Erzeugung detaillierter und ausführbarer Geschäftsprozessmodelle durch Modell-zu-Modell-Transformationen

Thomas Allweyer

Fachbereich Informatik und Mikrosystemtechnik
Fachhochschule Kaiserslautern
Standort Zweibrücken, Amerikastr. 1
66482 Zweibrücken
thomas.allweyer@fh-kl.de

Abstract: Es wird ein Ansatz vorgestellt, mit dem sich aus grobgranularen fachlichen Geschäftsprozessmodellen detaillierte, ggf. auch technisch verfeinerte, ausführbare Modelle in verschiedenen Zielnotationen erzeugen lassen. Hierzu wird ein der modellgetriebenen Software-Entwicklung vergleichbarer Ansatz gewählt, bei dem Modellelemente geeignet ausgezeichnet und dann mit Hilfe von parametrisierbaren Templates und Transformationsregeln in detaillierte Zielmodelle überführt werden. Vorteile sind eine Reduktion des Modellierungsaufwandes, einheitlichere und qualitativ hochwertigere Modelle und eine bessere und direktere Umsetzung fachlicher Modelle in ausführbare Prozessbeschreibungen.

1 Einleitung

Geschäftsprozesse werden zumeist auf verschiedenen Detaillierungsebenen modelliert. Die Spanne reicht von unternehmensweiten Prozesslandkarten mit groben Prozessclustern bis hin zu detaillierten Prozessbeschreibungen mit genauer Angabe von beteiligten Mitarbeiterrollen, Input- und Outputdaten, aufzurufenden Systemtransaktionen, usw. Einen sehr hohen Detaillierungsgrad weisen insbesondere Modelle auf, die von Business Process Management-Systemen (BPMS) direkt ausgeführt werden können. Hier müssen alle technischen Details, wie z. B. Parameter mit ihren Datentypen, die exakte Formulierung von Bedingungen usw. definiert sein, damit eine eindeutige und vollständige Ausführungsvorschrift für die verwendete Process Engine vorliegt.

Die meisten Methoden und Werkzeuge zur Geschäftsprozessmodellierung sehen daher die Möglichkeit einer hierarchischen Modellierung vor. In der Regel kann einer grobgranularen Funktion ein Modell eines Detailprozesses hinterlegt werden. Auch den Funktionen eines solchen Detailprozesses lassen sich wiederum detailliertere Prozessmodelle hinterlegen, usw. Auf diese Weise können prinzipiell beliebig tiefe Prozesshierarchien aufgebaut werden. In der Praxis findet man meist drei bis sechs Hierarchiestufen von der gesamten Unternehmensprozesslandkarte bis zum feinsten Detailmodell.

Um eine gewisse Einheitlichkeit zu erreichen, werden zumeist Modellierungskonventionen aufgestellt, in denen festgelegt wird, welche Details auf welcher Ebene modelliert werden sollen [Al05a].

Wie die Konsistenz zwischen über- und untergeordnetem Modell sichergestellt wird, ist ganz unterschiedlich geregelt: Z. T. gibt es die Möglichkeit, jedem Modellelement beliebige Modelle mit beliebigen Inhalten zu hinterlegen. In diesem Fall ist der Modellierer komplett selbst für das Zusammenpassen von unter- und übergeordnetem Modell verantwortlich. Im Falle von stärker formalisierten Modellierungsnotationen (z. B. UML Aktivitätsdiagramm[1] oder BPMN[2]) erzwingt die Notation die Einhaltung von Konsistenzregeln beim Verfeinern, indem In- und Output einer Funktion auch im hinterlegten Modell als In- und Output des Detailprozesses vorkommen müssen.

Das Erstellen sehr detaillierter Modelle ist recht aufwändig, weshalb eine Aufwands-/ Nutzen-Abschätzung erforderlich ist, um zu entscheiden, ob sich der Aufwand für einen sehr hohen Detaillierungsgrad lohnt, oder ob eine vergleichsweise grobe Modellierung ausreicht. Für manche Zwecke, wie genaue Analysen (z. B. mit Hilfe von Simulation), die Spezifikation von Informationssystemen, oder die Ausführung durch ein BPMS, sind hingegen zwangsläufig hinreichend detaillierte Modelle mit einer genauen Einhaltung von bestimmten Formalismen erforderlich, ohne die eine korrekte Simulation oder Ausführung nicht möglich sind. Problematisch ist, dass sich eine einheitliche Modellierung über sehr viele, zumeist von verschiedenen Modellierern erstellte Detailmodelle hinweg kaum sicherstellen lässt. Vergleichbare Sachverhalte werden daher oftmals sehr verschieden dargestellt. Auch die Sicherstellung der Qualität ist bei einer sehr großen Zahl detaillierter Modelle schwierig, wenn diese Modelle von Hand erstellt werden.

Erschwert wird die geschilderte Problematik, wenn Prozessmodelle für unterschiedliche Zwecke in verschiedenen Notationen und/oder Tools erstellt werden müssen. Häufig werden innerhalb desselben Unternehmens sowohl High End-Modellierungs-Suiten als auch einfache Grafiktools, spezielle Analyse- und Simulationswerkzeuge und die Modellierungskomponenten von Business Process Management-Systemen verwendet. Zumeist nutzen diese Tools verschiedene Notationen, oder sie verlangen auch bei gleichen Notationen die Einhaltung spezifischer Regeln oder die Verwendung von proprietären Notations-Erweiterungen. Hier ergibt sich die Problematik des Austausches von Modellen zwischen unterschiedlichen Modellierungstools sowie der Konvertierung zwischen verschiedenen Notationen (oder gar einer Übersetzung zwischen verschiedenen Modellierungs-Stilen innerhalb derselben Notation).

Hierfür existiert bereits eine Reihe von Ansätzen, z. B. zur Konvertierung von EPK in BPMN oder BPEL[3] und umgekehrt (z. B. [Ko05], [MZ05]). Diese Ansätze verfolgen die Zielsetzung einer möglichst guten eins-zu-eins-Übersetzung der Modellinhalte in die jeweilige Zielnotation. Andererseits ist eine solche in vielen Fällen gar nicht sinnvoll.

[1] UML steht für die in der objektorientierten Software-Entwicklung gebräuchliche „Unified Modeling Language“ [Om07], vgl. auch [Hi05], [Ru07].

[2] BPMN steht für die „Business Process Modeling Notation“, eine grafische Notation für die Modellierung von (insbesondere ausführbaren) Geschäftsprozessen [Om06a], vgl. auch [Ha05].

[3] BPEL steht für „Business Process Execution Language“, eine XML-basierten Sprache zur Spezifikation der Orchestrierung von Web Services im Rahmen ausführbarer Geschäftsprozesse [Oa07], vgl. auch [Ha05].

Wird ein Detailmodell eines bestimmten Prozesses nur für ein spezielles Tool, z. B. für die Ausführung in einem spezifischen BPMS, benötigt, so ist es nicht erforderlich, das betreffende Modell zunächst etwa als EPK zu erstellen und dann mit oftmals problembehafteten Konvertierungsmechanismen in die von dem BPMS benötigte Zielnotation zu überführen. Stattdessen genügt es in diesem Fall, das Detailmodell von Vornherein in der Modellierungskomponente des BPMS und der dort eingesetzten Notation zu erstellen. Wendet man dieses Verfahren (jedes Teil-Modell nur in einem Tool) konsequent an, so treten andererseits die oben diskutierten Probleme der Konsistenz und Einheitlichkeit durch die mangelnde Tool- und Methodenintegration in noch stärkerem Maße auf.

Werden verschiedene Methoden und Tools verwendet, so werden diese i. d. R. auch von verschiedenen Personen beherrscht und angewandt. Die Schwierigkeiten in der Kommunikation von Modellierern mit ganz unterschiedlichen Methoden- und Toolkenntnissen verschärfen die Problematik weiter. Leicht führt diese Situation dazu, dass die in den verschiedenen Tools erstellten Modelle völlig unabhängig voneinander (weiter) entwickelt werden, so dass überhaupt keine übergreifende Modellintegration und somit keine Gesamtsicht auf die Geschäftsprozesse mehr möglich ist.

Trotz der Entwicklung leistungsfähiger und flexibler Business Process Management-Systeme (BPMS) mit grafischen Modellierungsoberflächen bleibt in einem solchen Fall die viel beklagte Kluft zwischen Fachbereich und IT weitgehend bestehen: Die Entwicklung ausführbarer Prozesse erfolgt durch BPMS-Experten, der Bezug zu den fachlichen Geschäftsprozessmodellen fehlt weitgehend, oder muss durch aufwändige organisatorische Maßnahmen sichergestellt werden.

Die beiden genannten Problemfelder, nämlich die effiziente Erstellung qualitativ hochwertiger Detailmodelle einerseits, und die tool- und methodenübergreifende Modellintegration andererseits, sind eng miteinander verknüpft. Der im Folgenden zur Diskussion gestellte Ansatz ist dazu geeignet, einen Lösungsbeitrag für beide Problemfelder zu liefern.

Bei der Erarbeitung dieses Ansatzes wurde folgendermaßen vorgegangen: Zunächst wurden ausgehend von realen Modellen aus der Praxis Beispielmodelle für ausgewählte Sachverhalte erstellt. Anhand dieser wurden Art und Umfang der notwendigen Übergänge von grobgranularen Modellen zu detaillierten fachlichen und zu ausführbaren Modellen untersucht. Anschließend wurden existierende Ansätze zur Verfeinerung und Transformation von Modellen daraufhin analysiert, ob sie geeignet sind, die erforderlichen Übergänge zu schaffen. Als Ergebnis wurde ein Ansatz der Modell-zu-Modell-Transformation entwickelt, der sich an das Vorgehen der modellgetriebenen Software-Entwicklung anlehnt und die Grundidee auf die Transformation von Geschäftsprozessmodellen überträgt. Dieser Ansatz wurde mit Hilfe der entwickelten Beispielmodelle konkretisiert und validiert.

2 Modell-Transformationen in der Software-Entwicklung

Die Grundidee des vorgestellten Ansatzes ist es, durch Modell-zu-Modell-Transformationen detaillierte und ggf. ausführbare Prozessmodelle aus grobgranularen, abstrakteren Prozessmodellen zu generieren, anstatt diese manuell zu modellieren. Die Idee lehnt sich an aktuelle Ansätze zur modellgetriebenen Software-Entwicklung an, wie sie unter den Stichworten „Model Driven Architecture“ (MDA, [Om03]) oder „Model Driven Software Development“ (MDSD) diskutiert werden [St07], und überträgt diese in den Bereich der Geschäftsprozessmodellierung.

In der modellgetriebenen Software-Entwicklung werden Modelle verwendet, um daraus in einem oder mehreren Transformationsschritten Code zu generieren. Die Modelle werden typischerweise in UML oder einer domänenspezifischen Sprache (Domain Specific Language, DSL) erstellt. Sie weisen einen wesentlich höheren Abstraktionsgrad als der erzeugte Code auf. So wird etwa aus einer modellierten UML-Klasse mit ihren Attributen nicht nur eine entsprechende Java-Klasse generiert. Es können vielmehr zugleich auch ein Datenbankschema für die persistente Speicherung, der Zugriffscode für die Datenbank, der Code für die Darstellung und Manipulation entsprechender Objekte in der Benutzeroberfläche usw. generiert werden.

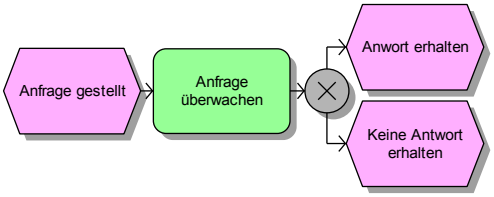
Hierzu werden Code-Templates verwendet. Ein Generator füllt diese Templates mit den Inhalten des Ausgangsmodells und fügt die generierten Code-Teile konsistent zusammen. Hierfür müssen geeignete Transformationsregeln definiert sein, die vom Generator ausgeführt werden. Aus ein und demselben Modell können durch Verwendung verschiedener Templates und Transformationsregeln ganz unterschiedliche Artefakte generiert werden, u. a. auch Code in diversen Programmiersprachen und für verschiedene technische Plattformen.

Z. T. ist es erforderlich, die Modellinhalte im Hinblick auf die Transformation speziell zu markieren, um z. B. festzulegen, für welche der modellierten Klassen ein persistenter Speicherungsmechanismus erzeugt werden soll. Die UML bietet für eine solche Auszeichnung beliebiger Modellelemente das Konstrukt des „Stereotyps“ an. Zusätzlich erforderliche Parameter für die Transformation können als so genannte „Tagged Values“ (Eigenschaftswerte) angegeben werden.

Bei den Transformationen muss es sich nicht ausschließlich um Transformationen von grafisch orientierten Modellen in Code handeln, vielmehr können beispielsweise auch Text-basierte Beschreibungen (z. B. XML-Dateien) transformiert oder erzeugt werden, und es sind auch Modell-zu-Modell-Transformationen möglich, wobei aus einem abstrakteren Modell ein konkreteres Modell erzeugt wird. Der von der Object Management Group (OMG) definierte MDA-Ansatz sieht explizit mehrere Modelltypen mit unterschiedlichen Abstraktionsgraden vor, wobei Computation Independent Models (CIM) nacheinander in Platform Independent Models (PIM) und schließlich in Platform Specific Models (PSM) transformiert werden können. Im Sinne der modellgetriebenen Software-Entwicklung kann der Übergang von grobgranularen zu detaillierteren Geschäftsprozessmodellen ebenfalls als Modell-zu-Modell-Transformation aufgefasst werden.

3 Beispiel: Überwachung von Anfragen

Die entwickelte Vorgehensweise wird anhand eines Beispiels erläutert. Häufig ist es erforderlich, die Reaktion auf eine Anfrage, eine Bestellung, eine versandte Rechnung o. ä. zu überwachen. Hierzu muss überprüft werden, ob die gewünschte Antwort, Lieferung oder Zahlung innerhalb einer gewissen Frist eingegangen ist. Gegebenenfalls muss nachgefragt oder gemahnt werden. Oftmals wird eine solche Überwachung pauschal als einzelne Funktion modelliert. Die wesentlichen Details können in Form eines beschreibenden Textes hinterlegt werden.



Beschreibung der Funktion „Anfrage überwachen“:

Falls keine Antwort eingeht, werden bis zu zwei Nachfragen gestellt: Nach 3 Wochen und ggf. nach 2 weiteren Wochen. Erfolgt auch dann noch keine Reaktion, so wird nach weiteren 2,5 Wochen eine Absage versandt.

Geht eine Antwort erst danach ein, so wird diese wegen verspäteten Eingangs abgelehnt.

Abbildung 1: „Anfrage überwachen“ in EPK sowie Beschreibung der Details

Abbildung 1 stellt eine solche pauschale Modellierung einer Überwachungsfunktion dar. In vielen Fällen mag diese Modellierung ausreichen, wenn sich die zuständigen Mitarbeiter bei der Prozessdurchführung eigenverantwortlich um die geeignete Realisierung des Detailablaufs kümmern.

Eine Möglichkeit, die Anfrage-Überwachung detailliert zu modellieren, ist in Abbildung 2 dargestellt. Bei der Darstellung dieses Ablaufs mit Hilfe der EPK ergeben sich eine Reihe methodischer Fragestellungen, insbesondere hinsichtlich der Darstellung von zeitlichen Ereignissen, der Auswertung von Bedingungen beim Eintreten von Ereignissen sowie dem Abbrechen von Teilprozessen. Diese Fragestellungen sind jedoch nicht Gegenstand des vorliegenden Beitrags und werden daher nicht weiter vertieft. Zielsetzung bei der Erstellung des gezeigten Modells war eine möglichst gute Verständlichkeit. Im Hinblick auf eine formale Analysierbarkeit, eine Simulation oder Ausführung des Prozesses wäre die gewählte Art der Darstellung eher ungeeignet.

Im Vergleich zur groben Modellierung in Abbildung 1 ist das detaillierte Modell wesentlich komplexer und daher zunächst auch unübersichtlicher und weniger leicht zu verstehen. Von daher wird man sich in fachlich orientierten Modellen, die vor allem dem Prozessverständnis dienen, häufig auf die grobe Darstellung beschränken.

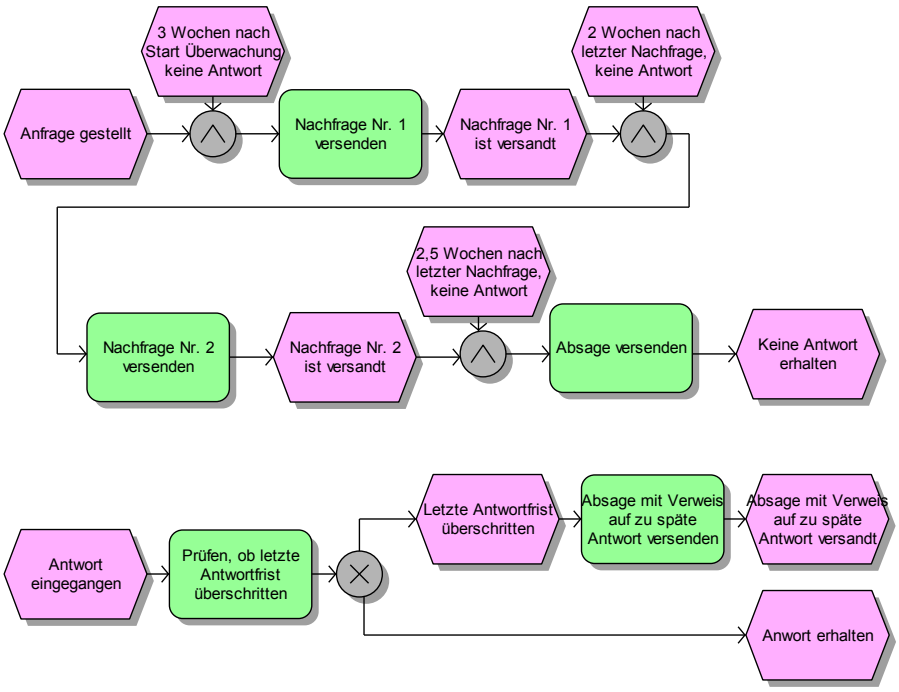


Abbildung 2: Ausführliche Modellierung der Überwachung einer Anfrage

Dennoch ist in vielen Fällen auch ein detailliertes Modell erforderlich, wenn z. B. eine detaillierte Handlungsanweisung für angelerntes Personal benötigt wird, wenn der genaue Prozessablauf aus Gründen gesetzlicher Nachweispflichten verbindlich dokumentiert sein muss, oder wenn der Prozess automatisiert von einer Process Engine ausgeführt werden soll. Im Falle der Automatisierung wird das ausführliche Modell i. d. R. nicht in Form einer EPK benötigt, sondern in einer Ausführungssprache, z. B. BPEL. Die im Folgenden vorgestellte Vorgehensweise lässt sich prinzipiell auch für andere Zielsprachen und –notationen als die EPK durchführen.

Das grobe Prozessmodell aus Abbildung 1 lässt sich auf verschiedene Arten in ein detailliertes Modell umsetzen. In Abbildung 2 wurden die in der Beschreibung enthaltenen Parameter (Zahl der Nachfragen, Fristen) direkt in Form des Kontrollflusses umgesetzt. Eine alternative Modellierung für die obere Hälfte der Abbildung 2 zeigt Abbildung 3. Hier wurden die Parameter aus der eigentlichen Kontrollflusslogik ausgelagert. Die in den Funktionen und Ereignissen genannte Maximalzahl und die Fristen müssen bei der Prozessdurchführung dem Informationsobjekt „Anfrage-Parameter“ entnommen werden.

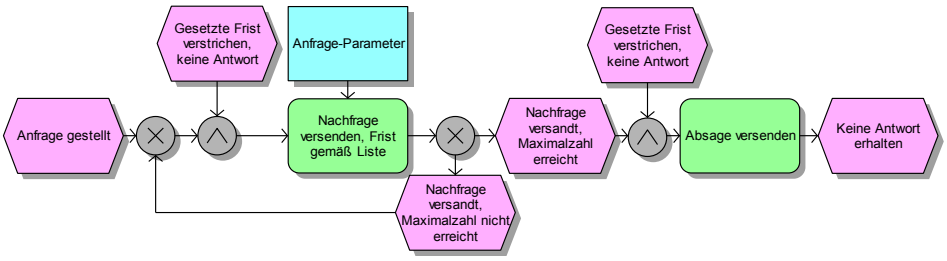


Abbildung 3: Alternatives Modell zur oberen Hälfte von Abbildung 2

Vorteil ist eine größere Flexibilität: Ändern sich lediglich die Parameter, so muss das Modell im Gegensatz zu Abbildung 2 nicht geändert werden. Vorteil der Variante aus Abbildung 2 hingegen ist die größere Aussagekraft des Modells an sich. Auch lassen sich reine Kontrollflusskonstrukte direkter in ausführbare Modelle umsetzen, wohingegen die Parameter-Abfrage und –Auswertung häufig zusätzlichen Realisierungsaufwand bedeuten.[4]

Im Folgenden soll gezeigt werden, wie ein derartiges Detail-Modell mit Hilfe einer geeigneten Modell-zu-Modell-Transformation automatisch generiert werden kann.

Ausgangspunkt der Transformation ist das Grobmodell aus Abbildung 1. Mit Hilfe einer Transformation soll daraus das detaillierte Modell aus Abbildung 2 erzeugt werden. Um eine entsprechend Transformation definieren zu können, muss im Ausgangsmodell ein bestimmtes Muster verwendet werden. Für das hier diskutierte Beispiel der Überwachung einer Anfrage o. ä. ist dieses Muster relativ simpel (linke Seite von Abbildung 4): Es handelt sich um eine Funktion mit zwei alternativen Folge-Ereignissen, wobei das eine Folge-Ereignis das positive Ergebnis einer erfolgreichen Überwachung repräsentiert, das andere das negative Ergebnis, d. h. es ist keine Antwort o. ä. eingegangen. Das auslösende Ereignis spielt hierbei keine wesentliche Rolle, es repräsentiert lediglich den vorangehenden Teil des Prozessmodells, das zum Starten der Funktion führt. Hierbei könnte es sich etwa auch um mehrere durch Konnektoren miteinander verbundene Ereignisse handeln.

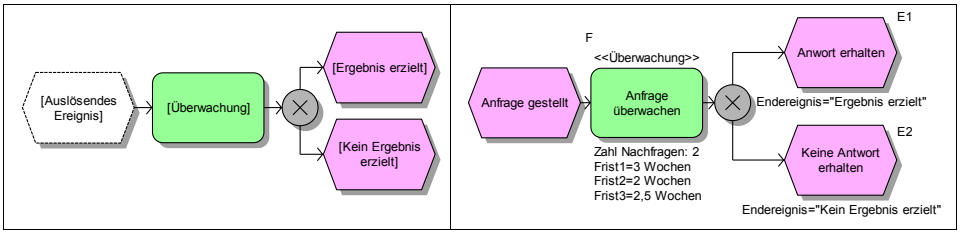


Abbildung 4: Muster „Überwachung“ (links), Anwendung des Musters in einer EPK (rechts)

[4] Die Auslagerung von Regeln aus der eigentlichen Prozesslogik lässt sich mit Hilfe von Business Rules Management-Systemen realisieren, vgl. z. B. [De05], [SG06], [Wa07].

Auf der rechten Seite von Abbildung 4 ist die konkrete Anwendung dieses Musters in einer als Ausgangsmodell dienenden EPK dargestellt. Das herkömmliche Modell aus Abbildung 1 muss um einige transformationsspezifische Informationen ergänzt werden. Zum einen muss angegeben werden, welches Muster verwendet werden soll. Hierfür wurde das aus der UML stammende Konstrukt der „Stereotypen“ verwendet, die dazu dienen, Modellelemente nach beliebigen Kriterien zu kategorisieren. In Beispiel wurde die Funktion F mit dem Stereotyp «Überwachung» ausgezeichnet. Weiterhin muss eine Zuordnung der verschiedenen Modellelemente zu den betreffenden Elementen des Musters erfolgen. Im Beispiel wurden hierzu die beiden Ereignisse E1 und E2 mit einem ebenfalls der UML entlehnten Tagged Value „Endereignis“ versehen. Auch die benötigten Parameter (Zahl der Nachfragen und Fristen) sind in Form von Tagged Values angegeben.

Abbildung 5 zeigt, was zur Transformation des derart angereicherten Ausgangsmodells noch benötigt wird. Zunächst ist es sinnvoll, das Ausgangsmodell zu validieren. Hierdurch wird sichergestellt, dass es richtig strukturiert ist und alle erforderlichen Informationen enthält, um erfolgreich transformiert zu werden. In Abbildung 6 sind entsprechende Validierungsregeln für die Verwendung des Stereotyps «Überwachung» dargestellt.

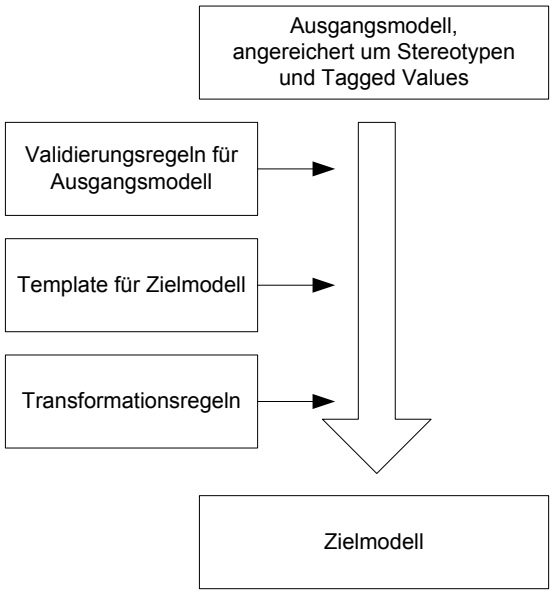


Abbildung 5: Modell-zu-Modell-Transformation

```
- Die Funktion mit dem Stereotyp „Überwachung“ (hier als F
  bezeichnet) verfügt über:
  1. einen Tagged Value „Zahl Nachfragen“ mit einem
     Integer-Wert > 1
  2. So viele Tagged Values „Frist [i]“ wie die Zahl der
     Nachfragen (Typ Zeitdauer).
- Auf die Funktion folgt eine XOR-Verknüpfung, die genau zwei
  Folge-Ereignisse hat.
- Eines der beiden Folge-Ereignisse (hier als E1 bezeichnet)
  hat einen Tagged Value „Endereignis“ mit dem Wert „Ergebnis
  erzielt“.
- Das andere der beiden Folge-Ereignisse (hier als E2
  bezeichnet) hat einen Tagged Value „Endereignis“ mit dem
  Wert „Kein Ergebnis erzielt“.
```

Abbildung 6: Validierungsregeln für die Verwendung des Stereotyps «Überwachung»

Basis für die Erstellung des Zielmodells ist das in Abbildung 7 gezeigte Template. Es enthält bereits die Struktur und die wesentlichen Elemente des Zielmodells. Eine Reihe von Platzhaltern muss jedoch noch mit konkreten Werten gefüllt werden. Außerdem enthält es eine Sequenz, die ggf. mehrfach wiederholt werden kann. Hier wurde die ausführliche Modellierung der Überwachung gewählt (vgl. Abbildung 2). Alternativ hätte auch die kompaktere Darstellung nach Abbildung 3 verwendet werden können.

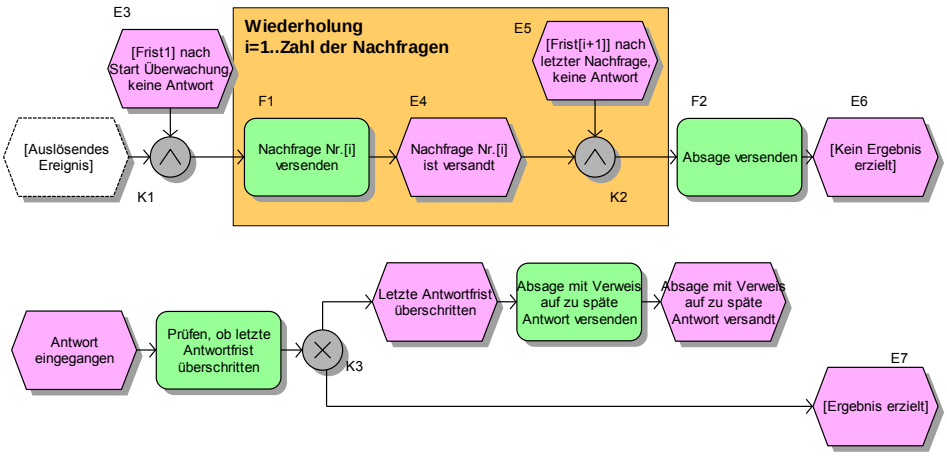


Abbildung 7: Template für die detaillierte Modellierung einer Überwachung

Bei der eigentlichen Transformation wird dieses Template mit Hilfe der in Abbildung 8 dargestellten Transformationsregeln und des Ausgangsmodells (rechte Seite von Abbildung 4) angepasst und in das in Abbildung 2 gezeigte Zielmodell überführt.

```
- Verbinde die in F eingehende(n) Kante(n) mit Konnektor K1
  statt mit F.
- Entferne F.
- Ersetze in E3 „[Frist1]“ durch den Wert des Tagged Value
  Frist1 von F.
- Setze i=1. Solange i kleiner oder gleich der Zahl der
  Nachfragen ist, wiederhole Folgendes:
    • Erzeuge neue Objekte und Kanten für die Vorlagen im
      Rechteck „Wiederholung“.
    • Wenn i=1: verbinde K1 mit F1, sonst verbinde K2 der
      vorangehenden Wiederholung mit F1.
    • Ersetze in F1 und E4 jeweils „[i]“ durch den Wert von i.
    • Ersetze in E5 „Frist[i+1]“ durch den Wert des Tagged
      Value Frist[i+1] von F.
- Verbinde K2 der letzten Wiederholung mit F2.
- Verbinde F2 mit E2 statt mit E6.
- Verbinde K3 mit E1 statt mit E7.
```

Abbildung 8: Transformationsregeln für den Stereotyp «Überwachung»

Die dargestellten Transformationsregeln ersetzen die ursprüngliche Funktion im Ausgangsmodell durch die neu erzeugten Modellelemente. Es wird also kein neues Modell erzeugt, sondern das vorhandene Modell modifiziert. Im vorliegenden Fall könnte man die Transformationsregeln auch derart verändern, dass stattdessen ein neues Modell erzeugt würde, das der Funktion „Anfrage überwachen“ hinterlegt wird. Es sind jedoch andere Fälle denkbar, bei denen das Ausgangsmuster beispielsweise mehrere Funktionen umfasst, so dass das erzeugte Modell nicht als Detaillierung einer einzelnen Funktion dargestellt werden kann.

4 Transformationen beim Übergang zu ausführbaren Modellen

Soll ein auf fachlicher Ebene modellierter Geschäftsprozess durch ein Business Process Management-System (BPMS) unterstützt werden, so ist die Überführung in eine ausführbare Prozessbeschreibung erforderlich. Liegt das Prozessmodell nicht in einer von der Process Engine des verwendeten Systems direkt ausführbaren Notation vor, so ist zunächst eine Transformation in eine andere Notation oder Beschreibungssprache notwendig, z. B. von EPK zu BPMN oder BPEL.

Wie in der Einleitung bereits ausgeführt, geht es nicht darum, eine möglichst identische Konvertierung der Inhalte aus einer Notation in eine andere durchzuführen. Stattdessen soll aus einem grobgranularen, fachlichen Modell (hier in Form einer EPK) ein detailliertes, ausführbares Modell in der Notation des Zielsystems erzeugt werden.

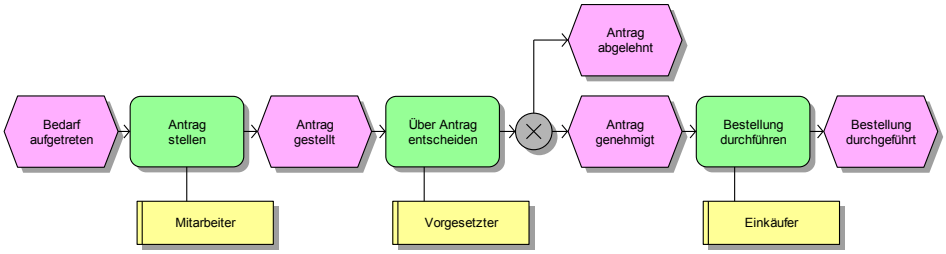


Abbildung 9: Fachliche EPK eines Ablaufs, der durch Workflow unterstützt werden soll

Abbildung 9 zeigt das als EPK modellierte fachliche Modell eines einfachen Beschaffungsprozesses. Soll dieser Prozess nun mit Hilfe des Systems Intalio|BPMS workflowunterstützt ablaufen, so ist eine Überführung in das BPMN-Modell nach Abbildung 10 erforderlich.

Bereits auf den ersten Blick wird deutlich, dass eine reine 1:1-Umsetzung der EPK-Logik in die BPMN nicht ausreicht. Zur Modellierung von Workflows, die mit menschlichen Benutzern interagieren, verlangt Intalio die Aufteilung in einen System-Pool für die Modellierung des eigentlichen Kontrollflusses sowie einen separaten Pool für jede Mitarbeiter-Rolle. Eine einzelne von einem Mitarbeiter durchzuführende Funktion muss in bis zu vier BPMN-Tasks aufgeteilt werden.

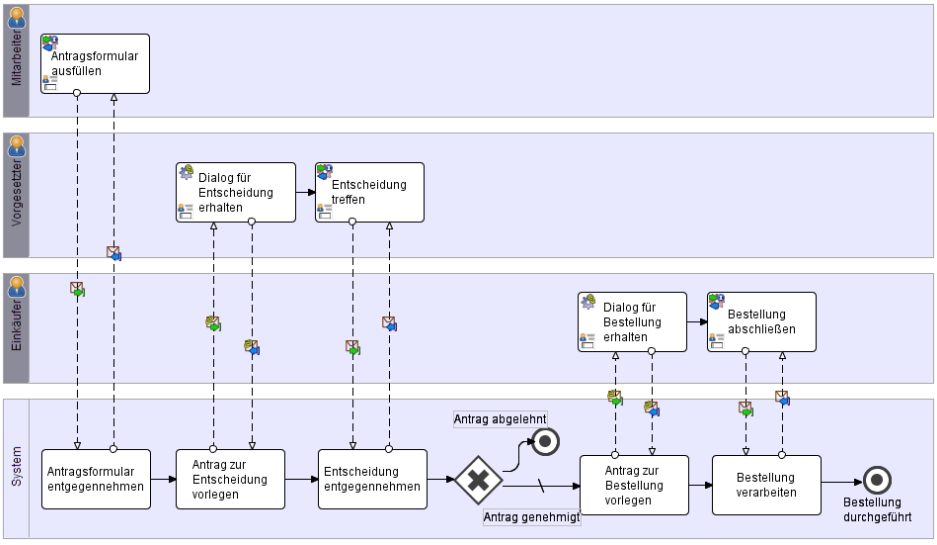


Abbildung 10: Umsetzung der EPK aus Abbildung 9 in ein ausführbares BPMN-Modell (für Intalio|BPMS)

Beispielsweise werden für die Funktion „Über Antrag entscheiden“ vier Tasks benötigt. Der erste Task „Antrag zur Entscheidung vorlegen“ übermittelt die benötigten Daten, der zweite Task erstellt den Dialog und trägt die Aufgabe in die Task-Liste der betreffenden

Rolle ein, der dritte Task „Entscheidung treffen“ beinhaltet das Ausfüllen und Absenden des Dialogs, und der vierte Task „Entscheidung entgegennehmen“ ist schließlich für die Verarbeitung des Ergebnisses zuständig.

Mit Hilfe des vorgestellten Ansatzes lässt sich eine derartige Aufteilung einer Funktion in vier durch Kontroll- und Nachrichtenflüsse verbundene Tasks automatisieren. Hierzu kann ein geeigneter Stereotyp „Workflow-Funktion“ definiert werden, für den ein BPMN-Template mit den vier Funktionen sowie geeignete Transformationsregeln existieren müssen.

Darauf aufbauend kann ein Stereotyp für die Entscheidungsfunktion mit genau zwei möglichen Ergebnissen definiert werden, der die Umsetzung in die BPMN-Logik regelt. Z. B. muss sichergestellt werden, dass die Entscheidungsfunktion eine Boole’sche Variable als Ergebnis liefert, die mit Hilfe geeignet formulierter Bedingungen steuert, welche der aus dem Gateway ausgehenden Kontrollflusskanten gewählt wird.

Im Gegensatz zu einer 1:1-Umsetzung von EPK-Modellen in eine ausführbare Notation können mit dem vorgestellten Ansatz in einem Schritt auch notwendige Verfeinerungsschritte beim Übergang von einem fachlichen in ein detailliertes, ausführbares Modell durchgeführt werden.

Dieser Schritt ist vergleichbar mit der im Software Engineering erfolgten Entwicklung weg von der Code-Generierung in traditionellen CASE-Tools hin zur Model Driven Architecture (MDA). Traditionelle CASE-Tools boten eine 1:1-Umsetzung grafischer Modelle (z. B. UML-Modelle) in Programmcode. Dies bedeutete, dass die Modelle auf dem gleichen Abstraktions- und Detaillierungsgrad wie der endgültige Programmcode sein mussten. Der MDA-Ansatz hingegen erlaubt die Generierung umfangreicheren Programmcodes aus abstrakteren Modellen. Hierdurch werden insbesondere bei Mehrfachverwendung der Modelle Produktivitäts- und Qualitätsvorteile erzielt. Mit Hilfe verschiedener Templates und Transformationsregeln kann z. B. aus ein und demselben Modell Code für ganz verschiedene Zielplattformen erzeugt werden.

Dies kann mit dem vorgestellten Ansatz ebenso für die Prozessausführung erreicht werden. Beispielsweise können fachliche Prozesse teilweise von unterschiedlichen Process Engines ausgeführt werden. Neben eigenständigen BPMS können innerhalb eines Prozesses z. B. auch Workflow-Komponenten eines Content Management-Systems oder eines Anwendungssystems zum Einsatz kommen.

Herkömmlich führt der doch erhebliche manuelle Aufwand bei der Umsetzung fachlicher Modelle in ausführbare Modelle häufig dazu, dass die ausführbaren Modelle von Grund auf neu entwickelt, in der Ausführung getestet und verbessert werden, und fachliches und ausführbares Modell völlig voneinander entkoppelt werden.

5 Anwendungsmöglichkeiten fachlicher Stereotypen

In der modellgetriebenen Software-Entwicklung werden Stereotypen eher für implementierungsbezogene Kategorisierungen verwendet, z. B. um anzugeben, dass Objekte einer bestimmten Klasse persistent gespeichert werden sollen. Bei der Software-Generierung wird für diese Klasse dann der notwendige Code für die betreffenden Datenbankzugriffe erzeugt.

Das in Abschnitt 4 vorgestellte Beispiel des Übergangs von fachlichen zu ausführbaren Modellen entspricht dieser Idee. Anstelle von Code in einer herkömmlichen Programmiersprache wird eine Implementierung in Form eines von einer Process Engine ausführbaren Modells erzeugt.

Das in Abschnitt 3 vorgestellte Beispiel des Stereotyps «Überwachung» bezieht sich hingegen auf fachliche Aspekte. Die prinzipielle Vorgehensweise ist dennoch die gleiche. In beiden Fällen wird nämlich eine Konkretisierung und Detaillierung des betreffenden grobgranularen, abstrakteren Modellelements vorgenommen, einmal in Form von Programmcode (bzw. ausführbaren Prozessmodellen), im anderen Fall in Form von detaillierteren fachlichen Anweisungen. Das Detailmodell kann im übertragenen Sinne als eine „organisatorische Implementierung“ der übergeordneten Funktion aufgefasst werden.

Die durch Stereotypen in der Software-Generierung bezeichneten implementierungsbezogenen Konzepte, sind recht klar definier- und abgrenzbar, und sie werden vielfach verwendet. Beispiele sind neben der bereits genannten Persistenz Stereotypen für verschiedene Elemente der Benutzungsoberfläche, für diverse Aufgaben der Verwaltung von Objekten durch eine Middleware (z. B. Kommunikation, Sicherheit, Verteilung) oder für Schnittstellen.

Die für die Transformation fachlicher Prozessmodelle in ausführbare Modelle verwendbaren Stereotypen ergeben sich aus den Funktionalitäten der Process Engines. Neben von Benutzern ausführbaren Workflow-Tasks sind beispielsweise Konzepte wie der Aufruf komplett automatisierter Funktionen, der Versand und Empfang von Nachrichten, Vertreterregelungen, Eskalationsmechanismen, langlaufende Transaktionen oder Ausnahmebehandlungen abzubilden.

Für die Generierung detaillierter fachlicher Modelle sind vergleichbar gut abgrenzbare Muster weniger offensichtlich. Beispiele könnten sein:

- Die Behandlung von Stornierungen und Abbrüchen, die an beliebigen Stellen eines Teilprozesses auftreten können.
- Die Verarbeitung von Änderungen, z. B. eines erteilten Auftrages, an beliebigen Stellen eines Teilprozesses.
- Überprüfungen und Kontrollschritte mit anschließenden Freigaben und ggf. Überarbeitungen.

- Prozesse, deren Bearbeitungsobjekte unter gewissen Umständen aufgeteilt werden können, wobei anschließend jede Teilmenge für sich bearbeitet wird, z. B. das Splitten von Aufträgen in verschiedene Teillieferungen.

Die Identifikation und Analyse derartiger Konzepte wird auch Gegenstand weiterer Forschung sein. Insbesondere ist zu prüfen, in wieweit sich für die einzelnen Konzepte Muster finden lassen, die tatsächlich in verschiedenen Prozesskontexten sinnvoll wiederverwendet werden können.

Es ist auch möglich, ein und dieselbe grobgranulare fachliche Funktion einerseits in Form eines ausführbaren Modells zu implementieren und andererseits organisatorisch mit Hilfe eines detaillierten fachlichen Modells umzusetzen. Hierzu kann das Grobmodell unverändert bleiben, es müssen lediglich die Templates und Transformationsregeln ausgetauscht werden. Insofern lassen sich die beiden vorgestellten Konzepte auch gemeinsam anwenden, wenn etwa nur ein Teil der Prozesse mit Hilfe eines BPMS automatisiert wird, ein anderer Teil hingegen manuell ausgeführt wird.

6 Zusammenfassung und Ausblick

Im vorliegenden Beitrag wurde ein Ansatz vorgestellt, detaillierte und ggf. ausführbare Prozessmodelle durch Modell-zu-Modell-Transformationen aus grobgranularen Prozessmodellen zu generieren. Hierzu werden Templates und Transformationsregeln verwendet. Die Anwendung dieses Ansatzes wurde anhand der Generierung einer Detail-EPK für die Überwachung einer Anfrage sowie anhand der Umsetzung einer einfachen EPK in ein ausführbares BPMN-Modell demonstriert.

Der Ansatz ist dazu geeignet, den Aufwand für die Erstellung von detaillierten Modellen zu reduzieren. Die Einheitlichkeit, Konsistenz und Qualität der Detailmodelle wird erhöht, und der Übergang von fachlichen zu eher technischen, ausführbaren Modellen wird erleichtert.

Wie bereits erläutert lehnt sich der Ansatz an Verfahren zur modellgetriebenen Software-Entwicklung an. Hierzu wurden u. a. einige Elemente der MDA übernommen. Das vorgestellte Verfahren setzt jedoch das MDA-Konzept der OMG nicht komplett um. So wurde beispielsweise auf die Spezifikation von Ziel- und Quellsprache (EPK bzw. BPMN mit den beschriebenen Erweiterungen) mit Hilfe eines MOF-konformen Metamodells[5] und die Repräsentation und Verarbeitung der Modelle mit Hilfe von XMI[6] verzichtet. Ebenso wurde zunächst keine konkrete Transformationssprache verwendet, sondern eine Formulierung der Transformationsregeln in Form von Pseudocode vorgenommen. Beides wäre jedoch ohne weiteres möglich und könnte etwa im Vorfeld einer Software-Implementierung der beschriebenen Transformationen hilfreich sein.

[5] MOF steht für „Meta Object Facility“, einen OMG-Standard zur Beschreibung von Metamodellen [Om06b].
[6] XMI steht für „XML Metadata Interchange“, einen OMG-Standard für den Austausch von Modellen [Om05].

Im Sinne der Referenzmodelladaption lässt sich die dargestellte Vorgehensweise auch als Instanziierung von Referenzmodellen interpretieren[7]: Hierbei kann das in Abbildung 7 gezeigte Template als Referenzmodellbaustein aufgefasst werden, der mit Hilfe des vorgestellten Transformationsmechanismus an den konkreten Anwendungskontext angepasst wird. Die Besonderheit des vorgestellten Ansatzes liegt einerseits in eben diesem, der modellgetriebenen Software-Entwicklung angelehnten Mechanismus begründet, andererseits in der Verwendung von grobgranularen Prozessmodellen mit Stereotypen und Tagged Values zur Spezifikation der durchzuführenden Instanziierung.

Das vorgestellte Verfahren ist zunächst unabhängig von den verwendeten Modellierungssprachen und kann damit insbesondere auch unabhängig von bestimmten Interpretationen der durch eine EPK ausgedrückten Semantik durchgeführt werden. Bei der Erstellung konkreter Templates und Transformationsregeln wird es in der Praxis erforderlich sein, gewisse Annahmen über die zugrunde liegende Semantik zu machen. Ersteller von Templates und Transformationsregeln einerseits sowie Modellierer grobgranularer Prozesse (und damit Anwender der Transformationen) andererseits müssen über ein einheitliches Verständnis der jeweils intendierten Bedeutung verfügen. Dies muss nicht unbedingt über eine formale Definition der Semantik erfolgen, in der Praxis kann eine informale Erläuterung, z. B. anhand von Beispielen, für den Modellierer sogar nützlicher sein.

Ein zentraler Gegenstand der weiteren Arbeiten wird insbesondere die Validierung des Ansatzes und seiner praktischen Anwendbarkeit sein. Hierzu werden weitere, komplexere Anwendungsfälle identifiziert und in Form von Templates und Transformationsregeln umgesetzt. Hierbei soll der Ansatz auch erweitert werden, um nicht nur eine reine Umsetzung des reinen Kontrollflusses zu ermöglichen, sondern auch weitere Modellinhalte wie organisatorische Informationen, Input- und Outputdaten u. ä. zu transformieren. Im Zusammenspiel mit modellgetriebener Software-Entwicklung im eigentlichen Sinne könnte dann beispielsweise nicht nur der Kontrollfluss für ein Workflow Management-System generiert werden, sondern auch ein gewisser Anteil an Anwendungslogik, wie z. B. Benutzerdialoge für die Bearbeitung der einzelnen Arbeitsschritte (vgl. [Al05b]). Hierbei sollen geeignete Muster sowie Modellierungsrichtlinien für die Transformations-geeignete fachliche Modellierung entwickelt werden. Nicht zuletzt werden entsprechende Transformatoren softwaretechnisch implementiert, wobei vorhandene Modellierungswerkzeuge und Transformationsframeworks einbezogen werden sollen.

Literaturverzeichnis

[Al05a] Allweyer, T.: Geschäftsprozessmanagement – Strategie, Entwurf, Implementierung, Controlling. W3L-Verlag, Herdecke Bochum, 2005.

[Al05b] Allweyer, T.: Maßgeschneiderter Methodeneinsatz für die Entwicklung betriebswirtschaftlicher Software. In: Scheer, A.-W.; Jost, W.; Wagner, K. (Hrsg.): Von Prozessmodellen zu lauffähigen Anwendungen. Springer, Berlin Heidelberg New York, 2005, S. 173-195.

[7] Vgl. die Übersicht über Verfahren zur Referenzmodelladaption in [Be04], S. 42f.

[Be04] Becker, J.: Data Warehousing und Referenzmodellierung: Ever Tried to Build a House Without a Construction Plan? In: Becker, J. et al. (Hrsg.) Working Paper No. 1. European Research Center for Information Systems (ERCIS). Münster 2004, S. 37-48.

[De05] Debevoise, T.: Business Process Management with a Business Rules Approach. Implementing the Service Oriented Architecture. Business Knowledge Architects, Roanoke, 2005.

[Ha05] Havey, M.: Essential Business Process Modeling. O'Reilly, Sebastopol, 2005.

[Hi05] Hitz, M. et al.: UML @ Work. 3. Aufl., dpunkt Verlag, Heidelberg, 2005.

[Ko05] Kopp, O.: Abbildung von EPKs nach BPEL anhand des Prozessmodellierungswerkzeugs Nautilus. Diplomarbeit. Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2005.

[MZ05] Mendling, J.; Ziemann, J.: Transformation of BPEL Processes to EPCs. In: Nüttgens, M.; Rump, F. J. (Hrsg.): EPK 2005 Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten. 4. Workshop der Gesellschaft für Informatik e.V. (GI) und Treffen ihres Arbeitskreises "Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (WI-EPK)", Hamburg, 2005, S. 41-53.

[Oa07] OASIS (Hrsg.): Web Services Business Process Execution Language Version 2.0. OASIS, Billerica, 2007.

[Om03] OMG (Hrsg.): MDA Guide Version 1.0.1. OMG, Needham, 2003.

[Om05] OMG (Hrsg.): MOF 2.0/XMI Mapping Specification, v.2.1. OMG, Needham, 2005.

[Om06a] OMG (Hrsg.): Business Process Modeling Notation Specification. Version 1.0. OMG, Needham, 2006.

[Om06b] OMG (Hrsg.): Meta Object Facility (MOF) Core Specification. Version 2.0. OMG, Needham, 2006.

[Om07] OMG (Hrsg.): Unified Modeling Language: Superstructure. Version 2.1.1. OMG, Needham, 2007.

[Ru07] Rupp, C.; Queins, S.; Zengler, B.: UML 2 glasklar. Praxiswissen für die UML-Modellierung. 3. Aufl., Hanser, München Wien 2007.

[SG06] Schacher, M.; Grässle, P.: Agile Unternehmen durch Business Rules. Der Business Rules Ansatz. Springer, Berlin Heidelberg New York, 2006.

[St07] Stahl, T.; Völter, M.; Efftinge, S.: Modellgetriebene Softwareentwicklung. Techniken, Engineering, Management. 2. Auflage. Dpunkt Verlag, Heidelberg, 2007.

[Wa07] Wagner, J.: Agile Geschäftsprozesse durch Integration von Geschäftsregeln. IDS Scheer AG, Saarbrücken, 2007.