Validierung von UML-Modellen und OCL-Bedingungen

Mark Richters and Martin Gogolla

Universität Bremen, FB 3 Informatik
Postfach 330440, 28334 Bremen
{mr|gogolla}@informatik.uni-bremen.de,
WWW home page: http://www.db.informatik.uni-bremen.de

Zusammenfassung Die Unified Modeling Language (UML) ist ein allgemein akzeptierter Standard für die Modellierung von Software-Systemen und wird als solcher von einer großen Anzahl von CASE-Tools unterstützt. Oft weisen UML-Werkzeuge nur wenig Unterstützung für die Validierung von Modellen in frühen Entwurfsphasen auf. Es gibt im allgemeinen auch keine nennenswerte Unterstützung für Integritätsbedingungen, die in der Object Constraint Language (OCL) geschrieben sind. Wir stellen daher einen Ansatz für die Validierung von UML-Modellen und OCL-Bedingungen vor, der diese Unterstützung bereitstellt. Das USE-Werkzeug (UML-based Specification Environment) bietet einen Animator zum Simulieren von UML-Modellen und einen OCL-Interpreter für die Überprüfung von Bedingungen. Einzelne Zustände eines laufenden Systems können untersucht und auf Übereinstimmung mit dem Modell überprüft werden.

1 Einleitung

Die Unified Modeling Language (UML) [9] ist allgemein als Standard für die Modellierung von Softwaresystemen anerkannt. Viele CASE-Tools erleichtern das Erstellen und Dokumentieren von UML-Diagrammen. Oft wird außerdem automatische Code-Generierung und Reverse Engineering angeboten. Häufig weisen diese Werkzeuge jedoch nur wenig Unterstützung für die Validierung von Modellen in frühen Entwurfsphasen auf. Es gibt im allgemeinen auch keine Unterstützung für Integritätsbedingungen, die in der Object Constraint Language (OCL) spezifiziert sind. Die Übersetzung von OCL-Bedingungen in Programmcode als Teil der Code-Generierung erscheint zwar generell möglich, dennoch betrachten wir die Validierung von Modellen und zusätzlichen Bedingungen als eine Aufgabe, die möglichst noch vor der Implementierungsphase durchgeführt werden sollte. Fehler im Entwurf können dadurch früher entdeckt und rechtzeitig korrigiert werden.

Wir stellen im folgenden einen Ansatz zur Validierung von UML-Modellen und OCL-Bedingungen vor, der auf Animation basiert. Dazu haben wir ein Werkzeug entwickelt, das Entwickler in diesem Prozeß unterstützt. Die wesentlichen Komponenten des USE-Tools (UML-based Specification Environment) sind ein Animator für die Simulation von UML-Modellen und ein OCL-Interpreter zum Überprüfen der Bedingungen. Ein UML-Modell wird hierbei als Beschreibung eines Systems betrachtet. Systemzustände sind Ausschnitte (Schnappschüsse) eines laufenden Systems zu bestimmten Zeitpunkten. Diese können manipuliert, untersucht und auf Übereinstimmung mit dem Modell geprüft werden. Das Werkzeug setzt Ideen und Ergebnisse um, die aus der Arbeit zur Formalisierung von OCL und der Entwicklung eines Metamodells für OCL entstanden sind [11, 12].

Das Validierungswerkzeug kann allgemein auf Modelle aus beliebigen Anwendungsbereichen angewandt werden. Als speziellen Fall haben wir es auf Teile des UML 1.3-Metamodells und dessen Wohlgeformtheits-Regeln angewandt [13]. Damit konnte eine gründliche und systematische Prüfung der OCL-Bedingungen im UML-Standard durchgeführt werden. Als praktische Konsequenz ergibt sich daraus, daß z. B. die Wohlgeformtheit von beliebigen Modellen hinsichtlich des UML-Standards automatisch geprüft werden kann, indem diese Modelle als Instanz des UML-Metamodells betrachtet und validiert werden.

Gegenwärtig sind nur wenige Werkzeuge erhältlich, die speziell zum Analysieren von UML-Modellen und OCL-Bedingungen geeignet sind. Als Ursache kommt hierfür vermutlich die nicht genügend präzise Formulierung der UML- und OCL-Semantik in Betracht. Eine wohldefinierte Semantik ist aber Grundvoraussetzung für die Entwicklung von Werkzeugen zur Analyse von Modellen. Beispiele für Arbeiten zur Definition einer präzisen Semantik sind [3, 11]. Einige verwandte Arbeiten im Bereich Werkzeugunterstützung für die Überprüfung von UML-Modellen sind die folgenden. Alcoa ist ein Werkzeug für die Analyse von Objektmodellen [5]. Alcoa verwendet nicht OCL, sondern die Sprache Alloy, die auf Z basiert. RTUML [7] ist eine UML-Variante, die besonders auf die Entwicklung echtzeitfähiger Systeme ausgerichtet ist. Es bietet eine Methode für die werkzeuggestützte Verifikation von Entwurfsaspekten mittels PVS. An der Universität Dresden wurde ein OCL-Compiler, ein Code-Generator und eine Laufzeitbibliothek in Java entwickelt [4]. Eine erste Version eines kommerziellen Werkzeugs für die Animation von UML-Modellen mit OCL-Unterstützung ist unter dem Namen ModelRun von Boldsoft erhältlich [1].

Die nachfolgenden Abschnitte sind wie folgt aufgebaut. In Abschnitt 2 beschreiben wir unseren Ansatz für die Validierung von UML-Modellen und OCL. Abschnitt 3 gibt eine Übersicht über die Architektur des USE-Werkzeugs. In Abschnitt 4 zeigen wir anhand einer Fallstudie die wesentlichen Eigenschaften des Werkzeugs in Bezug auf den Validierungsprozeß. Wir schließen mit einer Zusammenfassung und ziehen einige Schlußfolgerungen für zukünftige Arbeiten.

2 Validierung mit USE

Die Validierung von Entwürfen erfolgt mit dem Ziel, Fehler im Entwurf zu entdecken und zu vermeiden, bevor die Implementierungsphase beginnt. Es gibt verschiedene Arten der Validierung. In diesem Zusammenhang verstehen wir unter Validierung folgendes. Systemzustände werden als prototypische Instanzen eines Modells erzeugt und auf Konsistenz mit dem vorgegebenen Modell überprüft. Das Ergebnis einer Modellvalidierung kann zu verschiedenen Konsequenzen hinsichtlich des Entwurfs führen. Falls Systemzustände auftreten, die einen gewünschten Zustand der zu modellierenden Anwendung darstellen, dabei jedoch Integritätsbedingungen verletzen, kann dies bedeuten, daß die Bedingungen zu einschränkend sind, oder das Modell in anderer Hinsicht nicht angemessen ist. In diesem Fall muß man den Entwurf modifizieren, z. B. indem die Integritätsbedingungen abgeschwächt werden. Umgekehrt können Bedingungen zunächst auch zu schwach formuliert sein, so daß unerwünschte Systemzustände nicht ausgeschlossen werden. Durch eine restriktivere Formulierung der Bedingungen läßt sich auch dieses Problem beheben.

Wenn während der Validierungsphase unerwünschte Zustände erkannt sowie unerwünschte Zustände abgewiesen werden, läßt sich daraus allerdings nicht die Korrektheit eines Modells im formalen Sinne folgern. Es ist damit lediglich die Korrektheit des Modells hinsichtlich der untersuchten Fälle nachgewiesen. Im Gegensatz zu einer formalen Verifikation lassen sich jedoch auch nicht-formale Anforderungen validieren. Der Prozeß der Validierung stellt außerdem geringere Anforderungen an den Entwickler und erfordert keine Vorkenntnisse in formalen Methoden.

Abbildung 1 zeigt die wesentlichen Anwendungsfälle (Use Cases) für die Validierung eines Modells mit dem USE-Werkzeug. Zunächst kann eine Beschreibung (Spezifikation) des Modells durch das Validierungssystem geprüft werden. Der Anwendungsfall *check specification* beinhaltet eine Syntax-, Typ- und Semantikprüfung. Die Syntaxprüfung (*syntax check*) stellt die Wohlgeformtheit der Spezifikation sicher, die durch die Grammatik der Spezifikationssprache bestimmt ist. Diese Grammatik ist im wesentlichen eine Obermenge der OCL-Grammatik [8, 14], erweitert durch Sprachkonstrukte für die Beschreibung der Struktur eines Systems mithilfe von Klassen und Assoziationen. Die Typprüfung (*type check*) gewährleistet die korrekte Typisierung von OCL-Ausdrücken. Schließlich werden als Teil der Semantikprüfung (*semantic check*) kontext-sensitive Bedingungen überprüft. Diese Bedingungen basieren auf den Wohlgeformtheitsregeln, die in der UML-Semantik definiert sind [10]. Ein Beispiel für solch eine Wohlgeformtheitsregel ist die Anforderung, daß eine Generalisierungshierarchie keinen Zyklus enthalten darf.

Nachdem eine Spezifikation erfolgreich überprüft wurde, kann der Entwickler mit der Erzeugung und Änderung von Systemzuständen fortfahren (siehe Anwendungsfall *change system state*). Ein Systemzustand kann verändert werden durch das Erzeugen und Löschen von Objekten, das Einfügen und Entfernen von Verbindungen (*links*) zwischen Objekten, und das Setzen von Attributwerten einzelner Objekte. Ein Entwickler kann jederzeit Systemzustände auf Gültigkeit überprüfen (*check system state*). Eine solche Überprüfung beinhaltet zwei Schritte. Zunächst müssen modellinhärente Bedingungen verifiziert werden (*check model-inherent constraints*). Eine modellinhärente Bedingung ist eine Integritätsbedingung, die für alle UML-Modelle gilt und implizit durch die Definition der UML vorgegeben ist. Beispielsweise werden die vorhandenen Ver-

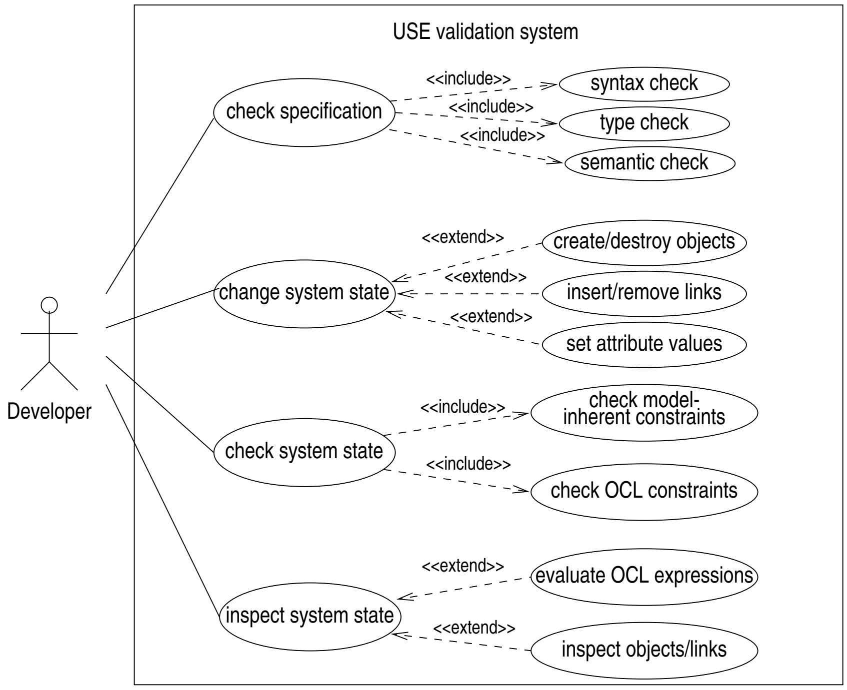

Abbildung1. Die wichtigsten Anwendungsfälle mit USE

bindungen zwischen Objekten mit den Multiplizitäten im Modell verglichen. Die Zuordnung von Objektverbindungen muß mit diesen Multiplizitäten übereinstimmen. In der zweiten Phase werden alle explizit vom Entwickler vorgegebenen OCL-Bedingungen ausgewertet (*check OCL constraints*). Wenn eine dieser Bedingungen falsch ist oder ein undefiniertes Ergebnis aufweist, wird der Systemzustand als ungültig betrachtet.

Der Anwendungsfall *inspect system state* beschreibt verschiedene Möglichkeiten, Informationen über einen Systemzustand zu erhalten. Für den Entwickler bildet dies ein wichtiges Hilfsmittel, um die Auswirkungen von Zustandsänderungen zu untersuchen und nachzuvollziehen. Von besonderer Bedeutung ist dies im Falle von ungültigen Systemzuständen. Dann steht der Entwickler vor der Aufgabe, die Ursache für diesen Fehler zu finden. Das Untersuchen eines Systemzustands beinhaltet das Inspizieren individueller Objekte, deren Attributwerte und Verbindungen. Eine weitere effiziente Methode zur Untersuchung ist die Verwendung von OCL als Anfragesprache. Nehmen wir als Beispiel ein Modell an, in dem jedes Objekt der Klasse A mindestens eine Verbindung zu einem Objekt der Klasse B haben muß, d. h. das Assoziationsende der Klasse B hat die Multiplizität `1..*`. Wenn diese Bedingung durch einen Systemzustand verletzt wird, können die Objekte der Klasse A, die keine Verbindung zu einem B

Objekt haben, einfach durch den OCL-Ausdruck `A.allInstances->select(a | a.b->size = 0)` ermittelt werden.

3 Die USE-Architektur

Einen Überblick über die Architektur von USE bietet Abbildung 2. Die obere Hälfte der Abbildung kann als Beschreibungsschicht und die untere Hälfte als Interaktionsschicht aufgefaßt werden. Die Beschreibungsschicht ist für die Verarbeitung einer Modellspezifikation verantwortlich. Die Hauptkomponente ist ein Parser, der Spezifikationen in der USE-Syntax einliest und eine abstrakte Syntax-Repräsentation des Modells erzeugt. Eine USE-Spezifikation definiert die strukturellen Anteile eines Modells wie z. B. Klassen und Assoziationen. Außerdem können OCL-Ausdrücke angegeben werden, um sowohl Bedingungen als auch Operationen ohne Seiteneffekte zu definieren.

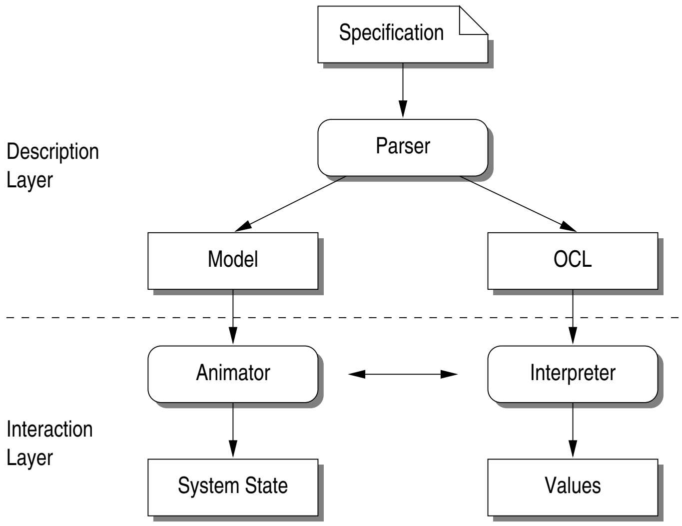

Abbildung2. Überblick der USE-Architektur

Die Ausgabe des Parsers ist eine abstrakte Darstellung einer Spezifikation, die das Modell und OCL-Ausdrücke umfaßt. Die Repräsentation eines Modells wird durch eine Teilmenge des Core packages des UML-Metamodells ausgedrückt [10, S. 2-13]. In der Festlegung der Teilmenge sind jene Modellelemente ausgeschlossen worden, die nicht während der Analyse- und frühen Entwurfsphase des Software-Entwicklungsprozesses benötigt werden. Modellelemente wie z. B. Permission, Component und Node werden sinnvoll erst in späteren Entwurfs- und Implementierungsphasen eingesetzt. Die abstrakte Syntax von OCL-Ausdrücken orientiert sich stark an dem OCL-Metamodell, das wir in [12] vorgestellt haben.

Die Interaktionsschicht ist verantwortlich für das dynamische Verhalten und die statischen Eigenschaften eines Modells. Die Hauptaufgabe der Animations-Komponente ist das Erzeugen und Verändern von Systemzuständen. Ein Systemzustand ist ein Schnappschuß eines Systems zu einem bestimmten Zeitpunkt. Der Systemzustand beinhaltet eine Menge von Objekten und eine Menge von *links*, welche die Objekte miteinander verbinden. Die Entwicklung eines Systems über die Zeit produziert eine Sequenz von Systemzuständen. Jeder Systemzustand muß wohlgeformt sein, d. h. er muß strukturell der Modellbeschreibung entsprechen, und er muß alle OCL-Bedingungen erfüllen. Außerdem muß der Übergang zwischen einzelnen Zuständen konform sein mit der Spezifikation des dynamischen Verhaltens. Die Beschreibung dynamischen Verhaltens und dessen Überprüfung ist in USE noch nicht realisiert.

Die Interpreter-Komponente ist für die Auswertung von OCL-Ausdrücken verantwortlich. Ein Ausdruck kann ein Teil einer Bedingung zur Einschränkung der Menge aller möglichen Systemzustände sein. Die Auswertung von OCL-Ausdrücken zur Überprüfung eines Systemzustands wird von der Animations-Komponente an den Interpreter delegiert. Der Interpreter wird außerdem für ad hoc-Anfragen verwendet. Der Benutzer kann mithilfe von Anfragen Objekte und deren Eigenschaften genauer untersuchen.

Die beiden von *Model* und *OCL* ausgehenden Zweige in Abbildung 2 sind eng miteinander verknüpft. Z. B. hängt die Darstellung von Modellen von OCL ab, da Operationen einer Klasse durch OCL-Ausdrücke definiert werden können. Umgekehrt besteht eine Abhängigkeit in die andere Richtung, weil OCL-Ausdrücke sich fast immer auf Teile eines Modells beziehen. Es können jedoch im allgemeinen sowohl Modelle angegeben werden, die keine OCL-Anteile besitzen, als auch OCL-Ausdrücke, die kein Anwendungsmodell benötigen. Die Umsetzung verschiedener grundlegender Berechnungsverfahren mit OCL (z. B. Sortieren, Bestimmung der transitiven Hülle einer Relation [6]) führt zu interessanten Erkenntnissen und kann durchgeführt werden, ohne ein bestimmtes Modell zu benötigen.

Auch die Animations-Komponente und der Interpreter arbeiten eng zusammen. Erstere überläßt dem Interpreter die Auswertung von OCL-Ausdrücken. Andererseits benötigt der Interpreter Informationen über den aktuellen Systemzustand, z. B. wenn ein Ausdruck ausgewertet wird, der sich auf den Attributwert eines Objekts bezieht.

4 Fallbeispiel

In diesem Abschnitt zeigen wir die Validierung eines UML-Modells anhand eines kleinen Fallbeispiels. Wir beginnen mit einem Modell eines einfachen Unternehmens, das durch ein Klassendiagramm und zusätzliche Integritätsbedingungen beschrieben wird. Das Modell wird textuell in der USE-Notation spezifiziert. Diese Spezifikation dient als Eingabe für das Validierungswerkzeug. In einer interaktiven Sitzung werden eine Reihe von Systemzuständen produziert, indem Objekte erzeugt und Verbindungen hergestellt werden. Schließlich zeigen wir die

Überprüfung eines Systemzustands und erläutern, wie das Werkzeug die Untersuchung von Zuständen und die Fehleranalyse unterstützt.

Abbildung 3 zeigt ein UML-Klassendiagramm des Fallbeispiels. Angestellte arbeiten in Abteilungen und haben Attribute für Namen und Gehalt. Eine Abteilung kontrolliert Projekte, an denen wiederum beliebig viele Angestellte arbeiten können. Sowohl Abteilungen als auch Projekte haben Attribute, die das verfügbare Budget bestimmen.

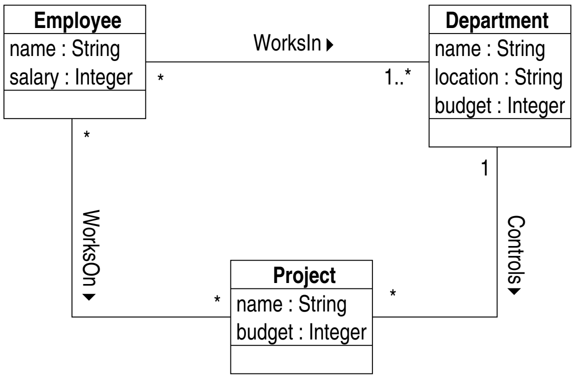

Abbildung3. Klassendiagramm für ein einfaches Unternehmensmodell

Die Übersetzung des Klassendiagramms in die textuelle USE-Notation führt zu der Spezifikation in Abbildung 4. Die Spezifikation enthält Definitionen für alle Klassen und Assoziationen. Die Beschreibung einer Klasse enthält hier lediglich Attribute. Assoziationen verweisen auf die beteiligten Klassen. Multiplizitäten sind jeweils in eckigen Klammern angegeben. UML-Konzepte wie Generalisierung, Aggregation, Komposition, Operationen, Rollennamen usw. werden zwar nicht in diesem Beispiel verwendet jedoch prinzipiell von USE unterstützt.

Um das Beispiel interessanter zu gestalten, fügen wir Bedingungen hinzu, die nicht grafisch im Klassendiagramm ausgedrückt werden können. Ein System, welches das gegebene Modell implementiert, muß die folgenden fünf Bedingungen erfüllen.

[1] Gehalt und Budget sind immer positive Zahlen.
[2] Eine Abteilung hat mindestens so viele Mitarbeiter wie Projekte.
[3] Ein Mitarbeiter, der an mehr Projekten arbeitet als andere Mitarbeiter, verdient ein höheres Gehalt.
[4] Das Budget eines Projekts darf nicht das Budget der kontrollierenden Abteilung übersteigen.
[5] Projektmitarbeiter müssen in der Abteilung arbeiten, welche das Projekt kontrolliert.

```
model Company

class Employee
attributes
    name : String;
    salary : Integer;
end

class Department
attributes
    name : String;
    location : String;
    budget : Integer;
end

class Project
attributes
    name : String;
    budget : Integer;
end

association WorksIn between
    Employee[*];
    Department[1..*];
end

association WorksOn between
    Employee[*];
    Project[*];
end

association Controls between
    Department[1];
    Project[*];
end
```

Abbildung4. Spezifikation des Beispiels in USE-Syntax

Für diese Bedingungen haben wir OCL-Ausdrücke formuliert, die als Invarianten einzelnen Klassen zugeordnet werden. Die Spezifikation aus Abbildung 4 wird nun erweitert durch die Integritätsbedingungen in Abbildung 5. Jede Invariante besitzt einen Namen, der einen direkten Bezug zu der obigen Liste ermöglicht. Die Anforderung [1] wird dabei auf drei OCL-Ausdrücke abgebildet (i1a, i1b, i1c), da sie eine Bedingung für alle drei Klassen ausdrückt.

```
constraints

context Department
    inv i1a: self.budget >= 0
    inv i2: self.employee→size >= self.project→size

context Employee
    inv i1b: self.salary >= 0
    inv i3: Employee.allInstances→forAll(e1, e2 |
            e1.project→size > e2.project→size implies e1.salary > e2.salary)

context Project
    inv i1c: self.budget >= 0
    inv i4: self.budget <= self.department.budget
    inv i5: self.department.employee→includesAll(self.employee)
```

Abbildung5. Spezifikation der OCL-Bedingungen

Nach dem Start von USE mit der Beispiel-Spezifikation liegt zunächst ein leerer Systemzustand vor, in dem keine Objekte und keine Verbindungen existieren. Im nächsten Schritt fügen wir dem System Objekte hinzu und verbinden sie miteinander. Es gibt drei Möglichkeiten zur Veränderung eines Systemzustands: (1) Objekte erzeugen und löschen, (2) Attributwerte verändern, (3) Verbindungen einfügen und entfernen. Abbildung 6 zeigt einen Screenshot von USE mit einem Systemzustand, nachdem mehrere Objekte und Verbindungen erzeugt wurden.

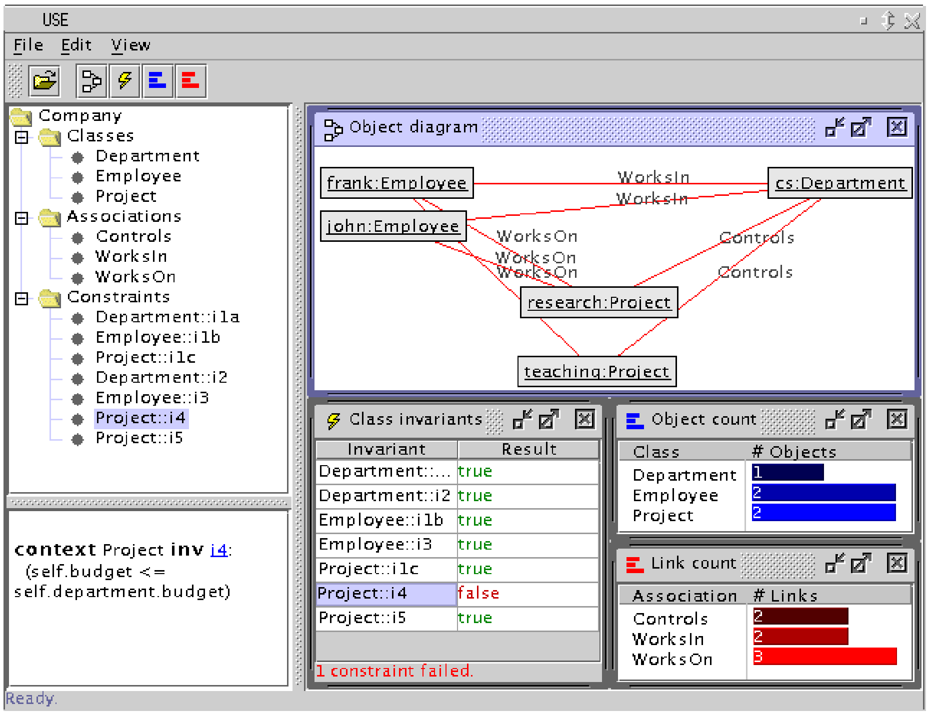

Abbildung6. USE Benutzungsschnittstelle mit Visualisierung eines Systemzustands

Auf der linken Seite bietet die Benutzungsschnittstelle eine Baumdarstellung der im Modell definierten Klassen, Assoziationen und Integritätsbedingungen. Der Ausschnitt darunter zeigt die Definition der aktuell ausgewählten Komponente (hier die Invariante `Projekt::i4`). Auf der rechten Seite befinden sich verschiedene Sichten auf den aktuellen Systemzustand. Diese Sichten werden automatisch aktualisiert, wenn sich das System verändert. Der Anwender kann aus einer Menge verschiedener Sichten wählen, die jeweils spezielle Aspekte eines Systemzustands hervorheben. In diesem Beispiel wird oben ein Objektdiagramm dargestellt, links darunter eine Liste von Invarianten mit ihren Ergebnissen sowie rechts unten zwei Sichten, welche die Anzahl von Objekten und Verbindungen anzeigen.

Die hervorgehobene Bedingung `Projekt::i4` in der Anzeige der Klasseninvarianten wurde zu falsch ausgewertet, d. h. der gezeigte Systemzustand ist ungültig. Die Information, daß eine Invariante verletzt wurde, gibt im allgemeinen keinen direkten Hinweis auf die Ursache des Problems. Die Invariante ist in diesem Fall relativ einfach (siehe Abb. 5), anhand der Spezifikation können wir daher folgern, daß es mindestens ein Projekt mit einem Budget geben muß, welches das Budget der kontrollierenden Abteilung übersteigt. Eine Überprüfung aller Projekte muß daher früher oder später eines ergeben, das die Bedingung verletzt. Für größere Systeme wird dieses jedoch recht schnell eine mühsame Aufgabe. Unser Werkzeug bietet daher eine spezielle Unterstützung für die Analyse von OCL-Ausdrücken an.

Die Auswertung eines OCL-Ausdrucks kann im Detail nachvollzogen werden durch einen sogenannten Evaluations-Browser. Dieser bietet eine baumartige Sicht auf den Auswertungsvorgang. Abbildung 7 zeigt einen Browser für die verletzte Invariante. Der Wurzelknoten zeigt den vollständigen OCL-Ausdruck der als Rumpf der Invariante definiert wurde (tatsächlich wird der Originalausdruck zunächst in einen Ausdruck expandiert, der ohne Kontext auskommt). Untergeordnete Knoten im Browser repräsentieren Teilausdrücke des Ausdrucks im Elternknoten. Die Auswertung des `forAll`-Ausdrucks erfordert die Auswertung zweier Teilausdrücke: die Ausgangsmenge der `forAll`-Operation (`Project.allInstances`) sowie deren Argument (`self.budget <= self.department.budget`). Die Belegung der Variable `self` offenbart schließlich, daß es sich im Beispiel um das Forschungsprojekt handelt, welches die Invariante verletzt.

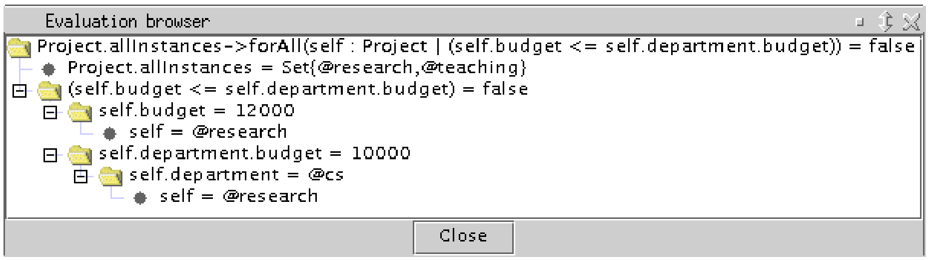

Abbildung7. OCL Evaluations-Browser mit Auswertungsergebnissen von Teilausdrücken

5 Fazit

Wir haben einen werkzeuggestützten Ansatz zur Validierung von UML-Modellen und OCL-Bedingungen vorgestellt. Die hier vorgestellten Ideen wurden in dem

USE-Werkzeug implementiert. Die Funktionalität und Möglichkeiten von USE haben wir anhand von Anwendungsfällen und mit einem Fallbeispiel erläutert. Das Werkzeug wurde außerdem von uns verwendet, um einen Teil des UML-Metamodells inklusive umfangreicher OCL-Bedingungen zu prüfen [13]. Als Ergebnis stellten wir eine Anzahl von Fehlern in den OCL-Ausdrücken im Standarddokument fest. Da das Metamodell als Beschreibung aller UML-Modelle genutzt werden kann, ermöglicht das Werkzeug eine mechanisierte Prüfung der Übereinstimmung beliebiger Modelle mit dem Standard.

Es gibt verschiedene denkbare Erweiterungen für USE. Das Ableiten von Testfällen aus einer Modellbeschreibung könnte zu einem gewissen Grad zur Automatisierung des Validierungsvorgangs beitragen. Eine andere Erweiterung ist die Anwendung der Validierungstechniken auch auf die Implementierungsphase. Dazu könnte Code generiert werden, der den aktuellen Zustand eines Systems widerspiegelt. Die Zustandsfolgen könnten dann gleichzeitig mit USE beobachtet und analysiert werden. Dieser Ansatz hat den Vorteil, daß eine aufwendige Übersetzung von OCL-Bedingungen in Programmcode entfällt, da ein OCL-Interpreter bereits Bestandteil von USE ist. Wünschenswert wäre auch eine Untersuchung von OCL-Bedingungen in Hinblick auf Eigenschaften wie Konsistenz. Zur Zeit existiert jedoch keine klare Definition des Konsistenzbegriffes für OCL. Eine Diskussion auch zu anderen aktullen OCL-Themen findet sich im OCL Semantik-FAQ [2].

Danksagung

Wir danken Frank Finger, Heinrich Hussmann und Jos Warmer für hilfreiche Diskussionen im Zusammenhang mit OCL.

Literatur

[1] BoldSoft. Modelrun, 2000. Internet: `http://www.boldsoft.com/products/modelrun/index.html`.

[2] Tony Clark, Stuart Kent, Jos Warmer, et al. OCL Semantics FAQ, Workshop on the Object Constraint Language (OCL) Computing Laboratory, University of Kent, Canterbury, UK. Internet: `http://www.cs.ukc.ac.uk/research/sse/oclws2k/index.html`, March 2000.

[3] Andy Evans and Stuart Kent. Core meta-modelling semantics of UML: The pUML approach. In Robert France and Bernhard Rumpe, editors, *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings*, volume 1723 of *LNCS*, pages 140–155. Springer, 1999.

[4] Frank Finger. Design and implementation of a modular OCL compiler. Diplomarbeit, Dresden University of Technology, Department of Computer Science, Software Engineering Group, Germany, March 2000.

[5] Daniel Jackson, Ian Schechter, and Ilya Shlyakhter. Alcoa: the Alloy constraint analyzer. In *Proc. International Conference on Software Engineering, Limerick, Ireland, June 2000*, 2000. (to appear).

[6] Luis Mandel and María Victoria Cengarle. On the expressive power of OCL. In *FM'99 - Formal Methods. World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September 1999. Proceedings, Volume I*, volume 1708 of *LNCS*, pages 854–874. Springer, 1999.

[7] Darmalingum Muthiayen. *Real-Time Reactive System Development - A Formal Approach Based on UML and PVS*. PhD thesis, Department of Computer Science at Concordia University, Montreal, Canada, January 2000.

[8] OMG. Object Constraint Language Specification. In *OMG Unified Modeling Language Specification, Version 1.3, June 1999* [9], chapter 7.

[9] OMG, editor. *OMG Unified Modeling Language Specification, Version 1.3, June 1999*. Object Management Group, Inc., Framingham, Mass., Internet: `http://www.omg.org`, 1999.

[10] OMG. UML Semantics. In *OMG Unified Modeling Language Specification, Version 1.3, June 1999* [9], chapter 2.

[11] Mark Richters and Martin Gogolla. On formalizing the UML Object Constraint Language OCL. In Tok Wang Ling, Sudha Ram, and Mong Li Lee, editors, *Proc. 17th Int. Conf. Conceptual Modeling (ER'98)*, volume 1507 of *LNCS*, pages 449–464. Springer, 1998.

[12] Mark Richters and Martin Gogolla. A metamodel for OCL. In Robert France and Bernhard Rumpe, editors, *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings*, volume 1723 of *LNCS*, pages 156–171. Springer, 1999.

[13] Mark Richters and Martin Gogolla. Validating UML models and OCL constraints. In *UML'2000 - Third International Conference on the Unified Modeling Language, 2-6th October, 2000, York, UK.*, LNCS. Springer, 2000.

[14] Jos Warmer and Anneke Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1998.