Generierung von UML-Modellen aus formalisierten Anwendungsfallbeschreibungen

Mario Friske, Bernd-Holger Schlingloff
Fraunhofer FIRST
Kekuléstraße 7
D-12489 Berlin
{mario.friske|holger.schlingloff}@first.fraunhofer.de

Abstract: In diesem Kurzbeitrag zeigen wir, wie textuelle Anwendungsfallbeschreibungen werkzeuggestützt in UML-Aktivitätsdiagramme überführt werden können. Mit Hilfe eines von uns entwickelten Werkzeuges formalisieren wir zunächst die Beschreibung der Interaktionen, um anschließend aus dem so erstellten formalisierten Anwendungsfallmodell mittels automatisierter Transformation ein UML-Modell zu generieren. Die notwendige Transformation definieren wir in abstrakter Form und setzen diese mit verschiedenen Transformationswerkzeugen um.

1 Motivation

Anwendungsfälle (engl. Use Cases) sind ein weit verbreitetes Mittel zur Strukturierung und Spezifikation funktionaler Anforderungen an Softwaresysteme. Ein Anwendungsfall definiert eine Menge von Interaktionen zwischen Akteuren und dem System, die dazu dient, ein bestimmtes fachliches Ziel zu erreichen. Anwendungsfälle können als textuelle Beschreibungen oder in Form von Diagrammen, z. B. als Sequenz- oder Aktivitätsdiagramme, oder als Kombination von Text und Diagrammen spezifiziert werden.

Oftmals werden Anwendungsfälle als erster Schritt einer Systementwicklung in der Analysephase erstellt, um einen Überblick über die geforderte Systemfunktionalität zu gewinnen. Gängige Werkzeuge zur Anforderungsanalyse (z. B. Telelogic DOORS) unterstützen die Verwaltung und Verlinkung von Anwendungsfällen, erlauben es jedoch nicht, diese direkt für die Systementwicklung oder den Systemtest weiter zu verarbeiten.

Die interaktive Überführung von informellen Anwendungsfallbeschreibungen in eine formale Zwischenrepräsentation wurde von uns in einer früheren Publikation [FS05] diskutiert. In der modellbasierten Entwicklung werden häufig grafische Darstellungsformen verwendet, insbesondere die Unified Modeling Language (UML). Die grafische Darstellung eines Ablaufs als Diagramm bietet oftmals den Vorteil des leichteren Überblicks. Eine einfache Möglichkeit der Überführung von formalisierten textuellen Beschreibungen in eine grafische UML-Repräsentation wäre daher wünschenswert.

2 Der Use Case Validator

Mit dem *Use Case Validator* (UCV) wird bei Fraunhofer FIRST ein Werkzeug zum anwendungsfallbasierten Systemtest entwickelt. Ziel ist dabei unter anderem der Blackbox-Test eingebetteter Systeme. Der UCV basiert auf der *Eclipse*-Plattform [Ecl] und verfolgt einen modellbasierten Ansatz, in welchem Datenstrukturen durch Metamodelle definiert sind und mittels Modelltransformationen weiterverabeitet werden. Als Grundlage verwenden wir das *Eclipse Modeling Framework* [BEG+03] und darauf basierende Transformationsmaschinen, von denen wir zu Experimentierzwecken mehrere in den UCV eingebunden haben [FH06].

Der UCV ermöglichst es dem Nutzer, Anwendungsfallbeschreibungen unter Verwendung von Entwurfsinformation interaktiv zu formalisieren. Zur Veranschaulichung greifen wir das Beispiel aus [FS05] wieder auf, den in Abbildung 1 gezeigten Anwendungsfall „Record a Message“. Dieses ursprünglich aus [PL99] stammende Beispiel ist Teil der funktionalen Spezifikation eines digitalen Sound-Recorders und dient dort zur Veranschaulichung einer UML-basierten Entwicklungmethodik.

```
1. The user selects a message slot from the message
   directory.
2. The user presses the 'record' button.
3. If the message slot already stores a message, it is
   deleted.
4. The system starts recording the sound from the
   microphone until the user presses the 'stop' button,
   or the memory is exhausted.
```

Abbildung 1: Der Anwendungsfall „Record a Message“ in Textform

Zur Formalisierung mit dem UCV werden zunächst Systemfunktionen und -reaktionen bestimmt. Danach wird der Kontrollfluss ermittelt und anschließend die Systemfunktionen den Ablaufschritten zugeordnet. Dabei werden schrittweise die Mehrdeutigkeiten der textuellen Repräsentation interaktiv aufgelöst. Bei der Formalisierung des Kontrollflusses wird intern ein Kontrollflussmodell aufgebaut, welches für den Anwender des UCV nicht unmittelbar sichtbar ist. In der in Abbildung 2 gezeigten Benutzeroberfläche wird der Kontrollfluss nur durch Umklammerung und Einrückungen von Schritten repräsentiert, was in Abbildung 3 nochmal im Detail dargestellt wird. Die einzelnen Ablaufschritte werden formalisiert, indem ihnen durch den Anwender aufrufbare Systemfunktionen und mögliche Systemreaktionen zugeordnet werden, siehe auch [FS05]. Systemfunktionen- und reaktionen können optional auch Parameter enthalten, an welche Variablen gebunden werden. Im Beispiel wird auf diese Art und Weise durch die Variable `Slot` der Datenaustausch zwischen den Schritten realisiert. Das Ergebnis der Bearbeitung mit dem UCV ist ein formalisiertes Anwendungsfallmodell, welches durch das in [FP05] beschriebene Metamodell definiert ist und den Kontrollfluss und die einzelnen Ablaufschritte umfasst.

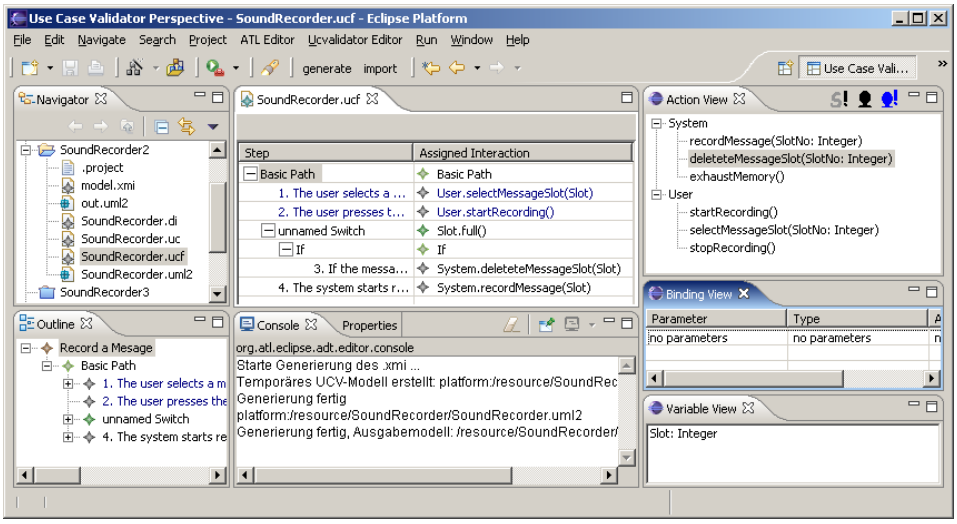

Abbildung 2: Die Benutzeroberfläche des Use Case Validator

```
User.selectMessageSlot(Slot)
User.startRecording()
if(Slot.full())
  System.deleteMessageSlot(Slot)
Exception(System.exhaustMemory()||User.stopRecording())
  System.recordMessage(Slot)
```

Abbildung 3: Die Formalisierung des Anwendungsfalls „Record a Message“

3 Generierung von UML-Modellen

Aus diesen formalisierten Anwendungsfällen wollen wir nun UML-Aktivitätsdiagramme mittels Modelltransformation generieren. Dieses Problem ist vergleichbar zur Generierung von Kontrollflussgraphen aus Programmen. Mit dem vorgestellten Ansatz können beliebige, durch ein Metamodell definierte, visuelle Repräsentationen generiert werden. UML-Aktivitätsdiagramme sind eine natürliche und weit verbreitete grafische Darstellungsform für Anwendungsfälle.

Die Transformation spezifizieren wir zunächst als abstrakte Transformationsbeschreibung, um eine einfache Abbildung auf verschiedene Modelltransformationssprachen zu ermöglichen. Diese Transformationsbeschreibung haben wir in Anlehnung an [ATL05] erstellt. Der Pseudocode für die Transformation ist in Abbildung 4 dargestellt. Er beschreibt die Abbildung von Elementen im Ausgangsmodell, dem Metamodell des UCV [FP05], auf Elemente im Zielmodell, dem Metamodell des Eclipse-UML2-Projekts [Hus04]. Eine einzelne Transformationsregeln ist dabei folgendermaßen zu lesen: Für jedes links beschriebene Element aus dem Ausgangsmodell sind alle rechts von den zugehörigen Pfeilen (⇒)

UCF2UML-AD: Formalisierte Anwendungsfallbeschreibung in UML2-Aktivitätsdiagramm

Kurze Beschreibung: Die Transformation UCF2UML-AD generiert UML2-Aktivitäten für alle formalisierten Anwendungsfälle im Ausgangsmodell. Diese Aktivitäten modellieren den Kontrollfluss der Anwendungsfälle.

Ausgangs-Metamodell: Metamodell des Use Case Validator: http://ucvalidator.ecore

Ziel-Metamodell: UML 2.0: http://www.eclipse.org/uml2/1.0.0/UML
Webseite des Eclipse-UML2-Projekts: http://www.eclipse.org/uml2

Transformationsregeln:

• ***UCFModel***	⇒	***Model***
• ***UCFormalization*** mit enthaltenem ***Block***	⇒	namensgleiche ***Activity***, enthalten in obigem ***Model***; alle anderen zu erstellenden Elemente sind in dieser ***Activity*** enthalten, soweit nicht anders angegeben
	⇒	***ActivityPartition*** für die Systemreaktionen
	⇒	***InitialNode*** und ***ActivityFinalNode***
	⇒	***ControlFlow*** vom ***InitialNode*** zu aus dem Block erstellten Knoten, von dort zum ***ActivityFinalNode***
• ***Step*** mit enthaltener ***Interaction***	⇒	***CallOperationAction*** namensgleich mit ***Interaction***
• ***Sequence*** mit enthaltenen Blöcken	⇒	***ControlFlow*** zwischen aus den Blöcken erstellten Knoten
• ***Switch*** mit enthaltenen Blöcken	⇒	***DecisionNode***
	⇒	***MergeNode***
	⇒	***ControlFlow*** zwischen ***DecisionNode***, aus den Blöcken erstellten Knoten und ***MergeNode***
• ***Loop*** mit enthaltenem ***Block***	⇒	***MergeNode***
	⇒	***DecisionNode***
	⇒	***ControlFlow*** von ***MergeNode*** zu aus dem ***Block*** erstellten Knoten, von dort zum ***DecisionNode***, von dort zum ***MergeNode***
• ***Actor***	⇒	***ActivityPartition***
• ***Variable***	⇒	namensgleicher ***CentralBufferNode***
• ***Parameter***, in ***Interaction*** und damit ***Step*** enthalten	⇒	namensgleicher ***InputPin*** bzw. ***OutputPin*** enthalten in erstellter ***CallOperationAction***
• ***Binding*** mit Referenzen auf ***Variable*** und ***Parameter***	⇒	***ObjectFlow*** zwischen aus den Referenzen erstelltem ***CentralBufferNode*** und ***Pin***

Abbildung 4: Abstrakte Transformationsspezifikation mittels Pseudocode

Elemente im Zielmodell zu erstellen. Dabei sind ggf. die in textueller Form beschriebenen Anweisungen zu berücksichtigen.

Die so spezifizierte Transformation erstellt für jeden formalisierten Anwendungsfall (***UC-Formalization***) eine Aktivität (***Activity***). In der Aktivität wird für jeden Ablaufschritt (***Step***) eine Aktion (*CallOperationAction*) und für jede Variable (***Variable***) ein Pufferknoten (***CentralBufferNode***) angelegt. Objektfluss und Kontrollfluss werden getrennt überführt. Der Kontrollfluss wird durch Kontrollflusskanten (***ControlFlow***) und Kontrollknoten (***DecisionNode, MergeNode***) realisiert. Der Objektfluss wird so umgesetzt, dass Pufferknoten durch Objektflusskanten (***ObjectFlow***) und Pins (***InputPin, OutputPin***) mit den Aktionen verbunden werden.

Dieser Pseudocode kann in ausführbaren Code für Transformationsmaschinen umgesetzt werden. Im UCV ist dies mit vier verschiedenen EMF-basierten Werkzeugen realisiert worden, welche mittels Wrapper-Plugin eingebunden wurden [Hil06]. Ein Erfahrungsbericht und Vergleich der verschiedenen Transformationswerkzeuge ist in [FH06] zu finden. Im vorliegenden Papier konzentrieren wir uns auf die Anwendung der Transformationssprache ATL [JK05], da sie frei verfügbar ist, zur Zeit stark weiterentwickelt wird und somit die Möglichkeit für weitere Experimente bietet.

Als Beispiel geben wir in Abbildung 5 einen Ausschnitt des ATL-Codes. Die gezeigte Transformationsregel überführt eine bedingte Verzweigung ***Switch*** in entsprechende UML-Konstrukte: ein Entscheidungsknoten ***DecisionNode*** und ein Ausgangknoten ***MergeNode*** werden angelegt und durch Kontrollflusskanten ***ControlFlow*** verbunden.

Das Ergebnis der Transformation einer formalisierten Anwendungsfallbeschreibung ist ein UML-2.0-Modell als Instanz des Metamodells des Eclipse-UML2-Projekts [Hus04]. Es liegt als Ecore vor und kann daher ohne weiteres als XMI-Datei serialisiert werden. XMI ist ein weit verbreitetes Format zum Speichern von Modellen [Obj03]. Das generierte Modell enthält die dem Anwendungsfall zugeordnete Aktivität, d.h. ihren Namen, die zugehörigen Aktivitätspartitionen, Aktionen und den Kontrollfluss. Abbildung 6 zeigt einen Ausschnitt der generierten XMI-Datei, in welchem die eine bedingte Verzweigung bildenden UML-Elemente dargestellt sind.

Dieses UML-Modell kann dann zur Visualisierung in ein handelsübliches UML-Modellierungswerkzeug geladen werden. Abbildung 7 zeigt eine solche Visualisierung des Transformationsergebnisses für das obige Beispiel, welches in diesem Fall mit Borland Together 2006 [Bor] erstellt wurde. Together bietet ein Autolayout für automatisch generierte Diagramme an, welches allerdings nur für sehr einfache Diagramme befriedigende Ergebnisse liefert. Deshalb wurde das Layout per Hand optimiert, um die Lesbarkeit zu verbessern. Die aktuell verfügbaren UML-Werkzeuge implementieren die in der UML2.0 definierten Sprachkonstrukte noch nicht in vollem Umfang, so wurde beispielsweise das Konstrukt *ExceptionHandler* in keinem der von uns betrachteten Werkzeugen unterstützt.

```
rule Switch2Node1 {
    from switch : ucv!Switch(switch.ElseBlock
                             ->oclIsUndefined())
    to
     -- Der Entscheidungsknoten
     dec  : uml!DecisionNode (
        name <- switch.Condition.description
     ),
     -- Der Mergeknoten führt die Pfade wieder zusammen,
     -- ist der Ausgangsknoten einer bedingten Anweisung
     out : uml!MergeNode(
        name <- 'end'+switch.name
     ),
     e1 : uml!ControlFlow(
        source <- dec,
        target <- switch.IfBlock.getValidBlocks()->first()
     ),
     e2 : uml!ControlFlow(
        source <- dec,
        target <- out
     ),
     e3 : uml!ControlFlow(
        source <-  thisModule.resolveTemp(
                        switch.IfBlock.getValidBlocks()
                        ->last(),'out'
                    ),
        target <- out
     )
}
```

Abbildung 5: Eine Transformationsregel aus der ausführbaren ATL-Transformation

4 Diskussion und Ausblick

In diesem Beitrag haben wir einen Ansatz vorgestellt, mit dem sich aus formalisierten Anwendungsfällen eine konsistente visuelle Repräsentation in UML automatisch generieren lässt. Anwendungskontext ist der Übergang von informellen textuellen Repräsentationen zu Modellen und ihren verschiedenen alternativen Repräsentationen. Während wir in vorhergehenden Arbeiten [FS05, FP05] den ersten Schritt, die Formalisierung, detailliert beschrieben haben, lag der Fokus dieses Papiers auf dem zweiten Schritt, dem Generieren verschiedenster Repräsentationen.

Der Abgleich von textuellen und grafischen Repräsentationen von Anwendungsfällen zählt nach wie vor zu den aktuellen Fragestellungen in der Forschung [GLM+05]. So schließt ein idealer iterativer Softwareprozess zwar die Anforderungen mit ein, MDA-Prozesse beginnen jedoch erst mit dem Analysemodell [KWB03]. Die Erstellung des plattformunabhängigen Modells aus den textuell beschriebenen Anforderungen bleibt offen. Mit unse-

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.0"
      xmlns:xmi="http://www.omg.org/XMI"
      xmlns:uml="http://www.eclipse.org/uml2/1.0.0/UML">
  ...
  <uml:CallOperationAction xmi:id="_4mAwt2_aEduZPozAIdpC6g"
    name="deleteteMessageSlot"
    incoming="_4mAwvW_aEduZPozAIdpC6g"/>
  <uml:DecisionNode xmi:id="_4mAwu2_aEduZPozAIdpC6g"
    name="Slot.full()"
    outgoing="_4mAwvW_aEduZPozAIdpC6g
              _4mAwvm_aEduZPozAIdpC6g"
    incoming="_4mAwtW_aEduZPozAIdpC6g"/>
  <uml:MergeNode xmi:id="_4mAwvG_aEduZPozAIdpC6g"
    name="endunnamed Switch"
    outgoing="_4mAwtm_aEduZPozAIdpC6g"/>
  <uml:ControlFlow xmi:id="_4mAwvW_aEduZPozAIdpC6g"
    source="_4mAwu2_aEduZPozAIdpC6g"
    target="_4mAwt2_aEduZPozAIdpC6g"/>
  ...
</xmi:XMI>
```

Abbildung 6: Ausschnitt aus der XMI-Repräsentation des generierten UML-Modells

rem Ansatz zeigen wir eine Möglichkeit auf, diesen Übergang zumindest in eine Richtung werkzeuggestützt zu vollziehen.

Mit einer den *Action Words* [Buw99] ähnlichen Methode formalisieren wir in Textform vorliegende Anwendungsfallbeschreibungen und bauen interaktiv ein formalisiertes Anwendungsfallmodell auf, welches durch ein Metamodell definiert ist. Im Gegensatz zu anderen Use-Case-Metamodellen, wie z. B. [Wil01], haben wir unser Metamodell unabhängig von dem UML-Metamodell definiert, wobei wir Wert auf eine kanonische Repräsentation anwendungsfallspezifischer Konzepte und des Kontrollflusses gelegt haben. Daher weist unser UCV-Metamodell auch Ähnlichkeiten hinsichtlich Abstraktionsgrad und Begrifflichkeiten mit Metamodellen zur Repräsentation von Programmiersprachen auf. Eine alternative Möglichkeit, ein formalisiertes Anwendungsfallmodell aufzubauen, wäre der Einsatz von schablonenbasierten Editoren. Diese werden vom Nutzer jedoch oft als zu einschränkend empfunden.

Aus dem formalisierten Anwendungsfallmodell generieren wir automatisch mittels Modelltransformation eine konsistente Repräsentation als UML-Aktivitätsdiagramm. Manuelle Schritte sind dabei nur bei der Handhabung der externen UML-Werkzeuge notwendig, z. B. beim Import und Layout. Prinzipiell sind diese Schritte jedoch auch automatisierbar. Bei der Diagrammgenerierung bestehen vielfältige Erweiterungsmöglichkeiten: Einerseits ist das Generieren weiterer Diagrammarten denkbar, wie Anwendungsfalldiagramme oder Sequenzdiagramme. Andererseits lassen sich so auch verschiedene Abbildungen von formalisierten Anwendungsfällen auf eine Diagrammart integrieren, um beispielsweise die

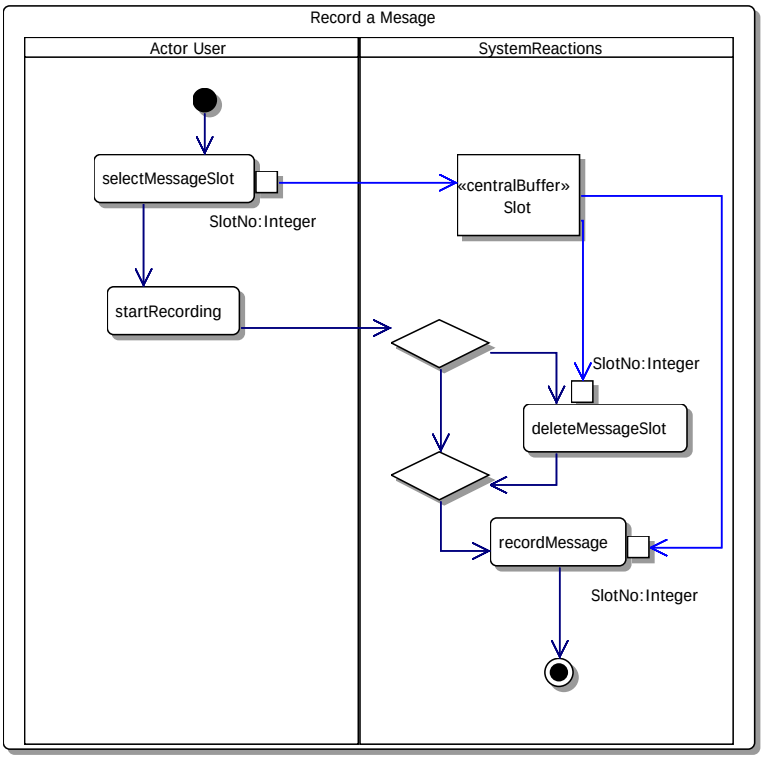

Abbildung 7: Ergebnis der Transformation ist die Visualisierung als UML-Aktivitätsdiagramm

fehlende Unterstützung spezieller UML-Konstrukte in einem Modellierungswerkzeug zu umgehen.

Durch den von uns gewählten Ansatz einer metamodellbasierten Transformation ist es möglich, auch Anwendungsfälle mit komplexeren Sprachkonstrukten zu berücksichtigen. Es ist weitere Forschungsarbeit notwendig, um die Möglichkeiten und Grenzen dieser Technik zu ermitteln.

Literatur

[ATL05] ATLAS Group, LINA & INRIA Nantes. ATL Transformation Description Template, Version 0.1, Dezember 2005.

[BEG+03] Frank Budinsky, Ray Ellersick, Timothy J. Grose, Ed Merks und David Steinberg. *Eclipse Modeling Framework*. The Eclipse Series. Addison-Wesley, 2003.

[Bor] Borland. Together 2006 Release 2 for Eclipse. http://www.borland.com/together/.

[Buw99] Hans Buwalda. Testing with Action Words. In Mark Fewster und Dorothy Graham, Hrsg., *Software Test Automation*, Kapitel 22, Seiten 446–464. Addison-Wesley, 1999.

[Ecl] Eclipse Foundation. Eclipse. http://www.eclipse.org/.

[FH06] Mario Friske und Konrad Hilse. Evaluation von Transformationsmaschinen in der modellbasierten Qualitätssicherung. Beiträge der 36. Jahrestagung der Gesellschaft für Informatik e.V. (Band2), LNI, Bd. P-94, Oktober 2006.

[FP05] Mario Friske und Holger Pirk. Werkzeuggestützte interaktive Formalisierung textueller Anwendungsfallbeschreibungen für den Systemtest. Beiträge der 35. Jahrestagung der Gesellschaft für Informatik e.V. (Band 2), LNI, Bd. P-68, September 2005.

[FS05] Mario Friske und Holger Schlingloff. Von Use Cases zu Test Cases: Eine systematische Vorgehensweise. In T. Klein, B. Rumpe und B. Schätz, Hrsg., *Tagungsband des Dagstuhl Workshops "Modellbasierte Entwicklung eingebetteter Systeme"(MBEES)*. Technische Universität Braunschweig, Januar 2005.

[GLM+05] Gonzalo Génova, Juan Llorens, Pierre Metz, Rubén Prieto-Díaz und Hernán Astudillo. Open Issues in Industrial Use Case Modeling. *Journal of Object Technology*, 4(6):7–14, August 2005. Special Issue: Use Case Modeling at UML-2004.

[Hil06] Konrad Hilse. Vergleich und Evaluation von Modelltransformationssprachen zur Generierung von UML-Diagrammen. Diplomarbeit, HU Berlin, August 2006.

[Hus04] Kenn Hussey. Getting Started with UML2. IBM, Juli 2004.

[JK05] Frédéric Jouault und Ivan Kurtev. Transforming models with ATL. In *Proceedings of Model Transformations in Practice Workshop*, 2005.

[KWB03] Anneke Kleppe, Jos Warmer und Wim Bast. *MDA Explained: The Model Driven Architecture - Practice and Promise*. Object Technology Series. Addison-Wesley, 2003.

[Obj03] Object Management Group. XML Metadata Interchange (XMI) Specification, Version 2.0 (formal/03-05-02). http://www.omg.org/, 2003.

[PL99] Ivan Porres Paltor und Johan Lilius. Digital Sound Recorder - A Case Study on Designing Embedded Systems Using the UML Notation. TUCS Technical Report No. 234, Turku Center for Computer Science, 1999.

[Wil01] Clay E. Williams. Toward a Test-Ready Meta-model for Use Cases. In Andy Evans, Robert France, Ana Moreira und Bernhard Rumpe, Hrsg., *Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists. Workshop of the pUML-Group held together with the UML 2001 October 1st, 2001 in Toronto, Canada*, LNI, Bd. P-7, Seiten 270–287, Oktober 2001.