Erweiterung eines MDSD-Systems zur Unterstützung von Produktlinien duch Feature-Modelle

Rolf-Helge Pfeiffer und Peter Hänsgen
Intershop Communications AG
07740 Jena
h.pfeiffer@uni-jena.de, p.haensgen@intershop.de

Abstract: Featuremodelle sind eine Schlüsseltechnologie für die Modellierung von Softwareproduktlinien. Sie können als höchste Abstraktionsstufe in der modellgetriebenen Softwareentwicklung eingesetzt werden. Besteht schon eine Infrastruktur zur modellgetriebenen Softwareentwicklung, d. h. Modelltransformatoren und Codegeneratoren existieren bereits, so wirft die Einführung von Softwareproduktlinien in eine solche Infrastruktur Probleme auf. Diese Arbeit beschreibt eine bestehende Infrastruktur zur modellgetriebenen Softwareentwicklung, mögliche Änderungen daran zur Unterstützung von Softwareproduktlinien und stellt eine Lösung mittels Aspektmodellen vor.

1 Einleitung

Softwareproduktlinien werden bisher auf Quellcodeebene [pur08] oder auf einzelnen Modellen [AC04], die zur Quellcodegenerierung genutzt werden, implementiert.

Das Konzept der *Modellgetriebenen Softwareentwicklung (MDSD)* wurde in den letzten Jahren so verbessert, dass es in der Praxis mehr und mehr akzeptiert und zur Softwareentwicklung eingesetzt wird. Im Zuge dieser Fortentwicklung sind Werkzeuge wie das *Eclipse Modeling Framework (EMF)* [emf08] oder *openArchitectureWare (oAW)* [oaw08b] entstanden und stehen für den Entwickler zur Verfügung. Das Hauptaugenmerk der Softwareentwicklung verschiebt sich deshalb von der Entwicklung von Quellcode hin zur Entwicklung von Modellen. Softwarehersteller wie Intershop tätigten hohe Investitionen, z. B. für die Entwicklung von Metamodellen, Modelltransformatoren und Codegeneratoren, um *MDSD* in den Softwareentwicklungsprozess zu integrieren. Aufgrund der Tatsache, dass Intershop Softwareprodukte herstellt, die eine Systemfamilie formen, und da einzelne Produkte einer Systemfamilie effizient duch den Einsatz von Produktlinien hergestellt werden können, ist der folgerichtige Schritt die Einführung der Unterstützung von Produktlinien in die bestehende *MDSD*-Infrastruktur.

Diese Arbeit beschreibt, wie eine bestehende *MDSD*-Infrastruktur angepasst werden kann, um Softwareproduktlinien zu unterstützen. Dazu werden um Aspektmodelle angereicherte Featuremodelle genutzt. Es stellt sich die Frage, wie die Informationen über Variabilität, die in den Featuremodellen ausgedrückt sind, in den Prozess der automatischen Softwaregenerierung einbezogen werden können.

Diese Arbeit ist wie folgt strukturiert: Im Abschnitt 2 werden wichtige Begriffe und Definitionen eingeführt. Abschnitt 3 stellt eine bestehende *MDSD*-Infrastruktur vor. Im darauffolgenden Abschnitt wird erklärt, wie Feature- und Aspektmodelle dazu genutzt werden können, um das Verhalten eines *MDSD*-Systems zu verändern. Anschließend werden themenbezogene Arbeiten in Abschnitt 5 vorgestellt und die Arbeit mit einem Fazit in Abschnitt 6 abgeschlossen.

2 Begriffe und Definitionen

2.1 Modellgetriebene Softwareentwicklung

Modellgetriebene Softwareentwicklung *(MDSD)* ist der Einsatz von Computerprogrammen zur Softwareentwicklung, welche Modelle ineinander transformieren oder beliebige textuelle Artefakte aus Modellen erzeugen. Modelle können u. a. verschiedene Abstraktionsebenen und technische Domänen beschreiben. Textuelle Artefakte sind Einheiten ohne ein präzise definiertes Metamodell, z. B. Quellcode in einer Programmiersprache, Konfigurationsdateien, Dokumentationen, u. s. w. .

2.2 Feature-Modellierung

Wir benutzen den Begriff der *Featuremodelle* analog zu [CA05]. Featuremodelle sind Featurediagramme plus zusätzliche Informationen. Unter *Features* verstehen wir Systemeigenschaften, die Gemeinsamkeiten und Unterschiede von Systemen einer Familie ausdrücken und die für bestimmte Parteien von Bedeutung sind. *Variantenmodelle* sind Instanzen von Featuremodellen, die eine gültige Auswahl von Features enthalten. *Featurediagramme* sind grafische Repräsentationen von Features und deren Relationen. Zur Darstellung von Featurediagrammen gibt es eine Vielzahl von Notationen, z. B. [KCH+90, KLD02, Rie03, GFd98, KC04].

2.3 Aspektorientierte Modellierung

Aspektmodelle sind Modelle, die sich auf ein anderes Modell beziehen, nämlich auf ein Basismodell. Sie können als Differenzenmodelle verstanden werden. Sie beschreiben eine Differenz zwischen einem Basismodell und dem Modell, das beim Mischen von Aspektmodellen in ein Basismodell entsteht. Aspektmodelle können also positive und negative Variabilität in Bezug auf ein Basismodell ausdrücken. Dabei bedeutet positive Variabilität das Hinzufügen von Modellelementen und negative Variabilität das Entfernen von Modellelementen eines Basismodells. Wir nennen den Prozess der Anwendung von positiver und negativer Variabilität auf ein Basismodell, anders als in aspektorientierten Programmiersprachen, nicht weben, sondern mischen, da aspektorientierte Programmiersprachen im

Allgemeinen nicht das Entfernen von Programmkonstrukten erlauben. Analog zu [GV07] und da wir *XWeave* zur Implementierung der Transformation des Mischens von Modellen verwenden, werden in unserer Arbeit Aspekte asymmetrisch in Basismodelle gemischt, d. h. die Operation des Mischens von Modellen (Operator: $\otimes$) ist unidirektional. Beispiel: **B** $\otimes$ **A = B'**, wobei **B** ein Basismodell und **A** ein Aspektmodell ist. Die umgekehrte Reihenfolge der Operanden ist ungültig.

3 Ein existierendes MDSD-System

Intershop ist ein führender Hersteller von E-Commerce-Software. *MDSD* wird als Schlüsseltechnologie zur Effizienzsteigerung in der Softwareentwicklung angesehen, darum wurde eine *MDSD*-Infrastruktur entwickelt, welche aus domänenspezifischen Modellen sowie kaskadierenden Modelltransformations- und Codegenerierungsschritten besteht. Abbildung 1 zeigt den schematischen Aufbau der aktuell verwendeten *MDSD* Infrastruktur. Die bunten Kästchen stellen domänenspezifische Modelle auf jeweils einem bestimmten Abstraktionsniveau dar. Die verschiedenen Abstraktionsniveaus werden in dieser Arbeit Modellierungsebenen genannt. Alle dargestellten Modelle wurden mit dem *Eclipse Modeling Framework (EMF)* entwickelt und basieren auf dem Metametamodell *Ecore* [emf08].

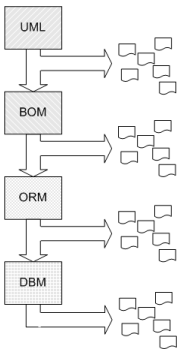

Abbildung 1: Ein *MDSD* System mit kaskadierender Transformation domänenspezifischer Modelle. Modelltransformatoren (vertikale Pfeile) und Codegeneratoren (horizontale Pfeile) wurden mittels *openArchitectureWare* [oaw08b] implementiert.

Im Folgenden werden die verschiedenen Modellierungsebenen genauer beschrieben:

DBM-Ebene: Auf dieser Modellierungsebene werden im *Database Model (DBM)* die Strukturen von Datenbanken definiert. Modelle dieser Ebene besitzen den geringsten Abstraktionsgrad innerhalb der Hierarchie, d. h. sie sind plattformspezifisch. Datenbankmodelle beschreiben z. B. Tabellen mit ihren Spalten und Constraints.

ORM-Ebene: Modelle der *Object Relational Mapping (ORM)*-Ebene dienen der Modellierung von persistenten Geschäftsobjekten. Sie bieten einen höheren Abstraktionsgrad als Modelle auf *DBM*-Ebene. Die zentralen Modellierungselemente sind *ORM*

Beans und deren Relationen. Auf dieser Ebene sind z. B. Konzepte wie Vererbung von *ORM Beans* und 1-zu-n Relationen vorhanden.

BOM-Ebene: Die Modelle der *Business Object Model (BOM)*-Ebene dienen der plattformunabhängigen Modellierung von persistenten Geschäftsobjekten und deren Relationen. *BOM*-Modelle bieten eine Sicht auf die darunter liegenden *ORM*-Modelle und eine Erhöhung des Abstraktionsgrades. So existieren z. B. Konzepte wie n-zu-m Relationen zwischen *BOM Beans* und Assoziations-*BOM Beans*.

UML-Ebene: Zur Modellierung und Präsentation der Struktur von Softwaresystemen werden *Unified Modeling Language (UML)*-Klassendiagramme verwendet. Für Modelltransformationen von *UML*- auf *BOM*-Ebene werden die Modellelemente verwendet, die durch entsprechende Stereotypen, z. B. «`BOM Bean`», gekennzeichnet sind.

Die Modelltransformatoren der bestehenden *MDSD*-Infrastruktur wurden mittels *XTend* [oaw08a] implementiert. Codegeneratoren zur automatischen Generierung von *Java* Quellcode, Dokumentationen, Konfigurationsdateien und anderer textueller Artefakte wurden mittels *XPand* [oaw08a] umgesetzt. Der in Abbildung 1 dargestellte Prozess von kaskadierenden Modelltransformationen und Codegenerierungsschritten wird durch einen *openArchitectureWare*-Workflow definiert. Die beschriebene *MDSD*-Infrastruktur ist zur Entwicklung verschiedenartiger, heterogener Softwaresysteme geeignet, welche die gleiche Architektur besitzen. Diese Infrastruktur ist folglich nicht dazu geeignet, verschiedene Produkte einer Softwaresystemfamilie, also Produkte, die viele Gemeinsamkeiten besitzen und sich nur in bestimmten Eigenschaften unterscheiden (homogene Systeme), effizient herzustellen. Intershop stellt E-Commerce-Systeme her, die alle Systeme einer Softwaresystemfamilie sind. Dadurch wird ein Produktlinienansatz zur Softwareherstellung interessant. Wie kann aber nun die bestehende *MDSD*-Infrastruktur verändert werden, um Variabilitäten zwischen einzelnen Produkten einer Familie zu berücksichtigen? Wie kann die existierende Infrastruktur angepasst werden, um Produktlinien zu unterstützen? Die folgenden Abschnitte stellen einen möglichen Lösungsansatz vor.

4 Steuerung eines MDSD-Systems durch Features

Zur Modellierung von Gemeinsamkeiten und Unterschieden von Softwaresystemen einer Softwaresystemfamilie können Featuremodelle eingesetzt werden. Soll ein Featuremodell ein bestimmtes Softwaresystem definieren und soll weiterhin die bestehende *MDSD*-Infrastruktur verwendet werden, so muss ein Featuremodell in der Art angewendet werden, dass es alle nötigen Informationen zur automatischen Generierung eines Softwaresystems in Form von Modellen enthält, und es muss in der Lage sein, ein *MDSD*-System so zu steuern, dass die in Aspektmodellen ausgedrückte Variabilität berücksichtigt wird.

Wir haben uns dazu entschieden, Features entsprechend der Definition von Featuremodellen mit zusätzlichen Informationen zu hinterlegen. Es sind unserer Ansicht nach zwei

Arten von Features notwendig, um ein *MDSD*-System für die Entwicklung von Softwareproduktlinien einzusetzen. Das sind:

Aspektfeatures: Ein Feature, welches mit einem Aspektmodell hinterlegt wurde, ist ein Aspektfeature. In unserer Arbeit werden Aspektmodelle dazu verwendet, um Variabilität zwischen Softwaresystemen einer Familie zu kapseln. Weiterhin steuern Aspektfeatures die neu in das *MDSD*-System eingefügte Modelltransformation des Mischens von Modellen. Diese Transformation ist durch den Operator $\otimes$ in Abbildung 2 markiert.

Steuerfeatures: Features, die steuern, ob ein bestimmter Modelltransformator oder Codegenerator in einem *MDSD*-System aufgerufen wird, heißen Steuerfeatures. In Abbildung 2 sind sie durch graue Kreise über Modelltransformatoren und Codegeneratoren symbolisiert.

Zentral für die Umsetzung des Produklinienansatzes ist in unserem Fall die Implementierung einer Modelltransformation, die es erlaubt, Aspektmodelle, welche Informationen über Variabilität enthalten, in ein Basismodell zu mischen. Damit kann ein maßgeschneidertes Modell erzeugt werden, aus dem anschließend das Softwaresystem generiert werden kann. Der Ansatz der Einführung einer neuen Transformation zum Mischen erlaubt die Wiederverwendung bereits bestehender Modelltransformatoren und Codegeneratoren.

4.1 Implementierung des Mischens von Aspektmodellen

Wie Abbildung 2 zeigt, nutzt unsere Implementierung zum Mischen von Modellen die *oAW* Komponenten *XVar* und *XWeave*. Im Folgenden wird erklärt, wie das Mischen von Modellen genau funktioniert.

4.1.1 Notation von Aspektmodellen

Um das Modellieren von Aspekten auf *UML*-Ebene zu ermöglichen, wurde ein *UML Profil* um Stereotypen, die das Ausdrücken von Regeln zum Mischen von Aspekt- und Basismodellen und das Ausdrücken von positiver und negativer Variabilität erlauben, erweitert. Alle im Folgenden vorgestellten Stereotypen sind auf die *UML*-Modellelemente *Class* und *Property* anwendbar. Weiterhin ist es möglich, sie parallel zu der Notation, die von *XWeave* [GV07] bereitgestellt wird, zu verwenden.

«BOMBaseElement» Ein Modellelement mit diesem Stereotypen bezieht sich auf genau ein Modellelement in einem Basismodell, das den gleichen Namen besitzt. Es wird genutzt, um einen Pointcut zu definieren, der genau einem Joinpoint im Basismodel entspricht.

«BOM XVar» Der Stereotyp besitzt die Property `nameOfOwningFeature`. Er dient der Spezifikation eines Elements im Basismodel, welches aus diesem entfernt wird, sobald das in `nameOfOwningFeature` angegebene Feature nicht im entsprechenden Variantenmodel enthalten ist.

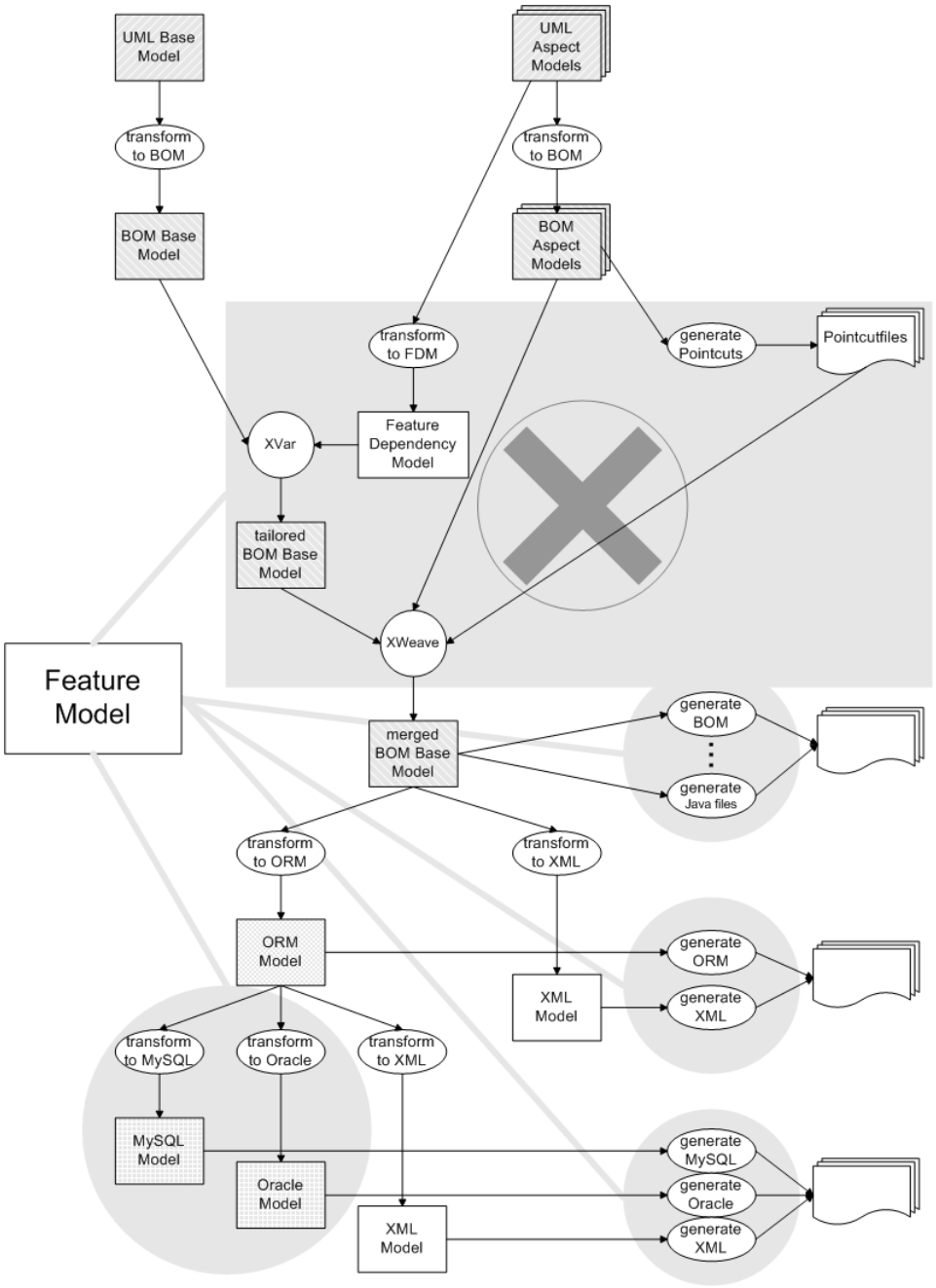

Abbildung 2: Erweiterung des *MDSD*-Systems aus Abbildung 1 zur Unterstützung von Produktlinien.

«BOM XWeave» Der Stereotyp besitzt die Property `pointcutSpec`. Er markiert Modellelemente, die zum Basismodell hinzugefügt werden sollen. Die Property `pointcutSpec` enthält die Deklaration einer Pointcutspezifikation. Ist diese Property leer, so wird das markierte Element im Basismodell zu dem Modellelement hinzugefügt, welches den gleichen Namen besitzt. Ist in der Property ein Stern ($\star$) enthalten, so wird das markierte Modellelement zu allen *Beans* des Basismodells hinzugefügt. Zur beliebigen Deklaration von Pointcutspezifikationen wird der Property der Name einer Pointcutfunktion gegeben, die mittels der *XWeave*-spezifischen Syntax manuell definiert werden muss.

4.1.2 Feature Dependency Model

Ein *Feature Dependency Model (FDM)* ist ein Modell, welches von *XVar* benötigt wird, um negative Variabilität auf ein Basismodell anzuwenden. Ein *FDM* stellt Abhängigkeiten zwischen einem Feature und einem oder mehren Modellelementen eines Basismodells dar. Für alle vorhandenen Aspektmodelle wird ein *FDM* genau einmal automatisch generiert. Dazu extrahiert eine Modelltransformation alle Elemente der Aspektmodelle, die mit dem Stereotyp «`XVar`» markiert sind, und erstellt aus dem Wert, der in der Property `nameOfOwningFeature` enthalten ist, einen entsprechenden Eintrag im *FDM*.

XVar bekommt als Eingabe ein Basismodell auf *BOM*-Ebene, ein Variantenmodell und ein *FDM* und generiert daraus ein temporäres Basismodell auf *BOM*-Ebene. Die Anwendung von *XVar* hängt nicht von den Metamodellen der entsprechenden Modelle ab, sondern setzt nur voraus, dass diese auf *Ecore* basieren.

4.1.3 Generierung von Pointcuts

Im Gegensatz zu aspektorientierten Programmiersprachen können alle Modellelemente eines Basismodells als Joinpoints fungieren. Dies wird durch die Nutzung von *XWeave* zur Implementierung des Mischens von Modellen möglich. Die Namen von Modellelementen in Aspektmodellen werden genutzt, um auf Pointcutspezifikationen zu verweisen, siehe dazu [oaw08a].

Für jedes Aspektmodell auf *UML*-Ebene wird automatisch eine Datei mit den entsprechenden Pointcutspezifikationen erstellt. Dies geschieht in Abhängigkeit zur Property `pointcutSpec` des Stereotyps «`BOM XWeave`». Ist diese leer, so wird eine Pointcutspezifikation generiert, die das markierte Modellelement zu genau dem Modellelement des Basismodells hinzufügt, welches den gleichen Namen besitzt. Ist in der Property `pointcutSpec` ein Stern ($\star$) enthalten, so wird eine Pointcutspezifikation generiert, die das markierte Element des Aspektmodells zu allen Klassen des Basismodells hinzufügt. Ist eine beliebige Zeichenfolge angegeben, so muss die Pointcutspezifikation manuell in einer extra Datei unter diesem Namen implementiert werden. Unsere Transformation bezieht automatisch generierte und manuell implementierte Pointcutspezifikationen bei der Anwendung von *XWeave* ein.

4.1.4 Transformation von Aspekmodellen auf *BOM*-Ebene

Alle Aspektmodelle werden einzeln von *UML* auf die *BOM*-Ebene transformiert. Dazu wurden die existierenden Transformatoren so erweitert, das aus der in Abschnitt 4.1.1 eingeführten Notation automatisch die von *XWeave* geforderte Notation erstellt wird.

Anschließend bekommt *XWeave* jeweils ein temporäres Basismodell auf *BOM*-Ebene, ein Aspektmodell auf *BOM*-Ebene sowie eine Datei mit Pointcutspezifikationen als Eingabe. Nachdem nacheinander, entsprechend ihrer Abhängigkeiten und wie vom Variantenmodell gefordert, die Aspektmodelle in das temporäre Basismodell gemischt wurden, ist der Prozess des Mischens von Modellen beendet und es ist ein Basismodell auf *BOM*-Ebene entstanden, das durch ein Variantenmodell definiert ist und welches für weitere Transformations- und Generierungsschritte verwendet wird. *XWeave* setzt ebenso wie *XVar* nur voraus, dass die Modelle, auf denen es angewandt wird, *Ecore* als Metametamodell besitzen.

4.1.5 Steuerfeatures

Steuerfeatures werden mittels *if*-Komponenten in *oAW*-Workflows implementiert. So wird es möglich, den Aufruf von bestimmten Transformatoren und Generatoren in Abhängigkeit von den ausgewählten Features eines Variantenmodells zu steuern. In Abbildung 2 symbolisieren die vier grauen Kreise Steuerfeatures.

5 Themenbezogene Arbeiten

5.1 pure::variants

pure::variants [pur08] ist ein Framework zum Variantenmanagement für *Eclipse*. Es bietet Möglichkeiten zur Erstellung von Featuremodellen und zur automatischen Generierung von Software aus bestimmten Variantenmodellen. Dazu werden textuelle Artefakte, z. B. Pakete, Klassen, Methoden und Aspekte einer Programmiersprache an bestimmte Features gekoppelt. Ein Transformator mischt die Artefakte entsprechend der Featureauswahl eines Variantenmodells. Wir benutzen *pure::variants* zur Erstellung von Featuremodellen und Variantenmodellen sowie einen entsprechenden Adapter, um Variantenmodelle in einem *oAW*-Workflow nutzen zu können.

5.2 FeaturePlugin

FeaturePlugin [AC04] ist ein *Eclipse* Plug-In zum Modellieren von Featuremodellen. Es besteht die Möglichkeit, es mit dem *Rational Software Modeler (RSM)* zu koppeln und so Software, die in *RSM* modelliert wurde, entsprechend ausgewählter Features automatisch zu generieren. Dazu werden Teile von *UML*-Modellen auf Features gemappt. In [CA05]

wird dieses Verfahren genauer beschrieben. Im Gegensatz zu *pure::variants* wird eine Modellebene zwischen Features und generierter Software eingefügt. Es wird aber nur negative Variabilität unterstützt.

5.3 openArchitectureWare

oAW [oaw08b] ist ein modulares *MDSD*-Framework. Es bietet eine Sprachfamilie zur Implementierung von Modeltransformatoren, Codegeneratoren und Modellvalidierern. Zur Verarbeitung von Workflows steht eine sogenannte "Workflow Engine" zur Verfügung. Diese ist beliebig erweiterbar, um proprietäre Workflowkomponenten zu unterstützen.

oAW ist in der Lage, Modelle verschiedenster Metametamodelle, wie z. B. *Ecore*, *XML* und *UML2* zu verarbeiten. Außerdem bietet es die Komponenten *XWeave* und *XVar*, um positive und negative Variabilität von Modellen auszudrücken.

6 Fazit

Mit dieser Arbeit haben wir gezeigt, dass es möglich ist, eine existierende *MDSD*-Infrastruktur, welche auf *Ecore* basierende domänenspezifische Modelle verarbeitet, so anzupassen, dass Softwareproduktlinien unterstützt werden. Dazu kann die Variabilität der einzelnen Systeme der Familie in Aspektmodelle gekapselt werden. Momentan unterstützt die neu implementierte Modelltransformation zum Mischen von Modellen nur die von uns genutzten domänenspezifischen Modelle. Allerdings ist diese Transformation leicht so anpassbar, dass alle auf *Ecore* basierenden Modelle unterstützt werden können.

Unser Ansatz kann auch zur Versionierung von Softwaresystemen verwendet werden, indem eine neue Version eines Systems durch die darin enthaltenen Features definiert wird.

Der von uns verfolgte Ansatz kapselt Variabilität von Softwaresystemen in Aspektmodellen. Dies ist günstig, wenn feinkörnige Variabilität durch Features ausgedrückt werden soll. Für den Fall, dass Features nur grobkörnige Variabilität beschreiben, könnte Variabilität in Komponentenmodelle gekapselt werden, und die Aufgabe des Mischens von Modellen entfällt.

Die Entscheidung, ob ein Feature ein Aspektfeature oder ein Steuerfeature ist und mit welchen Aspektmodellen ein Aspektfeature assoziert ist, wird momentan noch manuell vom Entwickler getroffen. Weiterhin werden Workflows wie in Abbildung 2 manuell implementiert. Zukünftig müsste untersucht werden, inwieweit *oAW*-Workflows automatisch generiert werden können. Gegenwärtig werden Basis- und Aspektmodelle unabhängig voneinander entwickelt. Der Entwicklungsprozess würde verbessert werden, wenn es möglich wäre, Features und Aspektmodelle während der Entwicklung miteinander zu verbinden, wie es z. B. *FeatureMapper* [HKW08] erlaubt. Zur Entwicklung von "großen" Softwaresystemfamilien, bei denen Aspektmodelle viele Abhängigkeiten untereinander besitzen, wäre eine Visualisierung dieser, vielleicht mittels [vis08], hilfreich.

Momentan wird nur eine Teilmenge von *UML*-Modellelementen für die Aspektmodellierung unterstützt. Diese Teilmenge ist ausreichend für unsere Zwecke, sie müsste allerdings noch für den allgemeinen Fall, z. B. für Verhaltensmodelle, angepasst werden.

Literatur

[AC04] Michal Antkiewicz und Krzysztof Czarnecki. FeaturePlugin: Feature Modeling Plug-In for Eclipse. In *eclipse '04: Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*, Seiten 67–72, New York, NY, USA, 2004. ACM.

[CA05] K. Czarnecki und M. Antkiewicz. Mapping Features to Models: A Template Approach Based on Superimposed Variants. In Michael Glück, Robert; Lowry, Hrsg., *Generative Programming and Component Engineering*, Jgg. Volume 3676/2005 of *Lecture Notes in Computer Science*, Seiten 422–437. Springer Berlin / Heidelberg, 2005.

[emf08] Eclipse Modeling Framework (EMF) website, June 2008. http://www.eclipse.org/modeling/emf/.

[GFd98] M. Griss, J. Favaro und M. d'Alessandro. Integrating Feature Modeling with the RSEB. In *Proceedings of theFifthInternational Conference on Software Reuse*, Seiten 76–85, Vancouver, BC, Canada, 1998.

[GV07] Iris Groher und Markus Voelter. XWeave: Models and Aspects in Concert. In *AOM '07: Proceedings of the 10th international workshop on Aspect-oriented modeling*, Seiten 35–40, New York, NY, USA, 2007. ACM.

[HKW08] Florian Heidenreich, Jan Kopcsek und Christian Wende. FeatureMapper: Mapping Features to Models. In *ICSE Companion '08: Companion of the 30th international conference on Software engineering*, Seiten 943–944, New York, NY, USA, 2008. ACM.

[KC04] U. Eisenecker K. Czarnecki, S. Helsen. Staged Configuration Using Feature Models. In *Software Product Lines*, Jgg. Volume 3154/2004 of *Lecture Notes in Computer Science*, Seiten 266–283. Springer Berlin / Heidelberg, 2004.

[KCH+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak und A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Bericht CMU/SEI-90-TR-21 ESD-90-TR-222, Software Engeneering Institue, Carnegie Mellon University, nov 1990. http://selab.postech.ac.kr/.

[KLD02] Kyo C. Kang, Jaejoon Lee und Patrick Donohoe. Feature-Oriented Product Line Engineering. *IEEE Software*, 19(4):58–65, 2002.

[oaw08a] openArchitectureWare Documentation Website, June 2008. http://www.eclipse.org/gmt/oaw/doc/.

[oaw08b] openArchitectureWare Website. http://www.openarchitectureware.org/, June 2008.

[pur08] pure::variants Website, June 2008. http://www.pure-systems.com/Variant_Management.49.0.html.

[Rie03] M. Riebisch. Towards a More Precise Definition of Feature Models. In *Modelling Variability for Object-Oriented Product Lines*, Seiten 64–76. BookOnDemand Publ. Co., 2003.

[vis08] The Visualizer Website, June 2008. http://www.eclipse.org/ajdt/visualiser/.