Bernd Oestereich

Objektorientierte Geschäftsprozessmodellierung und modellgetriebene Softwareentwicklung

Geschäftsprozessmodellierung, Softwareanalyse, Softwaredesign und Programmierung sind Disziplinen, die eng aufeinander aufbauen, zwischen denen aber oft tiefe Gräben liegen. Das führt zu Inkonsistenzen zwischen modellierten Geschäftsprozessen, Softwareanforderungen und realisierter Software. In diesem Beitrag zeigen wir auf, wie mit bewährten Ansätzen und standardisierten Notationen der Informatik funktionierende Übergänge zwischen den Disziplinen nachvollziehbar und systematisch herstellbar sind.

Inhaltsübersicht

1 Vom Prozess zum IT-System

Die Geschäftsprozessmodellierung ist aus Sicht der Informatik ein Randbereich. Meist wird im Rahmen der Anforderungsanalyse für ein Softwareentwicklungsprojekt eine begrenzte Geschäftsprozessmodellierung betrieben, um den Kontext des zu entwickelnden Systems zu analysieren. Im umgekehrten Fall werden Softwareentwicklungsprojekte aus einer Geschäftsprozessmodellierung abgeleitet, um Teilprozesse durch Automatisierung zu optimieren.

Für die Geschäftsprozessmodellierung existieren verschiedene Ansätze, Notationen und Vorgehensweisen, beispielsweise ereignisgesteuerte Prozessketten (EPKs), ARIS, PROMET und Ähnliches. Diese konzentrieren sich auf eine bestimmte Disziplin, die Geschäftsprozessmodellierung, und haben dort ihre Stärken. In den Disziplinen der Softwareentwicklung existieren jedoch andere Ansätze und Standards, beispielsweise die Unified Modeling Language (UML) und Objektorientierung.

Geschäftsprozessmodellierer und Softwareentwickler sind meistens verschiedene Personen, die traditionell mit verschiedenen Beschreibungssprachen, Methoden und Werkzeugen arbeiten. Auf der einen Seite existieren verschiedene proprietäre Typen von Ablaufdiagrammen und spezielle Werkzeuge (ARIS, Bonapart, Provision, Visio, Powerpoint etc.). Auf der anderen Seite existieren UML-Modellierungswerkzeuge und IDEs (Rose, XDE, Together, Innovator, Eclipse etc.).

Oft werden die Ergebnisse der Geschäftsprozessmodellierung von Softwareentwicklern nicht übernommen oder – wegen des speziellen für sie nicht verfügbaren Werkzeuges – gar nicht gewürdigt. Bestenfalls findet eine einmalige manuelle oder halbautomatische Datenübernahme statt, wobei deren Ergebnis wegen der großen semantischen Differenzen und nur rudimentären Transformationsregeln oftmals mehr Probleme als Nutzen bringt.

2 Geschäftsprozessmodellierung mit UML

Die Geschäftsprozessmodellierung beinhaltet viele Aktivitäten, die ähnlich oder identisch sind mit Aktivitäten der Softwareanforderungsanalyse. In beiden Disziplinen geht es vorwiegend um die Analyse und Beschreibung von Anforderungen und Abläufen. Der Schluss liegt nahe, bewährte Techniken der Informatik, vor allem Objektorientierung und UML, auch für die Geschäftsprozessmodellierung einzusetzen.

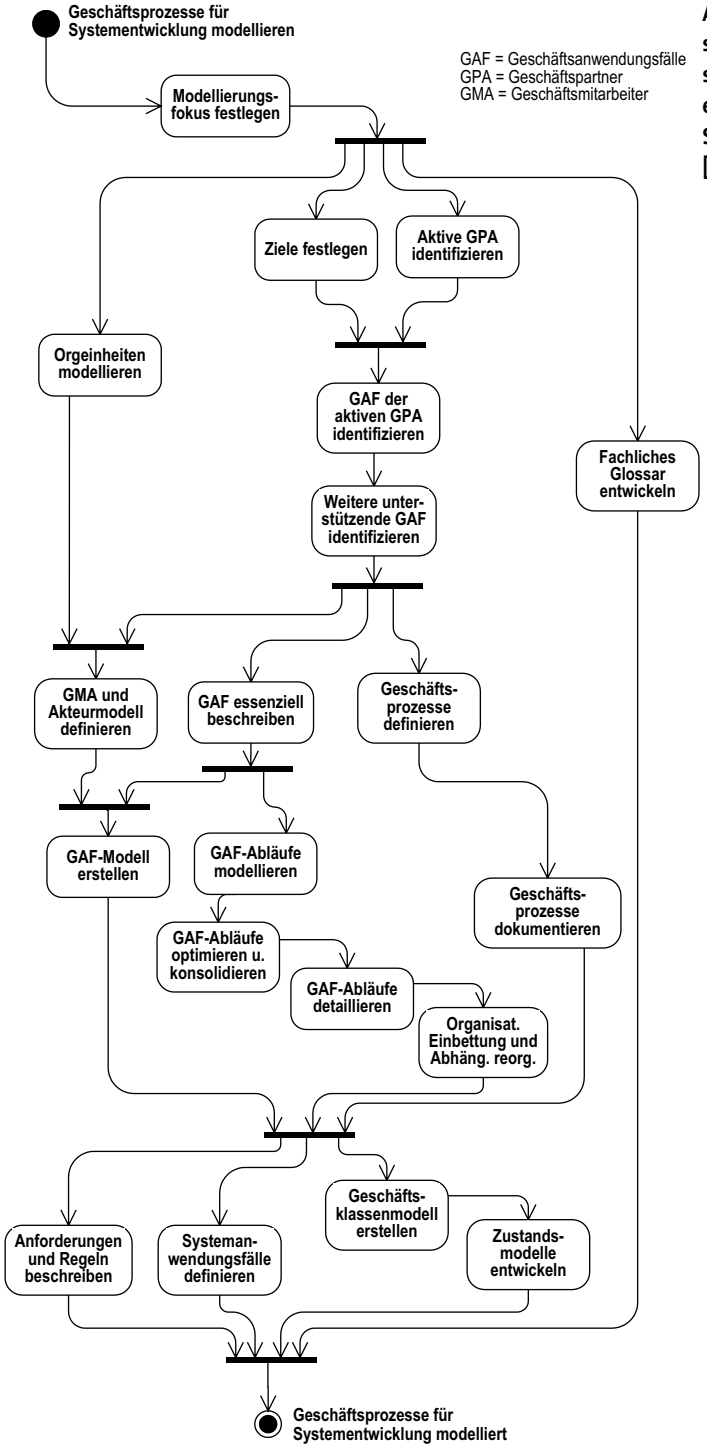

Abb. 1: Prozess der Geschäftsprozessmodellierung, speziell mit der Zielsetzung einer anschließenden Systementwicklung [Oestereich et al. 2003]

Die OOGPM-Methodik aus [Oestereich et al. 2003] ist ein solcher Ansatz. Die Zielrichtung, aufbauend auf der Geschäftsprozessmodellierung ein Softwaresystem zu entwickeln, ist eine mögliche Ausprägung der OOGPM-Methodik (vgl. Abb. 1). Die Ergebnisse der einzelnen Schritte aus Abbildung 1 basieren jeweils auf UML 2.0. Der gesamte obere Bereich in der Abbildung zeigt die eigentliche Geschäftsprozessmodellierung, die unteren vier Aktivitäten stellen den Übergang zur Softwareentwicklung dar.

Der Weg und Übergang von Geschäftsprozessen zur Softwareentwicklung ist aus methodischer Sicht klar strukturierbar und aufeinander aufbaubar. Eine in der Praxis gute methodische Durchgängigkeit setzt jedoch auch eine entsprechende übergreifende Notation voraus.

Ein großer Vorteil bei Verwendung von UML und Objektorientierung gegenüber spezifischen oder proprietären Geschäftsprozessmodellierungsmethoden ist das Vorhandensein einer durchgängigen Beschreibungssprache vom Geschäftsprozessmodell bis zur Systemrealisierung.

Durchgängigkeit bedeutet, dass in allen beteiligten Disziplinen einheitliche semantische Konzepte, Methoden und Notationen eingesetzt werden.

Der Bruch zwischen Geschäfts- und Softwaremodellierung lässt sich vermeiden, wenn auf beiden Seiten die gleichen oder verlustfrei abbildbare semantische Konstrukte verwendet werden, also idealerweise eine standardisierte Sprache wie die UML.

Im Kern ähnelt die OOGPM-Methodik dem Vorgehen in objektorientierten Softwareentwicklungsprojekten. Funktionale Anforderungen werden mit Anwendungsfällen und die jeweiligen detaillierten Abläufe mit Aktivitätsdiagrammen beschrieben. Die Strukturen der geschäftlichen Objekte sind in Klassendiagrammen modelliert. Die Ergebnisse der Geschäftsprozessmodellierung beinhalten also dieselben Artefakte wie die objektorientierte Softwareanalyse: Anwendungsfälle, Aktivitätsdiagramme und Klassen (vgl. Abb. 2).

Weitere Aktivitäten der OOGPM-Methodik führen beispielsweise zu Anforderungen wie Geschäftsregeln, einem Glossar und einem Modell der Organisationsstruktur. Hierzu existieren keine Standardmodellierungselemente in der UML. Die entsprechenden Modellelemente sind daher mit Hilfe der UML-eigenen Erweiterungsmechanismen (Stereotypen, Profile) einzurichten.

Ein wichtiges Ergebnis der Geschäftsprozessmodellierung sind natürlich die Geschäftsprozesse selbst. In der OOGPM-Methodik ist ein Geschäftsprozess eine Gruppe fachlich zusammengehöriger Geschäftsanwendungsfälle.

Die sachlogische Reihenfolge, in der die Geschäftsanwendungsfälle auftreten dürfen, kann in einem Aktivitätsdiagramm ausgedrückt werden. Im Geschäftsprozess Kfz-Vermietung findet »Kfz reservieren« vor »Kfz überlassen« und das wiederum vor »Kfz zurücknehmen« statt. Die Beschreibung dieser Abläufe ist z.B. für Workflow-Systeme wichtig.

Auch weitere Sachverhalte, wie organisatorische Einbettung, tangierte Systeme bzw. Schnittstellen zu anderen Systemen, Durchlauf- und Verweilzeiten können mittels UML-Erweiterungen definiert werden. Simulationen im Sinne ausführbarer Modelle (Executable UML) sind denkbar.

Für den Übergang von der Geschäftsprozess- zur Softwaremodellierung werden im Geschäftsprozessmodell die Bereiche identifiziert, die durch Softwaresysteme unterstützt werden sollen. Das können ganze Anwendungsfälle oder nur Teilabläufe sein. Die ausgewählten Modellelemente werden mit dem Stereotyp »system support« gekennzeichnet, um deutlich zu machen, dass diese Modellelemente auch im Modell der nächsten Disziplin repräsentiert werden sollen.

Abbildung 3 zeigt die semantischen Überlappungen der einzelnen Modelle. Im oberen

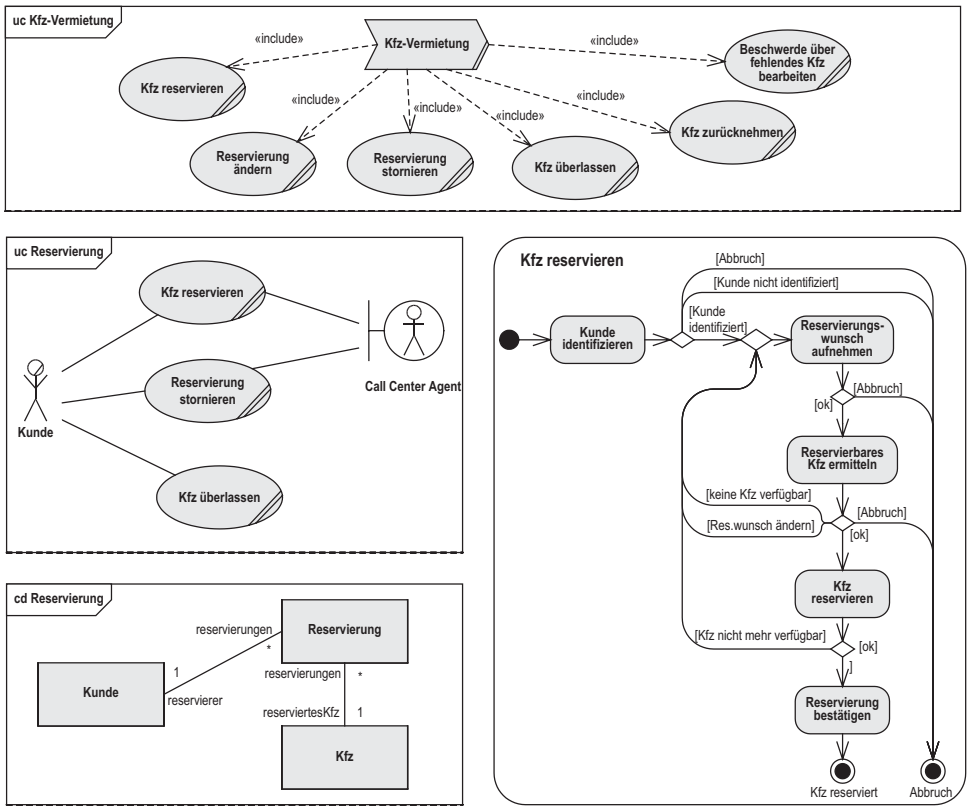

Abb. 2: Einige Ergebnisse der Geschäftsprozessmodellierung: Anwendungsfälle, Aktivitätsdiagramm, Geschäftsklassen

Bereich ist für jede Disziplin ein Modell dargestellt. Die Durchgängigkeit von Design- und Realisierungsmodell wird in der Informatik schon länger erfolgreich praktiziert, mit verschiedenen Ansätzen wie Roundtrip- und Forward-Engineering. Dabei können semantisch gleiche Modellinformationen in den unterschiedlichen Disziplinen durchaus unterschiedlich repräsentiert werden. In der UML ist eine Assoziation ein explizit definiertes Modellelement, in Java würde dies durch eine Menge von Operationen und Attributen repräsentiert.

In ähnlicher Weise existieren auch verschiedene Repräsentationsformen beispielsweise für Anwendungsfälle. In der UML würden Abläufe durch Aktivitätsmodelle beschrieben, in Java beispielsweise würde eine Anwendungsfallsteuerungsklasse implementiert.

Trotz der unterschiedlichen Repräsentationsformen in den verschiedenen disziplinspezifischen Modellen handelt es sich semantisch um denselben Sachverhalt. Sie sind aufeinander abbildbar bzw. zueinander transformierbar. Analog hierzu ist dies auch zwischen den Geschäftsprozess- und Softwareanalysemodellen möglich.

Hinzu kommen fachliche Referenzmodelle (z.B. ein bankfachliches Rahmenwerk) und technische Rahmenwerke (z.B. ein Persistenzrahmenwerk), die direkt verwendet oder erweitert werden.

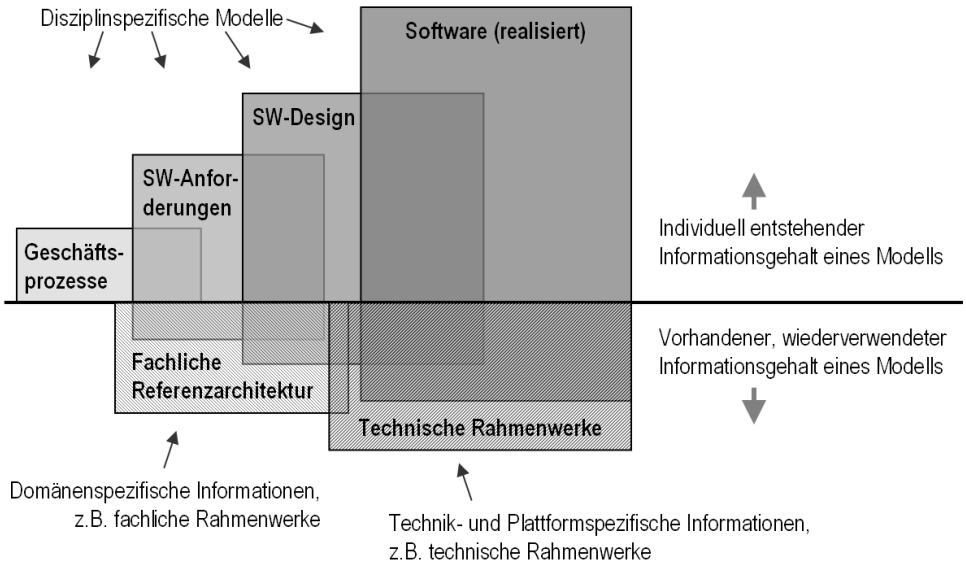

Abb. 3: Aufeinander aufbauende und sich überschneidende disziplinspezifische Modelle

In Abbildung 3 stellen die unterhalb der horizontalen Linien genannten Modelle bereits vorhandene Rahmenwerke oder Referenzlösungen dar, deren Inhalte für ein spezifisches Modell, bspw. eine konkrete Software, zu einem gewissen Teil (Schnittmenge der Bereiche) genutzt werden können.

Wenn diese Durchgängigkeit zwischen den disziplinspezifischen Modellen vorteilhaft ist, warum wird es denn in der Praxis nicht einfach so gemacht?

Zum einen liegt es wahrscheinlich daran, dass es historisch gesehen schon immer unterschiedliche Ansätze gab, und jeder an dem ihm Bekannten festhält. Auch innerhalb der Softwareentwicklung hat es einige Zeit gedauert, bis sich die UML als Standard durchsetzen und die Vielfalt der Vorgängersprachen ablösen konnte. Zum anderen war die UML aber bislang, das heißt in der Version 1.x, auch nicht ausdrucksstark, ausgereift und flexibel genug, um für andere Disziplinen attraktiv zu sein.

3 Neue Möglichkeiten durch UML 2.x

Mit UML 2.x [UML 2.0] stellt sich die Situation nun anders dar. Die UML dringt nun in weitere Anwendungsbereiche vor, beispielsweise in die Welt der Echtzeit- und eingebetteten Systeme, und mit der gerade entstehenden UML-Erweiterung SysML [SysML vers. 0.85] in den Bereich gemischter Systeme aus Software, Hardware, Organisation etc. und eben auch in den Bereich Geschäftsprozessmodellierung.

Um mit der möglichen Durchgängigkeit jedoch praktische Vorteile zu erschließen, sind einige Hürden zu nehmen, vor allem ist sowohl eine gute formale Fundierung als auch eine hohe Ausdrucksmächtigkeit der verwendeten semantischen Konstrukte und Notationen notwendig. Darauf aufbauend sind dann die Abbildungs- und Transformationsregeln zu entwickeln. Erst dadurch können die versprochenen Vorteile der Durchgängigkeit in der Praxis realisiert werden. UML 2 bietet das.

4 Automatisierte Transformationen

Liegen solche Regeln vor, können wir gleich noch einen Schritt weiter gehen und die definierten Transformationen automatisieren. Ähnlich wie aus einer UML-Assoziation oder einem Anwendungsfall im Designmodell die entsprechenden Java-Elemente automatisch nach definierten Regeln erzeugt werden, können auch Geschäftsklassen (Begriffsmodell) der Geschäftsprozessmodellierung in entsprechende Fachklassen des Softwareanforderungs- oder Softwaredesignmodells generiert werden.

Dabei ist zu beachten, dass

- nicht alle Informationen aus dem einen Modell in das nächste übertragen werden,
- sie nicht unbedingt in identischer Weise repräsentiert werden,
- zu den übertragenen Informationen gewöhnlich noch zahlreiche weitere, disziplinspezifische Informationen hinzukommen (Ergänzung bzw. Beimischung zusätzlicher Modellinformationen).

Zwischen den einzelnen Disziplinen stehen also jeweils modelltechnische Transformationen. Es bedarf klarer und theoretisch fundierter Festlegungen, wie welche Informationen in welchem disziplinspezifischen Modell repräsentiert werden und wie diese aufeinander abgebildet bzw. zueinander transformiert werden können.

Genau diese Transformationsmöglichkeiten zu schaffen war eines der Ziele, das die Object Management Group (OMG) mit der Entwicklung der neuen UML-Version 2.x erreichen wollte. Im Kontext der OMG wird im Hinblick auf eine bessere generative Softwareentwicklung von *Model Driven Architecture* (MDA) gesprochen. Ganz allgemein geht es bei MDA jedoch darum, Transformationen zwischen verschiedenen spezifischen Modellen zu ermöglichen, beispielsweise zwischen plattformunabhängigen (PIM) und plattformspezifischen (PSM) Modellen (vgl. z.B. [Sturm & Boger 2003]). Wir sprechen im Folgenden verallgemeinert von *modellgetriebener Softwareentwicklung*, denn statt PIM und PSM sind ebenso Transformationen zwischen disziplinspezifischen Modellen (DSM) möglich.

Die hier beschriebenen Transformationen stellen einen Vorwärtsgenerierungsansatz dar. Änderungen müssen bei diesem Ansatz an der Quelle vorgenommen werden. Wenn sich beispielsweise der Ablauf eines Geschäftsanwendungsfalles ändert, muss diese Änderung im Geschäftsprozessmodell durchgeführt werden und nicht im Softwareanalysemodell oder gar im Sourcecode selbst.

Modellelemente, die im Folgemodell weiterverwendet werden sollen, werden entsprechend gekennzeichnet, so z.B. mit dem Stereotyp »system support«, wenn ein Element des Geschäftsprozessmodells in das Softwareanforderungsmodell übertragen werden soll. Entsprechend werden Elemente, die ihren Ursprung in einer anderen Disziplin haben, ebenfalls gekennzeichnet, beispielsweise mit dem Stereotyp »gpm«, wenn ein Element im Softwareanforderungsmodell aus dem Geschäftsprozessmodell stammt.

Damit wird im Sinne des Vorwärtsgenerierungsansatzes auch ausgedrückt, dass diese Elemente nur im Ursprungsmodell verändert werden sollen. Erweiterungen sind mittels Spezialisierung (Vererbung) jedoch auch in den Folgemodellen zulässig (vgl. Abb. 4).

Neben Anwendungsfällen und Abläufen können auch weitere Informationen generativ aus dem Geschäftsprozessmodell in das Softwareanalysemodell übernommen werden. Wurde beispielsweise in Aktivitätsdiagrammen auch der Objektfluss modelliert, können automatisch entsprechende Fachklassen im Analysemodell angelegt werden.

Natürlich werden in der Analyse auch Anwendungsfälle, Abläufe und Strukturen identifiziert und beschrieben, die nicht in die Geschäftsprozessmodellierung gehören. Beispielsweise könnte das Softwaresystem eine Benutzerverwaltung haben, die zu Anwen-

dungsfällen wie »Benutzer anmelden« führt. In der Geschäftsprozessmodellierung würde man diesen Detaillierungsgrad nicht betrachten, sondern eher nur eine Anforderung formulieren, dass eine Benutzerverwaltung erforderlich ist. Auch das gehört zur semantischen Anreicherung bzw. Erweiterung eines disziplinspezifischen Modells.

Zwischen den einzelnen Teildisziplinen lässt sich prinzipiell eine ganze Kette von Transformationen definieren, so dass Informationen eines Modells, die auch für Modelle der Folgedisziplinen gültig sind, automatisch mitgenommen werden, aber Modellinformationen, die nur für ein spezielles Modell gelten, auch nur in diesem vorliegen.

Folgende Vorteile ergeben sich dann:

- Alle identifizierten, für die Folgedisziplinen relevanten Bereiche werden auch tatsächlich in das Folgemodell übernommen.
- Es entstehen keine Übertragungsfehler, wie sie beim manuellen Übertragen (Abschreiben) oder bei unzulänglichen Transformationsmöglichkeiten leicht auftreten können.
- Es können automatisch Bezüge zwischen den transformierten Elementen hergestellt werden (UML: »trace« – Abstraktionsbeziehung).

Beim Übergang zur Realisierung hat eine modellbasierte Entwicklung die größten Vorteile durch die Generierung eines Infrastrukturcoderahmens. Zur Vertiefung des Themas eignet sich [Stahl & Völter 2005].

Bei den heute üblichen Mehrschichtarchitekturen ist ein verhältnismäßig geringer Teil des Codes durch die Fachlichkeit motiviert. Etwa die Hälfte des Codes ergibt sich aus den zu verwendenden Entwurfsmustern und durch Infrastrukturnotwendigkeiten der Architektur und Plattform.

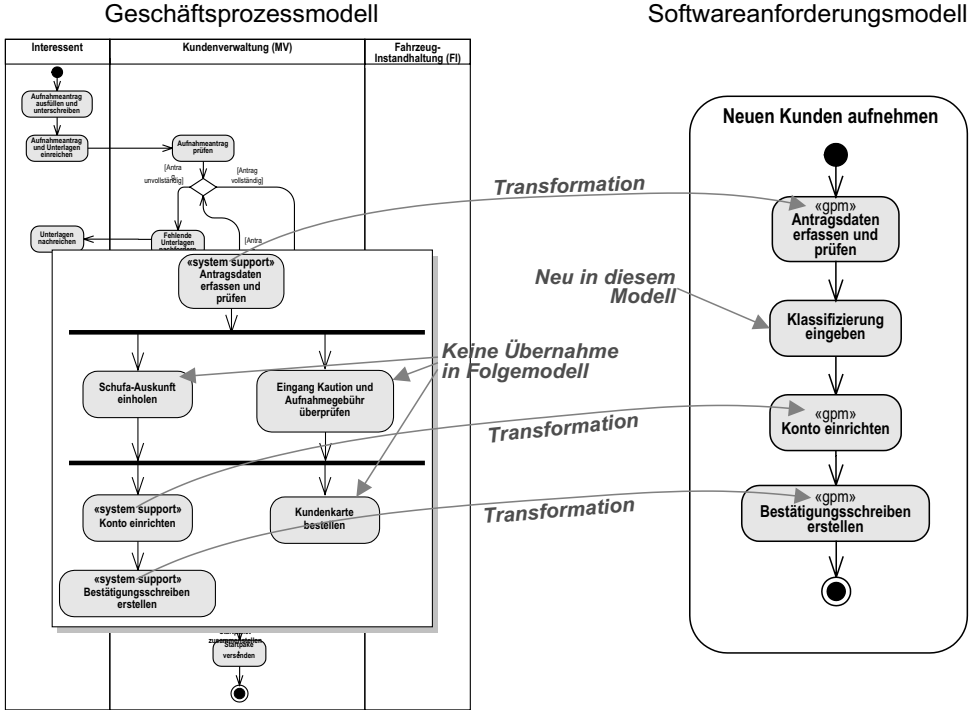

Abb. 4: Transformation eines geschäftlichen Ablaufes

Ziel einer modellgetriebenen Entwicklung ist es daher, diesen großen Teil des Codes automatisch herzustellen. Programmiersprachencode ist in diesem Sinne nur eine besondere Form von Modell.

Die in diesem Beitrag gezeigten Beispiele stammen aus einer prototypischen Entwicklung mit Hilfe der hier beschriebenen Verfahren und Techniken. Hier ist wichtig festzustellen, dass die Entwicklung der Transformationen, die Festlegung und Ausprägung der notwendigen und sinnvollen semantischen Konstrukte in der UML aufwendig sind und ein umfassendes Verständnis der UML 2.x erfordern.

Sind jedoch diese Festlegungen erst einmal vorhanden und ausgereift, lassen sich die bislang vorhandenen Informations- und Kommunikationsverluste zwischen den Disziplinen ganz erheblich reduzieren. Der Gesamtprozess wird effizienter und effektiver.

5 Literatur

[Oestereich et al. 2003] *Oestereich, B.; Weiss, C.; Schröder, C.; Weilkiens, T.:* Objektorientierte Geschäftsprozessmodellierung mit der UML. dpunkt.verlag, Heidelberg, 2003.

[Stahl & Völter 2005] *Stahl, T.; Völter, M.:* Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management. dpunkt.verlag, Heidelberg, 2005.

[Sturm & Boger 2003] *Sturm, T.; Boger, M.:* Softwareentwicklung auf Basis der Model Driven Architecture. In: HMD – Praxis der Wirtschaftsinformatik, 40. Jg., 2003, Heft 231, S. 38-45.

[SysML vers. 0.85] *o.V.:* Systems Modeling Language; *http://www.sysml.org*. Zugriff am 30. 12. 2004.

[UML 2.0] *OMG (Hrsg.):* Unified Modeling Language, *http://www.uml.org*. Zugriff am 30.12.2004.

Bernd Oestereich
Geschäftsführer
oose.de Dienstleistungen für
innovative Informatik GmbH
Oberstr. 14b
22144 Hamburg
boe@oose.de
www.oose.de