MOD2-SCM: Eine modellgetriebene Produktlinie für Softwarekonfigurationsverwaltungssyteme

Thomas Buchmann, Alexander Dotor

vorname.nachname@uni-bayreuth.de

Abstract: Heutige Softwarekonfigurationsverwaltungssysteme (SCMS) sind monolithische und schwer erweiterbare Systeme, die mit großem Aufwand an die modernen Entwicklungsprozesse angepasst oder sogar extra neu implementiert werden. Daher werden sie unsystematisch modifiziert und neu entwickelt. So ist es nahezu unmöglich, systematische Vergleiche zwischen bestehenden SCMS anzustellen. Zusätzlich sind die meisten SCM Verfahren nur implizit durch den Quellcode beschrieben, bzw. bei proprietären Systemen nur aus Black-Box-Tests ableitbar. Dies führt letztendlich dazu, dass die Methoden der SCM-Domäne nur aufwändig verglichen und erforscht werden können. Insbesondere die Abhängigkeiten unterschiedlicher Verfahren werden durch die enge Kopplung der monolithischen Systeme verschleiert. In diesem Papier beschreiben wir die Entwicklung einer modellgetriebenen Produktlinie für Softwarekonfigurationsverwaltungssysteme. Es wird ein allgemeines Featuremodell für SCM Systeme beschrieben und ein Prototyp für eine neue Generation von SCM Systemen vorgestellt. Durch die Beschreibung der Produktlinie in Modellen wird außerdem der Abstraktionsgrad erhöht und der Zugang zu existierenden Methoden durch die Verwendung graphischer Modelle erleichtert.

1 Einleitung

Es existieren zur Zeit über 70 verschiedene Softwarekonfigurationsverwaltungssysteme (kurz: SCMS), um die Softwareentwicklung unterstützen. SCMS sind selbst auch Softwaresysteme, deren Größe einige zehntausend bis zu mehreren Millionen Quelltext-Zeilen umfasst. Trotz dieses Umfangs handelt es sich meist um monolithische Systeme, deren Daten und Funktionen eng verschränkt sind und deren Verfahren lediglich implizit durch den Quellcode beschrieben werden. Eine Identifikation einzelner Methoden ist kaum möglich, ebensowenig wie eine Wiederverwendung bereits implementierter Verfahren. Dies führt dazu, dass für ein neues Verfahren, oder sogar nur für eine neue Kombination existierender Verfahren, ein SCMS von Grund auf neu implementiert wird. Die fehlende Modularität erschwert auch die Wiederverwendung eines SCMS für unterschiedliche Software-Entwicklungsprozesse, da bereits kleine Unterschiede zwischen den Prozessen die Entwicklung eines neuen SCMS erfordern können. Es besteht daher der Bedarf nach einer neuen Generation von SCMS, die durch ihre modulare Architektur (1) die Wiederverwendung bestehender Verfahren erlauben, (2) die Anpassbarkeit an Entwicklungsprozesse verbessern bzw. überhaupt erst ermöglichen und (3) die Erweiterbarkeit verbessern, um ein SCMS in möglichst vielen Entwicklungsprozessen einsetzen zu können.

Die modellgetriebene Modulare Produktlinie für Softwarekonfigurationsverwaltungssysteme (kurz: MOD2-SCM) [Dot11, BD09c] identifiziert Gemeinsamkeiten von SCMS auf Basis eines Entwicklungsprozesses für modellgetriebene Software Produktlinien (MOD-PL) [BDW10, Buc10]. Der Entwicklungsaufwand eines SCMS wird so reduziert, da gemeinsame Komponenten wiederverwendet und die Produkte aus dem Domänenmodell generiert werden (anstatt sie von Hand zu implementieren). So kann durch Konfiguration des Domänenmodells nach den Anforderungen eines Softwareentwicklungsprozesses ein SCMS speziell an den zu unterstützenden Prozess angepasst werden.

Im Folgenden geben wir einen Überblick über das verwendete Featuremodell für Softwarekonfigurationsverwaltungssysteme sowie das generisches Domänenmodell, das auf diesem Featuremodell aufbaut und die Grundlage für die Systemfamilie bildet. Das Featuremodell erlaubt nicht nur die systematische Beschreibung der generierten SCMS, sondern auch einen Vergleich und die Klassifikation bereits bestehender monolithischer Systeme. Durch das MOD2-SCM-Domänenmodell wird die SCM-Domäne mit Hilfe von Komponenten und Klassen beschrieben. Abhängigkeiten und Kopplungen werden dabei explizit identifiziert und reduziert. Dies führt zu einer vertieften Einsicht in die SCM-Domäne speziell hinsichtlich der Modularisierbarkeit, Erweiterbarkeit und Co-Evolution [BDW08b]. Durch die Verwendung von Storydiagrammen [ZSW99] zur Verhaltensmodellierung kann aus dem Domänenmodell ausführbarer Quelltext generiert werden.

2 Das MOD2-SCM Featuremodell

2.1 Domänenanalyse

Zur Analyse der SCM-Domäne wurde FORM [KKL+98] - eine Weiterentwicklung von FODA [KCH+90] verwendet. FORM unterscheidet vier Ebenen:

Capability Layer: Features auf dieser Ebene beschreiben die Produktlinie durch Dienste, Operationen und Qualitäten, die angeboten werden sollen.

Operating Environment Layer: Die Features, die hier spezifiziert werden, beschreiben die Produktlinie durch Eigenschaften der Hard- und Softwareumgebungen, in denen die konkreten Anwendungen später eingesetzt werden.

Domain Technology Layer: Die Techniken, die speziell in der Anwendungsdomäne verwendet werden, werden mit Features auf dieser Ebene beschrieben. In der MOD2-SCM Produktlinie werden hier etwa die Features zur Realisierung der unterschiedlichen Module (Produktraum, Versionsraum, etc.) definiert.

Implementation Technique Layer: Auf dieser Ebene beschreiben Features die Produktlinie mit Hilfe der verwendeten grundlegenden Techniken. Dabei handelt es sich nicht um domänenspezifische Techniken, sondern um Basistechniken der Datenübertragung und -verarbeitung.

Das Ergenis der Domänenanalyse nach FORM wird in einem Featuremodell festgehalten.

2.2 Featuremodell

Das Featuremodell wurde mit Hilfe des Werkzeugs *FeaturePlugin* [AC04] erstellt und umfasst 147 Features, sowie 34 aussagenlogische Einschränkungen. Dabei ist zu beachten, dass nicht alle Features zur Auszeichnung von Modellelementen verwendet werden. 45 Features sind Erweiterungspunkte, die lediglich zur Klassifikation dienen, und nicht implementiert wurden. Von den verbliebenen 102 Features sind jedoch nur 45 Merkmale für Featureannotationen verwendet worden. Dies hat mehrere Gründe: Erstens beeinflussen einige Merkmale über die aussagenlogischen Einschränkungen die Konfiguration. Zweitens dienen verpflichtende Merkmale zur Strukturierung der variablen Merkmale, wurden aber nicht zum Markieren verwendet. In MOD2-SCM dient das Featuremodell sowohl zur Klassifikation von SCMS als auch zur Auszeichnung des Domänenmodells (vgl. Abschnitt 3.2).

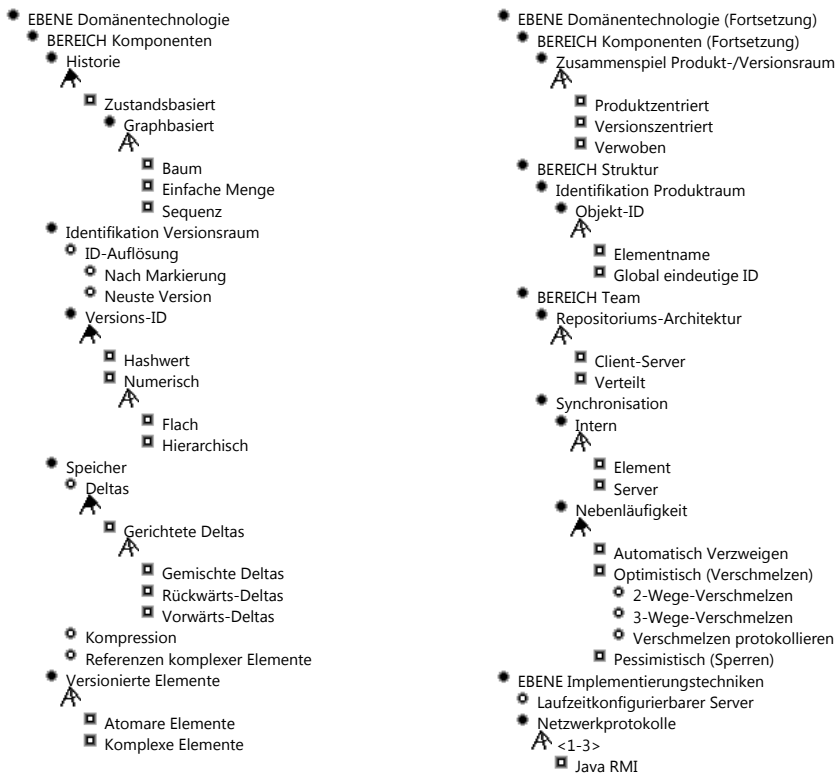

Abbildung 1: Ausschnitt aus dem Featuremodell der MOD2-SCM Produktlinie.

Der in Abbildung 1 gezeigte Ausschnitt des Featuremodells zeigt einen Teil der Ebene *Domain Technology Layer* (EBENE Domänentechnologie in der Abbildung). Diese Ebene enthält einen Bereich mit sämtlichen Komponenten, die für ein SCM-System benötigt werden: Die *Historie* zum Speichern der Versionen und ihrer Abhängigkeiten. Die *Identifikati-*

on [der Versionen im] Versionsraum, sowie zusätzliche *Meta-Informationen* zur Identifikation. Der *Speicher*, der die Versionen - meist platzsparend - persistenziert. Die Komplexität der *Versionierten Elemente* und die Möglichkeit parallel Entwicklungen durch *Verzweigen* zu unterstützen, sowie letztendlich das *Zusammenspiel von Produkt-/Versionsraum*. Im Bereich *Struktur* ist dies die *Identifikation [der Elemente im] Produktraum* und im Bereich *Team*: Die „Arbeitsbereich-Informationen" bestimmen, welche Informationen die SCMS-Operationen im Arbeitsbereich benötigen. Die *Repositoriums-Architektur* bestimmt den Aufbau und Kommunikationswege des SCMS, während die *Synchronisation* Verfahren zur Koordination mehrerer Entwickler beschreibt.

Aufgrund der erheblichen Größe des Featuremodells für MOD2-SCM kann im Rahmen dieses Aufsatzes nur ein kleiner Teil gezeigt werden. Das vollständige Featuremodell und die zugehörige Domänenanalyse ist in [Dot11] nachzulesen. Die Umsetzung dieser Features erfolgt im Domänenmodell.

3 Das ausführbare MOD2-SCM Domänenmodell

Das MOD2-SCM Domänenmodell beschreibt die Klassenstruktur bzw. das Verhalten der einzelnen Methoden mit Hilfe von Klassen- bzw. Storydiagrammen in Fujaba [Zün02]. Storydiagramme [ZSW99] dienen in Fujaba zur Modellierung des Verhaltens von genau einer Methode einer Klasse. Sie ähneln Interaktionsübersichtsdiagrammen in UML2 [OMG09]. Fujaba ist in der Lage aus Storydiagrammen ausführbahren Quelltext zu generieren, und somit modellgetriebenes Arbeiten zu unterstützen. Das Fujaba Modell wird durch ein Paketmodell aus dem MODPL-Paketdiagrammeditor ergänzt, das die grundlegende Architektur der MOD2-SCM Produktlinie mit Hilfe von Paketdiagrammen beschreibt. Dabei besteht eine 1:1-Verknüpfung zwischen den Paketen des MODPL-Paketmodells und des Fujaba-Modells (über den voll qualifizierten Namen). Zusätzlich ist auch jedes Klassendiagramm aus dem Fujaba-Modell genau einem Paket zugeordnet, dessen Inhalt es darstellt. Die Synchronisation der beiden Modelle erfolgt mittels des MODPL-Werkzeugkastens [Buc10, BDW10].

3.1 Architektur

3.1.1 Paketdiagramm

Bei der Erstellung des MOD2-SCM Paketmodells spielte auch die Variabilität und insbesondere eine ökonomische Auszeichnung mittels Featureannotationen eine Rolle. Daher wurden die Server-, Kommunikations- und Arbeitsbereich-Pakete ihren jeweiligen Modulen zugeordnet (anstatt z.B. drei Pakete für Server, Arbeitsbereich und Kommunikation zu wählen und die Komponenten darauf zu verteilen). Nur so ließ sich mit genau einer Featureannotation am entsprechenden Unterpaket das Domänenmodell konfigurieren, z.B. im Falle der Historie mit genau einem Feature am entsprechenden *history*-Unterpaket (vgl.

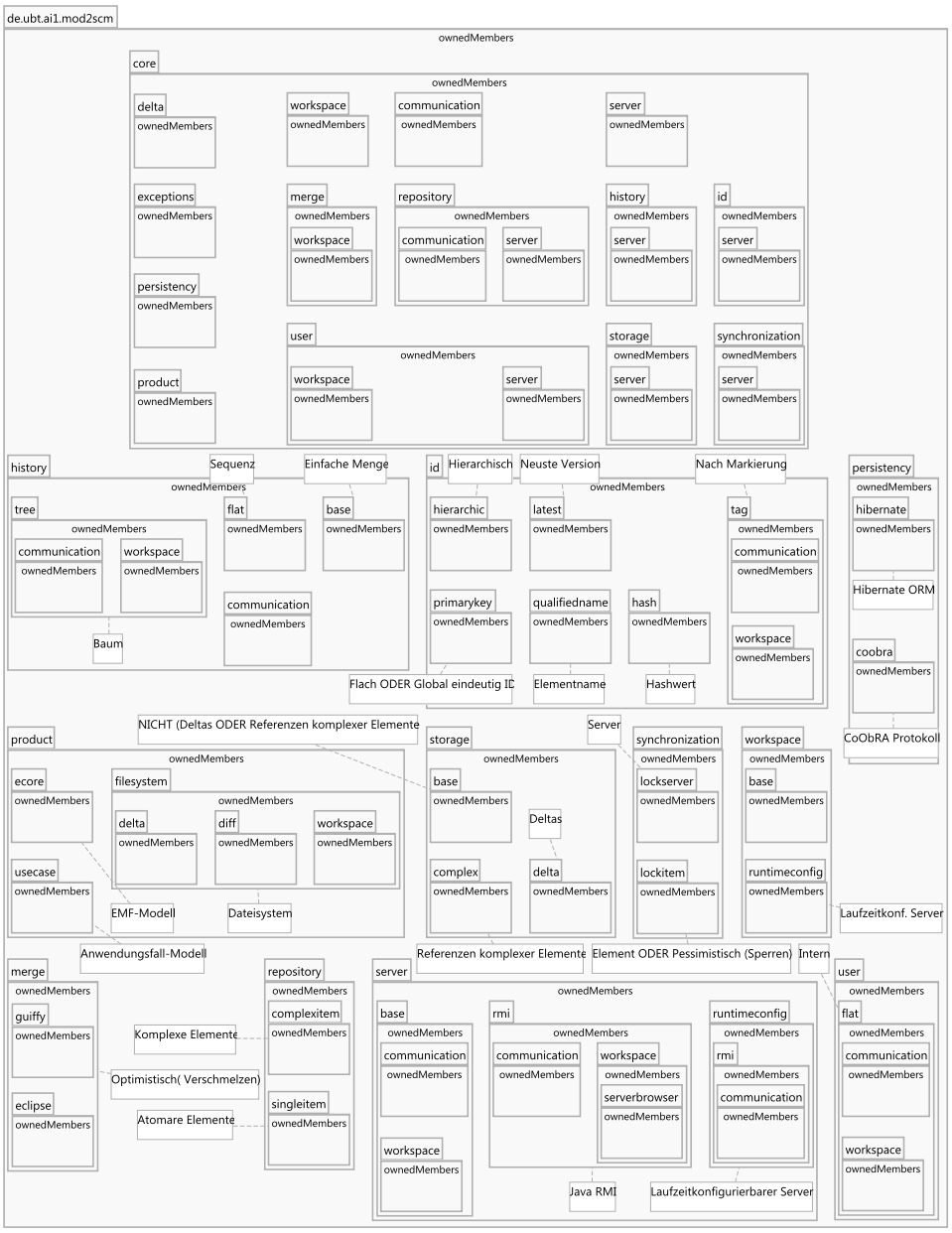

Abbildung 2: Das Paketdiagramm des MOD2-SCM Domänenmodells mit Featureannotationen.

Abbildung 2).

Abbildung 2 zeigt die Architektur des MOD2-SCM Domänenmodells. Es besteht aus 14 Kernmodulen und 29 Modulen der Modulbibliothek, die auf diesen Kernmodulen aufbau-

en. Einige Pakete sind hier bereits mit Featureannotationen ausgezeichnet. Bei der Auszeichnung werden die eindeutigen Bezeichner der jeweiligen Features aus dem Featuremodell und nicht der im Featuremodell angezeigte Name verwendet. Das von uns entwickelte Werkzeug stellt hierüber die Verknüpfung von Featuremodell und Domänenmodell her [BD09b, BDW10, Buc10]. Die Vergabe von Featureannotationen erfolgt durch Auswahl der jeweiligen Features im Featuremodell. Aufgrund von Platzrestriktionen können leider im Rahmen dieses Artikels keine weiteren Auszüge aus dem Domänenmodell (z.B. Klassen- oder Storydiagramme) präsentiert werden. Weiterführende Informationen zum Domänenmodell sind aber in [Dot11] nachzulesen.

3.1.2 Kontrolle der Kopplung

Für die Entwicklung einer Produktlinie für SCM-Systeme ist es essentiell wichtig, die Module, welche die unterschiedlichen Aufgabenbereiche eines SCM Systems realisieren, weitestgehend voneinander zu entkoppeln. Nur so ist es möglich, z.B. unterschiedliche Arten von Delta- und Versionsspeicherung orthogonal miteinander zu kombinieren. Daher müssen Abhängigkeiten zwischen solchen Modulen im Domänenmodell erfasst und womöglich beseitigt werden. Der von uns entwickelte Paketdiagrammeditor zur Unterstützung des *Modellieren im Großen* [BDK09, BDW10, Buc10] bietet hierfür eine Hilfestellung. Existierende Metriken zur Messung der Kopplung basieren i.d.R. auf Klassen und ihren Abhängigkeiten [BDW98]. Die Abhängigkeiten im MOD2-SCM Domänenmodell sind jedoch anhand der Pakete und ihren privaten Importbeziehungen beschrieben. Daher können sie analog zu den Abhängigkeiten definiert werden, die für die Existenz privater Paketimport-Beziehung zuständig sind, da sich diese unmittelbar auf die im Paket enthaltenen Klassen zurückführen lassen [Buc10]: Eine Abhängigkeit einer Klasse A von einer Klasse B, liegt genau dann vor, wenn (1) B ein Attributtyp von A ist, (2) B ein Rückgabetyp einer Methode von A ist (einschl. Ausnahmen), (3) B ein Parametertyp einer Methode von A ist, (4) B direkte Oberklasse von A ist, (5) B als Objekttyp in einem Storydiagramm von A verwendet wird. Die **Kopplungsstärke** $k(A;B)$ zwischen einer Klasse A und einer Klasse B lässt sich wie folgt berechnen:

$$
\begin{aligned}
k(A,B) &= |B\ Attributtyp\ in\ A| \\
&+ |B\ Rückgabetyp\ in\ A| \\
&+ |B\ Parametertyp\ in\ A| \\
&+ |B\ Oberklasse\ von\ A| \\
&+ |B\ Objekttyp\ in\ A|
\end{aligned}
$$

Abbildung 3 zeigt einen Ausschnitt aus dem Paketdiagramm des MOD2-SCM Domänenmodells und die unterschiedlich starke Kopplung zwischen den einzelnen Paketen. Die Liniendicke gibt dabei den Grad der Kopplung an. Hierbei kommen im graphischen Editor verschiedene Intervalle für die Darstellung der Liniendicke zum Einsatz. Dabei werden zunächst die Kopplungwerte aller Elemente in den jeweiligen Paketen ermittelt und summiert: Ist die Summe dann 1, so beträgt die Liniendicke ebenfalls 1. Die Liniendicke ist

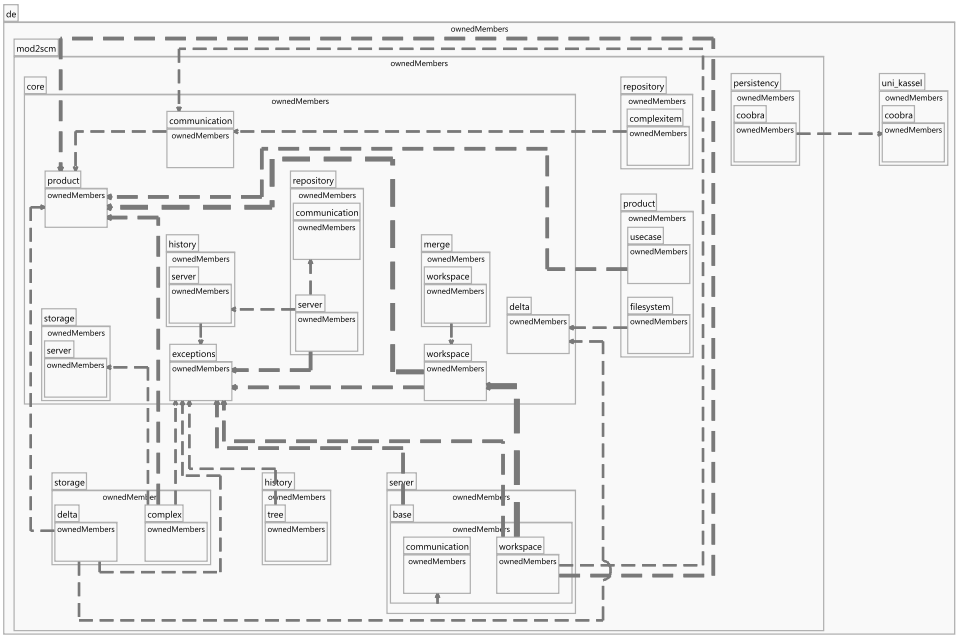

Abbildung 3: Grad der Kopplung zwischen den Paketen wird durch unterschiedlich dicke Linien ausgedrückt.

2, wenn die Summe Werte zwischen 2 und 9 annimmt, 4 bei Werten zwischen 10 und 19, usw. Das Maximum der Liniendicke beträgt 10 und gilt für Werte > 40.

Das vollständige MOD2-SCM Domänenmodell besteht aus 14 Kernmodulen und 29 Modulen in der Modulbibliothek (siehe auch Abb. 2). Zwischen ihnen bestehen 244 Abhängigkeiten, größtenteils zwischen Kernmodulen bzw. einem austauschbaren Modul zu seinem Kernmodul. Die Austauschbarkeit wird vor allem durch Schnittstellen-Abstraktion erreicht, so dass jedes Bibliotheksmodul von der Schnittstelle eines Kernmoduls abhängt. Es existieren kaum Kopplungen unter den Bibliotheksmodulen, somit sind diese gegen beliebige Module mit der gleichen Schnittstelle austauschbar. Anhand der Kopplungsanzahl und -stärke ließen sich auffällige Module identifzieren und genauer untersuchen. Da jedes Kernmodul einen Aufgabenbereich eines SCMS repräsentiert, lassen diese Kopplungen auch Rückschlüsse auf die Abhängigkeiten der SCM-Konzepte zu. Eine quantitative Analyse der Kopplungen liefert schnell gute Indikatoren für „kritische“ bzw. „interessante“ Pakete, lässt sich aber nur schwer zur Bewertung der Architekturqualität einsetzen. Zwischen den Modulen der Modulbibliothek existieren jedoch nur wenige Abhängigkeiten, so dass sich fast alle Module beliebig austauschen lassen. Die Entkopplung der einzelnen Module innerhalb der MOD2-SCM Produktlinie ist gelungen.

3.2 Verbindung mit dem Featuremodell

Die Verbindung zwischen dem Featuremodell und dem Domänenmodell wurde mit Hilfe des Werkzeugs *MODPLFeaturePlugin* [BD09a, BD09b, BDW10, Buc10] hergestellt. Hierzu wurden die entsprechenden Modellelemente mit den jeweiligen Features annotiert. Der Teil des Featuremodells, der mit dem Domänenmodell realisiert wurde, umfasst insgesamt 147 Features. Letztendlich war es aber nur erforderlich 57 Elemente des Domänenmodells manuell mit Features auszuzeichnen. Die Featureannotationen verteilen sich dabei wie folgt:

- 20 Pakete
- 8 Klassen
- 7 Methoden
- 22 Aktivitäten in Storymustern

Nur diese Modellelemente wurden manuell mit Features ausgezeichnet. Der MODPL-Werkzeugkasten unterstützt den Anwender durch automatische Propagation und Reparaturaktionen von Featureannotationen. Durch diesen Mechanismus wird sichergestellt, dass das konfigurierte Modell bzw. der konfigurierte Quelltext syntaktisch korrekt sind [BD09a, BD09b, BDW10, Buc10]. Hierbei werden Featureannotationen an abhängige Modellelemente automatisch übertragen, so dass bei der Konfigurierung keine Verletzungen des Metamodells entstehen.

Im MODPL-Werkzeugkasten bedeutet das Annotieren eines Modellelements, dass dieses nur dann in einem konfigurierten SCMS existiert, wenn das Feature Teil der Konfiguration ist [Buc10]. Soll ein Modellelement existieren, wenn ein Feature **nicht** in einer Konfiguration ist - z.B. als Alternative zu einem Modellelement mit ausgewähltem Feature - lässt sich dies nicht modellieren, da keine Annotation mit Negation existiert. Um diese Einschränkung zu umgehen, wurden Hilfsmerkmale im Featuremodell eingeführt, die über eine aussagenlogische Verknüpfung an das betroffene Feature gekoppelt sind, und immer den gegenteiligen Wert annehmen. Die von uns eingesetzte Software zur Featuremodellierung (*FeaturePlugin* [AC04]) unterstützt dies.

4 Vergleich mit anderen Ansätzen

4.1 Ähnliche Ansätze

4.1.1 Generische Versionsmodelle

Dieser Ansatz stellt keinen klassischen Produktlinienansatz dar, vielmehr wird hier versucht, ein generisches Versionsmodell zu entwickeln. Generische Versionsmodelle verwenden ein einheitliches Meta-Datenmodell, das als Schema für die Implementierung

unterschiedlicher Datenmodelle eingesetzt wird. Ziel dieser Methode ist es, mit einem einzigen Verfahren möglichst viele Datenmodelle mit Hilfe elementarer Datentypen zu beschreiben. Wiederverwendung wird in dieser Methode durch die Wiederverwendung der Elementaroperationen erreicht. Prominente Vertreter dieses Ansatzes sind ECM/UVM [WMC01] und ICE [Zel95, ZS97].

4.1.2 Bamboo

Bamboo [WGP04] ist ein Rahmenwerk, bei dem die Funktionalität eines SCMS manuell implementiert und anschließend in Quelltextfragmente zerlegt wurde. Jedes dieser Fragmente wird nun einem oder mehreren SCMS zugeordnet, in denen es verwendet wird. Durch Auswahl einer bestimmten Implementierung, werden die benötigten Quellcode-Fragmente selektiert und zu einem SCMS zusammengesetzt. Wiederverwendung wird in dieser Methode durch Wiederverwendung von Quellcode-Fragmenten in mehreren manuellen Implementierungen erreicht. Doch obwohl Vorlagen verwendet werden, handelt es sich nicht um eine dynamische, vorlagenbasierte Quellcode-Generierung, da für jede Variante die Quellcode-Fragmente explizit vorliegen müssen [Guo08]. Bamboo verwendet sogenannte Containment Models, eine spezialisierten Fassung von Entity/Relationship Diagrammen, zur Beschreibung des Datenmodells [Whi02, Whi03].

4.1.3 VS-Gen

Quellcode-Generatoren erzeugen Quellcode aus grafischen Modellen, indem sie die Modellelemente auf Quellcode-Vorlagen abbilden. Durch Annotation der Modellelemente kann die Abbildung zusätzlich noch parameterisiert werden, um die Quellcode-Erzeugung zu steuern. Meist lässt sich jedoch nur der Quellcode für die Struktur und nicht für das Verhalten generieren, so dass das Verhalten mit Hilfe der Quellcode-Vorlagen spezifiziert wird. Mit Hilfe der Annotationen können dann die benötigten Quellcode-Fragmente in der Vorlage gewählt werden, um so - ähnlich wie bei der bedingten Übersetzung - das gewünschte Verhalten zu erzeugen. Wiederverwendung wird in dieser Methode durch die Wiederverwendung der Quellcode-Vorlagen beim Generieren eines SCMS erreicht. Durch Annotation der Modellelemente mit den Merkmalen einer Produktlinie ist diese Methode ein Schritt in Richtung modellgetriebener Produktlinien. VS-Gen [KG04, Kov05] stellt einen solchen Ansatz dar.

4.2 Bewertung

In den vergangenen Jahren, gab es einige unterschiedliche Ansätze, eine Produktlinie für Versionskontrollsysteme zu erstellen. Allerdings beruhen viele davon auf manueller Implementierung. In der Anwendungsentwicklung müssen die Quellcodefragmente händisch zu einem lauffähigen System kombiniert werden, was mit einem oftmals nicht unerheblichen Aufwand verbunden ist. In einigen Ansätzen wird auf die Verwendung von formalen Modellen komplett verzichtet, andere benutzen sie lediglich zur Dokumentation bzw. zur

Beschreibung der statischen Struktur. In dem hier vorgestellten Ansatz hingegen werden sowohl Struktur als auch Verhalten der Produktlinie für SCM-Systeme durch formale Modelle beschrieben. Da das Domänenmodell alle Varianten umfasst, reduziert sich somit die Anwendungsentwicklung auf einen reinen Konfigurierungsschritt, an dessen Ende vollautomatisch ausführbarer Quelltext erzeugt wird.

5 Zusammenfassung

Die MOD2-SCM Produktlinie ist eine modellgetrieben entwickelte komplexe Produktlinie. Sie zeigt, dass sich mit dem in [Buc10, BDW10] vorgestellten MODPL-Ansatz und -Werkzeugkasten Produktlinien modellgetrieben entwickeln lassen. Dieser Ansatz bietet mehrere Vorteile gegenüber herkömmlicher Produktlinien-Entwicklung: Größter Vorteil ist, dass das manuelle Ableiten bzw. Zusammensetzen eines Software-Produktes aus dem Domänenmodell durch eine automatisierte Konfiguration ersetzt wird. Dabei kann sowohl sofort lauffähiger Quellcode generiert werden als auch ein konfiguriertes Modell, um es mit produktspezifischen Komponenten zu erweitern [Buc10]. Weiterhin wurde das Domänenmodell vollständig modellgetrieben entwickelt.

Durch die explizite Verknüpfung der Modellelemente aus Featuremodell und Domänenmodells sind die Abhängigkeiten von Features und Modell-Elementen zurückverfolgbar. Und auch die Architektur der Produktlinie lässt sich anhand von Paketdiagrammen grafisch darstellen und ebenfalls explizit mit den Features in Verbindung bringen. Ebenso lässt sich die Konfiguration eines Produktes grafisch im Domänenmodell darstellen sowie verwendete und nicht verwendete Modellelemente parallel anzeigen [Buc10].

Im Verlauf des Projektes zeigte sich, dass durch umsichtige Nutzung von objektorientierten Modellierungstechniken und den bewussten Einsatz von Entwurfsmustern der Annotationsaufwand für den Benutzer äußerst gering gehalten werden kann. Obwohl die von uns entwickelten Werkzeuge zur Verbindung von Featuremodell und ausführbahrem Domänenmodell die Annotation von Modellelementen mit Features aus dem Featuremodell auf einer beliebigen Granularitätsstufe erlauben [Buc10, BDW10], reichte es in den meisten Fällen aus, Featureannotationen auf Paketebene zu vergeben [Dot11]. Auch zeigte sich, dass es meist ausreicht Modellelemente mit einzelnen Features zu annotieren. Komplexe Featureausdrücke waren nicht notwendig, da aussagenlogische Verknüpfungen bereits im Featuremodell vorgenommen werden können.

Weiterhin zeigte sich, dass sich einzelne Module eines Softwarekonfigurationsverwaltungssystems derart entkoppeln lassen, so dass aus den einzelnen Modulen eine Modellbibliothek als Grundlage einer Softwareproduktlinie erstellt werden kann. Um dies zu erreichen, war der von uns im Rahmen des Projektes entwickelte Paketdiagrammeditor [BDW08a, BDK09, BDW10, Buc10] sehr hilfreich, da er die Abhängigkeiten im Domänenmodell erfassen und visualisieren kann. Hierbei wurde ebenfalls deutlich, dass die in der UML Spezifikation [OMG09] definierten privaten Importbeziehungen zwischen Modellelementen am Besten für diese Zwecke geeignet sind, da sie, im Gegensatz zu den

öffentlichen Importbeziehungen, nicht transitiv arbeiten. Somit erreicht man durch die privaten Importe eine bessere Übersicht über erwünschte und unerwünschte Kopplungen im Domänenmodell.

Der aktuelle Stand der MOD2-SCM Produktlinie ermöglicht es, 290.304.000 verschiedene SCM Systeme zu generieren. Damit ist die Produktlinie so groß, dass sich nicht mehr alle Konfigurationen erzeugen und geschweige denn testen lassen. Lediglich mit Hilfe des laufzeitkongurierbaren Servers[1] können ca. 1.000 Kombinationen von Servermodulen getestet werden. Somit stellt das Testen ein noch offenes Forschungsproblem dar.

Als Benutzerschnittstelle für das MOD2-SCM System wurde ein Eclipse Plugin für eine spezielle Konfiguration mittels herkömmlicher Programmierung erstellt. Zukünftige Forschungsarbeiten könnten eine Ausdehnung der Produktlinie auf die Benutzerschnittstelle umfassen.

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.

[BD09a] Thomas Buchmann und Alexander Dotor. Constraints for a fine-grained mapping of feature models and executable domain models. Jgg. 1 of *CTIT Workshop Proceedings*, Seiten 9–17. CTIT, Juni 2009.

[BD09b] Thomas Buchmann und Alexander Dotor. Mapping Features to Domain Models in Fujaba. In Pieter van Gorp, Hrsg., *Proceedings of the 7th International Fujaba Days*, Seiten 20–24, Eindhoven, The Netherlands, November 2009.

[BD09c] Thomas Buchmann und Alexander Dotor. Towards a Model-Driven Product Line for SCM Systems. Jgg. 2, Seiten 174–181, San Francsisco, CA, USA, 2009. Software Engineering Institute, Carnegie Mellon University.

[BDK09] Thomas Buchmann, Alexander Dotor und Martin Klinke. Supporting Modeling in the Large in Fujaba. In Pieter van Gorp, Hrsg., *Proceedings of the 7th International Fujaba Days*, Seiten 59–63, Eindhoven, The Netherlands, November 2009.

[BDW98] Lionel C. Briand, John W. Daly und Jürgen Wüst. A Unified Framework for Cohesion Measurement in Object-OrientedSystems. *Empirical Softw. Engg.*, 3(1):65–117, 1998.

[BDW08a] Thomas Buchmann, Alexander Dotor und Bernhard Westfechtel. Experiences with Modeling in the Large in Fujaba. In Uwe Aßmann, Jendrik Johannes und Albert Zündorf, Hrsg., *Proceedings of the 6th International Fujaba Days*, Seiten 5–9, Dresden, Germany, 2008. Technische Universität Dresden.

[BDW08b] Thomas Buchmann, Alexander Dotor und Bernhard Westfechtel. MOD2-SCM: Experiences with co-evolving models when designing a modular SCM system. In Dirk Deridder, Jeff Gray, Alfonso Pierantonio und Pierre-Yves Schobbens, Hrsg., *1st International Workshop on Model Co-Evolution and Consistency Management (MCCM08)*, Jgg. 1, Seiten 50–65, Toulouse, France, September 2008.

[1] Der laufzeitkonfigurierbare Server erlaubt das Austauschen von einzelnen Modulen zur Laufzeit. In Verbindung mit JUnit4 können so parametrisierte Tests durchgeführt werden.

[BDW10] Thomas Buchmann, Alexander Dotor und Bernhard Westfechtel. Werkzeuge zur modellgetriebenen Entwicklung von Produktlinien: Ein Erfahrungsbericht am Beispiel von Versionskontrollsystemen. In Andreas Birk, Klaus Schmid und Markus Völter, Hrsg., *Tagungsband der PIK 2010 Produktlinien im Kontext*, 2010.

[Buc10] Thomas Buchmann. *Modelle und Werkzeuge für modellgetriebene Softwareproduktlinien am Beispiel von Softwarekonfigurationsverwaltungssystemen*. Doktorarbeit, Universität Bayreuth, September 2010.

[Dot11] Alexander Dotor. *Entwurf und Modellierung einer Produktlinie von Software-Konfigurations-Management-Systemen*. Doktorarbeit, Universität Bayreuth, Bayreuth, 2011. Submitted.

[Guo08] Ge Guozheng. *RHIZOME: A Feature Modeling and Generation Platform for Software Product Lines*. Dissertation, University of California, Santa Cruz, Dezember 2008.

[KCH+90] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak und A. Spencer Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Bericht, Carnegie-Mellon University Software Engineering Institute, November 1990.

[KG04] Jernej Kovse und Christian Gebauer. VS-Gen: A case study of a product line for versioning systems. In *Generative Programming and Component Engineering 2004, Proceedings*, Seiten 396–415, 2004.

[KKL+98] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Gerard Jounghyun Kim und Euiseob Shin. FORM: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.

[Kov05] J. Kovése. *Model-driven development of versioning systems*. Dissertation, Technische Universität Kaiserslautern, Kaiserslautern, 2005.

[OMG09] OMG. *OMG Unified Modeling Language (OMG UML), Superstructure V2.2*. OMG, 140 Kendrick Street, Building A, Suite 300 Needham, MA 02494 USA, Februar 2009. Version 2.2.

[WGP04] James E. Whitehead, GuoTheng Ge und Kai Pan. Automatic Generation of Version Control Systems. Bericht, University of California, 2004.

[Whi02] James E. Whitehead. Uniform comparison of data models using containment modeling. In *HYPERTEXT '02: Proceedings of the thirteenth ACM conference on Hypertext and hypermedia*, Seiten 182–191, New York, NY, USA, 2002.

[Whi03] James E. Whitehead. Uniform Comparison of Configuration Management Data Models. In *Software Configuration Management*, Seiten 70–85, 2003.

[WMC01] Bernhard Westfechtel, Björn P. Munch und Reidar Conradi. A Layered Architecture for Unfiorm Version Management. Jgg. 27, Seiten 1111–1133, Dezember 2001.

[Zel95] Andreas Zeller. A Unified Version Model for Configuration Management. In Gail Kaiser, Hrsg., *Proceedings of the 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Jgg. 20 (4) of *ACM Software Engineering Notes*, Seiten 151–160. ACM Press, Oktober 1995.

[Zün02] Albert Zündorf. *Rigorous Object Oriented Software Development*, 2002.

[ZS97] Andreas Zeller und Gregor Snelting. Unified Versioning through Feature Logic. *ACM Transactions on Software Engineering and Methodology*, 6(4):397–440, Oktober 1997.

[ZSW99] Alber Zündorf, Andy Schürr und Andreas J. Winter. Story Driven Modeling. Technical Report tr-ri-99-211, University of Paderborn, 1999.