P)$ erfüllen. Unter der Annahme, dass kein Prädikatsymbol sowohl in $\mathcal{M}$ als auch in $\mathcal{N}$ enthalten ist, ergibt sich dann für ein k-stelliges Prädikatsymbol P:

- $I_\mathcal{C}(P) = I_\mathcal{A}(P)$, falls P ist Prädikatsymbol in $\mathcal{M}$ und
- $I_\mathcal{C}(P) = I_\mathcal{B}(P)$, falls P ist Prädikatsymbol in $\mathcal{N}$.

D.h. die Menge $I_\mathcal{C}(P)$ enthält dieselben Elemente, gebildet aus denselben Individuen, wie in der ursprünglichen Interpretation. Diese Interpretation $I_\mathcal{C}(P)$ ist kompatibel mit der ursprünglichen Interpretation (z.B. $I_\mathcal{A}(P)$), da sie basierend auf dem neuen, potentiell größeren Definitionsbereich $U_\mathcal{C}$ dieselben Elemente enthält wie die Interpretation auf Basis des ursprünglichen, potentiell kleineren Definitionsbereichs.

Unter der Annahme, dass $\mathcal{M}$ und $\mathcal{N}$ keine gemeinsamen Funktionssymbole enthalten, gilt entsprechend für eine kompatible Interpretation einer Funktion f:

- $I_\mathcal{C}(f) = I_\mathcal{A}(f)$, falls f ist Funktionssymbol in $\mathcal{M}$ und
- $I_\mathcal{C}(f) = I_\mathcal{B}(f)$, falls f ist Funktionssymbol in $\mathcal{N}$.

Da f in der ursprünglichen Interpretation (z.B. $I_\mathcal{A}(f)$) nur für Argumente auf Basis des zugehörigen Universums (im Beispiel $U_\mathcal{A}$) definiert ist, ist sie auch in $I_\mathcal{C}(f)$ nur für Elemente des ursprünglichen Universums definiert. Für Argumente auf Basis von Elementen die nicht Teil des ursprünglichen Universums sind (im Beispiel Elemente aus $U_\mathcal{C} \setminus U_\mathcal{A}$) sei $I_\mathcal{C}(f)$ undefiniert.

Die Existenz von Prädikat- und Funktionssymbolen, die sowohl in einer Formelmenge $\mathcal{M}$ als auch in einer davon verschiedenen Formelmenge $\mathcal{N}$ existieren, wird für die Anwendung auf Artefakte der Softwareentwicklung ausgeschlossen. Falls zwischen Prädikat- oder Funktionssymbolen unterschiedlicher Formelmengen Namensgleichheiten bestehen, werden diese vor der Bildung einer kohärenten Obermenge durch Umbenennung der entsprechenden Symbole beseitigt. Werden unterschiedliche Formelmengen $\mathcal{M}$ und $\mathcal{N}$ zu einer kohärenten Obermenge $\mathcal{O}$ vereinigt und zu den Ausgangsmengen passende Strukturen $\mathcal{A}$ und $\mathcal{B}$ auf eine zur kohärenten Obermenge passende Struktur $\mathcal{C}$ abgebildet, dann ergibt sich aus dem Definitionsbereich der Funktionen und Prädikate in der gemeinsamen Struktur indirekt die Zugehörigkeit der Individuen zu ihrem ursprünglichen Universum. Das heißt, Individuen, die in der gemeinsamen Struktur ein Prädikat P erfüllen, dass nur für Individuen des Universums $U_{\mathcal{A}}$ definiert ist, können nur aus $U_{\mathcal{A}}$ stammen. Entsprechend lässt sich aus Formeln der kohärenten Obermenge $\mathcal{O}$, die Prädikate verknüpfen, von denen einige nur für Individuen eines Universums, andere nur für Individuen eines anderen Universums definiert sind, indirekt auf Zusammenhänge zwischen Individuen unterschiedlicher Universen schließen. Aus solchen Zusammenhängen zwischen Individuen unterschiedlicher Universen lässt sich auf Zusammenhänge unterschiedlicher Strukturen schließen. Solche Zusammenhänge zwischen Strukturen, sind für den Nutzer der gemeinsamen Struktur nur erkennbar, wenn er entweder (pragmatisch) die Zuordnung der Individuen der gemeinsamen Struktur zu ihren ursprünglichen Strukturen kennt oder wenn diese Zuordnung explizit in der gemeinsamen Struktur repräsentiert ist. Eine explizite Repräsentation dieser Zuordnung ist nicht nur sinnvoll, sondern sogar zwingend erforderlich, sobald Beziehungen zwischen Strukturen repräsentiert werden sollen, die ein Modell für dieselbe Formelmenge sind, also beispielsweise Beziehungen zwischen Artefakten desselben Artefakttyps. Ohne eine solche Zuordnung der Individuen zu ihrer ursprünglichen Struktur ließe sich nicht unterscheiden, ob eine Verknüpfung von Individuen eine Beziehung innerhalb desselben Artefakts oder eine artefaktübergreifende Beziehung darstellt. Daher fordern wir für eine gemeinsame Struktur, dass

- ihr Universum für jede Ausgangsstruktur $\mathcal{A}_i$ ein zusätzliches Individuum $Id_{\mathcal{A}_i}$ enthält, das die jeweilige Ausgangsstruktur eindeutig identifiziert,
- die kohärente Obermenge, zu der die gemeinsame Struktur passt, ein zweistelliges Prädikat *Strukturzuordnung* enthält,
- die gemeinsame Struktur eine zum Prädikat *Strukturzuordnung* passende Relation enthält, welche die Zuordnung eines Individuums zu seiner ursprünglichen

Struktur widerspiegelt.

Für den binären Fall, also die Zusammenführung zweier Strukturen $\mathcal{A}$ und $\mathcal{B}$ zu einer gemeinsamen Struktur $\mathcal{C}$ ergäbe sich dann:

- $U_\mathcal{A} \cup U_\mathcal{B} \cup \{Id_\mathcal{A}, Id_\mathcal{B}\} \subseteq U_\mathcal{C}$, und
- $I_\mathcal{C}(Strukturzuordnung) = \{(x,y) | \; x \in U_\mathcal{A} \cup U_\mathcal{B} \wedge$
  $y \in \{Id_\mathcal{A}, Id_\mathcal{B}\} \wedge$
  $x\, ist\, Element\, des\, Universums\, von\, y\}$

Während in den jeweiligen Ausgangsstrukturen implizit definiert ist, dass jedes Prädikat nur innerhalb "seiner" derzeitigen Struktur erfüllt ist, muss in einer gemeinsamen Struktur explizit definiert werden, in welcher der Teilstrukturen ein Prädikat für eine bestimmte Parameterkombination gilt. Dazu wird in der kohärenten Obermenge der beteiligten Formelmengen zu jedem Prädikat $P(x_1, \ldots, x_n)$ zusätzlich ein Prädikat $P'(x_1, \ldots, x_{n,}, a)$ definiert, dessen zusätzlicher Parameter die ursprüngliche Struktur identifiziert, aus der alle x_i stammen müssen, also $P'(x_1, \ldots, x_n, a) \leftrightarrow P(x_1, \ldots x_n) \wedge \forall x \in \{x_1, \ldots x_n\} : Strukturzuordnung(x, a)$. Unter Berücksichtigung der obigen Anforderungen lässt sich nun eine Relation zwischen Strukturen definieren.

Definition: Seien $\mathcal{A}$,$\mathcal{B}$ und $\mathcal{C}$ Strukturen, die zu Formelmengen $\mathcal{M}, \mathcal{N}$ und O passen und in denen die jeweiligen Formeln gelten, also $\mathcal{A} \models \mathcal{M}, \mathcal{B} \models \mathcal{N}$ und $\mathcal{C} \models \mathcal{O}$. $\mathcal{C}$ heisst genau dann *kohärente Superstruktur* von $\mathcal{A}$ und $\mathcal{B}$ oder $\mathcal{A} \underset{\mathcal{C}}{\circledcirc} \mathcal{B}$, wenn alle folgenden Bedingungen erfüllt sind

- $\mathcal{O}$ ist eine kohärente Obermenge von $\mathcal{M}$ und $\mathcal{N}$,
- $\mathcal{O}$ enthält ein Prädikat *Strukturzuordnung*,
- Zu jedem k-stelligen Prädikat P aus $\mathcal{M}$ oder $\mathcal{N}$ existiert in $\mathcal{O}$ ein k+1-stelliges Prädikat P', für das die Äquivalenz $P'(x_1, \ldots, x_k, a) \leftrightarrow P(x_1, \ldots x_k) \wedge \forall x \in \{x_1, \ldots x_k\} : Strukturzuordnung(x, a)$ gilt,
- $U_\mathcal{A} \cup U_\mathcal{B} \cup \{Id_\mathcal{A}, Id_\mathcal{B}\} \subseteq U_\mathcal{C}$,
- $I_\mathcal{C}(Strukturzuordnung) = \{(x,y) | \; x \in U_\mathcal{A} \cup U_\mathcal{B} \wedge$
  $y \in \{Id_\mathcal{A}, Id_\mathcal{B}\} \wedge$
  $x\, ist\, Element\, des\, Universums\, von\, y\}$
- jede Formel, die in $\mathcal{A}$ oder $\mathcal{B}$ erfüllt ist, ist auch in $\mathcal{C}$ erfüllt,

- es gibt mindestens eine Formel in $\mathcal{O}$, deren Interpretation in $\mathcal{C}$ eine nichtleere Relation zwischen Individuen aus $\mathcal{A}$ und Individuen aus $\mathcal{B}$ darstellt.

Die letzte Bedingung besagt umgangssprachlich, dass zwei Strukturen noch nicht zwingend zusammenhängen, wenn sie Modell einer kohärenten Formelmenge sind. Zusammenhängend sind sie erst dann, wenn die Struktur $\mathcal{C}$ Tupel aus Elementen enthält, die tatsächlich unterschiedliedlichen Ausgangsstrukturen zuzuordnen sind. Sei F eine Formel und sei $V(F)$ die Menge der Variablen von F. Dann verknüpft F die Strukturen $\mathcal{A}$ und $\mathcal{B}$ in $\mathcal{C}$, falls $\mathcal{C}$ ein Modell für F ist und für mindestens ein Paar $(x, y) \in V(F) \times V(F)$ gilt $(I_\mathcal{C}(x), I_\mathcal{C}(a)) \in I_\mathcal{C}(Strukturzuordnung) \wedge (I_\mathcal{C}(y), I_\mathcal{C}(b)) \in I_\mathcal{C}(Strukturzuordnung) \wedge I_\mathcal{C}(a) \neq I_\mathcal{C}(b)$. Eine solche artefaktübergreifende Relation nennen wir im Weiteren *Kohärenzrelation*.

Die oben für zwei Ausgangsstrukturen beschriebene Definition einer gemeinsamen Superstruktur lässt sich wie folgt für eine gemeinsame Superstruktur $\mathcal{C}$ einer beliebigen Menge von Ausgangsstrukturen $\mathcal{A}_1, \ldots, A_n$ verallgemeinern. Seien $\mathcal{A}_1, \ldots, A_n$ und $\mathcal{C}$ Strukturen, die zu Formelmengen $\mathcal{M}_1, \ldots, \mathcal{M}_n$ und $\mathcal{O}$ passen und in denen die jeweiligen Formeln gelten, also $\mathcal{A}_i \models \mathcal{M}_i$ und $\mathcal{C} \models \mathcal{O}$. $\mathcal{C}$ heisst genau dann *kohärente Superstruktur* von $\mathcal{A}_1, \ldots, \mathcal{A}_n$, wenn alle folgenden Bedingungen erfüllt sind:

- $\mathcal{O}$ ist eine kohärente Obermenge der zu $\mathcal{A}_1, \ldots, \mathcal{A}_n$ gehörigen Formelmengen $\mathcal{M}_1, \ldots, \mathcal{M}_n$.
- $\mathcal{O}$ enthält ein Prädikat $Strukturzuordnung$,
- Zu jedem k-stelligen Prädikat P aus $\mathcal{M}_1, \ldots, \mathcal{M}_n$ existiert in $\mathcal{O}$ ein k+1-stelliges Prädikat P', für das die Äquivalenz $P'(x_1, \ldots, x_k, a) \leftrightarrow P(x_1, \ldots x_k) \wedge \forall x \in \{x_1, \ldots x_k\} : Strukturzuordnung(x, a)$ gilt,
- $\bigcup_{i=1}^{n} U_{\mathcal{A}_i} \cup \{Id_{\mathcal{A}_1}, \ldots, Id_{\mathcal{A}_n}\} \subseteq U_\mathcal{C}$,
- $$I_\mathcal{C}(Strukturzuordnung) = \{(x, y) | \quad x \in \bigcup_{i=1}^{n} U_{\mathcal{A}_i} \wedge \; y \in \{Id_{\mathcal{A}_1}, \ldots, Id_{\mathcal{A}_n}\} \wedge \; x\, ist\, Element\, des\, Universums\, von\, y\}$$
- jede Formel, die in $\mathcal{A}_1, \ldots, \mathcal{A}_n$ erfüllt ist, ist auch in $\mathcal{C}$ erfüllt,
- es gibt Formeln in $\mathcal{O}$, deren Interpretation in $\mathcal{C}$ eine oder mehrere nichtleere Relationen definiert, so dass jedes $\mathcal{A}_i$ mit jedem $\mathcal{A}_j$ direkt oder transitiv über diese Relationen verknüpft ist.

Der letzte Punkt unterscheidet sich am stärksten vom binären Fall, also der Verknüpfung zweier Strukturen. Eine Superstruktur für n Ausgangsstrukturen gilt anschaulich dann als zusammenhängend, wenn der mit den $\mathcal{A}_i$ als Knoten und den Relationen, die Individuen unterschiedlicher $\mathcal{A}_i$ verbinden, als Kanten gebildete Graph zusammenhängend ist. Dies kann durch Relationen erreicht werden, die jeweils Individuen zweier Ausgangsstrukturen miteinander verknüpfen und durch die sich transitiv Verknüpfungen jeder Ausgangsstruktur mit jeder anderen ergeben oder durch Relationen, die direkt Individuen mehrerer Ausgangsstrukturen verknüpfen.

2.4 Semantische Kohärenz von Artefakten der Softwareentwicklung

Die formalen Grundlagen der vorigen beiden Abschnitte werden im Folgenden auf das Anwendungsgebiet der Softwareentwicklung übertragen.

Ein Artefakt ist eine Entität, die während der Entwicklung eines Softwareprodukts vom Softwareentwickler manuell oder mit Hilfe von Werkzeugen hergestellt wird. Ein Artefakttyp beschreibt gemeinsame Eigenschaften, die für eine Menge von Artefakten gelten sollen, unter anderem

- welches die atomaren Elemente sind, aus denen ein Artefakt aufgebaut sein kann,
- nach welchen Regeln atomare Elemente zu komplexeren Elemente kombiniert werden können,
- welche Bedingungen ein Artefakt erfüllen muss, damit es als wohlgeformt gilt.

Entsprechend gilt eine Entität genau dann als Artefakt eines Artefakttyps, wenn sie

- ausschließlich aus zulässigen atomaren Elementen des Artefakttyps aufgebaut ist,
- ihre atomaren Elemente gemäß der Regeln des Artefakttyps kombiniert wurden,
- sie alle Bedingungen erfüllt, welche die Wohlgeformtheit eines Artefakts beschreiben.

Soll die Wohlgeformtheit eines Artefakts formal überprüfbar sein, dann müssen

- die für die Prüfung erforderlichen Regeln des Artefakttyps auf Formeln eines formalen Systems abgebildet werden,

- die Eigenschaften des Artefakts auf eine Interpretation dieser Formeln des Artefakttyps abgebildet werden.

Anschließend kann für das Artefakt geprüft werden, ob die dem Artefakttyp entsprechenden Formeln in der dem Artefakt entsprechenden Interpretation gültig sind. Wäre das für die Prüfung gewählte formale System die Prädikatenlogik erster Stufe, dann

- entspräche der Artefakttyp einer Menge von prädikatenlogischen Formeln,
- entspräche ein konkretes Artefakt einer Struktur $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$.

Die Elemente des Artefakts beziehungsweise deren Identifikatoren bilden das Universum $U_\mathcal{A}$. Zu jeder für die Prüfung relevanten Artefakteigenschaft existiert ein Prädikatsymbol. Ist das Artefakt beispielsweise ein UML-Zustandsdiagramm, enthielte der Artefakttyp Prädikate wie ZUSTAND, TRANSITION, TRIGGER, AKTION usw. Durch $I_\mathcal{A}$ wird jedes dieser Prädikate mit einer Interpretation versehen, also mit einer Menge von aus $U_\mathcal{A}$ gebildeten Elementen, die das betreffende Prädikat im konkreten Artefakt erfüllen. Regeln werden als Formeln dargestellt z.B.

$$\forall x, y : ZUSTAND(x) \wedge ZUSTAND(y) \wedge name(x) = name(y) \rightarrow x = y$$

Das Beispiel zeigt mit *name* auch die Verwendung einer Funktion. Während die Interpretation eines Prädikatsymbols in einer konkreten Struktur die Zugehörigkeit von Elementen eines Universums zu einer bestimmten Menge ausdrückt oder bei mehrstelligen Prädikaten die Zugehörigkeit von Tupeln zu einer Relation, stellt die Interpretation einer Funktion eine Abbildung zwischen Elementen des Universums dar. Im Beispiel liefert die Interpretation von *name* zu einem Identifikator eines Diagrammelements den Namen des Diagrammelements. Funktionen werden sich auch bei der Bildung kohärenter Strukturen als nützlich erweisen, die artefaktübergreifende Zusammenhänge beschreiben. Mit ihrer Hilfe können Elemente der Ausgangsuniversen so auf Elemente des Universums einer gemeinsamen Struktur abgebildet werden, dass Vergleiche zwischen Elementen unterschiedlicher Artefakte möglich werden.

Ausgehend von den Definitionen der vorigen Abschnitte lässt sich nun der Begriff der semantischen Kohärenz zwischen Artefakten der Softwareentwicklung wie folgt definieren:

Definition Seien A und B Artefakte der Softwareentwicklung. Dann heißen A und B genau dann *semantisch kohärent*, wenn gilt:

- zu jedem Artefakt existiert ein Artefakttyp, der sich als Formelmenge $\mathcal{AT}_A$,$\mathcal{AT}_B$ in einem einheitlichen formalen System repräsentieren lässt und
- jedes Artefakt lässt sich in eine zu seinem Artefakttyp passende Struktur $\mathcal{S}_A, \mathcal{S}_B$ überführen, in der alle Formeln des Artefakttyps erfüllt sind, also $\mathcal{S}_A \models \mathcal{AT}_A$,$\mathcal{S}_B \models \mathcal{AT}_B$ und
- es existiert eine kohärente Superstruktur $\mathcal{S}_C$ zu $\mathcal{S}_A$und $\mathcal{S}_B$, also $S_A \underset{S_C}{\circledcirc} \mathcal{S}_B$.

S_C ist dabei Modell für eine kohärente Obermenge von $\mathcal{AT}_A$ und $\mathcal{AT}_B$, also $\mathcal{AT}_A \underset{\mathcal{KT}_C}{\otimes} \mathcal{AT}_B$ und $\mathcal{AT}_A \cup \mathcal{AT}_B \subseteq \mathcal{KT}_C$ und $\mathcal{S}_C \models \mathcal{KT}_C$. Ähnlich, wie $\mathcal{AT}_A$ und $\mathcal{AT}_B$ Artefakttypen repräsentieren, repräsentiert $\mathcal{KT}_C$ einen *Kohärenzbeziehungstyp*. Ein Tupel aus zwei oder mehr Artefakten wird im Folgenden als *Artefaktcluster* bezeichnet.

Definition Sei $AC = (A_1, \ldots, A_n)$ ein Artefaktcluster. AC heißt entsprechend genau dann *semantisch kohärent*, wenn gilt:

- zu jedem Artefakt existiert ein Artefakttyp, der sich als Formelmenge $\mathcal{AT}_{A_1}$, $\ldots$, $\mathcal{AT}_{A_n}$ in einem einheitlichen formalen System repräsentieren lässt und
- jedes Artefakt lässt sich in eine zu seinem Artefakttyp passende Struktur $\mathcal{S}_{A_1}, \ldots, \mathcal{S}_{A_n}$ überführen, in der alle Formeln des Artefakttyps erfüllt sind, also $\mathcal{S}_{A_1} \models \mathcal{AT}_{A_1}, \ldots, \mathcal{S}_{A_n} \models \mathcal{AT}_{A_n}$und
- zu $A_1, \ldots, A_n$ existiert eine kohärente Superstruktur.

Grafisch interpertiert ist der Cluster semantisch kohärent, wenn der mit den Artefakten als Knoten und aus den Kohärenzrelationen der kohärenten Superstruktur als Kanten gebildete Graph zusammenhängend ist. Während die Bezeichnung einer kohärenten Superstrukutur als *semantisch kohärent* redundant wäre, da kohärente Superstrukturen gerade als solche Strukturen definiert wurden, welche die semantische Kohärenz einer Formelmenge beweisen, ist die Bezeichnung eines Artefaktclusters als *semantisch kohärent* sinnvoll, um eindeutig zu kennzeichnen, dass die Kriterien für den Zusammenhalt und die Integrität des Clusters nicht nur formal spezifiziert sind (syntaktische Kohärenz), sondern, dass der Cluster diese Kriterien auch erfüllt (semantische Kohärenz).

3 Spezifikation und Nutzung von Modell-Suiten

Artefaktübergreifende Integritätsbedingungen wurden im vorigen Abschnitt als Sätze einer Logik beschrieben, die in einem Tupel aus mehreren Artefakten erfüllt sein sollen. Für Artefakte eines konkreten Artefaktclusters kann dann überprüft werden, ob sie die für diesen Cluster definierten Integritätsbedingungen erfüllen. Eine solche "Gültigkeitsprüfung" liefert dem Softwareentwickler Hinweise auf die derzeitige "Qualität" seiner Artefakte, sie bietet aber noch keine Unterstützung bei der Erstellung neuer Artefakte beziehungsweise neuer Artefaktcluster. Auch bei der Änderung bestehender Artefakte lässt sich mit den bisher beschriebenen Mitteln lediglich feststellen, ob ein kohärenter Artefaktcluster nach Änderung eines Artefakts des Clusters immer noch kohärent ist, nicht aber, welche Maßnahmen auszuführen sind, falls der Artefaktcluster durch die Änderung eines Artefakts nicht mehr kohärent wäre. Zur Lösung dieser Aufgabe wird im folgenden das Konzept der Modell-Suiten vorgestellt.

3.1 Artefaktcluster als Bestandteil von Modell-Suiten

Eine explizite Unterscheidung zwischen der Spezifikation von Kohärenz, ihrer Überwachung und ihrer Durchsetzung liefert folgende Beschreibung aus [17]. Dort besteht eine so genannte Modell-Suite aus

- einer Menge von Modellen $\{M_1, \ldots, M_n\}$,
- einem Schema für Assoziationen und Kollaborationen zwischen den Modellen,
- Controllern, welche für die Konsistenz oder Kohärenz der Modell-Suite sorgen,
- Bearbeitungsschemata, welche explizit die Bearbeitung und Weiterentwicklung der Modell-Suite beschreiben, und
- Tracern, mit denen sich die Herstellung von Kohärenz verfolgen lässt.

Ein Modell M wird dort als Struktur beschreiben,

- die auf einem Ausdruck in einer Sprache $\mathcal{L}_{\mathbb{S}}$ basiert, welcher die für die Sprache definierten Wohlgeformtheitsbedingungen $\sum_{\mathcal{L}_{\mathbb{S}}}$ erfüllt und
- die eine Menge von in der zur Sprache gehörigen Logik definierten Integritätsbedingungen $\sum_M$ erfüllt.

$\mathbb{S}$ ist die Signatur von $\mathcal{L}_{\mathbb{S}}$, also die Menge der Konstanten, Funktionssymbole und Relationssymbole beziehungsweise Prädikate, aus denen sich Ausdrücke in $\mathcal{L}_{\mathbb{S}}$ bilden lassen. Diese Definition des Modellbegriffs ist mit dem Artefaktbegriff der vorigen Abschnitte kompatibel, wenn man die artefaktbezogene Terminologie wie folgt deutet:

- Sowohl ein Artefakt als auch ein Modell sind mathematische Strukturen beziehungsweise lassen sich auf solche abbilden.
- Sowohl Artefakt als auch Modell besitzen eine Signatur. Im Falle des Artefaktbegriffs wurde unterstellt, dass die Signatur aus Symbolen besteht, die sich direkt als Bestandteile prädikatenlogischer Formeln interpretieren lassen. Beim Modellbegriff in [17] wird diese frühzeitige Festlegung auf eine bestimmte Logik vermieden.
- Sowohl Artefakte als auch Modelle besitzen einen Typ, also ein Tupel bestehend aus einer Sprache $\mathcal{L}_{\mathbb{S}}$ und einem Satz von Regeln $\sum_{\mathcal{L}_{\mathbb{S}}}$, nach denen die für diesen Typ spezifischen Aussagen dieser Sprache gebildet werden.
- Sowohl Artefakte als auch Modelle müssen Integritätsbedingungen erfüllen, die als Aussagen der Logik der entsprechenden Sprache formuliert sind.

Ein Artefaktcluster ließe sich entsprechend als die Menge der Modelle einer Modell-Suite auffassen. Die Menge der Kohärenzbeziehungstypen eines semantisch kohärenten Artefaktclusters entspräche dem Assoziationsschema einer Modell-Suite. Jeder Kohärenzbeziehungstyp definiert eine Menge von Bedingungen, welche die an der Beziehung beteiligten Artefakte erfüllen müssen, damit sie als kohärent gelten. Das Ergebnis der für Modell-Suiten eingeführten Operation zur Verknüpfung zweier Modelle $\boxtimes$ ist dabei vergleichbar mit der in Abschnitt 2.3 und 2.4 beschriebenen kohärenten Superstruktur von Artefakten.

Der für Modell-Suiten eingeführte Begriff des Kollaborationsschemas geht über die reine Spezifikation von Assoziationen zwischen Modellen und modellübergreifenden Bedingungen hinaus. Kollaboration zwischen Modellen beinhaltet neben der Spezifikation gültiger Zustände auch explizite Spezifikationen von ungültigen Zuständen und von Maßnahmen zu deren Korrektur oder Vermeidung. In [17] wird dazu der Begriff des Vertrages (contract) eingeführt. Ein Vertrag umfasst dort:

- eine Menge von Integritätsbedingungen,
- eine Beschreibung eines Mechanismus zu deren Durchsetzung,

- eine Beschreibung von Änderungsoperationen, durch die eine Modell-Suite ausgehend von einem kohärenten Startzustand in einen kohärenten Folgezustand überführt wird.

Übertragen auf die Bestandteile einer Modell-Suite

- sind die Integritätsbedingungen des Vertrages Bestandteil des Kollaborationsschemas der Modell-Suite,
- spezifiziert die Beschreibung des Durchsetzungsmechanismus des Vertrages das Verhalten von Controllern der Modell-Suite,
- sind die Beschreibungen kohärenter Änderungsoperationen Teil des Bearbeitungsschemas der Modell-Suite.

Als Tracer der Modell-Suite können die Daten aufgefasst werden, die bei der Erstellung und Änderung von Modellen entstehen und zur Bestimmung und gegebenenfalls Wiederherstellung der Kohärenz der Modell-Suite genutzt werden. Zwei Modelle eine Modell-Suite gelten in [17] als kohärent, wenn jede Änderung des einen Modells zu einer korrespondierenden Änderung des anderen Modells führt. Sind die Beschreibungen der kohärenzerhaltenden Änderungsoperationen eindeutig, lassen sich Änderungen eines Modells maschinell auf weitere Modelle übertragen. Sind unterschiedliche Operationen zur Wiederherstellung der Kohärenz nach Änderung eines Modells möglich, ist die Mitarbeit des Softwareentwicklers nötig. Dieser muss dann entweder manuell eine Operation aus der Liste der möglichen kohärenzerhaltenden Operationen auswählen oder er muss bei der Erzeugung der Modell-Suite bereits ein Bearbeitungsschema wählen, das solche Wahlmöglichkeiten ausschließt.

Um einen kohärenten Artefaktcluster zu einer Modell-Suite zu vervollständigen, müssten also die dem Artefaktcluster zu Grunde liegenden Kohärenzbeziehungstypen zu artefaktübergreifenden Verträgen erweitert werden. Ein solcher Vertrag darf nicht nur die Bedingungen enthalten, die erfüllt sein müssen, damit der Artefaktcluster als kohärent erkannt wird, sondern er muss auch Vorschriften enthalten, mit deren Hilfe sich Kohärenzverletzungen beheben lassen. Enthielte die für einen Artefaktcluster definierte Formelmenge eine Äquivalenz wie

$$R(a,b) \leftrightarrow \exists x, y : P(x,a) \wedge Q(y,b) \wedge f(x) = g(y) \wedge a \neq b$$

bei der a und b Artefakte bezeichnen. Diese ließe sich in zwei Implikationen zerlegen:

1. $(\exists x, y : P(x,a) \wedge Q(y,b) \wedge f(x) = g(y) \wedge a \neq b) \rightarrow R(a,b)$

2. $R(a,b) \rightarrow \exists x,y : P(x,a) \wedge Q(y,b) \wedge f(x) = g(y) \wedge a \neq b$

Ist der Artefaktcluster aus a und b kohärent, müssen beide Formeln erfüllt sein. Die Extension von P wird durch das Werkzeug geliefert, mit dem a erstellt wurde, die Extension von Q wird durch das Werkzeug geliefert, mit dem b erstellt wurde. Die Interpretation von f und g kann auch noch vom jeweiligen Werkzeug geliefert werden. Aber die Interpretation von R kann keines der beteiligten Werkzeuge liefern. Das heißt, der Mechanismus zur Kohärenzprüfung muss die Interpretation von R entweder aus den Daten der einzelnen Artefakte ableiten oder der Softwareentwickler muss diese Interpretation explizit vorgeben. Im ersten Fall lässt sich Formel 1) als Konstruktionsvorschrift für die Interpretation von R deuten, wenn man die linke Seite als Vorbedingung für die Erzeugung der rechten Seite auffasst, also "falls linke Seite erfüllt für ein Paar (a,b), dann füge der Menge R das Paar (a,b) hinzu". In dieser Leserichtung ließe sich also prüfen, ob eine gegebenes Artefaktpaar das Prädikat R erfüllt. Gibt der Softwareentwickler R selber vor, dann lässt sich Formel 2) als Anforderung interpretieren, die er bei der Erzeugung seiner Artefakte berücksichtigen muss, also "wenn die linke Seite gelten soll, dann müssen die beteiligten Artefakte auch die rechte Seite erfüllen". Aus dieser Leserichtung lassen sich zugehörige Änderungsoperationen ableiten:

- falls die Menge $P(x,a)$ leer ist, füge mindestens ein (x,a) zu P hinzu,
- falls die Menge $Q(y,b)$ leer ist, füge mindestens ein (y,b) zu Q hinzu,
- falls für kein Paar (x,y) gilt $f(x) = g(y)$, dann wähle ein Paar und ändere entweder x oder y so, dass $f(x) = g(y)$ gilt."

Für einen menschlichen Softwareentwickler könnten daraus schon Hinweise für folgende Entwicklungsschritte erzeugt werden (z.B. "Füge dem Klassendiagramm eine Klasse hinzu." oder "Der Name der Klasse mindestens eines Objekts im Objektdiagramm muss auch im zugehörigen Klassendiagramm auftauchen."). Soll die Herstellung von Kohärenz ebenfalls maschinell erfolgen, müssen nicht-deterministische Regeln wie "*Auswahl* eines Paares", "Ändere x *oder* y" durch deterministische Regeln ersetzt werden.

Während Kohärenzregeln in den bisherigen Beispielen überwiegend mit einer Interpretationssemantik versehen wurden, werden sie hier mit einer operationalen Semantik versehen [7]. Eine Regel wird also nicht genutzt, um zu prüfen, ob sie in einer als gegeben vorausgesetzten Interpretation erfüllt ist, sondern als Teil einer Vorschrift, mit deren Hilfe sich ein Zustand herbeiführen lässt, in dem die Regel erfüllt ist.

Als Ausgangspunkt einer Kohärenzprüfung wird der vom jeweiligen Entwicklungswerkzeug erzeugte Artefaktzustand genutzt. Die Kohärenzprüfung wird unabhängig von konkreten Entwicklungswerkzeugen spezifiziert und kann auch unabhängig von diesen durchgeführt werden. Inkonsistenzen werden nicht durch Verbot bestimmter Artefaktänderungen oder durch direkte Synchronisation unterschiedlicher Werkzeuge verhindert, sondern durch Zuordnung der verursachenden Artefaktelemente zu speziellen Prädikaten angezeigt. Dadurch sind auch Kohärenzprüfungen zwischen Artefakten möglich, die zunächst unabhängig entwickelt und erst später zu einem Artefaktcluster zusammengeführt wurden und die initial nicht kohärent sind. Artefaktelemente, die solche Prädikate erfüllen, die auf festgestellte Inkohärenzen hindeuten, kann der Softwareentwickler dann nach Ablauf der Prüfung manuell korrigieren oder durch Entwicklungswerkzeuge korrigieren lassen, die so erweitert wurden, dass sie für Prädikate bestimmter Fehlerfälle geeignete Korrekturoperationen vorschlagen und ausführen können.

3.2 Spezifikation und Nutzung von Modell-Suiten

Um eine Menge von Modellen wie im Beispiel aus Abbildung 2 zu einer Modell-Suite zusammenzufügen, sind folgende Schritte erforderlich:

- Definition bzw. Auswahl eines Assoziations- und Kollaborationsschemas,
- Definition bzw. Auswahl eines Bearbeitungsschemas für die konsistente Weiterentwicklung der Modell-Suite,
- Bereitstellung und Nutzung von Controllern, welche die Einhaltung des durch die Schemata definierten Vertrages gewährleisten,
- Bereitstellung und Nutzung von Tracern, welche eine Verfolgung der Entwicklung erlauben.

Für jedes Modell existiere ein entsprechender Modelltyp $\mathcal{T}_{\mathcal{L}_\mathbb{S}} = (\mathcal{L}_\mathbb{S}, \sum_{\mathcal{L}_\mathbb{S}})$ bestehend aus der basierend auf einer Signatur $\mathbb{S}$ definierten Sprache $\mathcal{L}_\mathbb{S}$ und einer Menge von Integritätsbedingungen $\sum_{\mathcal{L}_\mathbb{S}} \in \mathcal{L}(\sum_\mathbb{S}^{WellFormed})$, welche alle in $\mathcal{L}_\mathbb{S}$ erzeugten Modelle erfüllen müssen. Mit Hilfe der Typen der Einzelmodelle lässt sich nun der Typ der Modell-Suite definieren als Tupel $\mathcal{ST} = (\mathcal{T}_{\mathcal{L}_{\mathbb{S}_1}}, \ldots, \mathcal{T}_{\mathcal{L}_{\mathbb{S}_n}}, \sum_{\mathcal{L}_{\mathbb{S}_1},\ldots,\mathcal{L}_{\mathbb{S}_n}})$. Die Menge $\sum_{\mathcal{L}_{\mathbb{S}_1},\ldots,\mathcal{L}_{\mathbb{S}_n}}$ beinhaltet die modellübergreifenden Integritätsbedingungen, die also nicht Bestandteil eines einzelnen beteiligten Modelltyps sind. Der Typ der Modell-Suite definiert das Assoziationsschema der Modell-Suite.

Seien beispielsweise die Integritätsbedingungen des Modell-Suite Typs durch prädikatenlogische Formeln ausgedrückt und konkrete Modelle seien eine Interpretation

dieser Formeln. Dann bilden diese Modelle eine gültige Modell-Suite, falls alle durch den Typ der Modell-Suite definierten Formeln unter der Interpretation der konkreten Modelle erfüllt sind. Mit einer derartigen Erfüllbarkeitsprüfung lässt sich also feststellen, ob die Modell-Suite in ihrem aktuellen Zustand kohärent ist. Ändert sich eines der beteiligten Modelle, wird die Modell-Suite dadurch potentiell inkohärent. Um dies zu verhindern, müssen zulässige Änderungsoperationen oder Operationen zur Erhaltung der Kohärenz definiert werden. Da in der Sprache der Prädikatenlogik keine Elemente zur Darstellung von Begriffen wie Verpflichtung, Erlaubnis oder Verbot verfügbar sind, bietet sich zur Repräsentation derartiger Vertragsverpflichtungen die Verwendung deontischer Logik an. Abbildung 2 zeigt ein Beispiel einer solchen in deontischer Logik notierten Integritätsbedingung. Falls ein Zustand des Zustandsdiagramms so geändert wird, dass er nicht mehr in der Menge der aus dem Klassendiagramm ableitbaren Zustände enthalten wäre, dann ist die Änderung des Zustandsdiagramms verboten. Derartige Regeln für die konsistente Weiterentwicklung der Modell-Suite sind Bestandteil des Bearbeitungsschemas der Modell-Suite.

Um das Bearbeitungsschema anzuwenden, müssen Änderungen der beteiligten Modelle bemerkt werden (Tracer) und sie müssen zwischen den Modellen kommuniziert werden. Das Kollaborationsschema legt dazu fest, auf welche Art Änderungen kommuniziert werden (z.B. Master/Slave, Publish/Subscribe) und auf welche Weise eine koordinierte Änderung innerhalb der Modell-Suite abzulaufen hat (z.B. synchron/asynchron, transaktional).

Die Tracer einer Modell-Suite sind in einer konkreten Entwicklungsumgebung Softwarekomponenten, welche Änderungen an beteiligten Modellen registrieren und an andere Komponenten (Controller) kommunizieren könnnen. Die Controller einer Modell-Suite sind Softwarekomponenten, welche das Assoziations- und Bearbeitungsschema der Modell-Suite auf deren Teilmodelle anwenden.

Die Bestandteile einer Modell-Suite insbesondere ihre Schemata, Tracer und Controller müssen nicht für jede konkrete Modell-Suite erneut definiert werden. Tracer werden einmalig pro Modelltyp definiert. Controller können generisch für die zur Schemadefinition genutzte Sprache entwickelt werden und lassen sich danach mit konkreten Assoziations-, Bearbeitungs- und Kollaborationsschemata parametrisieren. Assoziations-, Bearbeitungs- und Kollaborationsschemata sind pro Modell-Suite Typ zu spezifizieren, können danach aber, wie der gesamte Modell-Suite Typ wiederverwendet werden.

4 Werkzeugunterstützung

Die praktische Spezifikation und Nutzung von Modell-Suiten in realen Arbeitsumgebungen erfordert sowohl Anpassungen der bereits existierenden Entwicklungswerkzeuge als auch der Art und Weise, in der diese Werkzeuge genutzt werden,

4.1 Veränderung herkömmlicher Arbeitsumgebungen

Wir gehen davon aus, dass eine Arbeitsumgebung zur Softwareentwicklung niemals aus einem einzigen Werkzeug besteht, mit dem sich alle Produkte des Softwareentwicklungsprozesses bearbeiten lassen. Eine solche Arbeitsumgebung besteht immer aus mehreren Werkzeugen, die jeweils optimal entweder an die Bearbeitung bestimmter Artefakte oder an die Unterstützung bestimmter Entwicklungsschritte angepasst sind. Als Artefakt werden im Folgenden alle Produkte aufgefasst, die während eines Entwicklungsschrittes hergestellt werden (von lat. arte factum - "durch Kunst Hergestelltes") und in einer maschinell verarbeitbaren Repräsentation vorliegen. Wir unterstellen mit dieser Begriffsbildung, dass Softwareentwicklung kein deterministischer, vollständig automatisierbarer Vorgang ist, sondern letztlich immer das Ergebnis einer kreativen Tätigkeit, nämlich der des Softwareentwicklers. Durch die Auswahl bestimmter Werkzeuge in der Anfangsphase eines Softwareentwicklungsprojekts wird bestimmt, welche Artefakte überhaupt produziert werden können, wie diese Artefakte repräsentiert werden und welche Arten von Beziehungen zwischen Artefakten verschiedener Werkzeuge denkbar sind. Sollen in einer solchen Arbeitsumgebung artefakt- und werkzeugübergreifende Konsistenzprüfungen stattfinden, dann muss die Arbeitsumgebung an diesen Zweck angepasst sein. Die erforderlichen Anpassungen lassen sich grob in

- Anpassungen der Werkzeuge,
- Ergänzungen der Arbeitsumgebung und
- projektspezifische Anpassungen

unterteilen. Werkzeugspezifische Anpassungen erweitern das jeweilige Werkzeug so, dass es Inhalte in einer Repräsentation produzieren kann, die für eine artefaktübergreifende Konsistenzsicherung innerhalb der Arbeitsumgebung geeignet ist. Außerdem wird das Werkzeuge um Funktionalität erweitert, durch die Änderungen eines Artefakts an andere Werkzeuge bzw. an eine artefaktübergreifende Konistenzprüfung kommuiziert werden können. Ein so erweitertes Werkzeug übernimmt damit unter anderem die Rolle eines Tracers einer Modell-Suite.

Eine Ergänzung der Arbeitsumgebung ist die Bereitstellung eines geeigneten Werkzeugs zur Verwaltung und Prüfung artefaktübergreifender Konsistenzbedingungen. Dieses übernimmt die Rolle eines Controllers im Sinne einer Modell-Suite. Die Definition artefaktübergreifender Integritätsbedingungen stellt eine weitere Ergänzung der Arbeitsumgebung dar. Die für eine Arbeitsumgebung getroffene Auswahl an Werkzeugen bestimmt, welche Artefakte produziert werden können und über welche Artefakteigenschaften überhaupt Konsistenzbedingungen formuliert werden können. Einmal konfiguriert ist eine solche Arbeitsumgebung aus einer konkreten Auswahl an Werkzeugen und einem konkreten Satz an zwischen den Artefakten der Werkzeuge definierten Konsistenzbeziehungen grundsätzlich für mehrere Softwareentwicklungsprojekte einsetzbar. Abhängig vom jeweiligen Projekt kann die Definition weiterer Konsistenzbeziehungen oder eine Änderung bestehender Definitionen von Konsistenzbeziehungen nötig werden. Konsistenzbedigungen können projektspezifisch verschärft, abgeschwächt, neu hinzugefügt oder ganz entfernt werden.

Vor Beginn der Entwicklungsarbeiten in einer Arbeitsumgebung mit artefaktübergreifender Konsistenzprüfung müssen also folgende Schritte stattgefunden haben:

1. Auswahl der zu verwendenden Werkzeuge,
2. Auswahl des Werkzeugs zur Verwaltung und Prüfung der artefaktübergreifenden Konsistenz,
3. Anpassung der unter 1) gewählten Werkzeuge an die Repräsentation des unter 2) gewählten Werkzeugs,
4. Definition der im Projekt zu prüfenden Konsistenzbedingungen.

Nicht alle Schritte sind bei jedem neuen Softwareentwicklungsprojekt zu durchlaufen. Hat sich eine bestimmte Kombination von Entwicklungswerkzeugen (Schritt 1) bewährt, kann davon ausgegangen werden, dass sie in mehr als einem Projekt zum Einsatz kommt. Auch die Festlegung auf ein bestimmtes Verfahren zur Konsistenzprüfung (Schritt 2), zum Beispiel auf Basis von Abstract State Machines [1], kann über mehrere Projekte beibehalten werden. Anpassungen eines Werkzeugs (Schritt 3) sind nur dann notwendig, wenn das betreffende Werkzeug die für die Konsistenzprüfung erforderliche Repräsentation nicht erzeugen kann. In Folgeprojekten werden solche Anpassungen des Werkzeugs nur dann noch erneut nötig, falls sich die zur Konsistenzprüfung genutzte Repräsentation im Folgeprojekt geändert hat oder gegen einen völlig andere ausgetauscht worden ist. Ergänzungen oder Veränderungen bestehender Konsistenzbedingungen (Schritt 4) sind in Folgeprojekten nur dann

notwendig, wenn neue Artefakttypen zur Arbeitsumgebung hinzukommen, deren Eigenschaften von den bisher definierten Konsistenzbedingungen nicht erfasst werden. Der Austausch eines Entwicklungswerkzeugs durch ein anderes, das Artefakte desselben Typs produziert, führt nicht zu einer Änderung bestehender Definitionen von Konsistenzbedingungen. Die für die Konsistenzprüfung genutzte Repräsentation von Artefakten dieses Typs ist unabhängig vom Werkzeug, mit dem das Artefakt produziert wird. Kommt in unterschiedlichen Softwareentwicklungsprojekten dieselbe Arbeitsumgebung oder zumindest eine Arbeitsumgebung auf Basis derselben Repräsentation von Konsistenzbeziehungen zum Einsatz, so besteht der Zusatzaufwand in der Anfangsphase des Projekts verglichen mit Projekten ohne diese Form der Integritätssicherung lediglich in der Auswahl und gegebenenfalls der Definition projektspezifischer Konsistenzbedingungen. Hat sich die in Schritt 2 gewählte Form der Konsistenzprüfung bewährt, so können die Schritte 1 und 2 in späteren Entwicklungsprojekten auch vertauscht sein. Das heißt, ausgehend von der gewählten Methode der Konsistenzsicherung werden gezielt solche Werkzeuge für die Arbeitsumgebung ausgewählt, welche die gewählte Form der Konsistenzsicherung bereits unterstützen.

Alle Daten, die aus Produkten bestehender Werkzeuge abgeleitet werden (z.B. die zur Konsistenzprüfung verwendete Repräsentation von Artefakten) oder die artefaktübergreifende Eigenschaften einer Modell-Suite beschreiben (z.B. Assoziations- und Bearbeitungsschema) werden von einer zentralen Instanz verwaltet, die allen beteiligten Werkzeugen zugänglich ist. Spezielle Content Management Systeme wie sie unter anderem in [14, 2] entwickelt werden, können als eine solche zentrale Instanz fungieren. Neben der Speicherung abgeleiteter und übergreifender Inhalte gestatten sie auch die Annotierung von Inhalten mit Schlagwörtern, die den Verwendungszweck der Inhalte näher beschreiben, und eine Verwaltung von Verweisen auf die ursprünglichen Artefakte, aus denen die im Content Management System verwalteten Daten abgeleitet sind. Inhalte, die der Entwickler als *wert*voll für seine Tätigkeit erachtet und die mit entsprechenden Schlagworten zur Kennzeichnung dieses Werts versehen sind, werden in der Terminologie derartiger begriffsorientierter Inhaltsverwaltungssysteme als *Assets* bezeichnet. Eine Arbeitsumgebung zur Softwareentwicklung, die mit Hilfe eines derartigen zentralen Verwaltungssystems für die Bearbeitung von Modell-Suiten angepasst wurde, folgt damit der in Abbildung 3 gezeigen Architektur. Bestehende Modellierungs- und Entwicklungswerkzeuge kommuizieren dabei mit einem zentralen Dienst, über den sowohl Verweise auf Modelldaten, deren Verschlagwortung- und Aggregation (Asset Service) als auch Spezifikation und Anwendung von Schemata einer Modell-Suite (Coherence Service) abgewickelt

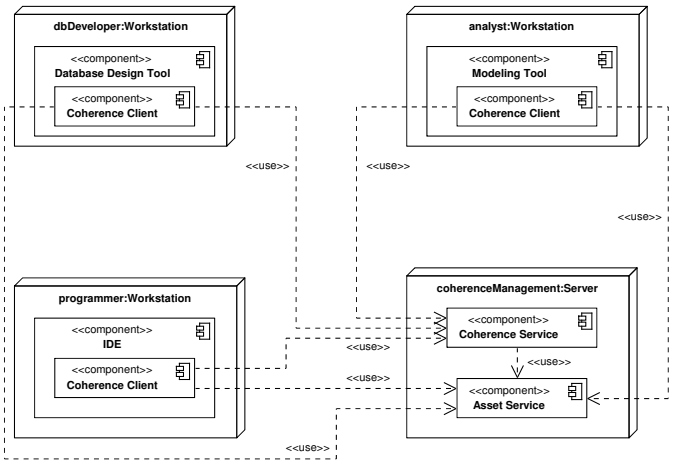

Abbildung 3: Komponenten einer integrierten Arbeitsumgebung

werden.

4.1.1 Anpassungen von Werkzeugen

Wird ein Softwareentwicklungswerkzeug erstmalig einer Arbeitsumgebung hinzugefügt, in welcher der Zusammenhalt von Artefakten unterschiedlicher Werkzeuge dokumentiert und deren Konsistenz überprüft werden soll, muss es für den Gebrauch in dieser Umgebung angepasst werden. Die erforderlichen Anpassungen des Werkzeugs werden unter dem Dach einer speziellen Komponente, des Coherence Clients, zusammengefasst. Wie die Funktionalität eines Coherence Clients implementiert wird, hängt von folgenden Faktoren ab:

- Identifizierbarkeit von Artefakten des Entwicklungswerkzeugs,
- Übersetzbarkeit von Artefakteigenschaften in eine für die Konsistenzprüfung geeignete Repräsentation,
- Erkennbarkeit von relevanten Artefaktänderungen.

Um Konsistenzbeziehungen zwischen Artefakten beschreiben zu können, müssen sich die beteiligten Artefakte identifizieren lassen. Um ein Artefakt überhaupt mit einem Werkzeug bearbeiten zu können, muss der Benutzer angeben können, welches Artefakt er bearbeiten möchte. Er übergibt dem Werkzeug also entweder einen Identifikator für ein bestehendes Artefakt oder er erzeugt ein neues Artefakt, während dessen Erstellung dann ein Identifikator vergeben wird. Dieser Identifikator kann ein Dateiname, ein Pfad innerhalb eines Dateisystems, ein Schlüsselwert einer Datenbanktabelle und ähnliches mehr sein. Es kann davon ausgegangen werden, dass jedes Werkzeug, mit dem sich Artefakte bearbeiten lassen, auch einen Mechanismus zur

Identifikation von Artefakten implementiert. Für das Entwicklungswerkzeug selber reicht es aus, wenn die Identifikation innerhalb seines spezifischen Arbeitsbereichs eindeutig ist, beispielsweise innerhalb des Dateisystem-Ordners, in dem das Entwicklungswerkzeug seine Artefakte ablegt. Für die Beschreibung einer werkzeugübergreifenden Konsistenzbeziehung reicht diese Form der Identifikation noch nicht aus. Innerhalb der globalen Arbeitsumgebung können Identifikatoren mehrfach auftreten, die innerhalb der lokalen Umgebung des jeweiligen Werkzeugs bereits eindeutig wären. Ein Beispiel wären zwei unterschiedliche Werkzeuge, die in ihrem jeweiligen Arbeitsbereich ihre Artefakte nach demselben Schema bennen und deren Artefakte nur anhand der unterschiedlichen Arbeitsbereiche der beiden Werkzeuge unterschieden werden könnten. Falls also die Werkzeuge einer Arbeitsumgebung noch keine artefakt- und werkzeugübergreifend eindeutigen Identifikatoren erzeugen können, muss für die Arbeitsumgebung ein entsprechend eindeutiges Identifikationsschema definiert werden. Jeder Coherence Client wird dann mit der nötigen Funktionalität versehen, um Abbildungen zwischen der Werkzeug-lokalen Identifikation und der globalen Identifikation der Arbeitsumgebung zu gewährleisten. Dieser innerhalb der Arbeitsumgebung eindeutigen Identifikatoren werden dann an den Asset Service der Arbeitsumgebung übermittelt.

Sollen innerhalb einer Arbeitsumgebung nicht nur artefaktübergreifende Beziehungen definiert, sondern auch Konsistenzprüfungen auf Basis dieser Beziehungen durchgeführt werden, müssen die an einer Konsistenzprüfung beteiligten Artefakte in einer Form beschrieben sein, die für eine solche Konsistenzprüfung geeignet ist. Beispielsweise lässt sich die zu prüfende Integritätsbedingung als Regel einer Abtract State Machine (ASM) formulieren. Falls ein Werkzeug seine Artefakte noch nicht in der für Konsistenzprüfungen genutzten Sprache darstellen kann, muss sein Coherence Client entsprechende Funktionalität zur Erzeugung einer solchen Repräsentation erhalten. Im Falle der ASM-basierten Konsistenzprüfung würde der Coherence Client dann Artefakteigenschaften beispielsweise als Wertbelegungen dynamischer ASM-Funktionen darstellen. Die Ausführung der Prüfung entspräche einem konkreten Lauf der ASM. Das Ergebnis der Prüfung ergäbe sich dann aus dem Endzustand der ASM.

Ein Entwicklungswerkzeug, das Artefakte in der oben beschriebenen Weise identifizieren und repräsentieren soll, muss auch relevante Veränderungen eines Artefakts erkennen können. Wird ein Artefakt mit dem Werkzeug bearbeitet, ändert sich möglicherweise auch die zur Konsistenzprüfung genutzte Repräsentation des Artefakts oder gar seine Identifikation. Der Coherence Client muss also die Konsistenz zwischen der "internen" Artefaktrepräsentation des Werkzeugs und der für

die "externe" Konsistenzprüfung genutzten Artefaktrepräsentation aufrechterhalten. Um solche Änderungen zu erkennen, müssen Ereignisse definiert werden, die der Coherence Client des Werkzeugs erkennen kann, beispielsweise das Speichern eines Artefakts oder das Verlassen eines Bearbeitungsdialogs. Bei Auftreten eines solchen Ereignisses synchronisiert dann der Coherence Client die Artefaktrepräsentation des Werkzeugs mit der werkzeugunabhängigen Artefaktrepräsentation der Arbeitsumgebung.

Die Erweiterung eines bestehenden Entwicklungswerkzeugs um einen Coherence Client setzt voraus, dass das jeweilige Entwicklungswerkzeug geeignete Schnittstellen bereitstellt, um die erforderlichen Daten zu gewinnen, die für eine einheitliche Identifikation und Repräsentation erforderlich sind. Entwicklungswerkzeuge, die auf einer offenen Plattform wie beispielsweise Eclipse basieren, erlauben das Hinzufügen neuer Funktionalität über so genannte PlugIns. Diese Module können Funktionalität anderer Module mitnutzen, sich also beispielsweise als Beobachter von Änderungsereignissen registrieren (vgl. Beobachter-Entwurfsmuster in [5]) und selber neue Funktionalität zur Plattform hinzufügen.

Kann ein Entwicklungswerkzeug die für den Coherence Client erforderlichen Daten nicht über interne Schnittstellen liefern, so lässt sich keine direkte Kommunikation zwischen Entwicklungswerkzeug und Coherence Service implementieren. Falls aber die Repräsentation dokumentiert ist, in der die mit dem Werkzeug erzeugten Artefakte gespeichert werden, lässt sich ein aus der Sicht des Werkzeugs externer Coherence Client implementieren. Dieser wäre ein eigenständiger Konverter, der unabhängig vom ursprünglichen Werkzeug arbeitet und der die werkzeugspezifische Repräsentation interpretiert und daraus eine für die Konsistenzprüfung geeignete Repräsentation erzeugt. Speichert beispielsweise das Werkzeug Artefakte als XML-Dateien und ist das Schema oder die Dokumententypdefiniton der XML-Dateien bekannt, können relevante Inhalte mit Hilfe eines XML-Parsers ausgelesen und in Elemente einer ASM-Spezifikation transformiert werden. Die Erkennung relevanter Artefaktänderungen würde in diesem Fall nicht über einen Benachrichtigungsmechanismus des Entwicklungswerkzeugs laufen (Push-Mechanismus), sondern der Konverter müsste selber in regelmäßigen Abständen oder auf Anfrage des Entwicklers überprüfen, ob sich das jeweilige Artefakt verändert hat (Pull-Mechanismus). Verfügt ein Entwicklungswerkzeug weder über dokumentierte Schnittstellen zu seiner Artefaktverarbeitung, noch über eine dokumentierte externe Repräsentation seiner Artefakte, ist es für eine werkzeugübergreifende Konsistenzprüfung nur bedingt geeignet. Der Entwickler könnte die für die Konsistenzprüfung erforderliche Repräsentation auch manuell erzeugen, müsste dann aber auch manuell die Konsistenz zwi-

schen werkzeugspezifischer und werkzeugunabhängiger Artefaktrepräsentation gewährleisten. Da dies einen aus Sicht des Entwicklers unerwünschten, wiederkehrenden Zusatzaufwand verursacht, würde man ein solches Werkzeug vermutlich nicht in einer Arbeitsumgebung verwenden, in der werkzeugübergreifende Konsistenzsicherung betrieben werden soll.

Alle in diesem Abschnitt beschriebenen Anpassungen eines Entwicklungswerkzeugs verursachen lediglich einmalig Entwicklungsaufwand. Entsprechend modifizierte Entwicklungswerkzeuge können danach in jeder Arbeitsumgebung eingesetzt werden, welche dieselbe formale Methode zur Konsistenzsicherung verwendet. Änderungen an bestehenden Coherence Clients werden nur dann nötig, wenn sich die Schnittstellen zur Asset- und Konsistenzverwaltung ändern oder wenn sich die formale Repräsentation von Konsistenzbeziehungen ändert, durch einen anderen Formalismus ersetzt oder um einen zusätzlichen Formalismus ergänzt wird.

4.1.2 Definition von Konsistenzbedingungen

Ein Aufwand, der einmalig für jede Arbeitsumgebung anfällt, ist die Beschreibung der artefaktübergreifenden Integritätsbedingungen, die in der Arbeitsumgebung bzw. für bestimmte Typen von Modell-Suiten gelten sollen. Anzahl, Form und Detailgrad dieser Integritätsbedingungen hängen von mehreren Einflussgrößen ab:

- Anzahl der in der Arbeitsumgebung auftretenden Artefakttypen,
- für die Spezifikation von Konsistenzbedingungen gewählte Sprache,
- der Fähigkeit der verwendeten Werkzeuge, Artefakte in der zur Konsistenzprüfung gewählten Sprache zu repräsentieren.

Die Anzahl der auftretenden Artefakttypen beeinflusst die Anzahl der möglichen Bezie-hungstypen zwischen Artefakten unterschiedlichen Typs. Sie wächst polynomiell mit der Anzahl der vorhandenen Artefakttypen. Im Fall von binären Artefaktbeziehungen wächst die Zahl der möglichen Beziehungstypen beispielsweise quadratisch mit der Zahl der vorhandenen Artefakttypen. In der Praxis wird nur ein Bruchteil aller möglichen Artefaktbeziehungen auch tatsächlich auftreten. Außerdem muss nicht jedes Artefakt in einer direkten, explizit repräsentierten Beziehung zu jedem beliebigen anderen Artefakt des Softwareentwicklungsprozesses stehen. Der Zusammenhalt zweier Artefakte kann sich auch transitiv ergeben. Zu beachten ist hierbei, dass die in dieser Arbeit vorgestellte Form der Kohärenzprüfung nur für Artefaktmengen gilt, die explizit zu einer entsprechenden Modell-Suite zusammengefasst wurden.

Existieren für einen Softwareentwicklungsschritt unterschiedliche Implementierungsalternativen, dann kann jede der Alternativen durch eine eigene Modell-Suite mit eigenen, anderen Alternativen widersprechenden Integritätsbedingungen dargestellt sein. Spielt ein Entwickler solche Alternativen durch, dann ist das Ausgangsartefakt an der Entstehung unterschiedlicher Zielartefakte beteiligt, die jeweils unterschiedlichen Implementierungsalternativen entsprechen. Dadurch sind die unterschiedlichen Zielartefakte indirekt über das Ausgangsartefakt verbunden. Würde man nun das Ausgangsartefakt mit allen Zielartefakten zusammenfassen und für diese Artefaktmenge die Konsistenzbedingungen aller beteiligten Modell-Suiten überprüfen, ergäben sich Widersprüche, obwohl alle Artefakte bezogen auf die einzelnen Modell-Suiten konsistent wären. Solche Inkonsistenzen ließen sich beheben, wenn beispielsweise eine Kennzeichnung der unterschiedlichen Implememententierungsalternativen Teil der jeweiligen Konsistenzbeziehung wären. Dies würde aber bedeuten, dass nicht nur alle möglichen Kombinationen bekannter Artefakttypen, sonden auch noch alle möglichen Kombinationen bekannter Artefaktbeziehungen zu komplexeren Modell-Suiten vor Beginn der Softwareentwicklung vorausgesehen und formalisiert werden müssen. Die Gewinnung einer "globalen Modell-Suite", die alle Artefakte des Entwicklungsprozesses umfasst und kohärent ist, ist folglich ein wenig realistisches Szenario.

Die zur Spezifikation von Konsistenzbeziehungen gewählte Sprache bestimmt, wie Konsistenzbedingungen formuliert werden müssen und welche Konsistenzbedingungen sich überhaupt formulieren lassen. Eigenschaften konkreter Artefakte müssen sich auf Ausdrücke der gewählten Sprache abbilden lassen und die entsprechenden Ausdrücke müssen sich in einer Weise zu formalen Aussagen kombinieren lassen, welche eine Bestimmung der Gültigkeit einer Integritätsbedingung erlauben.

Eine weitere Voraussetzung für die Konsistenzprüfung innerhalb einer Arbeitsumgebung besteht darin, dass sich nicht nur Aussagen über die in der Umgebung auftretenden Artefakte formal formulieren lassen, sondern dass die beteiligten Werkzeuge solche formalen Aussagen auch tatsächlich produzieren. Die Qualität solcher Aussagen wird wesentlich vom Funktionsumfang des Coherence Clients des jeweiligen Werkzeugs bestimmt. Dieser kann nur solche Beschreibungen in der formalen Sprache der Arbeitsumgebung produzieren, für die das jeweilige Entwicklungswerkzeug auch die nötigen Daten liefert. Je mehr Daten das jeweilige Werkzeug liefert, desto spezifischere Aussagen kann sein Coherence Client produzieren.

Zwischen den von Coherence Clients produzierbaren Aussagen und den in der formalen Sprache formulierbaren Integritätsbedingungen besteht eine wechselseitige Abhängigkeit. Können beispielsweise unterschiedliche Werkzeuge dasselbe Artefakt

bearbeiten, produzieren dabei aber unterschiedlich detaillierte formale Aussagen, dann kann der Entwickler einer Konsistenzbeziehung entscheiden, ob er eine Konsistenzbeziehung:

- auf Basis der detaillierten Beschreibung des einen Werkzeugs,
- auf Basis der gröberen Beschreibung des anderen Werkzeugs,
- oder auf Basis gemeinsamer Eigenschaften beider Beschreibungen formuliert.

Je nachdem, welche dieser Möglichkeiten gewählt wird und mit welchem Werkzeug das Artefakt dann tatsächlich bearbeitet wird, ändert sich auch die Aussagekraft einer Konsistenzprüfung. Basiert die Konsistenzprüfung auf einer detaillierten Artefaktbeschreibung, die das verwendete Werkzeug nicht liefern kann, schlägt die Konsistenzprüfung möglicherweise fehl oder ist nicht ausführbar. Basiert die Konsistenzprüfung auf einer groben Artefaktbeschreibung, das Werkzeug liefert aber feinere Angaben, bleiben unter Umständen Teile der Artefaktbeschreibung für die Konsistenzprüfung ungenutzt. Der Detaillierungsgrad, der bei der Formulierung von Konsistenzbeziehungen genutzt wird, kann in einem weiten Bereich schwanken. Folgende Liste zeigt beispielhaft einige mögliche Granularitätsstufen für artefaktübergreifende Beziehungen:

- grob, untypisiert, keine Konsistenzprüfung:
  Die Beziehung dokumentiert, dass Artefakte zueinander in Beziehung stehen. Ob die Artefakte einem bestimmten Typ angehören oder zwischen den Eigenschaften der Artefakte bestimmte Integritätsbedingungen erfüllt sind, wird nicht geprüft.

- grob, typisiert, keine über die Typprüfung hinausgehende Konsistenzprüfung:
  Der Typ der Konsistenzbeziehung schreibt nicht nur vor, dass Artefakte zueinander in Beziehung stehen, sondern auch, welchem Artefakttyp die Artefakte angehören müssen. Eine derartige Beziehung kann also nur zwischen Artefakten bestimmter Typen hergestellt werden.

- fein, auf Standardeigenschaften basierende Konsistenzprüfung:
  Zusätzlich zum Typ der an der Beziehung beteiligten Artefakte definiert der Beziehungstyp auch eine Integritätsbedingung. Existiert zum jeweiligen Artefakt eine Repräsentation, die mit dem Formalismus zur Konsistenzprüfung kompatibel ist, kann die Einhaltung der Integritätsbedingung zwischen den an dieser Beziehung beteiligten Artefakten geprüft werden. Werden in der Konsistenzbedingung ausschließlich Artefakteigenschaften genutzt, die für Artefakte

des jeweilgen Typs in offiziellen Standards festgelegt sind, dann kann die Konsistenzprüfung für alle Werkzeuge genutzt werden, die sich an den jeweiligen Standard halten.

- sehr fein, spezialisierte Konsistenzprüfung:
  Die für eine Konsistenzbeziehung definierte Integritätsbedingung nutzt spezielle Artefakteigenschaften, die nur für Artefakte eines speziellen Werkzeugs definiert sind. Hierdurch lässt sich eine enge Kopplung von Entwicklungswerkzeugen erreichen. Allerdings sinkt dadurch auch die Wiederverwendbarkeit von Integritätsbedingungen, da sie in diesem Fall nur für eine spezielle Arbeitsumgebung mit einem speziellen Satz von Werkzeugen definiert sind.

Die Notwendigkeit, für eine artefaktübergreifende Konsistenzsicherung entsprechende Beziehungen und Integritätsbedingungen zu definieren oder aus bestehenden Definitionen auszuwählen beeinflusst den Ablauf herkömmlicher Entwicklungsprozesse und damit die Nutzung der Arbeitsumgebung. Der folgende Abschnitt fasst die entsprechenden Veränderungen zusammen.

4.2 Veränderung herkömmlicher Arbeitsweisen

Die Veränderung einer Arbeitsumgebung durch zusätzliche Mechanismen zur Konsistenzprüfung bewirkt Veränderugen der Arbeitsweise der Softwareentwickler in einer solchen Umgebung.

4.2.1 Definition von Beziehungstypen

Vorgehensmodelle für die Softwareentwicklung wie das V-Modell [4] oder der Rational Unified Process [6] sehen am Anfang der Entwicklung eine Phase des Zurechtschneiderns (Tailoring) des Vorgehensmodells für das konkrete Entwicklungsprojekt vor. In diese Phase fällt auch die Festlegung, welche Artefakttypen während des Entwicklungsprozesses entstehen können oder sollen. Die Definition oder Auswahl artefaktübergreifender Integritätsbedingungen fällt auch in diese Phase. Sobald bekannt ist, welche Typen von Artefakten entstehen können, lässt sich auch festlegen, zwischen welchen Artefakttypen Konsistenzbeziehungen möglich sind. Die möglichen Beziehungstypen stellen dabei eigenständige Artefakttypen dar, die der Menge der bereits ausgewählten Artefakttypen hinzugefügt werden. Sofern in dieser Phase bereits die Arbeitsumgebung im Sinne einer Menge einzusetzender Werkzeuge feststeht, können diese bereits auf ihre Unterstützung für die bekannten Beziehungstypen hin überprüft und gegebenenfalls angepasst werden. Nach der Anpassung des

Vorgehensmodells kann es instanziiert werden, es wird also eine konkrete Entwicklung bzw. ein konkretes Vorgehen begonnen. Während der Instanziierung werden weitere Festlegungen getroffen, unter anderem

- welche Artefakte im einzelnen zu produzieren sind,
- für welchen Entwicklungsschritt welches Artefakt relevant ist,
- in welcher Beziehung Artefakte zueinander stehen sollen.

Aus der Menge der möglichen Artefakttypen und -beziehungen wird in dieser Entwicklungsphase also noch einmal eine Auswahl der tatsächlich zu produzierenden Artefakte und der tatsächlich einzuhaltenden Artefaktbeziehungen getroffen. Ein Großteil der zu prüfenden Integritätsbedingungen lässt sich damit festlegen, noch bevor das erste Artefakt produziert wird. Da Konsistenzbeziehungen zwischen konkreten Artefakten hier ihrerseits als Artefakte aufgefasst werden, lassen sie sich auch während des Entwicklungsprozesses noch um spezielle Konsistenzprüfungen erweitern. Dies kommt insbesondere solchen Entwicklungsprozessen zu Gute, bei denen Artefakte iterativ überarbeitet und inkrementell verfeinert werden.

4.2.2 Erzeugen konkreter Artefaktbeziehungen

Die kausale Reihenfolge von Entwicklungsschritten ändert sich durch die Einführung zusätzlicher Konsistenzprüfungen nicht. Die Gewinnung der für die Konsistenzprüfung notwendigen Artefaktdaten erfolgt automatisiert durch den Coherence Client des jeweiligen Entwicklungswerkzeugs. Die Ausführung der Konsistenzprüfung kann ebenfalls vom Coherence Client zu bestimmten Zeitpunkten automatisch angestossen werden, beispielsweise beim Speichern, Einchecken oder Kompilieren eines Artefakts. Für den Entwickler ändert sich bei der Bearbeitung existierender Artefakte lediglich die Menge der Meldungen, die er nach einem entsprechenden Ereignis bekommt. Neben den bisherigen Fehlermeldungen, Warnungen und Reports erhält er jetzt zusätzlich die Ergebnisse der artefaktübergreifenden Konsistenzprüfung.

Ein geringer Zusatzaufwand entsteht für den Entwickler beim Erzeugen neuer Artefakte. Diese verfügen noch über keinerlei Beziehungen zu anderen Artefakten. Der Entwickler muss also aus den möglichen Beziehungstypen für das neue Artefakt diejenigen auswählen, die künftig für das Artefakt geprüft werden sollen. Diese Auswahl kann auch für alle Artefakte dieses Typs fest vorgegeben werden, so dass ein Coherence Client beim Erzeugen eines neuen Artefakts automatisch alle für diesen Artefakttyp vorgeschriebenen Beziehungen anlegen kann. Diese vom Entwickler

manuell oder automatisch festgelegten Beziehungen sind noch nicht vollständig initialisiert. Ihnen fehlen noch die übrigen an der Beziehung beteiligten Artefakte. Existieren die fehlenden Artefakte noch gar nicht, braucht der Entwickler zunächst nichts weiter zu tun. Die artefaktübergreifenden Beziehungen seines neuen Artefakts sind partiell initialisiert und damit zunächst nur eingeschränkt überprüfbar. Existieren aber bereits Artefakte, deren Typ zur fraglichen Beziehung passt und die ihrerseits eine partiell initialisierte Beziehung desselben Typs besitzen, dann erzeugt der Entwickler keine neue, partiell initialisierte Konsistenzbeziehung, sondern er fügt sein neues Artefakt der partiell initialisierten Beziehung des anderen Artefakts hinzu und vervollständigt damit dessen Beziehung. Dadurch, dass dem Entwickler nicht alle existierenden Artefakte eines bestimmten Typs angezeigt werden, sondern nur diejenigen, die quasi noch auf eine Vervollständigung ihrer Beziehungen warten, reduziert sich die Anzahl der auswählbaren Artefakte. Die Grad der möglichen Unterstützung bei der Herstellung neuer Artefaktbeziehungen wird durch das eingesetzte Content Management System bestimmt.

4.2.3 Nutzung bestehender Konsistenzbeziehungen

Sind Artefakte einmal erfasst und über entsprechende explizite Beziehungen verbunden, lassen sich diese Beziehungen in weiteren Entwicklungsschritten nutzen. Eine Anwendung ist die so genannte Impact Analysis, also die Bewertung der Auswirkungen, die eine Veränderung eines Artefakts hätte. Abbildung 4 zeigt ein Beispiel.

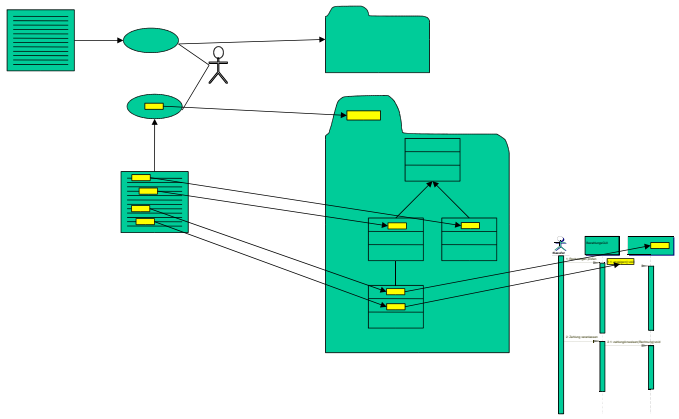

Abbildung 4: Modelle einer Modell-Suite.

Einige Artefakte, hier ein Text, ein Use Case-Diagramm, Klassendiagramme und ein Interaktionsdiagramm sind erzeugt und als Assets erfasst worden. Dabei wurden speziell die markierten Teile erfasst und zueinander in Beziehung gesetzt. Die Richtung und Bedeutung der einzelnen Beziehungen werden vom Benutzer definiert

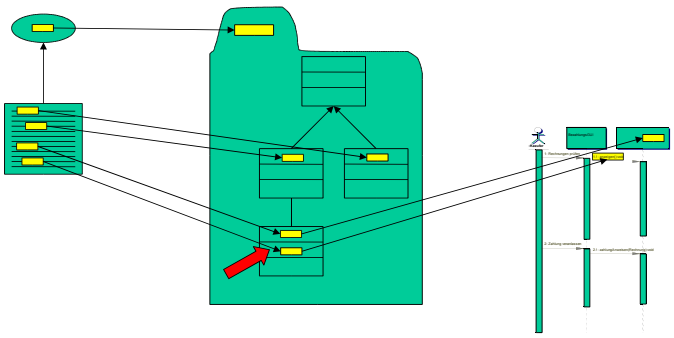

Abbildung 5: Ein Modell wird verändert.

(z.B. “wird modelliert durch”). Angenommen, eines der gezeigten Artefakte soll nun in einer folgenden Iteration der Softwareentwicklung an der in Abbildung 5 gezeigten Stelle verändert werden. Eine Änderung in einem Artefaktbestandteil, im Beispiel die Veränderung einer Methodensignatur in einem Klassendiagramm, bedeutet auch eine Veränderung des umgebenden Artefakts. Nicht nur die einzelne Methode, auch die Signatur der umgebenden Klasse ändert sich. Innerhalb desselben Artefakts werden solche Auswirkungen typischerweise vom Entwicklungswerkzeug erkannt und behandelt. Auswirkungen zwischen Artefakten unterschiedlicher Werkzeuge können so nicht behandelt werden. Wurde das im Beispiel gezeigte Interaktionsdiagramm mit einem anderen Werkzeug erstellt, wird die notwendige Anpassung nach einer Änderung des Klassendiagramms nicht erkannt. Hier kommen nun die zuvor erfassten Beziehungen zum Tragen. Bevor der Entwickler das Klassendiagramm ändert, kann er sich eine Liste aller Artefakte anzeigen lassen, die von diesem Artefakt abhängen. In Abbildung 6 sind die relevanten Beziehungen gestrichelt dargestellt. In einem vorwärtsgerichteten, iterativen Entwicklungsprozess würde die ursprüngliche Artefaktversion nicht gelöscht, sondern durch eine neue, veränderte Version ersetzt werden. Für die neue Artefaktversion ist also zu prüfen, ob die in der Vorgängerversion noch konsistenten Beziehungen zu Folgeartefakten auch nach der Änderung noch gültig sind. Das Asset-Verwaltungssystem liefert dazu eine Agenda aller Beziehungen, die auf mögliche Konsistenzverletzungen zu prüfen sind. Ist auch die Prüfung maschinell durchführbar, beispielsweise in Form einer ASM-Spezifikation, wird sie ausgeführt und dem Entwickler das Ergebnis angezeigt.

Ist die ursprüngliche Konsistenzbeziehung durch die neue Artefaktversion nicht mehr gegeben, müssen auch für die betroffenen Folgeartefakte neue Versionen erzeugt werden. Wie in Abbildung 7 gezeigt, wird dann diese neue Version des Folgeartefakts, im Beispiel das Interaktionsdiagramm, mit der neuen Version des Vorläuferartefakts verbunden. Die neuen Beziehungen sind auch konsistent, da das Folgeartefakt explizit zur Wiederherstellung der Konsistenz zum Vorläuferartefakt erzeugt

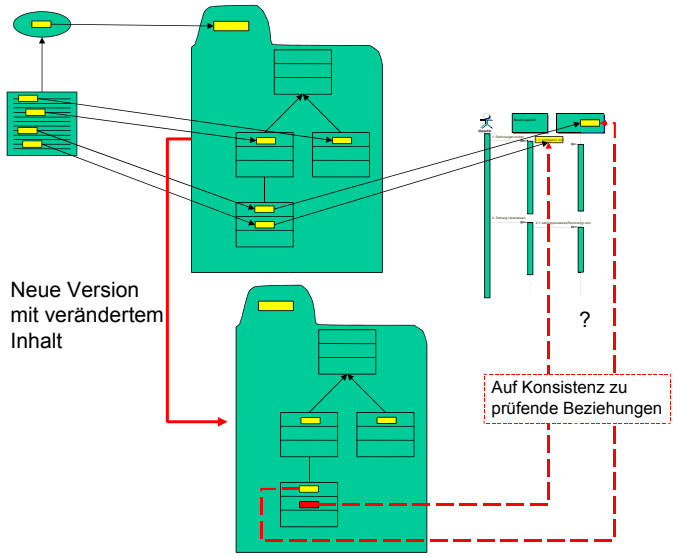

Abbildung 6: Von Änderung potentiell betroffene Teilmodelle.

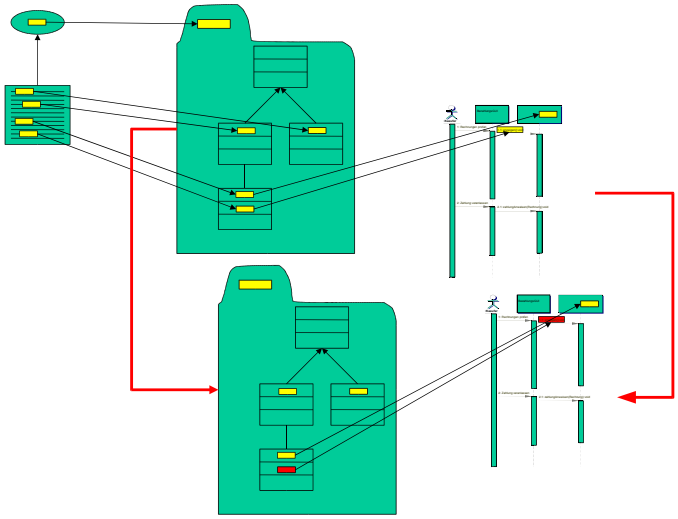

Abbildung 7: Konsistenz durch neue Version des abhängigen Modells wiederhergestellt.

wurde.

Hat die Veränderung eines Artefakts hingegen keine Auswirkungen auf die Gültigkeit der auf der Agenda befindlichen Beziehungen, kann eine neue Artefaktversion erzeugt werden, ohne dass zusätzliche Beziehungen oder neue Folgeartefakte hergestellt werden müssen. Die ursprünglichen Beziehungen bleiben erhalten, lediglich zwischen den beiden unterschiedlichen Versionen des geänderten Artefakts existiert eine Beziehung, die den Versionssprung anzeigt. Die Forderung nach Kohärenz, nach dem konsistenten Zusammenhalt aller Artefakte ist hierdurch bereits erfüllt.

4.2.4 Prüfung konkreter modellübergreifender Konsistenzbedingungen

Zur Veranschaulichung der Prüfung artefaktübergreifender Konsistenzbeziehungen seien die in Abbildung 8 gezeigten Artefakte gegeben. Das linke der beiden Artefakte

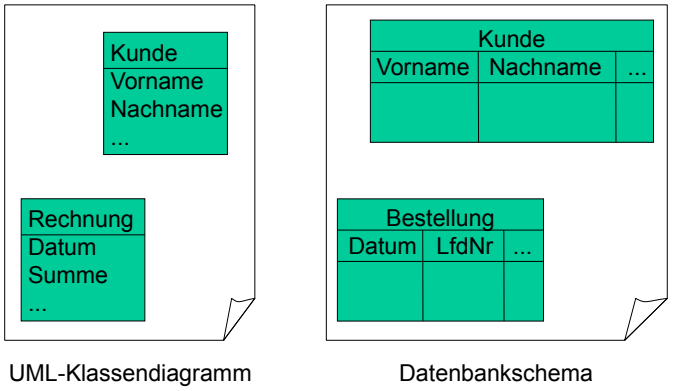

Abbildung 8: Auf Konsistenz zu prüfende Artefakte

sei ein Klassendiagramm. Seine physische Repräsentation wird durch das Werkzeug bestimmt, mit dem es erstellt wurde, beispielsweise eine XMI-Datei [10]. Das rechte Artefakt symbolisiere ein Datenbankschema. Das Datenbankschema könnte durch das SQL-Skript beschrieben sein, mit dem es erzeugt wurde. Zwischen diesen beiden Artefakten sollen folgende Konsistenzbedingungen geprüft werden:

- Für jede Klasse innerhalb des UML-Klassendiagramms muss eine korrespondierende (namensgleiche) Relation im Datenbankschema existieren.
- Für jede Relation innerhalb des Datenbankschemas muss eine korrespondierende (namensgleiche) Klasse innerhalb des Klassendiagramms existieren.

In einem ersten Schritt werden die Eigenschaften des jeweiligen Artefakts in die gewählte einheitliche Repräsentation überführt. Im Beispiel in Abbildung 9 werden alle Bezeichner in Elemente einer Domäne *Names* übersetzt. Diese Bezeichner bilden den Definitionsbereich für dynamische Funktionen einer Abstract State Machine

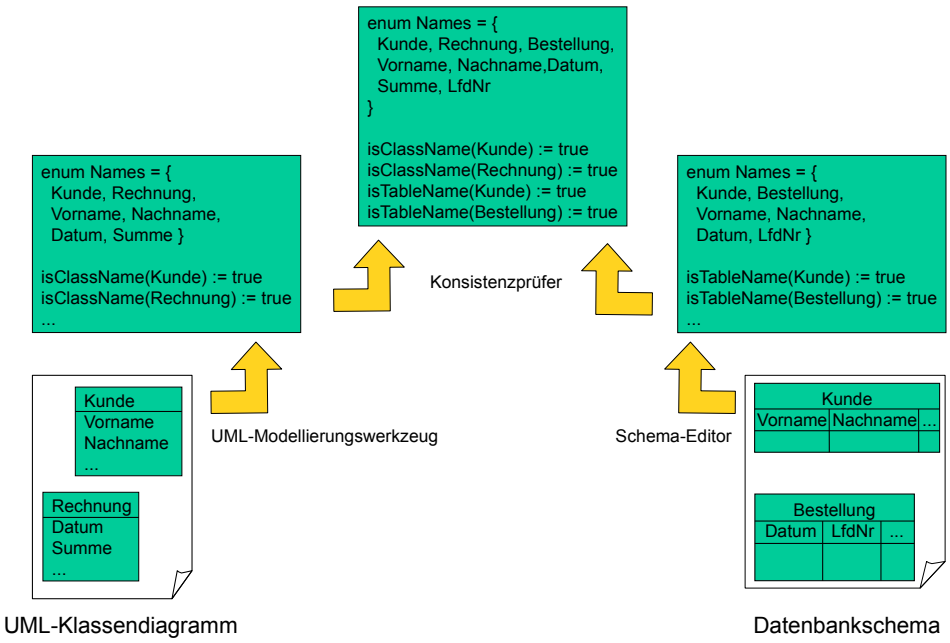

Abbildung 9: Abbildung von Artefakteigenschaften auf einheitliche Repräsentation

(ASM), welche die Zuordnung des Namens zu einer Artefakteigenschaft symbolisieren. Beschreibt ein Bezeichner eine bestimmte Eigenschaft, wird der korrespondierenden dynamischen Funktion für diesen Bezeichner der Wert *true* zugewiesen. Die Menge der dynamischen Funktionen ist spezifisch für den jeweiligen Artefakttyp und kann unabhängig von konkreten Konsistenzbedingungen definiert werden. Während *isClassName* im Beispiel eine Funktion wäre, die ausschließich im Kontext des Klassendiagramms auftritt, kann eine Beschreibung eines Datenbankschemas eine Funktion *isTableName* verwenden, die im Kontext eines Klassendiagramms nicht anzutreffen wäre. Die Abbildung von der werkzeugspezifischen Repräsentation in die werkzeugübergreifende Repräsentation nimmt das jeweilige Werkzeug selber vor, zum Beispiel automatisch beim Speichern des jeweiligen Artefakts. Liegen beide Artefakte in der einheitlichen Repräsentation vor, können diese von einem speziellen Konsistenzprüfer zusammengeführt werden. Im Falle der hier gewählten ASM-Repräsentation definieren die zusammengeführten Eigenschaften beider Artefakte den Startzustand einer ASM. Damit auf Basis dieses Zustandes eine Konsistenzprüfung ausgeführt werden kann, muss diesem Startzustand eine Beschreibung der zu prüfenden Konsistenzregel hinzugefügt werden (Abbildung 10). Diese kann unabhängig von den konkreten Artefakten formuliert und verwaltet werden. Allerdings ist für ihre Formulierung die Kenntnis der für die beteiligten Artefakttypen definierten Domänen und Funktionen erforderlich.

Gegeben sein nun folgender Zustand:

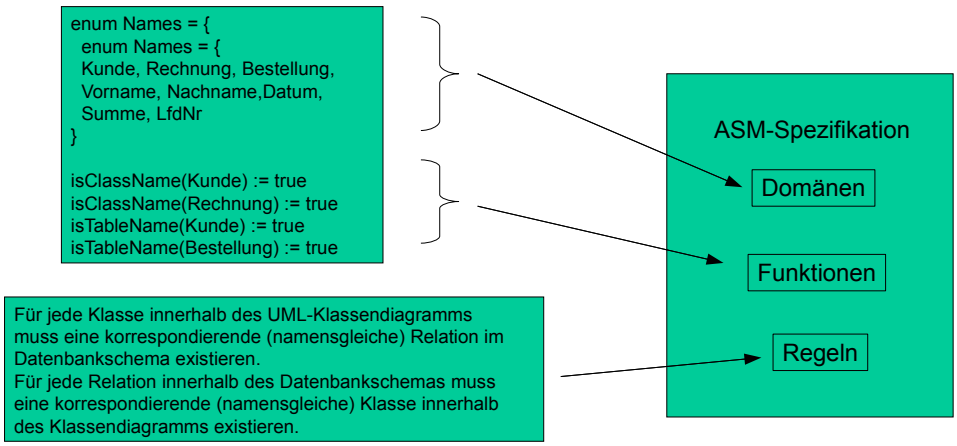

Abbildung 10: Zusammenführung von Artefakteigenschaften und Konsistenzregeln

- Jedes Artefakt liege in der einheitlichen Repräsentation vor.
- Die zu prüfende Regel liege in der einheitlichen Repräsentation vor.
- Artefakt- und Regelbeschreibung seien als Inhalte einer begriffsorientierten Inhaltsverwaltung erreichbar.
- Artefakt- und Regelbeschreibung seien durch eine Asset-Beschreibung verknüpft, welche die Zusammengehörigkeit der Artefakte und die zu prüfende Konsistenzbedingung repräsentiert.

Ausgehend von diesem Zustand kann dann ein Konsistenzprüfer alle beteiligten ASM-Beschreibungen lesen und zu einer ASM-Gesamtspezifikation zusammenfügen. Abbildung 11 zeigt für das Beispiel eine mögliche Gesamtspezifikation der zu prüfenden Konsistenzbedingung. Diese kann nun direkt von einer entsprechenden Laufzeitumgebung ausgeführt werden. Das Ergebnis der Prüfung, also der für die beteiligten Werkzeuge relevante Endzustand, lässt sich aus den Werten der dynamischen Funktionen nach Ausführung der Prüfung auslesen. Im Beispiel ist folgende Konsistenzbedingung zu erfüllen

$$\forall x \in Names : isTableName(x) \wedge isClassName(x)$$

Diese wäre für die im Beispiel gezeigten Artefakte nicht erfüllt. Daraus kann der Entwickler aber noch nicht zwingend erkennen, wodurch diese Bedingung verletzt ist und wie sich die Konsistenzverletzung beheben lässt. Betrachtet man diese Bedingung als Teil eines artefaktübergreifenden Vertrages (vgl. Abschnitt 3) müsste sie noch um einen Mechanismus zu ihrer Durchsetzung ergänzt werden. Im Beispiel geschieht dies durch Einführung einer dynamischen Funktion *contractReaction*. Diese bildet das konsistenzverletzende Element der Domäne *Names*, auf ein Element der

```
/* CoreASM specification */

CoreASM SampleContractCheck

use StandardPlugins

function isClassName:Names -> Boolean
function isTableName:Names -> Boolean
function contractReaction:Names -> ContractReactions

enum Names = { Kunde, Rechnung, Bestellung, Vorname, Nachname,
               Datum, Summe, LfdNr  }

enum ContractReactions = {
  createCorrespondingTable, createCorrespondingClass
}

init InitRule

rule customInits =
par
  isTableName(Kunde) := true
  isTableName(Bestellung) := true
  isClassName(Kunde) := true
  isClassName(Rechnung) := true
endpar

rule InitRule =
seqblock
  customInits
  derivedInits
  program(self) := @MainProgram
endseqblock

rule derivedInits =
forall n in Names do par
  if (isTableName(n) = undef) then isTableName(n) := false
  if (isClassName(n) = undef) then isClassName(n) := false
endpar

rule MainProgram =
seqblock
  forall n in Names do seqblock
    isConsistent(n) := isTableName(n) and isClassName(n)
    if not isConsistent(n) then
      if isTableName(n) then
        contractReaction(n) := createCorrespondingClass
      else if isClassName(n) then
        contractReaction(n) := createCorrespondingTable
  endseqblock
  program(self) := undef
endseqblock
```

Farbkodierung:
•Blau: Funktions- und Regeldefinitionen
•Rot: Artefakteigenschaften

Abbildung 11: ASM-Code für Konsistenzprüfung

Domäne *ContractReactions* ab. Anhand des Funktionswerts lässt sich die Ursache der Konsistenzverletzung und damit auch eine mögliche Maßnahme zu ihrer Korrektur erkennen. Nach einem Durchlauf des ASM-Hauptprogramms ist damit die folgende Nachbedingung erfüllt:

$$\begin{aligned}\forall x \in Names: \quad & (isTableName(x) \wedge isClassName(x)) \vee \\ & (isTableName(x) \wedge contractReaction(x) = createCorrespondingClass) \vee \\ & (isClassName(x) \wedge contractReaction(x) = createCorrespondingTable)\end{aligned}$$

Erfüllen alle Elemente aus *Names* die Integritätsbedingung $isTableName(x) \wedge isClass\ Name(x)$, ist ihr zugehöriger Funktionswert in *contractReaction* undefiniert. Erfüllt ein Element aus *Names* diese Integritätsbedingung nicht, dann ist $contractReaction(x) = createCorrespondingClass$ oder $contractReaction(x) = createCorrespondingTable$. Der Funktionswert suggeriert hier schon eine bestimmte Änderungsoperation aus der Menge der konsistenzerhaltenden Operationen des Vertrages. Diese Bezeichnungen wurden hier nur zur Veranschaulichung gewählt. *ContractReactions* könnte auch künstliche Werte wie $\{Inkonsistenz1, Inkonsistenz2\}$ enthalten. Wie diese Werte interpretiert werden, wird entweder vom Entwickler entschieden oder durch ein Werkzeug, das mit einer Interpretation für diese Werte erweitert wurde. Wird für ein $x \in Names$ der Funktionswert $contractReaction(x) = createCorrespondingClass$ ermittelt, dann existiert keine Klasse, die mit demselben Namen benannt ist. Diese Inkonsistenz lässt sich auf unterschiedliche Arten beheben, zum Beispiel

- durch Einfügen einer neuen Klasse mit dem fraglichen Namen,
- durch Entfernen der Tabelle mit diesem Namen, dann wäre auch keine korrespondierende Klasse nötig,
- durch Umbennennen eine Klasse, so dass ihr Name mit dem der Tabelle identisch ist,
- durch Umbenennen der Tabelle, so dass ihr Name mit dem einer existierenden Klasse identisch ist.

Nach jeder dieser Änderungsoperationen wäre die Konsistenzbedingung für das fragliche Element erfüllt. Welche diese Operationen tatsächlich ausgeführt wird, ist nicht mehr Teil der Konsistenzprüfung, sondern, in der Terminologie der Modell-Suiten gesprochen, Teil eines Bearbeitungsschemas der Modell-Suite [17]. Ein solches Schema könnte festlegen, dass immer dann, wenn das Fehlen eines korrespondierenden Artefaktelements festgestellt wird, das fehlende Artefaktelement erzeugt wird. In einem

alternativen Schema könnte festgelegt sein, dass Artefaktelemente, zu denen kein korrespondierendes Element in einem anderen Artefakt existiert, gelöscht werden. Ein drittes Schema könnte die Erzeugung einer Fehlermeldung vorschreiben, welche den Benutzer auf die Inkonsistenz hinweist und ihn zu einer manuellen Korrektur auffordert. Soll die Korrektur von Konsistenzverletzungen in die beteiligten Entwicklungswerkzeuge integriert werden, dann muss das jeweilige Entwicklungswerkzeug mindestens ein Bearbeitungsschema implementieren. Dasselbe Werkzeug könnte auch unterschiedliche Schemata implementieren. Würde man dann das gewählte Bearbeitungsschema zusammen mit den Artefaktdaten als ASM-Spezifikation repräsentieren, könnte die ASM-Konsistenzprüfung auch abhängig vom gewählten Bearbeitungsschema unterschiedliche *ContactReactions*-Werte bestimmen. Die Auswahl einer geeigneten Änderungsoperation wäre dann nicht mehr allein dem jeweiligen Entwicklungswerkzeug überlassen, sondern könnte werkzeugunabhängig im Rahmen des ASM-Programms spezifiziert werden.

5 Zusammenfassung

Wir haben gezeigt, wie sich Teilmodelle, die während der Entwicklung eines Softwaresystems entstehen, zu einer Modell-Suite zusammenfassen lassen. In einer derartigen gemeinsamen Struktur lässt sich nicht nur die Zusammengehörigkeit der Teilmodelle im Sinne einer Koexistenz ausrücken, sondern in dieser Struktur sind auch die Randbedingungen einer konsistenten Ko-Evolution der beteiligten Modelle spezifiziert. Eine derartige Ko-Evolution wird in Arbeitsumgebungen für Softwarentwickler bisher nicht oder bestenfalls partiell unterstützt.

Wir haben an Beispielen gezeigt, dass eine derartige Unterstützung unter Zuhilfenahme bereits existierende Werkzeuge und Content-Management-Systeme möglich ist. Durch Definition spezifischer Typen von Modell-Suiten, erhält der Softwareentwickler eine Art Skelett, entlang dessen er seine Entwicklungsschritte planen, ausführen und ihre Gültigkeit überprüfen kann. Die Gefahr der Entstehung unerwünschter oder gar widersprüchlicher Modellierungsergebnisse wird reduziert. Risiken für die spätere Implementierung eines Softwaresystems werden verringert oder gar vollständig vermieden.

Literatur

[1] Börger, Egon: The ASM Refinement Method. In: *Formal Aspects of Computing* 15 (2003), Nr. 2-3, S. 237–257

[2] Bossung, Sebastian ; Sehring, Hans-Werner ; Skusa, Michael ; Schmidt, Joachim W.: Conceptual Content Management for Software Engineering Processes. In: *Proceedings of the 9th East-European Conference on Advances in Databases and Information Systems, ADBIS 2005* Bd. 3631, Springer, 2005 (Lecture Notes in Computer Science), S. 309 ff

[3] Dahanayake, Ajantha ; Thalheim, Bernhard: Co-Evolution of (Information) System Models. In: *Enterprise, Business-Process and Information Systems Modeling* Bd. 50, Springer, 2010 (Lecture Notes in Business Information Processing), S. 314–326

[4] Dröschel, Wolfgang ; Heuser, Walter: *Inkrementelle und objektorientierte Vorgehensweisen mit dem V-Modell 97*. Oldenbourg, 1998

[5] Gamma, E. ; Helm, R. ; Johnson, R. ; Vlissides, J.: *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1996 (Professional Computing Series)

[6] Jacobson, Ivar ; Booch, Grady ; Rumbaugh, James: *The Unified Software Development Process*. Reading, Massachusetts : Addison-Wesley, 1999 (Object Technology Series)

[7] *Kapitel* 11 – Truth Functional Forms. In: Lover, Robert: *Elementary Logic: For Software Development*. Springer, 2008

[8] Mittelstrass, Jürgen (Hrsg.): *Enzyklopädie Philosophie und Wissenschaftstheorie, Band 2 H-O*. Bibliographisches Institut Mannheim, 1984

[9] Mittelstrass, Jürgen (Hrsg.): *Enzyklopädie Philosophie und Wissenschaftstheorie, Band 4 Sp-Z*. Bibliographisches Institut Mannheim, 1996

[10] Object Management Group: *XML Metadata Interchange (XMI) Specification*. `http://doc.omg.org/formal/03-05-02.pdf`. Version: Mai 2003, Abruf: 18.12.2010

[11] Salay, Rick ; Mylopoulos, John ; Easterbrook, Steve: Using Macromodels to Manage Collections of Related Models. In: *Advanced Information Systems Engineering, 21st International Conference CAiSE 2009* Bd. 5565, Springer, 2009 (Lecture Notes in Computer Science), S. 141–155

[12] Schmidt, Peggy ; Thalheim, Bernhard: Management of UML Clusters. In: Abrial, Jean-Raymond (Hrsg.) ; Glässer, Uwe (Hrsg.): *Rigorous Methods for*

Software Construction and Analysis Bd. 5115, Springer, 2009 (Lecture Notes in Computer Science), S. 111–129

[13] Schöning, Uwe: *Logik für Informatiker.* 5. Auflage. Spektrum, 2000

[14] Sehring, Hans-Werner: *Konzeptorientierte Inhaltsverwaltung – Modell, Systemarchitektur und Prototypen*, Technische Universität Hamburg-Harburg, Diss., 2003. `http://www.sts.tu-harburg.de/phd-theses/2003/Sehr03.pdf`, Abruf: 18.12.2010

[15] *Kapitel* 25 – Higher-Order Logic. In: Shapiro, Stewart: *The Oxford Handbook of Philosophy of Mathematics and Logic.* Oxford University Press, 2007 (Oxford handbooks in philosophy)

[16] Simons, Anthony J. H. ; Graham, Ian: 30 Things That Go Wrong in Object Modelling with UML 1.3. In: Kilov, Haim (Hrsg.) ; Rumpe, Bernhard (Hrsg.) ; Simmonds, Ian D. (Hrsg.): *Behavioral Specifications of Businesses and Systems.* Kluwer Academic Publishers, 1999, S. 237–257

[17] Thalheim, Bernhard: Model Suites / Department of Computer Science, Christian-Albrechts-University Kiel. Version: 2010. `http://www.is.informatik.uni-kiel.de/~thalheim/ModelEngineering.pdf`, Abruf: 18.12.2010. 2010. – Forschungsbericht