Konsistenzmanagement von objektorientierten Verhaltensmodellen

Jochen M. Küster

IBM Zurich Research Laboratory, CH-8803 Rüschlikon, Schweiz
jku@zurich.ibm.com

Abstract: In der modellbasierten Softwareentwicklung wird ein Softwaresystem durch verschiedene Teilmodelle auf unterschiedlichen Abstraktionsstufen und aus verschiedenen Sichten modelliert. Das Ziel von Konsistenzmanagement im Entwicklungsprozess ist es, eine gewisse Form von Konsistenz der Teilmodelle zu gewährleisten. Da objektorientierte Verhaltensmodelle keine eindeutige formale Semantik haben und sehr unterschiedlich eingesetzt werden, ist ein Konsistenzmanagement dieser Modelle besonders schwierig. In der Arbeit werden sowohl Grundlagen als auch Techniken für ein systematisches Konsistenzmanagement entwickelt und am praktischen Beispiel erprobt.

1 Einführung

Das Erstellen und Analysieren von Modellen zur Qualitätsverbesserung ist in vielen Ingenieursdisziplinen fester Bestandteil des Entwicklungsprozesses. In der Softwaretechnik hat sich die objektorientierte Modellierung mit der Unified Modeling Language (UML) [UML03] zur festen Größe auch im industriellen Umfeld entwickelt.

In objektorientierter Analyse und Entwurf wird ein Softwaresystem mit Hilfe verschiedener Teilmodelle auf unterschiedlichen Abstraktionsebenen und aus verschiedenen Sichten beschrieben. So werden beispielsweise Klassendiagramme zur Strukturbeschreibung eingesetzt und Verhaltensdiagramme wie Statecharts oder Sequenzdiagramme zur Beschreibung des Verhaltens. Dabei wird in einem Entwicklungsprozess üblicherweise ein aus verschiedenen Teilmodellen bestehendes Modell iterativ verfeinert.

Teilmodelle eines Modells und auch Modelle verschiedener Abstraktionsstufen sind nicht voneinander unabhängig. Sie können Teile des Systems widersprüchlich beschreiben, wenn nicht Maßnahmen zur Konsistenzsicherstellung der Modelle durch ein Konsistenzmanagement im Entwicklungsprozess getroffen werden. Dabei ist das erforderliche Konsistenzmanagement von der eingesetzten Modellierungssprache abhängig.

Die Modellierungssprache UML integriert verschiedene Teilsprachen und ermöglicht daher die Konstruktion von Modellen, die aus verschiedenen Teilmodellen bestehen. Die Syntax der Sprache UML wird durch ein Metamodell definiert, ihre Semantik durch eine informale Beschreibung. Durch das Metamodell wird eine eingeschränkte Grundform von Konsistenz der Modelle definiert. Konsistenz der Teilmodelle bezüglich ihrer Seman-

tik wie beispielsweise die Konsistenz des Verhaltens, das in Sequenzdiagrammen und Statecharts modelliert wird, wird in der UML Sprachdefinition nicht definiert. Die Aufgabe der Konsistenzsicherstellung wird daher dem eigentlichen Softwareentwickler auferlegt.

Im Allgemeinen wird Konsistenz in der Softwaretechnik durch eine geeignete Abbildung der Teilmodelle in einen gemeinsamen semantischen Bereich, definiert durch eine Sprache mit formaler Syntax und Semantik, sichergestellt. So definieren beispielsweise Zave et al. [ZJ93] Konsistenz durch eine Abbildung in Logik. Dieser Ansatz erfordert jedoch die aufwändige Definition einer logikbasierten Semantik der Modelle, die bei der UML nicht vorhanden ist. Moreira und Clark [MC94] übersetzen objektorientierte Modelle in die formale Sprache LOTOS und benutzen existierende LOTOS Model Checker zum Auffinden von Inkonsistenzen. Grosse-Rhode [GR01] integriert verschiedene Teilmodelle mit Hilfe des formalen Bereichs der Transformationssysteme. Hierfür stehen jedoch keine Werkzeuge zur automatischen Überprüfung bereit und der Ansatz ist somit in der Praxis nicht direkt einsetzbar. Bezüglich von UML-basierten Modellen gibt es eine Reihe von Ansätzen, die sich jeweils mit einem speziellen Konsistenzproblem befassen. Hier sei Li et al. [LL99] genannt, die das Konsistenzproblem von Zeitbedingungen in UML-Sequenzdiagrammen behandeln und die Konsistenz durch eine Abbildung in ein System von linearen Ungleichungen definieren.

Trotz dieser Vorarbeiten bleibt es jedoch unklar, wie grundsätzlich Konsistenz von objektorientierten Modellen in einem Entwicklungsprozess sichergestellt werden kann. Die Konsistenzsicherstellung ist insbesondere schwierig, da

- die Semantik der Teilmodelle wie Sequenzdiagramme und Statecharts in der UML nicht formal definiert ist,
- es keinen eindeutigen Entwicklungsprozess gibt. Vielmehr werden objektorientierte Modelle sehr unterschiedlich in verschiedenen Entwicklungsprozessen eingesetzt. Auch innerhalb eines Entwicklungsprozesses werden Modelle auf unterschiedlichen Abstraktionsstufen mit unterschiedlicher Semantik eingesetzt. Beispielsweise werden Sequenzdiagramme zur Beschreibung von Anwendungsfällen in der Anforderungsdefinition und auch zur Beschreibung von Testfällen verwendet.
- zunehmend objektorientierte Modelle in verschiedenen Anwendungsbereichen eingesetzt werden, in denen dann unterschiedliche Konsistenzeigenschaften von Interesse werden.

Diese Beobachtungen sind die Ausgangsbasis für die diesem Dokument zugrundeliegende Dissertation [Küs04]. In der Dissertation wird Konsistenzmanagement von objektorientierten Verhaltensmodellen untersucht und folgende wesentlichen Ergebnisse erarbeitet:

- Ein generischer Begriff der Konsistenz wird für den Bereich der Softwaretechnik etabliert, so dass die Vielzahl der auftretenden Konsistenzprobleme klassifiziert werden können,

- ein Ebenenmodell für Konsistenzmanagement wird entwickelt, aus dem Anforderungen für ein Konsistenzmanagement von objektorientierten Verhaltensmodellen abgeleitet werden können,
- eine allgemein anwendbare Methodik für Konsistenzmanagement wird definiert, die es erlaubt, einen beliebigen Entwicklungsprozess so zu verbessern, dass konsistente Modelle erzeugt werden,
- ein Ansatz zur Abbildung von UML Modellen in formale Sprachen zur Verifikation von Konsistenzbedingungen wird entwickelt,
- die Grundzüge eines Konsistenzmanagementwerkzeuges werden beschrieben und in einem Forschungsprototypen realisiert.

Die weitere Struktur dieses Beitrages ist wie folgt: Zunächst erklären wir in Abschnitt 2 die Grundproblematik von Konsistenz in modellbasierter Softwareentwicklung, klären Begriffe und entwickeln ein Ebenenmodell für Konsistenzmanagement. In Abschnitt 3 fassen wir die Aktivitäten der allgemein anwendbaren Methodik zur Definition des Ebenenmodells für Konsistenzmanagement zusammen. Schließlich führen wir in die Technik der Modelltransformationen ein, die zur Abbildung von Modellen in formale Sprachen genutzt werden.

2 Konsistenz und Konsistenzmanagement

Die Modellierung eines Systems aus verschiedenen Sichten und auf verschiedenen Abstraktionsstufen im Softwareentwurf führt zwangsläufig zum Auftreten von Konsistenzproblemen. Abbildung 1 zeigt hierzu zwei solche Teilmodelle, die im objektorientierten Entwurf verwendet werden: Ein Sequenzdiagram beschreibt Interaktionen zwischen einer Menge von Instanzen. Weiterhin werden Interaktionen zwischen zwei Instanzen auch durch Protokolle in Statecharts modelliert.

In der Softwaretechnik wurde schon früh erkannt, dass vollständige Konsistenz nur mit großem Aufwand erreicht werden kann [Bal91]. Andererseits ist das Erkennen von möglichen Inkonsistenzen sehr wichtig, da anderenfalls z. B. die Codegenerierung aus verschiedenen inkonsistenten Teilmodellen zu einem System führt, das nicht korrekt arbeitet. Ein weiteres mögliches Problem durch Inkonsistenzen ist, dass das System die Anforderungen nicht erfüllt, die in einem Teilmodell modelliert sind. Beispielsweise könnte in Abbildung 1 das Protokoll, das durch das Statechart spezifiziert wird, ein Teil einer fundamentalen Anforderung sein.

Gegenwärtig werden Modelle der Sprache UML in der Softwaretechnik sehr unterschiedlich eingesetzt, und die Aufgabe des Konsistenzmanagements wird dem einzelnen Entwickler übertragen. So wird das obige Problem weder durch die Sprachdefinition noch durch kommerzielle Werkzeuge behandelt.

Im Allgemeinen werden verschiedene Teilmodelle in der Softwaretechnik als konsistent aufgefasst, wenn sie in ein einzelnes Modell mit einer wohldefinierten Semantik integriert

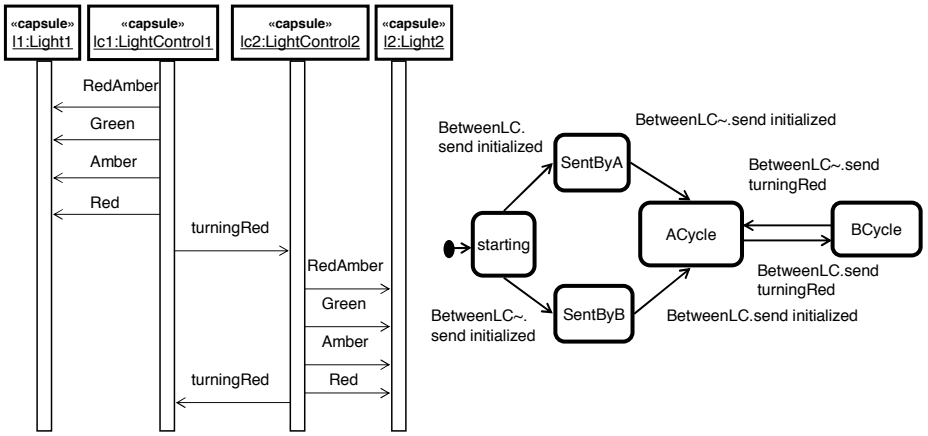

Abbildung 1: Beispiel für ein Konsistenzproblem

werden können [1]. Die geforderte Form der Integration hängt dabei vom Typ des Modells, dem Entwicklungsprozess und dem Anwendungsbereich ab. Wir können zwischen Konsistenzbedingungen und Konsistenzkonzepten wie folgt unterscheiden:

- Eine Konsistenzbedingung ist ein Prädikat, das definiert, ob ein Modell bezüglich dieser Konsistenzbedingung konsistent ist. Im Allgemeinen kann zwischen Konsistenzbedingungen auf der Syntax und der Semantik eines Modells unterschieden werden. Konsistenzbedingungen auf der Syntax können beispielsweise in der UML durch OCL Bedingungen ausgedrückt werden. Semantische Konsistenzbedingungen erfordern die Definition einer semantischen Abbildung in einen geeigneten semantischen Bereich und die Definition von Bedingungen in diesem Bereich.
- Verschiedene Konsistenzbedingungen können zu einem Konsistenzkonzept gruppiert werden.

Diese Herangehensweise an die Definition von Konsistenzkonzepten ist notwendig, weil für ein und dasselbe Modell jeweils unterschiedliche Konsistenzkonzepte, abhängig vom Entwicklungsprozess und vom Anwendungsbereich, denkbar sind.

Die Motivation für die Definition eines Konsistenzkonzepts bezeichnen wir als Konsistenzeigenschaft. Eine Konsistenzeigenschaft ist eine informale, modellunabhängige Beschreibung einer Eigenschaft, die ein aus Teilmodellen bestehendes Modell besitzen soll. Eine Konsistenzeigenschaft kann informal dadurch charakterisiert werden, indem man beschreibt, was durch sie sichergestellt wird. Beispiele für Konsistenzeigenschaften sind syntaktische Korrektheit, Verhaltenskonsistenz oder Zeitkonsistenz.

[1] vgl. dazu Kapitel 3 der Dissertation

Auf der Basis von Konsistenzeigenschaften und den dazugehörigen Konsistenzkonzepten können Konsistenzchecks definiert werden. Ein Konsistenzcheck ist eine operationale Beschreibung der notwenigen Aktivitäten, die zum Prüfen von Konsistenzbedingungen auf einem Modell erforderlich sind. Konsistenzchecks können durch Model Checker, Theorembeweiser oder durch entsprechende Algorithmen realisiert werden und auch Modelltransformationen der Modelle in semantische Bereiche einschließen, um das Prüfen von semantischen Konsistenzbedingungen zu realisieren.

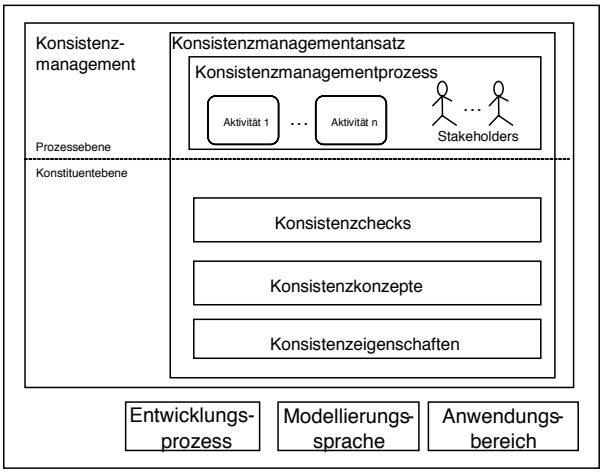

Abbildung 2: Das Ebenenmodell für Konsistenzmanagement

Die Definition der Begriffe Konsistenzeigenschaft, Konsistenzkonzept, Konsistenzbedingung und Konsistenzcheck bildet die Basis zur Definition von Konsistenzmanagement[2]. Konsistenzmanagement umfasst dabei auch die Definition eines Konsistenzmanagementprozesses, der beschreibt, wie in einem Entwicklungsprozess durch Ausführung geeigneter Aktivitäten Konsistenz erreicht wird. Wir können daher unterschiedliche Ebenen unterscheiden (siehe Abbildung 2):

Gegeben einen Entwicklungsprozess, eine Modellierungsprache und einen Anwendungsbereich, erfordert ein umfassendes Konsistenzmanagement zunächst die informale Beschreibung der Konsistenzeigenschaften. Diese können dann zur Definition von Konsistenzkonzepten und Konsistenzchecks genutzt werden. Weiterhin müssen Aktivitäten definiert und in den bestehenden Entwicklungsprozess integriert werden. Diese Aktivitäten beinhalten die Ausführung entsprechender Konsistenzchecks und legen fest, wie mit Inkonsistenzen umgegangen werden soll.

Dieses Ebenenmodell für Konsistenzmanagement wird in der Dissertation zur Ableitung von Anforderungen genutzt, die zur Bewertung existierender Ansätze von Konsistenzmanagement herangezogen werden. Weiterhin bildet es die Basis für die nachfolgend beschriebene allgemeine Methodik.

[2]Unter Konsistenzmanagement verstehen wir hier den Bereich der Softwaretechnik, der sich mit Techniken zur Konsistenzsicherstellung befasst.

3 Eine allgemeine Methodik für Konsistenzmanagement

Der Ansatz für die Methodik ist der folgende: Die grundlegende Frage, die bei der Verwendung von verschiedenen Teilmodellen und einer gegebenen Konsistenzeigenschaft beantwortet werden muss, ist, ob eine Integration der Teilmodelle existiert, die die Konsistenzeigenschaft erfüllen. Obwohl es keine formal definierte Semantik von UML Modellen gibt, kann zur Definition von Konsistenz eine geeignete Formalisierung konstruiert werden. Hierbei ist es möglich, sich auf die Aspekte, die zum Konsistenzproblem beitragen, zu beschränken.

Das entwickelte Ebenenmodell stellt eine systematische Basis für eine allgemeine Methodik für Konsistenzmanagement bereit, wenn es gelingt, dieses in Abhängigkeit von einem konkreten Entwicklungsprozess, einer Modellierungssprache und einem Anwendungsbereich aufzubauen. Dazu werden im Folgenden eine Reihe von Aktivitäten definiert[3]:

Aktivität 1: Identifikation von Typen von Konsistenzproblemen Das Ziel dieser Aktivität ist die Identifikation aller relevanten Typen von Konsistenzproblemen, die im Entwicklungsprozess auftreten. Die Basis für diese Identifikation ist eine systematische Untersuchung der Teilmodelle bezüglich der Modellierungsaspekte, die sie unterstützen. Wenn verschiedene Teilmodelle gleiche oder verwandte Aspekte unterstützen oder es eine Überlappung der Konsistenzeigenschaft und eines Aspektes gibt, können Konsistenzprobleme auftreten. Diese Konsistenzprobleme müssen identifiziert, klassifiziert und informal beschrieben werden.

Aktivität 2: Formalisierung von Typen von Konsistenzproblemen Diese Aktivität zielt auf die Konstruktion einer geeigneten Formalisierung eines Konsistenzproblems ab. Als erstes muss ein geeigneter formaler semantischer Bereich gewählt werden. Ein formaler semantischer Bereich muss als Anforderung zunächst die Aspekte der Teilmodelle unterstützen. Weiterhin muss es möglich sein, die informal beschriebenen Konsistenzbedingungen zu formalisieren. Schließlich muss entsprechende Werkzeugunterstützung für die Verifikation von Konsistenzbedingungen vorhanden sein. Nach der Auswahl des semantischen Bereichs müssen die Teilmodelle in diesen abgebildet und formale Konsistenzbedingungen definiert werden.

Aktivität 3: Operationale Spezifikation von Modelltransformationen Jedes formale Konsistenzkonzept muss so transformiert werden, dass die Teilmodelle automatisch in den semantischen Bereich abgebildet werden können. Zu diesem Zweck müssen Modelltransformationen für die Teilmodelle definiert werden.

Aktivität 4: Spezifikation von Konsistenzchecks Für jeden Typ von Konsistenzproblem muss ein Konsistenzcheck spezifiziert werden, der die Konsistenzbedingungen überprüft. Außerdem muss definiert werden, was beim Auftreten von Inkonsistenzen geschehen soll.

[3] Grundzüge der Methodik wurden in [EKGH01] vorgestellt.

Aktivität 5: Einbettung von Konsistenzmanagement in den Entwicklungsprozess Für jeden Typ von Konsistenzproblem muss entschieden werden, wann er im Entwicklungsprozess behandelt werden soll. Die Reihenfolge der Konsistenzchecks muss festgelegt werden und entsprechend Konsistenzmanagementaktivitäten definiert werden. Diese beinhalten das Ausführen der Konsistenzchecks sowie die Behandlung von Inkonsistenzen. Die Konsistenzmanagementaktivitäten müssen in den existierenden Entwicklungsprozess integriert werden.

Das Ergebnis dieser Aktivitäten der Methodik ist ein Entwicklungsprozess der einen konkreten Konsistenzmanagementprozess enthält. In der Dissertation werden die einzelnen Aktivitäten durch klare Anweisungsschritte verfeinert. Weiterhin werden zur Unterstützung verschiedene Techniken entwickelt.

Wir besprechen nun eine mögliche Anwendung der allgemeinen Methodik auf die in Abb. 1 dargestellte Situation, unter der Annahme, dass diese in der Aktivität 1 als mögliches Konsistenzproblem erkannt worden ist. Als (informale) Konsistenzbedingung fordern wir, dass das Verhalten, welches im Statechart beschrieben wird, in Sequenzdiagrammen eingehalten wird.

In Aktivität 2 würde nun dieses Konsistenzproblem formalisiert werden. Dazu müsste ein geeigneter semantischer Bereich gewählt werden, in dem die Konsistenzbedingung formal ausgedrückt werden kann, beispielsweise eine Prozessalgebra wie die Sprache CSP [Hoa85]. Teil der Formalisierung wäre auch die Definition einer Abbildung von Sequenzdiagrammen und Statecharts in diese Sprache. In der Aktivität 3 würde diese Abbildung als Modelltransformation automatisiert und in Aktivität 4 würde auf Basis dieser Modelltransformation und durch Anbindung des existierenden Model Checkers FDR [For97] ein entsprechender Konsistenzcheck definiert. Schließlich würde man in Aktivität 5 eine Konsistenzmanagementaktivität, die das Ausführen des Konsistenzchecks beinhaltet, in einen (existierenden) Entwicklungsprozess integrieren.

Offensichtlich ist die Durchführung der Aktivitäten der Methodik aufwendig. Es ist jedoch möglich, einige der Aktivitäten durch entsprechende Werkzeuge zu unterstützen. Dazu wurde im Rahmen einer Projektgruppe das Werkzeug *Consistency Workbench* [EHK03] entwickelt. Außerdem kann durch Einführung geeigneter Standards der Austausch und die Wiederverwendung von bereits erfolgreich eingesetzten Konsistenzkonzepten (inklusive der sehr aufwändigen Modelltransformationen) und der darauf basierenden Konsistenzchecks ermöglicht werden.

Im Folgenden werden wir uns auf die Definition und Ausführung von regelbasierten Modelltransformationen konzentrieren, die in Aktivität 3 benötigt werden.

4 Definition und Ausführung von regelbasierten Modelltransformationen

Ein wesentlicher Bestandteil der Methodik ist die Abbildung von Modellen in formale semantische Bereiche. Gegenwärtig erlangen Modelltransformationen auch im Rahmen der modellgetriebenen Architektur vermehrte Aufmerksamkeit [MDA03].

In der Dissertation wird eine Form der Modelltransformation entwickelt, die es dem Softwareingenieur erlaubt, Modelltransformationen von UML Modellen in geeignete semantische Bereiche regelbasiert und implementierungsunabhängig zu spezifizieren. Wesentliche Anforderungen an die Technik zur Spezifikation von Modelltransformation sind, dass sie auf Modellierungsebene (d. h. ohne Programmierung) definiert werden, dass Eigenschaften wie Konfluenz analysiert werden können und dass sie automatisch ausgeführt werden können.

Im entwickelten Ansatz wird eine Modelltransformation mit einer Reihe von so genannten zusammengesetzten Regeln [KHE03] definiert. Eine zusammengesetzte Regel zur Übersetzung von UML Modellen in CSP Modelle besteht aus einem UML und einem CSP Teil. In Abbildung 3 sind zwei zusammengesetzte Regeln zur Übersetzung von UML Statecharts nach CSP dargestellt. Der UML Teil besteht aus einem UML Muster spezifiziert auf Metamodellebene und beschreibt üblicherweise Ausschnitte aus dem Metamodell (abstrakte Syntax). So ist L_S dem Metamodell für Statecharts entnommen. Der CSP Teil besteht aus einer textuellen Ersetzungsregel. Diese beschreibt, welcher Teil des CSP Modells bei Anwendung der Regel entsprechend ersetzt wird. Gekoppelt sind die beiden Teile durch die Verwendung von gemeinsamen Variablen. Weiterhin können CSP Regeln Nichtterminale enthalten, die durch nachfolgende Regeln ersetzt werden. Variablen werden durch Klammern gekennzeichnet (beispielsweise ist $< SMName >$ eine gemeinsame Variable).

Die Anwendung einer solchen zusammengesetzten Regel erfolgt in folgenden Schritten:

1. Das UML Modell wird nach einem Auftreten des UML Musters durchsucht.
2. Bei Auftreten des Musters werden die Variablen mit Werten belegt.
3. Die CSP Regel wird mit den Werten der Variablen instanziiert.
4. Das CSP Modell wird nach einem Auftreten der linken Seite der CSP Regel durchsucht.
5. Bei Auftreten der linken Seite wird im CSP Modell die linke Seite durch die rechte Seite der Regel ersetzt.

Bei der Definition einer Modelltransformation mit Hilfe eines regelbasierten Ansatzes müssen grundsätzlich die Terminierung und die Konfluenz des Transformationssystems sichergestellt werden. Im Allgemeinen ist der Aspekt der Terminierung unentscheidbar. Jedoch ist es möglich, bei Gewährleistung dieser die Konfluenz mit Hilfe von so genannten kritischen Paaren zu berechnen. In der Dissertation werden dazu Modelltransformation mit Hilfe der Theorie der getypten Graphtransformationssysteme formalisiert und eine

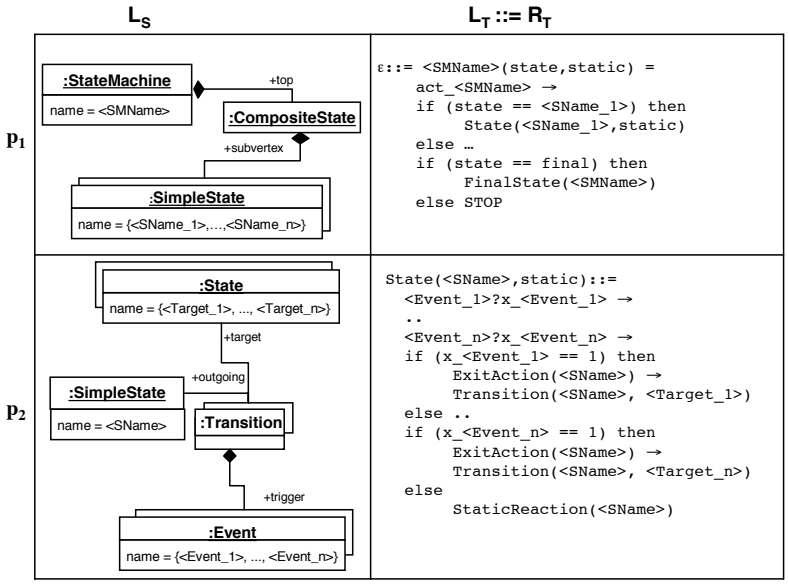

Abbildung 3: Beispielregel aus der Modelltransformation

Reihe von hinreichenden Bedingungen definiert, die Konfluenz und Terminierung sicherstellen. Diese Bedingungen können von dem Softwareentwickler bei der Definition der Regeln überprüft werden. Eine solche Bedingung ist beispielsweise, dass einen Regel nur höchstens einmal auf ein Modell angewandt werden kann.

Das beschriebene Konzept für die Definition der Modelltransformationen wurde für Statecharts nach CSP praktisch erprobt. Dies hat zur Definition von ungefähr 60 Regeln geführt, die in [HKB+03] dokumentiert sind. Praktisch wurden diese Regeln in der Consistency Workbench definiert. Auf dieser Basis wurden anschließend Konsistenchecks definiert und in der Consistency Workbench ausgeführt. Verfeinerungsbedingungen wurden durch die Anbindung des Model Checkers FDR überprüft und die Ergebnisse in der Workbench dargestellt.

Literatur

[Bal91] R. Balzer. Tolerating Inconsistency. In *Proceedings of 13th International Conference on Software Engineering, Austin, Texas*, Seiten 158–163, Mai 1991.

[EHK03] G. Engels, R. Heckel und J. M. Küster. The Consistency Workbench - A Tool for Consistency Management in UML-based Development. In P. Stevens, J. Whittle und G. Booch, Hrsg., *UML 2003 - The Unified Modeling Language. 6th International Conference, Proceedings*, Jgg. 2863 of *LNCS*, Seiten 356–359. Springer-Verlag, Oktober 2003.

[EKGH01] G. Engels, J. M. Küster, L. Groenewegen und R. Heckel. A Methodology for Specifying and Analyzing Consistency of Object-Oriented Behavioral Models. In V. Gruhn, Hrsg., *Proceedings of the 8th European Software Engineering Conference (ESEC)*, Seiten 186–195. ACM Press, Oktober 2001.

[For97] Formal Systems Europe (Ltd). *Failures-Divergence-Refinement: FDR2 User Manual*, 1997.

[GR01] M. Grosse-Rhode. Integrating Semantics for Object-Oriented System Models. In F. Orejas, P. G. Spirakis und J. van Leeuwen, Hrsg., *Proceedings of ICALP'01*, LNCS 2076, Seiten 40–60. Springer-Verlag, Juni 2001.

[HKB+03] R. Heckel, J. M. Küster, N. Bandener, B. Gueldali, I. Jahnich, C. Koepke und M. Weking. Automatische Qualitätssicherung von UML Modellen. Bericht der Projektgruppe. Technischer Bericht TR-RI-03-245, Universität Paderborn, Dezember 2003.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[KHE03] J. M. Küster, R. Heckel und G. Engels. Defining and Validating Transformations of UML Models. In J. Hosking und P. Cox, Hrsg., *Proceedings of IEEE Symposium on Human Centric Computing Languages and Environments (HCC 2003)*, Seiten 145–152. IEEE Computer Society, Oktober 2003.

[Küs04] J. M. Küster. *Consistency Management of Object-Oriented Behavioral Models*. Dissertation, Universität Paderborn, März 2004.

[LL99] X. Li und J. Lilius. Timing Analysis of UML Sequence Diagrams. In Robert France und Bernhard Rumpe, Hrsg., *UML'99 - The Unified Modeling Language. Second International Conference, Proceedings*, Jgg. 1723 of *LNCS*, Seiten 661–674. Springer-Verlag, Oktober 1999.

[MC94] A. Moreira und R. Clark. Combining Object-Oriented Modeling and Formal Description Techniques. In M. Tokoro und R. Pareschi, Hrsg., *Proceedings of ECOOP'94*, Jgg. 821 of *LNCS*, Seiten 344 – 364. Springer-Verlag, 1994.

[MDA03] Object Management Group. *MDA Guide Version 1.0.1*, Juni 2003.

[UML03] Object Management Group (OMG). *UML 2.0 Superstructure Final Adopted Specification. OMG document pts/03-08-02*, August 2003.

[ZJ93] P. Zave und M. Jackson. Conjunction as Composition. *ACM Transactions on Software Engineering and Methodology*, 2(4):379–411, Oktober 1993.

Jochen Malte Küster wurde am 11. Mai 1974 in Bielefeld geboren. Nach dem Abitur studierte er im akademischen Jahr 1993/94 Mathematics and Computer Science am University College London. Danach wechselte er zur Universität Paderborn. Nach einem von dem DAAD geförderten Aufenthalt an der Carleton University in Kanada im akademischen Jahr 1998/99 schloss er im Januar 2000 sein Studium der Informatik mit Nebenfach Mathematik ab. Anschließend war er bis Juni 2004 als wissenschaftlicher Mitarbeiter im C-LAB und am Lehrstuhl für Datenbank- und Informationssysteme an der Universität Paderborn beschäftigt, wo er im März 2004 promovierte. Im Juli 2004 wechselte Jochen Küster als Postdoktorant zum IBM Forschungslabor in Rüschlikon, Schweiz. Seit Dezember 2004 ist er dort festangestellter Mitarbeiter und beschäftigt sich mit Modelltransformationen und ihrer Anwendung auf Geschäftsprozesse.