Zur Validierung von Kompositionsstrukturen in UML mit USE

Lars Hamann, Martin Gogolla, Mirco Kuhlmann
Universität Bremen, Fachbereich Informatik
AG Datenbanksysteme, D-28334 Bremen
{lhamann|gogolla|mk}@informatik.uni-bremen.de

Abstract: In der Softwareentwicklung rücken Modelle zunehmend in den Fokus des Entwicklungsprozesses. Dadurch steigen auch die Anforderungen an deren Qualität. Mit dem an der Universität Bremen entwickelten UML/OCL-Werkzeug USE können bereits bestimmte Qualitätsaspekte von Modellen statisch und dynamisch analysiert werden. Dieser Artikel beschreibt neue Modellierungselemente der UML 2 und zeigt, welchen Beitrag eine Weiterentwicklung von USE auf dem Weg zu einer integrierten Semantik der UML 2 Kompositionsstrukturen leisten kann.

1 Einleitung

In der Softwareentwicklung verlagert sich der Fokus zunehmend von der code-zentrierten Entwicklung, bei der Modelle (wenn überhaupt) nur für den Entwurf oder die Dokumentation benutzt werden, hin zur modell-zentrierten Entwicklung, bei der formale Modelle, aus denen auch Quellcode generiert werden kann, im Mittelpunkt stehen; aktuelle Schlagwörter dafür sind *Model Driven Development* bzw. *Model Driven Architecture* (MDD/MDA). Der Begriff „formales Modell“ bedeutet, dass das Modell einen Aspekt der zu erstellenden Software vollständig beschreibt [SVEH07, S. 11]. Ein Modell kann auf verschiedene Arten durch textuelle oder grafische Modellierungssprachen definiert werden, wobei im Kontext der Softwareentwicklung die *Unified Modeling Language* (UML) [OMG09b] weit verbreitet ist. Durch die zentrale Stellung von Modellen erhält auch deren Qualität eine größere Bedeutung, da diese essentiell für die weitere Verarbeitung (Codegenerierung oder Interpretation des Modells) ist. Die Qualität der (statischen) Softwarestruktur hat einen großen Einfluss auf die Gesamtqualität des resultierenden Systems, da diese (in der Regel) den Grundstein für dynamische Vorgänge innerhalb des Systems bildet.

Die Möglichkeiten der Strukturbeschreibung wurden in UML 2 auch im Hinblick auf die modellgetriebene Entwicklung gegenüber der UML 1 erweitert. In diesem Artikel liegt der Fokus auf den Erweiterungen im Bereich der Darstellung von Kompositionsstrukturen und deren Auswirkungen auf das Klassenmodell. Während UML 1 für deren Darstellung nur Komposition und Aggregation zwischen Klassen zur Verfügung stellte, können diese in UML 2 auch mit Hilfe von sogenannten *Composite Structures* [OL06, HMPW04] dargestellt werden. Die bisher vorhandenen Modellierungsmöglichkeiten zeigten vor allem bei der Verbindung von Elementen auf der gleichen Dekompositionsebene Schwä-

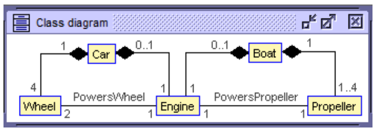

Abbildung 1: UML 1 Kompositionen

chen [Boc04]. Der neue Ansatz erlaubt eine detaillierte Beschreibung von Kompositionen in einem UML Modell. Dabei können komplexe Kompositionsstrukturen innerhalb einer einzelnen Klasse dargestellt werden, was gegenüber der klassischen „Teil-Ganzes-Beziehung“ präzisere Spezifikationen von Modellen erlaubt. Damit wird es möglich, über die UML 2 Modelleinschränkungen auszudrücken, die vorher als zusätzliche Einschränkungen z. B. in OCL [WK03, OMG06] angegeben werden mussten. Im Folgenden wird ein kurzer Überblick über die neuen Modellierungselemente gegeben; eine detailliertere Einführung findet sich in [Boc04]. Das in Abbildung 1 dargestellte Beispiel (mit geringfügigen Änderungen übernommen aus [Boc04]) zeigt Kompositionen, wie sie in UML 1 möglich waren. Dadurch, dass die Assoziationen global für Klassen definiert sind, sich also auf alle Instanzen der Klassen beziehen, können sich in Abhängigkeit von Instanzeigenschaften verschiedene Probleme ergeben:

- Ein Motor (Engine) innerhalb eines Auto-Objekts könnte die Räder eines anderen Auto-Objekts antreiben oder sogar die Schiffsschraube eines Bootes.
- Ein Motor muss Räder *und* Propeller antreiben.

Eine gültige Modellinstanz, die sicherlich so nicht gewünscht aber erlaubt ist, ist in Abbildung 2 gezeigt. Dabei treibt der Motor *e1* sowohl zwei Räder (*w1* und *w2*) des Autos *car* als auch die Schiffsschraube *p1* des Bootes *boat* an, obwohl der Motor *e1* zum Objekt *car* gehört. Entsprechende Unstimmigkeiten finden sich beim Motor *e2*. Weiterhin werden alle vier Räder von *car* angetrieben, obwohl nur zwei Räder mit dem zum Auto gehörenden Motor verbunden sind.

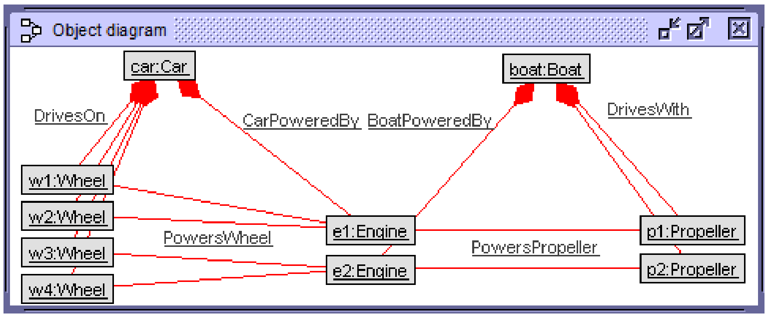

Abbildung 2: Objektdiagramm

Durch Anpassung der Multiplizitäten (u. a. auf 0..1 statt 1 bei den Assoziationen `PowersWheel` und `PowersPropeller` an den Assoziationsenden des Motors) und zusätzliche OCL-Constraints kann dieses Modell entsprechend korrigiert werden. Ein mögliches OCL-Constraint, welches definiert, dass ein Motor innerhalb eines Autos zwei Räder antreibt und keine Schiffsschrauben, wäre:

```
context Engine inv inCarRequiresTwoPoweredWheels:
 self.car.isDefined() implies
   self.wheel->size() = 2 and self.propeller->size() = 0
```

Dieses Problem kann auch durch weitere Spezialisierungen der Klasse `Engine` in `CarEngine` und `BoatEngine` und weiteren Assoziationen gelöst werden.

Das zweite angesprochene Problem, dass ein Motor Antriebsteile aus einem anderen Fahrzeug des selben Typs antreiben könnte, kann jedoch nur mit Constraints gelöst werden und nicht durch eine Veränderung der Vererbungsbeziehungen:

```
context Engine inv powersSameCar:
 self.car.isDefined() implies
  self.wheel->forAll(w:Wheel | w.car=self.car)

context Engine inv powersSameBoat:
 self.boat.isDefined() implies
  self.propeller->forAll(p:Propeller | p.boat=self.boat)
```

Ein Nachteil der angegebenen Constraints und der Spezialisierung in Auto- und Bootmotoren im Hinblick auf die Wiederverwendung und auf eine komponentenbasierte Entwicklung ist, dass der Entwickler eines Motors bereits zur Entwicklungszeit wissen muss, in welchen Kontexten ein Motor eingesetzt wird. Im Falle der Spezialisierung stellt sich zusätzlich die Frage, ob es tatsächlich eine Unterscheidung der Motorenarten geben soll oder ob dies nur eine technisch motivierte Vererbungsbeziehung wäre, was im Allgemeinen vermieden werden sollte.

Mit dem Kompositionsmodell der UML 2 kann die gewünschte Struktur ohne zusätzliche Constraints, also nur mit Hilfe von grafischen Mitteln, ausgedrückt werden. Die in Abbildung 3 dargestellten Structured Classifier stellen dies dar, indem sie den Blickwinkel auf individuelle Instanzen von Klassen (in diesem Fall Autos und Boote) legen. Für ein Auto bedeutet dies z. B., dass es genau einen Motor, zwei Vorder- und zwei Hinterräder besitzt und der Motor die beiden Vorderräder des Autos antreibt. Weiterhin wird laut [Boc04] sichergestellt, dass ein Motor in einem Auto nichts anderes in diesem oder einem anderen Auto oder einem Boot antreibt. Die Unterteilung in Vorder- und Hinterräder, sowie der alleinige Antrieb der Vorderräder sind in dieser Modellierung hinzugekommen, da sie sich hier adäquat darstellen lassen. In der vorangegangenen Modellierung wäre dazu eine Unterteilung der Komposition zwischen `Car` und `Wheel` in zwei Kompositionen mit den Multiplizitäten 0..1 auf Seiten von `Car` nötig. Auch hier müsste über ein zusätzliches Constraint die Anforderung realisiert werden, dass ein Rad immer zu einem Auto gehört. Die eingebetteten Rechtecke innerhalb von `Car` und `Boat` stellen keine Klas-

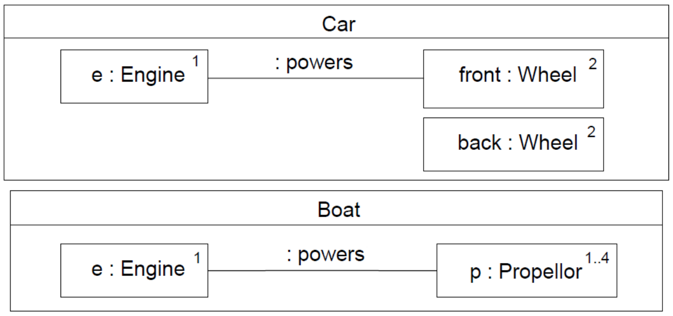

Abbildung 3: UML 2 Kompositionen

sen dar, sondern *Parts*. Die Verbindung zwischen Parts ist ein *Connector*. Im Gegensatz zu Assoziationen verbinden Konnektoren nicht zwei beliebige Instanzen der verbundenen Klassen, sondern Instanzen, die im jeweiligen Kontext als der angegebene Part auftreten (siehe [OMG09b, S. 174]).

Für die modellgetriebene Softwareentwicklung sind zusätzlich die neu hinzugekommenen *Ports* von Bedeutung, da sie eine komponentenbasierte Architektur unterstützen. Sie ermöglichen es, Komponenten zu orchestrieren, sodass eine lose Kopplung erreicht werden kann. Ports stellen Interaktionspunkte eines Classifiers dar, über die der jeweilige Classifier mit seiner Umgebung oder seinen internen Parts interagieren kann. Ein Port kann dafür bereitgestellte und benötigte Dienste spezifizieren (*Required/Provided Interfaces*). Eine detaillierte Beschreibung von Ports sprengt den Rahmen dieses Artikels und daher verweisen wir auf [HMPW04].

2 Wechselwirkungen zwischen Kompositions- und Klassendiagramm

Dass das Kompositionsdiagramm und das Klassendiagramm nicht unabhängig voneinander sind, liegt unter anderem an der Verwendung von Assoziationen als Typen von Konnektoren. Die in Abbildung 3 gezeigten Konnektoren verwenden als Typ die Assoziation `powers`, was durch den vorangestellten Doppelpunkt ausgedrückt wird; vor dem Doppelpunkt kann ein Name für den Konnektor stehen. Dies impliziert die Anwesenheit einer Assoziation zwischen `Engine` und `Wheel` sowie zwischen `Engine` und `Propeller` mit dem Namen `powers`. Damit unabhängig vom Kontext eines Motors von diesem zu seinen angetriebenen Teilen navigiert werden kann, um z. B. Zusicherungen für die an einen Motor angeschlossenen Teile definieren zu können, bietet es sich an, für die Klassen `Wheel` und `Propeller` eine gemeinsame Oberklasse zu definieren, z. B. `PowerTransmitter`. Die Assoziation `powers` kann dann zwischen `PowerTransmitter` und `Engine` definiert werden, wie es in Abbildung 4 anhand eines Screenshots aus der nächsten USE Version dargestellt ist. Dabei dürfen sich die Multiplizitäten der Assoziation und die Multiplizitäten der Konnektoren nicht widerspre-

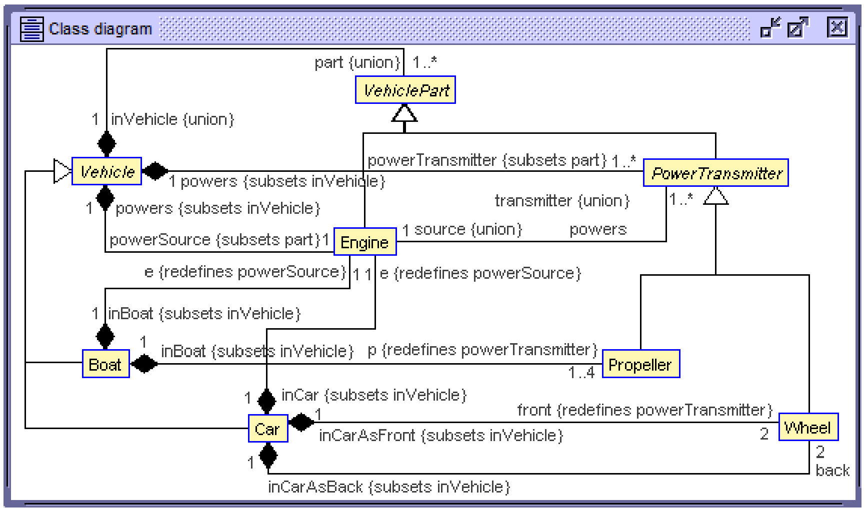

Abbildung 4: Erweitertes Klassendiagramm

chen. Die Konnektormultiplizitäten müssen entweder den gleichen oder einen kleineren Zustandsraum definieren [OL06]. Für Motoren können dann Einschränkungen definiert werden, sodass z. B. nur für die Leistung des Motors geeignete Teile angeschlossen werden dürfen:

```
context Engine inv noOverkill:
  self.transmitter->forAll(pt:PowerTransmitter |
    pt.allowedHorsePower >= self.horsePower)
```

Für komplexere Modelle müssen im Klassendiagramm zusätzliche Constraints an Assoziationsenden angegeben werden. Dies sind vor allem die in UML 2 hinzugekommenen Constraints der Form {*subsets* associationEnd}, {*redefines* associationEnd} und {*union*} (siehe [AS07], [Ame09]). Alle drei Constraints schränken die Verwendung der Assoziationsenden ein. Während *subsets* und *redefines* die Semantik eines spezialisierten Assoziationsendes verändern, verändert *union* die Semantik an einem allgemeineren Assoziationsende.

Das in Abbildung 4 dargestellte Klassendiagramm verwendet diese Möglichkeiten, um die in Abbildung 3 dargestellte Kompositionsstruktur auf Klassenebene abzubilden. Die Komposition zwischen `Vehicle` und `VehiclePart` beinhaltet alle Teile eines Gefährts. Da durch spätere Spezialisierungen verschiedene Arten von Teilen hinzukommen können, sind die Assoziationsenden als {*union*} markiert. Das bedeutet, dass das Navigationsergebnis von `part` aus dem Kontext einer `Vehicle`-Instanz die Vereinigung aller Teilmengen ist, die sich durch die Navigation über Enden mit der Markierung {*subsets part*} ergeben. Die andere Navigationsrichtung von `VehiclePart` zu `Vehicle` kann nur ein einzelnes Objekt ergeben, da das Assoziationsende eine Multiplizität von 1 hat. Für die Dokumentation macht es aber durchaus Sinn {*union*} anzugeben. Die UML Spezifikation [OMG09b] ist an dieser Stelle sehr frei interpretierbar. Während in der Spezifikation explizit erwähnt wird, dass sich die beschriebenen Constraints ausschließlich auf Assoziationsenden anwenden lassen und nicht auf Assoziationen (siehe [OMG09b, S. 40]), ergeben sich trotzdem Anforderungen an die zu den Enden gehörende Assoziation und an die gegenüberliegenden Assoziationsenden. So müssen alle Typen an den Enden einer Assoziation zueinander passend sein [OMG09a, S. 113]. Auffällig ist hierbei auch, dass die Typsicherheit nur in der UML Infrastructure gefordert wird, obwohl die jeweiligen Abschnitte über subsetting etc. in der Super- und Infrastructure nahezu identisch sind. Gar nicht in der UML Spezifikation angegeben sind Implikationen für die anderen Assoziationsenden. In [AP08] setzen sich die Autoren intensiv mit diesem Thema auseinander.

Der Unterschied von {*redefines*} und {*subsets*} kann mit der Komposition zwischen `Car` und `Wheel` für die Vorderräder veranschaulicht werden. Das Assoziationsende `front` ist mit {*redefines* powerTransmitter} und nicht mit {*subsets* powerTransmitter} markiert. Damit wird gefordert, dass alle Links zwischen einem Auto und einem kraftübertragenden Teil Räder sein müssen. Bei einer Markierung mit {*subsets*} würde stattdessen gefordert werden, dass die Menge der über die Assoziation verbundenen Räder eine Teilmenge der allgemeineren Assoziation sein müssen. Es wären also weiterhin Verbindungen zwischen Autos und anderen kraftübertragenden Teilen möglich (Schiffsschrauben fallen hier heraus, da diese an einer anderen Komposition teilnehmen müssen und eine Instanz nur an einer Komposition teilnehmen darf).

Der Vorteil der mit {*union*} gekennzeichneten Assoziationsenden gegenüber Enden ohne Markierung liegt in der möglichen automatischen Verarbeitung. Ein Modellinterpreter oder ein Codegenerator kann diese Markierung berücksichtigen und automatisch die Vereinigungsmenge berechnen bzw. Quellcode für die Berechnung erzeugen. Ohne diese Markierung muss die Berechnung manuell erfolgen und kann somit leichter zu Fehlern führen.

3 Werkzeugunterstützung

Schon das einfache Beispiel aus den vorangegangenen Abschnitten verdeutlicht, dass die Auswirkungen von Kompositionsstrukturen auf das Klassendiagramm nicht trivial sind und gut durchdacht werden müssen. Um bestimmte Sachverhalte zu validieren ist die Erzeugung von Szenarien der Modellnutzung hilfreich. Leider fehlt es hier, nicht nur im Zusammenhang mit den neuen Kompositionsstrukturen, an einer geeigneten Werkzeugunterstützung [OL06, BO06].

Das an der Universität Bremen entwickelte UML/OCL-Werkzeug USE [USE, GBR07] erlaubt es, auf Basis der UML und der OCL, Modelle zu spezifizieren und Instanziierungen (Objektdiagramme) dieser zu erzeugen. Dadurch unterstützt USE Entwickler bei der Validierung ihrer Modelle, indem die erzeugten Zustände auf ihre Gültigkeit hin überprüft und Anfragen an das Modell gestellt werden können. So erhält der Entwickler frühzeitig eine Rückmeldung über die Qualität seines Modells. Weiterhin können endliche Zustandsräume über einen Generator geprüft werden [GBR03]. Dies ermöglicht es z. B. zu überprüfen, ob zu einem spezifizierten Modell ein gültiger Zustand existiert, also die Einschränkungen auf dem Modell nicht zu stark und nicht inkonsistent sind.

Bisher werden in USE die herkömmlichen Kompositionen und Aggregationen (*black* and *white diamonds*) unterstützt. Im Rahmen der Weiterentwicklung von USE ist die Integration der neuen Kompositionsstrukturen (vor allem Structured Classifier) geplant. Die Integration ist jedoch nicht das einzig verfolgte Ziel, vielmehr sollen durch die schrittweise Implementierung evtl. vorhandene Widersprüche in der UML Spezifikation aufgedeckt und behoben, sowie fehlende Angaben definiert werden. Dass die UML Spezifikation allein für die Umsetzung nicht ausreicht, zeigte sich bereits bei der Umsetzung der vorgestellten Constraints für Assoziationsenden. So ist z. B. unklar wie sich ein Validationswerkzeug verhalten muss, falls ein mit *subsets* markiertes Assoziationsende mit einem Assoziationsende ohne eine solche Markierung verbunden ist. Abbildung 5 zeigt eine solche Situation, bei der ein Werkzeug drei Möglichkeiten hat zu reagieren:

1. Einen Fehler melden und das Modell abweisen.
2. Das Assoziationsende `c` implizit mit *subsets* markieren und damit für die Berechnung der Vereinigung am Assoziationsende `a` verwenden.
3. Das Modell ohne Änderungen akzeptieren und das Assoziationsende nicht in die Berechnung der Vereinigung einbeziehen.

Das in Abbildung 5 auf der rechten Seite gezeigte Objektdiagramm ist nur im zweiten Fall gültig. Im dritten Fall entspricht die Menge der Objekte am Assoziationsende `a` nicht der Menge aller unterteilenden (subsetting) Enden. Da es keine Assoziationsenden gibt, die `a` unterteilen, ist die Vereinigungsmenge leer. Die Navigation `d1.a` ergibt aber `Set{c1}`, was ein Widerspruch ist. Im USE System wurde die dritte Variante realisiert, um die Transparenz der Modelleigenschaften zu wahren.

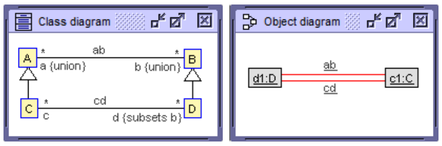

Abbildung 5: Nicht angegebenes subsets Constraint

Das vorangegangene Beispiel verdeutlicht die Möglichkeiten der statischen Validierung innerhalb eines Werkzeugs. Eine der angesprochenen Stärken von USE ist zusätzlich die Validierung eines Modells zur Ausführungszeit. Dazu können Systemzustände erzeugt und deren Struktur gegen die Einschränkungen im entsprechenden Modell geprüft werden. Eine Validierung zur Laufzeit kann Fehler aufdecken, die eine rein statische Analyse nicht findet. Ein Beispiel anhand des Klassendiagramms für Fahrzeuge verdeutlicht dies:

Fügt der Benutzer einen neuen Fahrzeugtyp hinzu, welcher eine andere Antriebsart verwendet, so muss auch eine neue redefinierende Assoziation für die Verbindung zwischen `Vehicle` und `PowerTransmitter` definiert werden, da eine einmal redefinierte Assoziation schreibgeschützt ist, um das Substitutionsprinzip zu gewährleisten [Boc04]. Wird keine redefinierende Assoziation modelliert und zur Ausführungszeit versucht ein Link zwischen dem neuen spezialisierten `Vehicle` und dessen speziellen `PowerTransmitter` zu erzeugen verhindert USE dieses. Dadurch erhält der Benutzer eine frühzeitige Rückmeldung, dass das von ihm erstellte Modell einen gewünschten Zustand nicht darstellen kann.

Durch die schrittweise Implementierung der Erweiterungen wird eine klar definierte Semantik der dazugehörigen UML Sprachelemente entwickelt, sodass mit USE ein Validierungswerkzeug mit einer integrierten Semantik von Structured Classifier und deren Elementen wie Ports, Parts, und Connectors verfügbar sein wird. Anhand von Szenarien können so z. B. Widersprüche in der Definition von Kompositionsstrukturen und Klassenmodellen aufgezeigt werden. Grundlegende Arbeiten dafür sind bereits umgesetzt, wie an den gezeigten Einschränkungen an Assoziationsenden zu sehen ist. In der nächsten USE Version wird es volle Unterstützung von *subsets*, *union* und *redefines* geben.

4 Fazit und Ausblick

Die in UML 2 hinzugekommenen Sprachelemente erweitern die Ausdrucksmöglichkeit der UML im Hinblick auf eine komponentenbasierte Architektur. Wie in diesem Artikel gezeigt, ist die UML Spezifikation allerdings nicht an allen Stellen eindeutig. Wir haben gezeigt, dass grundlegende Arbeiten zur Unterstützung von Structured Classifier im USE System umgesetzt worden sind und welche Interpretationsspielräume die UML Spezifikation einem Werkzeugentwickler lässt.

Mittelfristig soll in USE die Definition von Kompositionsstrukturen und deren Instanziierung möglich sein, so dass Inkonsistenzen zwischen diesen und dem Klassenmodell erkannt werden können. Interessant wird dabei auch die Frage sein, inwieweit bestimmte Angaben im Kompositionsdiagramm automatisch in das Klassenmodell übernommen werden können. Weiterhin ist die Integration von OCL in den Kontext der Kompositionsstrukturen geplant, bei der sich unter anderem die Frage stellt, ob z. B. Ansätze wie Komponenteninvarianten [HBKW01] übernommen werden können bzw. müssen. Die vollständige Integration des Konzepts der Kompositionsstrukturen mit der Unterstützung von Ports und Protokollautomaten (Protocol State Machines) ist das langfristige Ziel der USE-Entwicklung, sodass komponentenbasierte Modelle validiert werden können.

Literatur

[Ame09] C. Amelunxen. *Metamodel-based Design Rule Checking and Enforcement*. Dissertation, Technische Universität Darmstadt, 2009. Dissertation.

[AP08] Marcus Alanen und Ivan Porres. A metamodeling language supporting subset and union properties. *Software and Systems Modeling*, 7(1):103–124, feb 2008.

[AS07] C. Amelunxen und A. Schürr. Formalizing Model Transformation Rules for

UML/MOF 2. *IET Software Journal*, 2(3):204–222, June 2007. Special Issue: Language Engineering.

[BO06] J. M. Bruel und I. Ober. Components Modeling in UML 2. *Studia Jurnal*, 1:79–90, 2006.

[Boc04] Conrad Bock. UML 2 Composition Model. *Journal of Object Technology*, 3(10):47–73, Between November and December 2004.

[GBR03] Martin Gogolla, Jörn Bohling und Mark Richters. Validation of UML and OCL Models by Automatic Snapshot Generation. In Grady Booch, Perdita Stevens und Jonathan Whittle, Hrsg., *Proc. 6th Int. Conf. Unified Modeling Language (UML'2003)*, Seiten 265–279. Springer, Berlin, LNCS 2863, 2003.

[GBR07] Martin Gogolla, Fabian Büttner und Mark Richters. USE: A UML-Based Specification Environment for Validating UML and OCL. *Science of Computer Programming*, 69:27–34, 2007.

[HBKW01] Rolf Hennicker, Hubert Baumeister, Alexander Knapp und Martin Wirsing. Specifying Component Invariants with OCL. In *GI Jahrestagung (1)*, Seiten 600–607, 2001.

[HMPW04] Øystein Haugen, Birger Møller-Pedersen und Thomas Weigert. *Structural Modeling with UML 2.0*, Kapitel 3, Seiten 53–76. Springer US, 2004.

[OL06] Ian Oliver und Vesa Luukala. On UML's Composite Structure Diagram. In *Fifth Workshop on System Analysis and Modelling*, Kaiserslautern, Germany, 2006.

[OMG06] *Object Constraint Language 2.0*. Object Management Group (OMG), Mai 2006. `http://www.omg.org/spec/OCL/2.0`.

[OMG09a] *UML Infrastructure 2.2*. Object Management Group (OMG), Februar 2009. `http://www.omg.org/spec/UML/2.2/Infrastructure/PDF/`.

[OMG09b] *UML Superstructure 2.2*. Object Management Group (OMG), Februar 2009. `http://www.omg.org/spec/UML/2.2/Superstructure/PDF`.

[SVEH07] Thomas Stahl, Markus Völter, Sven Efftinge und Arno Haase. *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. Dpunkt Verlag, 2007.

[USE] A UML-based Specification Environment. Internet. `http://sourceforge.net/projects/useocl/`.

[WK03] J. Warmer und A. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 2003. 2nd Edition.