In Jürgen Ebert and Franz Lehner, editors, Proc. 3rd Workshop on Software-Reengineering (WSR), Universität Koblenz, Technischer Bericht

Selektive Darstellung von Programmstrukturen mit UML

Ralf Kollmann & Martin Gogolla
Universität Bremen
Fachbereich Mathematik/Informatik
Postfach 330440, D-28334 Bremen
{kollmann|gogolla}@informatik.uni-bremen.de

1 Einleitung

Die Verwendung von Reverse Engineering-Techniken zur Untersuchung von nicht-trivialen Software-Systemen erfordert oft die Handhabung von großen Informationsmengen. Sollen diese zum Zwecke der Redokumentation[BBE+96] durch Diagramme dargestellt werden, so ist eine Verdichtung oder Vorselektion relevanter Informationen sinnvoll. Die Größe und Komplexität von automatisch generierten Diagrammen führt sonst oft dazu, daß die Lesbarkeit verringert und das Verständnis erschwert wird.

In diesem Papier geben wir zunächst einen Überblick über die Arbeit im Projekt UML-AID und stellen dann Ansätze zur Vorselektion und Transformation von durch Reverse Engineering erhobenen Informationen vor, welche dann durch Diagramme der Unified Modeling Language (UML)[OMG99c] [OMG99d] dargestellt werden sollen.

2 Das Forschungsprojekt UML-AID

Im Rahmen des Projektes UML-AID (*Abstrakte Implementierung von und Dokumentation mit UML)* an der Universität Bremen werden zwei Forschungsbereiche zunächst unabhängig bearbeitet, wobei jedoch die Arbeitsergebnisse als Grundlage für weitere Forschungen in beiden Bereichen dienen.

Zentrales Thema beider Projektzweige ist die UML, eine Sprache, die dem Entwurf und der Dokumentierung von Software-Artefakten dient und durch verschiedene Diagrammarten Mittel zur Verfügung stellt, um diese graphisch darzustellen.

Ein Ziel des Projektes ist – für ausgewählte Diagrammarten – die Beschreibung eines eindeutigen Kerns von Sprachkonstrukten, aus dem die weiteren sprachlichen Mittel für die jeweilige Diagrammart abgeleitet werden können. Im Stil der UML Semantik-Dokumente soll die Beschreibung zunächst auf semi-formaler Ebene mittels UML durchgeführt werden. In einem weiteren Schritt soll dann eine formale Semantik basierend auf Graph-Transformationsregeln erstellt werden.

Die aktuelle Arbeit zielt auf die formale Beschreibung von UML Statechart- und Klassen-Diagrammen mittels Graphtransformation ab. Basierend auf bestehenden Ansätzen für Klassendiagramme wird zunächst untersucht, wie komplexe syntaktische Mittel der Klassen- und Statechart-Notation durch die elementaren Sprachmittel – einen Teil des bereits oben erwähnten Kerns – repräsentiert werden können. Eine formale Untermauerung der aus diesem Ansatz hervorgehenden Regeln ist ebenso Ziel der Arbeit wie ein vollständiger Ansatz zur Erzeugung von korrekten Statecharts durch Graphtransformationsregeln.

Als weiteres Forschungsthema werden Techniken des Reverse Engineerings bzw. der Redokumentation unter Verwendung der Unified Modeling Language untersucht und entwickelt. Programmcodes von C++ und Java-Programmen sollen analysiert und deren Informationsgehalt zum Zwecke der Dokumentation in UML-Diagrammen repräsentiert werden. Von besonderer Bedeutung ist die korrekte und einheitliche Abbildung der durch den Programmcode gegebenen Spezifikation in entsprechende UML-Konstrukte, wobei Wert auf eine möglichst gute Ausnutzung des vollständigen Satzes der UML-Sprachmittel gelegt wird. Wie wir an späterer Stelle beschreiben, kann die so erreichbare Abstraktion der Darstellung sowohl zur Komprimierung von bestimmten Diagrammen, als auch zur klareren Beschreibung der Semantik dienen [GK00].

Der momentane Themenschwerpunkt liegt auf der Entwicklung eines Metamodells für Java [KG01b]. Durch diesen Ansatz stehen sowohl auf Seite der Programmiersprache als auch der graphischen Beschreibungssprache vergleichbare Meta-Strukturen zur Verfügung, so daß unter Verwendung eines Satzes von Transformationsregeln eine Übersetzung von Java nach UML möglich ist. Dazu werden zunächst mittels Reverse Engineering die relevanten Informationen über die statische Struktur und ggf. über das dynamische Verhalten aus dem zu untersuchenden Programm extrahiert. In diesem Vorgang setzen wir momentan die folgenden Methoden ein:

- Durch einen Bytecode-Parser werden die statische Struktur und Teile des dynamischen Verhaltens in Form von statischen Traces analysiert. Wir untersuchen hier die Objekt-Interkommunikation durch Methodenaufrufe. Diese Methode hat den Vorteil, daß kein Quellcode für die Extraktion von Informationen vorliegen und daß Programm nicht laufen muß.
- Ein auf der *Java Platform Debugging Architecture* (JPDA) basierendes Programm zur Extraktion von Programm-Traces erlaubt zur Laufzeit das Sammeln von Informationen über dynamische Typen und vor allem über das dynamische Programmverhalten durch Erzeugung von dynamischen Programm-Traces. Der Vorteil der wesentlich genaueren Information gegenüber statischen Traces wird durch die Tatsache relativiert, daß sich die gesammelten Informationen jeweils nur auf einen einzelnen Programm-Lauf beziehen. Es ergibt sich das Problem der Abdeckung aller theoretisch möglichen Programm-Abläufe.
- Durch die Verwendung eines Sourcecode-Parsers stehen weitere Informationen zur Verfügung wie z.B. Übergabe von Variablen-Bezeichnern und insbesondere Informationen über den Kontroll-Fluß von Methoden. Wir verwenden den Parser ANTLR[Par89], um abstrakte Syntaxbäume (AST, *engl. abstract syntax tree*) zu erzeugen. In einem zweiten Schritt werden durch einen weiteren Parser die Informationen des Java-Programms aus dem AST extrahiert und als Instanz des Java-Metamodells abgelegt.

Um die aus der Verwendung dynamischer Traces resultierenden Probleme zu umgehen, versuchen wir zunächst, die relevanten Informationen ohne Verwendung der JPDA allein aus dem Source- bzw. Bytecode zu extrahieren.

Die Informationen werden nun als Instanz des Java Meta-Modells abgelegt und dann mittels der Transformationsregeln in eine Instanz des UML Meta-Modells übersetzt. Hierzu werden verschiedene Transformationsalgorithmen eingesetzt: je nach gewünschter Sicht auf die extrahierten Informationen können so verschiedene Programmaspekte visualisiert werden. Z.B. kann die statische Programmstruktur in Form von Klassen- und Objektdiagrammen dargestellt werden. Des weiteren wurden Algorithmen zur Repräsentation des dynamischen Verhaltens durch Interaktionsdiagramme erarbeitet. Durch Verwendung von Kollaborationsdiagrammen ist eine kombinierte Darstellung von statischen und dynamischen Aspekten möglich.

3 Ansätze zur Informationsverarbeitung

Sollen Diagramme zur Darstellung komplexer Programmstrukturen verwendet werden, so verwenden wir verschiedene Methoden, um die dem Diagramm zugrunde liegenden Informationen vor der graphischen Darstellung so zu bearbeiten, daß eine Darstellung sinnvoll ist. Wir unterscheiden zunächst zwischen zwei Ansätzen zur Verarbeitung der anfallenden Informationen, die sich bezüglich Umfang und Motivation teilweise unterscheiden.

Abstraktion. Hier werden die als Instanz des UML Metamodells vorliegenden Programminformationen nach bestimmten Mustern durchsucht: Übereinstimmungen mit diesen erlauben die Verwendung abstrakterer Notationen. Auf diese Weise können z.B. Qualifier, bidirektionale Assoziationen und Kompositionen in Klassendiagrammen erkannt werden. Als Ansatz dienen die in [GR99] beschriebenen Äquivalenzregeln zur Ersetzung von komplexen Notationselementen durch einfachere. Diese Regeln sind semantikerhaltend, wobei das “added value” der komplexeren Notation durch Ausdrücke der OCL (*Object Constraint Language*) [OMG99a] beschrieben wird.
Die folgende Regel zeigt z.B. die Transformation eines Qualifiers in eine Assoziationsklasse:

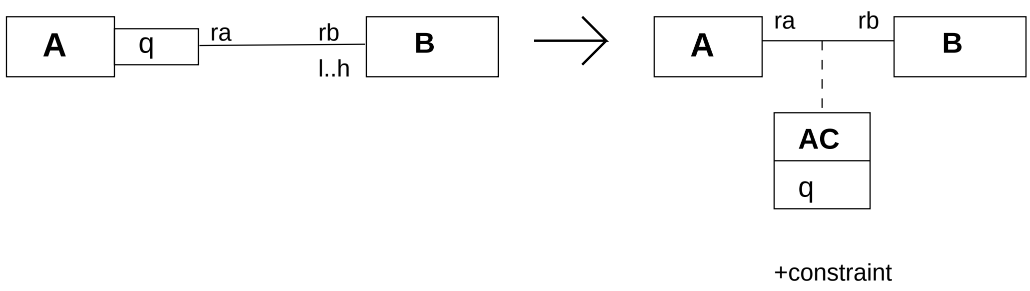

Abbildung1. Transformationsregel für Qualifier

Um die Äquivalenz der beiden Seiten zu gewährleisten, muß auf der rechten Seite folgender OCL-Ausdruck gelten:

```
A.allInstances->forAll(a|
 a.ac->forAll(ac|
   a.rb->select(b|
    b.ac->exists(ac'|ac'.q=ac.q and ac'.ra=a))->size>=l
   and
```

```
    a.rb->select(b|
      b.ac->exists(ac'|ac'.q=ac.q and ac'.ra=a))->size<=h
  )
)
```

Für die Redokumentation wenden wir diese Regeln in umgekehrter Richtung an, d.h. ausgehend von elementaren Notationselementen erfolgt die Transformation hin zu komplexeren, wenn o.g. Muster gefunden wurden. Auch hier wäre eine semantikerhaltende Transformation durch Verwendung von OCL-Ausdrücken möglich, jedoch erscheint uns dieser Schritt nicht immer Sinnvoll, da durch Hinzufügen der komplexen OCL-Ausdrücke das ursprüngliche Ziel, die Programmstruktur besser verständlich zu machen, verfehlt wird.

Die folgende Transformation in Form einer *double pushout rule* [Roz97] gibt beispielhaft eine Abstraktionsregel für UML Klassendiagramme an: Sie beschreibt die Zusammenfassung zweier gerichteter Assoziationen zu einer Bidirektionalen.

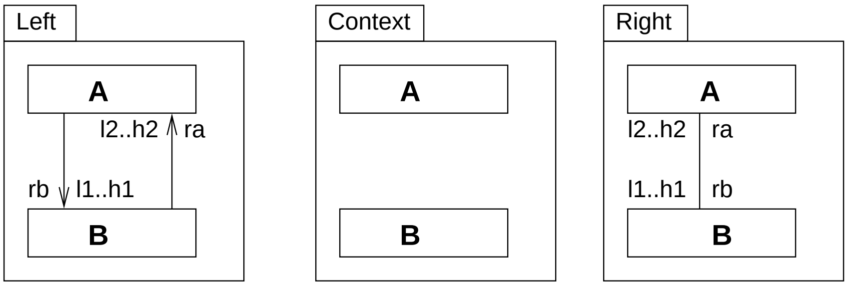

Abbildung2. Transformation von gerichteten Assoziationen

Während beim obigen Beispiel in Abbildung 1 die rechte Seite der Transformation durch einen OCL-Ausdruck ergänzt wird, müssen hier zunächst bestimmte Bedingungen für die linke Seite gelten, damit die beschriebene Transformation durchgeführt werden kann. Des weiteren können weitere Regeln für die Erkennung semantischer Zusammenhänge angegeben werden: Im Falle der Transformation in Abbildung 2 können z.B. Paare von den Assoziationen zugehörigen Rollennamen vorgegeben werden (z.B. Braut/Bräutigam, Arbeitgeber/Arbeitnehmer). Nur wenn solch ein Paar gefunden wird, wird die Transformation durchgeführt [KG01a].

Selektion. Insbesondere um einen ersten Überblick über die Architektur eines komplexen Systems zu erlangen, jedoch auch bei der gezielten Untersuchung von Teilbereichen, ist eine selektive Darstellung der Programm-Informationen sehr hilfreich.

Bei Verwendung einer manuellen Selektion obliegen die Auswahlentscheidungen dem Benutzer. Dieser kann auf semantisches Wissen zurückgreifen, welches dem Analyseprogramm nicht oder nur eingeschränkt zur Verfügung steht. Eine manuelle Überarbeitung aller Informationen ist jedoch sehr aufwendig und für komplexe Systeme nicht realistisch. Jedoch ist dieser Ansatz in Verbindung mit anderen Methoden sinnvoll: So kann eine Vorselektion aufgrund von Transformationsregeln getroffen werden, wobei in durch die Software nicht entscheidbaren Fällen der Benutzer hinzugezogen wird. Hierbei kann der Grad der Benutzer-Interaktion vorgegeben werden: während bei sehr großen Systemen möglicherweise aus Aufwandsgründen auf eine intensivere Interaktion des Benutzers verzichtet wird, mag diese zur gezielten Untersuchung von überschaubaren Strukturen wie z.B. Design Patterns [GHJV95] in verstärktem Maße sinnvoll sein.

Ein weiterer Ansatz zur Darstellung großer Informationsmengen bedient sich der räumlichen Perspektive. In diesem Kontext wird in unserer Arbeitsgruppe die in [OMG99c] erwähnte drei-dimensionale Darstellung von UML-Diagrammen erforscht[GRR99][RG00].

4 Zusammenfassung

Wir haben beschrieben, wie verschiedene Ansätze zur Selektion und Transformation von durch Reverse Engineering gesammelten Informationen dazu beitragen können, die Menge der durch den Programmcode gegebenen Informationen in komprimierter Weise darzustellen, um Lesbarkeit und Verständnis zu erleichtern. Wir unterscheiden zwischen Abstraktion, wobei komplexere Elemente der UML Notation von Klassendiagrammen zum Einsatz kommen, und selektiver Darstellung, wobei die Transformation der Daten mit einer manuellen Selektion kombiniert wird. Beim Einsatz dieser Techniken hat sich gezeigt, daß durch diese Herangehensweise im Hinblick auf die o.g. Ziele bessere Ergebnisse erreicht werden können. Eine genauere Erfassung der erzielten Verbesserungen ist Gegenstand weiterer Untersuchungen.

Literatur

[BBE+96] Ulrike Baumöhl, Jens Borchers, Stefan Eicker, Knut Hildebrand, Reinhard Jung, and Franz Lehner. Einordnung und Terminologie des Software Reengineering. *Informatik-Spektrum*, 19:191–195, 1996.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison Wesley, Reading, MA, 1995.

[GK00] Martin Gogolla and Ralf Kollmann. Re-Documentation of Java with UML Class Diagrams. In Eliot Chikofsky, editor, *Proc. 7th Reengineering Forum, Reengineering Week 2000 Zürich*, pages REF 41–REF 48. Reengineering Forum, Burlington, Massachusetts, 2000.

[GR99] Martin Gogolla and Mark Richters. Transformation Rules for UML Class Diagrams. In Jean Bézivin and Pierre-Alain Muller, editors, *Proc. 1st Int. Workshop Unified Modeling Language (UML'98)*, volume 1618 of *LNCS*, pages 92–106. Springer, Berlin, 1999.

[GRR99] Martin Gogolla, Oliver Radfelder, and Mark Richters. Towards Three-Dimensional Representation and Animation of UML Diagrams. In Robert France and Bernhard Rumpe, editors, *Proc. 2nd Int. Conf. Unified Modeling Language (UML'99)*, pages 489–502. Springer, Berlin, LNCS 1723, 1999.

[KG01a] Ralf Kollmann and Martin Gogolla. Application of UML Associations and Their Adornments in Design Recovery. In *Proc. 8th Working Conference on Reverse Engineering (WCRE)*, 2001.

[KG01b] Ralf Kollmann and Martin Gogolla. Capturing Dynamic Program Behaviour with UML Collaboration Diagrams. In Pedro Sousa and Jürgen Ebert, editors, *Proc. 5th European Conference on Software Maintenance and Reengineering*, pages 58–67. IEEE, Los Alamitos, 2001.

[OMG99a] OMG. Object Constraint Language Specification. In *OMG Unified Modeling Language Specification, Version 1.3, June 1999* [OMG99b], chapter 7.

[OMG99b] OMG, editor. *OMG Unified Modeling Language Specification, Version 1.3, June 1999*. Object Management Group, Inc., Framingham, Mass., Internet: http://www.omg.org, 1999.

[OMG99c] OMG. UML Notation Guide. In *OMG Unified Modeling Language Specification, Version 1.3, June 1999* [OMG99b], chapter 3.

[OMG99d] OMG. UML Semantics. In *OMG Unified Modeling Language Specification, Version 1.3, June 1999* [OMG99b], chapter 2.

[Par89] Terence Parr. ANTLR Website, 1989. http://www.antlr.org.

[RG00] Oliver Radfelder and Martin Gogolla. On Better Understanding UML Diagrams through Interactive Three-Dimensional Visualization and Animation. In Vito Di Gesu, Stefano Levialdi, and Laura Tarantino, editors, *Proc. Advanced Visual Interfaces (AVI'2000)*, pages 292–295. ACM Press, New York, 2000.

[Roz97] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation - Volume 1: Foundations*. World Scientific, 1997.

[Sys00] Tarja Systä. *Static and Dynamic Reverse Engineering Techniques for Java Software Systems*. PhD thesis, Department of Computer and Information Science, University of Tampere, Finland, 2000.