MDA in der Praxis
Erfahrungen mit modellgetriebener Softwareentwicklung in Großprojekten bei der Karstadt Warenhaus AG

Ralf Schäftlein, Thomas Mahler

Java Innovation Center
Itellium Systems & Services GmbH
Theodor-Althoff-Straße 2
45133 Essen
ralf.schaeftlein@itellium.com
thomas.mahler@itellium.com

Abstract: In diesem Text beschreiben wir die Einführung eines modellgetriebenen Softwareentwicklungsprozesses bei der Karstadt Warenhaus AG. Nach einer Schilderung der Rahmenbedingungen und der spezifischen Anforderungen skizzieren wir den gewählten Lösungsansatz. Zentrale Verfahren und Werkzeuge werden näher erläutert. Abschließend fassen wir die Erfahrungen zusammen, die in mehreren Projekten mit dem Ansatz gesammelt wurden.

1 Ausgangssituation

Im Jahr 1999 beschloss die Karstadt Warenhaus AG einen Relaunch der gesamten bestehenden Prozesslandschaft. Dieser Relaunch sollte den geänderten Anforderungen an die Prozesskette des Warenhauses sowie den sich verändernden Zielsetzungen des KarstadtQuelle Konzerns Rechnung tragen. Da die bestehenden IT-Systeme nicht an die neuen Anforderungen angepasst werden konnten, mussten sie abgelöst werden.

Kern des Relaunchs war das Warenwirtschaftssystem. Da sich im Markt kein Produkt finden ließ, dass die warenwirtschaftlichen Prozesse eines Warenhauses durchgängig abbildet, wurde beschlossen, gemeinsam mit der SAP AG die SAP RETAIL Lösung entsprechend zu erweitern. Die Einführung dieses Systems soll Ende 2005 abgeschlossen sein.

Durch die Änderungen an den zentralen Warenwirtschaftsprozessen wurden auch bei den angrenzenden Systemen, insbesondere den Filial- und Logistiksystemen, umfangreiche Veränderungen im Bereich der Schnittstellen zur Warenwirtschaft notwendig. Da diese Systeme ebenfalls an geänderte organisatorische Prozesse angepasst werden mussten, technologisch aber überaltert waren, entschied man sich auch hier für eine vollständige Ablösung der Systeme. Da im Markt keine adäquaten Standardlösungen gefunden werden konnten, war man bei diesen Systemen auf Entwicklung in Eigenregie, teilweise mit externer Unterstützung angewiesen.

1.1. Herausforderungen

Aus der geschilderten Ausgangslage ergaben sich eine ganze Reihe von technischen und organisatorischen Herausforderungen.

Zum einen mussten Projektlenkungsstrukturen aufgebaut werden, die die Multiprojektorganisation über die einzelnen Großvorhaben hinweg sicherstellt. Insbesondere die zeitliche Abstimmung der Projekte zueinander, sowie die inhaltliche Abstimmung von Schnittstellen und Prozessabläufen müssen in Projektübergreifenden Gremien geleistet werden.

Zum anderen mussten tragfähige technische Lösungen entwickelt werden. Eine besondere Herausforderung war die Schnittstellenintegration der alten und neuen Systeme. Das Risiko einer BigBang-Umstellung der gesamten Systemlandschaft wurde als zu groß eingestuft. Daher wurden Konzepte für eine zeitliche entkoppelte Einführung der einzelnen Systeme benötigt. Beispielsweise wurde das Filialsystem bereits in 2004 umgestellt, während das alte Warenwirtschaftssystem erst Ende 2005 abgelöst werden soll. Die Schnittstellen des neuen Filialsystems mussten daher so flexibel gestaltet werden, dass zunächst eine Anbindung an die Hostbasierte alte Warenwirtschaft und nach der Umstellung eine Anbindung an SAP Retail möglich ist.

Um diese und weitere komplexe Anforderungen zu erfüllen, wurde in einem separaten Projekt ab 2001 eine EnterpriseApplicationIntegration(EAI)-Plattform aufgebaut. Eingesetzt wird die Software ICAN von SeeBeyond. Der gesamte Datenaustausch zwischen den neuen, den noch zu migrierenden und den bleibenden Bestandssystemen wird auf diese Integrationsplattform gehoben.

Eine dritte große Herausforderung stellte das Softwareengineering für die in Eigenregie zu entwickelnden Systeme dar.

Zum einen galt es, die in den Karstadt Fachbereichen entwickelten organisatorischen Prozesse inhaltlich vollständig, über alle Teilprojekte hinweg einheitlich zu dokumentieren. Die Dokumentation sollte über den gesamten Lebenszyklus der Software die implementierten Prozesse exakt beschreiben.

Bei früheren Individualentwicklungen war man nach dem klassischen Wasserfallmodell mit den groben Stufen Vorstudie, Anforderungsanalyse, Systementwurf, Implementierung, Integration, Auslieferung vorgegangen. Die aus der Literatur hinlänglich bekannten spezifischen Schwächen des Wasserfallmodells [Ba98, S.101] sollten in den neuen Projekten vermieden werden. Insbesondere die folgenden Punkte führten zu Problemen:

- Die „no-change-assumption“ kollidiert mit der sich rasch verändernden Prozesswelt eines Handelskonzerns. Nach Abschluss der Analysephase wurden früher Changes stillschweigend in der Implementierungsphase nachgereicht, ohne jedoch Eingang in die Dokumentation zu finden. Bereits bei der Betriebseinführung war dann die Prozessdokumentation nicht mehr synchron mit dem Implementierungsstand. Dies erhöhte die Wartungskosten.

- Die Fachbereiche wünschen eine direkte Einbeziehung in die Gestaltung von Abläufen, Benutzeroberflächen und in die Evaluierung von Prototypen. Solche Feedback-Schleifen ließen sich in der Wasserfallkaskade nicht zufriedenstellend abbilden. Gerade unter dem hohen Zeitdruck der aktuellen Projekte war hier eine Verbesserung notwendig.

- Es gab keine durchgängige Methode. Die Anforderungsanalyse erfolgte textuell. Im Entwurf wurden eine Reihe unterschiedlicher Methoden und Notationen verwendet (Strukturierte Analyse, ERM, Objektorientierte Analyse). Gewünscht war ein einheitlicher methodischer Rahmen, der eine einheitliche Dokumentation über alle Phasen und Teams hinweg ermöglicht und der nicht nur Verbote ausspricht, sondern produktive Konstruktionsregeln vorgibt.
  Besonders wichtig war hier, dass die Dokumentationen von allen Projektbeteiligten (Fachbereiche, Systemanalytiker, Softwareentwickler, sowie externe Partner) verstanden, akzeptiert und bearbeitet werden müssen.

- Im bestehenden Vorgehensmodell war Qualitätssicherung immer nur als nachgelagerte Aktivität vor der Abnahme der jeweiligen Phasenergebnisse vorgesehen. Qualitätsverbesserungen konnten so nur in sehr langen Zeiträumen erreicht werden. Angestrebt wird eine konstruktive, kontinuierliche Qualitätssicherung als integraler Bestandteil des Vorgehens.

Die neuen Individuallösungen sollten in verschiedenen Teams vom konzerneigenen Systemhaus Itellium entwickelt werden. Aufgrund begrenzter Ressourcen war es notwendig, externe Mitarbeiter in die Teams zu integrieren und zum Teil die Entwicklung komplett extern zu vergeben.

Trotz der heterogenen Teamstrukturen sollten wesentliche Vorgehensweisen und Qualitätsziele nicht kompromittiert werden.

- Alle Systeme sollen nach einem einheitlichen und durchgängigem Verfahren entworfen werden.

- Die Software der einzelnen Systeme soll eine einheitlich Softwarearchitektur aufweisen.

- Die Systeme müssen bei minimalem Portierungsaufwand auf unterschiedlichen Systemplattformen betrieben werden können (Z.B. CORBA auf HP Nonstop oder J2EE Server auf Intel-Linux).

- Die Systeme müssen sich nahtlos in übergreifendene Strukturen wie EAI-Bus, Single-Sign-On integrieren.

- Die Teams sollen mit einer einheitlichen Werkzeugkette arbeiten.

- Entwickler sollen ohne Umstellungsschwierigkeiten zwischen den Projekten wechseln können.

- Die Systeme sollen mit einheitlichen Verfahren und Werkzeugen getestet werden können.
- Die Aufwände in der Wartungsphase sollen minimiert werden durch einheitliche Dokumentation, Architektur, Verfahren und Werkzeuge.

2. Der Lösungsansatz

Für die oben genannten Anforderungen an Vorgehensmodell, Architektur und Methodik mussten praxisgerechte Lösungen gefunden werden, die engverzahnt zusammenarbeiten.

Auf der Ebene des Vorgehensmodells verabschiedete man sich vom Wasserfallmodell zugunsten eines agilen iterativ-inkrementellen Modells.

Auf der Ebene der Softwarearchitektur entschied man sich für strategische Festlegung auf eine J2EE basierte Architektur. Wesentliche Designvorgaben sind dabei in einem eigenentwickelten Infrastruktur-Framework namens JACK (Java Architektur für KarstadtQuelle) implementiert.

Auf der Ebene Methodik wurde ein modellgetriebener, generativer Ansatz gewählt, der sich unmittelbar an die Konzepte der Model-Driven Architecture (MDA) der Object Management Group (OMG) anlehnt. Wir verzichten hier aus Platzgründen auf eine ausführlichere Darstellung und verweisen auf die einschlägige Literatur [KWB03].

Die Anforderungsanalyse erfolgt bei diesem Ansatz vollständig in UML [OE04]. Die resultierenden Modelle, die zunächst nur fachliche Informationen enthalten, werden mittels Abbildungsvorschriften in weitere Modelle umgeformt oder in konkrete Implementierungen überführt. Siehe Abbildung 1.

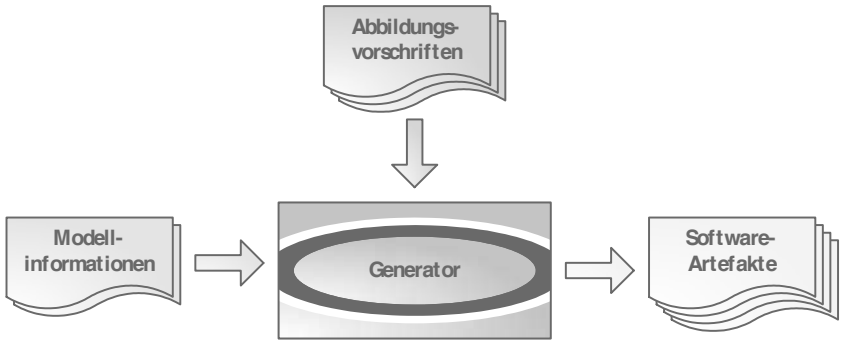

Abbildung 1: Modellgetriebene, generative Softwareentwicklung

Der OMG-Ansatz sieht vor, dass aus dem plattformunabhängigen Analysemodell (PIM) ein plattformspezifisches Modell (PSM) generiert wird, welches ausschließlich auf die Ausführung in einer Ziellaufzeitumgebung ausgerichtet ist [SOL01]. Siehe Abbildung 2.

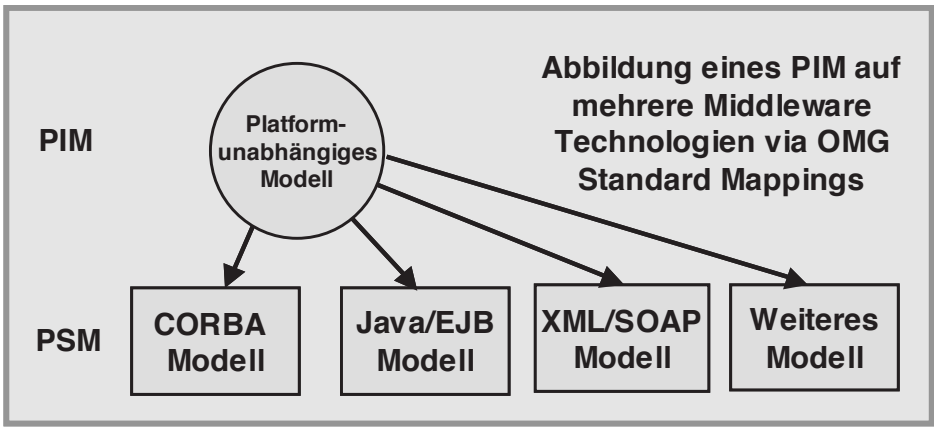

Abbildung 2: Der OMG Ansatz. Darstellung folgt [SOL01] S.16ff.

Der bei Itellium gewählte Ansatz weicht an einer Stelle signifikant von diesem ursprünglichen Ansatz ab. Hier wird aus dem plattformunabhängigen Analysemodell (PIM) Java Code generiert, der auf das JACK-Framework aufsetzt. Auch dies ist ein plattformspezifisches Modell (PSM). Jedoch ist in dem generierten JACK-Code kein plattformspezifischer Code (z.B. CORBA- oder EJB-Stubs) enthalten. Siehe Abbildung 3.

Dieses Design hat spezifische Auswirkungen:

- Der MDA-Generator muss Code für das JACK Framework generieren können.
- Der generierte Code ist frei von Infrastrukturcode, behält daher immer einen nachvollziehbaren Bezug zum Analysemodell. CORBA- oder EJB-Infrastrukturcode wird durch plattformspezifische Generatoren ergänzt.
- Das JACK Framework muss die Laufzeitumgebungen transparent kapseln und gleichzeitig für eine performante Ausführung sorgen.
- Know-how über die spezifische Ausführungsplattform wird aus dem MDA Generator verlagert in das Framework.
- Die Implementierung von Businesslogik bleibt unabhängig von der Laufzeit-umgebung.
- Implementierter Code kann durch Konfigurationsänderungen auf anderen Middleware Plattformen ablaufen.

Im weiteren Verlauf des Textes wird zunächst das Zusammenwirken von Framework und MDA-Ansatz vorgestellt. Anschließend fassen wir unsere bisherigen Erfahrungen mit diesem Ansatz zusammen.

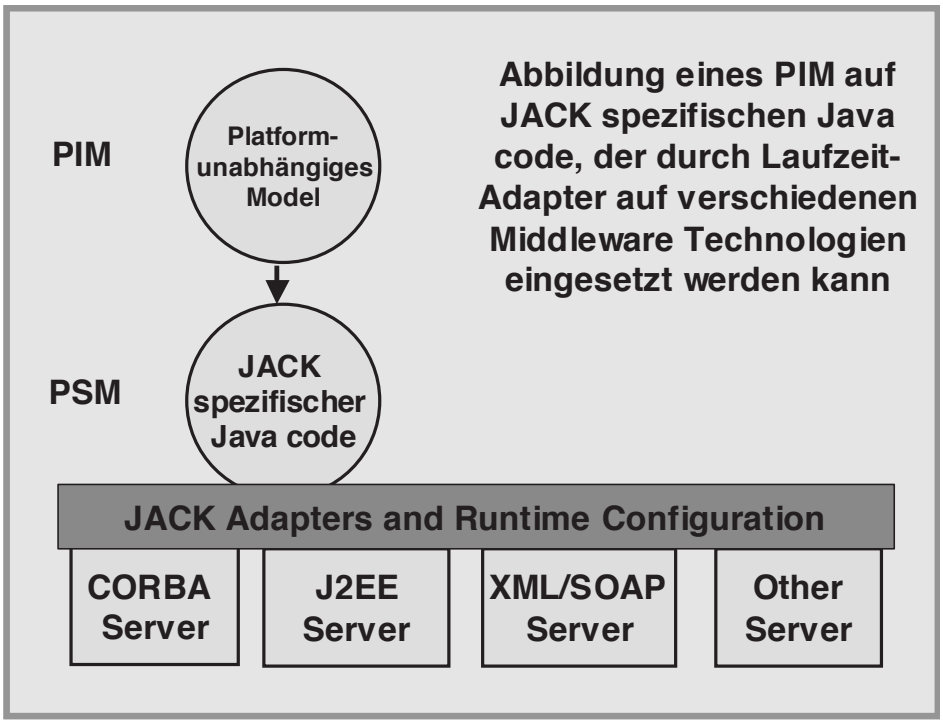

Abbildung 3: Der Itellium Ansatz

3. Framework und Generator

3.1. JACK

Das JACK- Framework (Java Architektur für KarstadtQuelle) wurde im Auftrag der Karstadt Warenhaus AG als Architekturrahmen für unternehmenskritische J2EE-Anwendungen von der Itellium GmbH konzipiert und implementiert. An der Erstellung waren die Beratungshäuser Gebit, Viadee und Unilog beteiligt. In das Design des Frameworks sind mehrjährige Erfahrungen der genannten Firmen mit Java-basierter Softwareentwicklung in Großprojekten eingeflossen.

Das Design des Frameworks musste eine Reihe strategischer Ziele berücksichtigen. Es sollte für neue J2EE Anwendungen eine stabile und skalierbare Basis bieten. Es sollte spezielle Unterstützung für die Modularisierung und flexible Konfiguration von Businesslogik und Workflows bieten. Ein modellgetriebenes Vorgehen sollte optimal unterstützt werden.

Die Abbildung 4 zeigt den groben Aufbau des JACK Architekturrahmens. Auf der obersten Schicht sind mögliche Clients aufgeführt außer Java- und Webclients sind auch Webservicesclients wie der SeeBeyond EAI-Bus abgedeckt.

Für den GUI-Bereich bietet JACK MVC-basierte Konstrukte für die Implementierung von Swing- und Struts-basierten Views (V) und Controllern (C) [Gam95].

Für den Zugriff der Controller auf die Geschäftslogik, das Model (M), sind verschiedene Mechanismen konfigurierbar. Es besteht die Möglichkeit mit einem lokalen Methodenaufruf innerhalb der selben JVM auf das Model zuzugreifen. Für den Zugriff auf entfernte Objekte werden die RPC-Mechanismen CORBA, Java RMI, EJB Session Beans und SOAP unterstützt. Der gesamte Zugriff erfolgt für den Controller transparent. D.h. der Anwendungscode bleibt vollständig frei von middlewarespezifischem Infrastrukturcode.

Auf der Modellschicht stellt JACK einen Rahmen für die Strukturierung von Businesslogik zur Verfügung, der grob zwischen Aktivitäts- und Entitätsobjekten unterscheidet. Entitätsobjekte, Entities, sind persistierbare Datencontainer. Der Zugriff auf Attribute erfolgt Javabeans-konform. Entities entsprechen dem PoJO (Plain Old Java Objects) Ansatz, der auch für EJB 3.0 angestrebt wird [AlCrMa03]. Sie sind daher nicht an einen Persistenzmechanismus gekoppelt.

Aktivitätsobjekte, die Activities, beinhalten die eigentliche Geschäftlogik, sie steuern den konkreten fachlichen Ablauf. Activities erzeugen oder laden Entities, und manipulieren und speichern sie.

Der Zugriff der Activities auf persistente Medien wie Relationale Datenbanken oder Verzeichnisdienste erfolgt transparent durch einen Persistenzlayer. Diese Schicht abstrahiert Transaktions- und Abfrageoperationen in einer an ODMG und JDO angelehnten API und stellt konkrete Adapterimplementierungen für verschiedene objektrelationale Mapping-Werkzeuge und LDAP zur Verfügung.

Zusätzlich zu diesem Architekturrahmen stellt das Framework noch Services für Konfiguration, Objekterzeugung, Logging, Authentifizierung und Autorisierung zur Verfügung.

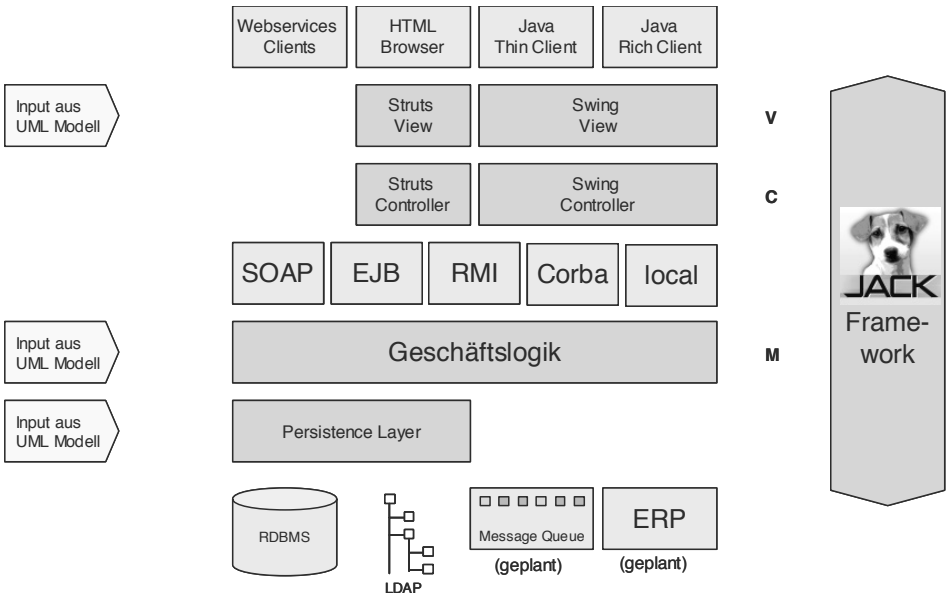

Abbildung 4: Der Architekturrahmen des JACK-Frameworks

3.2. CoJACK

Aufgabe des CoJACK (COdegenerator für JACK) Generators ist es, UML Modelle in Code für die JACK Plattform zu transformieren. In Abbildung 4 ist durch die Pfeile auf der linken Seite bezeichnet, welche Teile der Software generiert werden können. Im Moment ist die Verwendung von UML-Klassendiagrammen unterstützt. Geplant ist die Unterstützung für Aktivitätsdiagramme.

Das UML Modell wird durch die Verwendung spezifischer Stereotype und tagged values um Informationen für den Generator angereichert.

Der Generator arbeitet in zwei Phasen. In der ersten Phase werden UML XMI Daten in ein proprietäres Zwischenformat überführt. Dieser Schritt erlaubt es, den Generator an spezifische Eigenheiten verschiedenster UML Werkzeuge anzupassen [OMG02d].

In der zweiten Phase werden auf dem Zwischenformat weitere Transformationen durchgeführt. Diese Transformation werden definiert durch konfigurierbare Cartridges für verschiedene Zielformate. Zur Zeit können folgende Zielformate erzeugt werden: Struts JSP Code, Views, Controller und Models sowie dazugehörige Konfigurations-dateien, Activities- und Entities, Mappingrepository für das Objektrelationale Mapping, sowie ein HTML Glossar.

Der generierte Code enthält klar hervorgehobene Implementierungsstellen, sogenannte geschützte Bereiche, an denen die Entwickler dann die eigentliche fachliche Logik ausimplementieren.

Für alle Transformationsschritte werden durchgängig compiled stylesheets (XSLTC) verwendet. Der Generator ist vollständig durch ANT steuerbar und lässt sich damit in Continuous-Integration-Umgebungen einbetten. Der Gesamtablauf ist in der Abbildung 5 schematisch dargestellt.

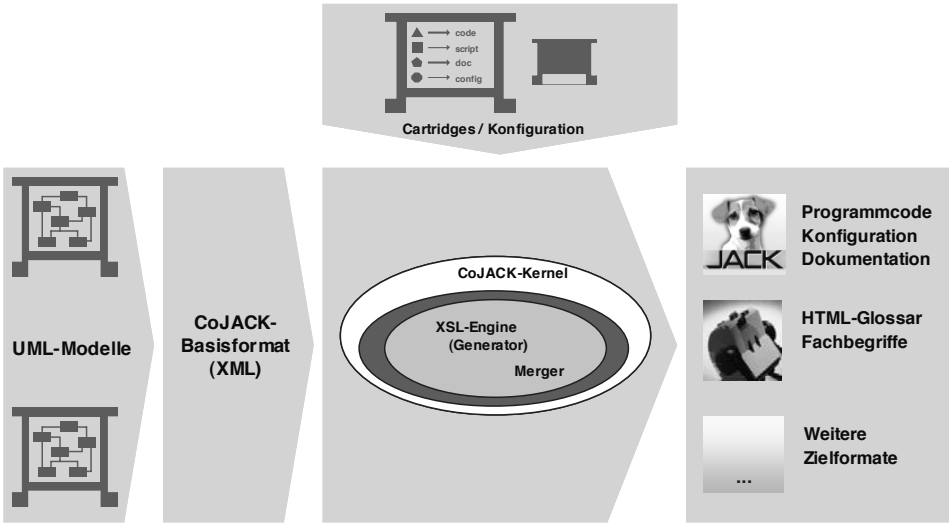

Abbildung 5: Der CoJACK Generierungsablauf

Die Abbildung 6 zeigt beispielhaft die CoJack Transformation eines UML Modells. Das Modell enthält hier nur eine Klasse `BuchExplAusleiher` des Stereotyps `Activity`. Anhand der in der entsprechenden Cartridge definierten Transformationsvorschriften für diesen Stereotyp generiert CoJack folgenden Java Code: Ein Komponenteninterface bestehend aus den Interfaces `BuchExplAusleiher` und `BuchExplAusleiherOperations`, eine abstrakte Basisklasse `BuchExplAusleiherBase`, und eine konkrete Implementierungsklasse `BuchExplAusleiherImpl` sowie eine Factoryklasse `BuchExplAusleiherHome`. Der generierte Code erweitert Basisklassen und Interfaces des JACK-Frameworks, in der Abbildung sind JACK-Klassen und generierter Code durch eine gestrichelte Linie getrennt. Der Implementierungscode für modellierte Methoden wird vom Softwareentwickler in die geschützten Bereiche der Klasse `BuchExplAusleiherImpl` eingefügt.

Eventuell benötigter Remotingcode (z.B. für EJB Session Beans oder CORBA Stubs und Skeletons) wird zur Zeit nicht von CoJACK sondern von platformspezifischen Generatoren erzeugt.

CoJACK unterstützt das Merging von existierendem Implementierungscode und neu generierten Code. Dadurch ist gewährleistet, dass UML Modelle und Implementierungscode parallel weiterentwickelt werden können und zu definierten Zeitpunkten wieder synchronisiert werden können.

Der Merge-prozess basiert auf dem Einsatz von geschützten Bereichen im Quellcode. Innerhalb dieser geschützten Bereiche kann vom Softwareentwickler Implementierungscode eingefügt werden. Das Merging basiert auf folgendem Algorithmus:

Aus einem initialen UML-Modell M_0 entsteht durch n inkrementelle Veränderungen durch den UML Modellierer das Modell M_n. Dieses Modell wird durch die in CoJACK Cartridges hinterlegten Abbildungsvorschriften überführt in eine Quellecodegeneration C_n Die Softwareentwickler fügen ihren Implementierungscode in die geschützten Bereiche dieser Codegeneration ein; so entsteht eine Modifikation C_n' Nach der Erstellung einer neuen Modellversion M_{n+1}. erzeugt der CoJACK Generator eine neue Quellcodegeneration C_{n+1}.

Der Merger ersetzt nun jeden geschützen Codebereich c_{n+1} in C_{n+1}. durch den entsprechenden geschützen Codebereich c_n' in C_n' . So wird sichergesetellt, dass Quellcodemodifikationen aus C_n' in die resultierende Quellcodeversion $C_{n.+1}'$ übernommen werden. Quellcodemodifikationen in geschützten Bereichen c_n' für die es keine Entsprechungen in C_{n+1} gibt (zum Beispiel bei Veränderungen von Methodensignaturen oder Packagenamen), werden vom Merger ignoriert und müssen von den Softwareentwicklern manuell bearbeitet werden.

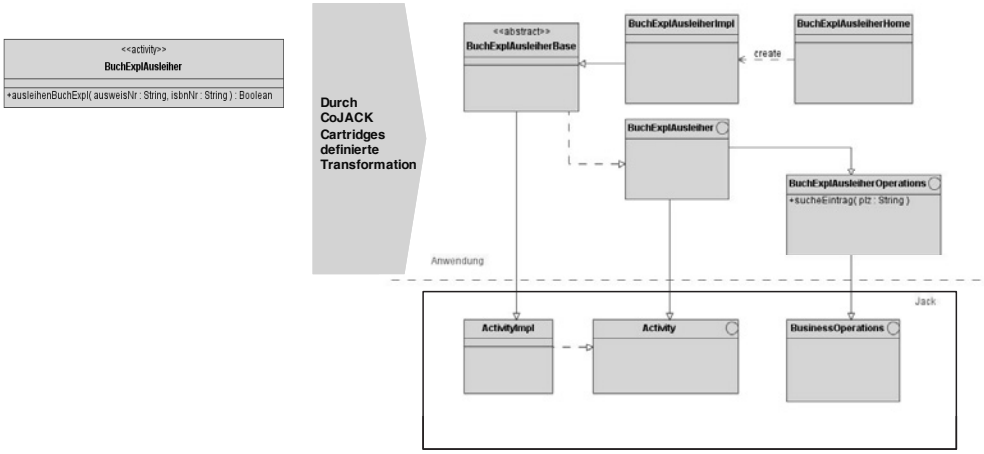

Abbildung 6: CoJACK Transformation eines UML Modells

4. Lessons learnt

Inzwischen sind einige Projekte auf Basis des JACK/CoJACK Ansatzes implementiert und in den Praxiseinsatz überführt worden. Die Projekte haben alle wesentlichen fachlichen, zeitlichen und monetären Ziele erreicht. Nach Abschätzungen und Untersuchungen in den verschiedenen Projekten ergibt sich folgendes Bild.

In unseren Projekten haben wir es bisher nicht geschafft, durch den MDA-Ansatz die Aufwände der Analyse- und Implementierungsphasen gegenüber dem herkömmlichen Vorgehen zu reduzieren [Bu03]. Die Aufwände bewegen sich nach den Abschätzungen auf einem nur leicht reduzierten Niveau. Gleichzeitig wurden aber die Qualität der Ergebnisse von Analyse und Implementierung deutlich verbessert. Die Folgekosten für Betrieb und Wartung werden von den Projekten als signifikant niedriger eingestuft.

Sehr positiv eingeschätzt wird, dass Quellcode jetzt über alle Projekte einheitlich strukturiert ist. Die Einarbeitungszeit von Entwicklern, die in andere Teams wechseln, hat sich drastisch verkürzt. Für neue Mitarbeiter oder Juniorentwickler ergeben sich aus den klar vorgegebenen Implementierungsstellen einheitliche Realisierungs-Strukturen und -Muster.

Alle Projekte bestätigen, dass es tatsächlich gelungen ist, UML Modelle und Quellcode über den gesamten Lebenszyklus der Software synchron zu halten. Einzelne Entwickler bemängeln eine geringe Flexibilität bei „schnellen“ Änderungen, bei denen nicht nur Implementierungscode, sondern auch modellierte Schnittstellen verändert werden. In der Tat verzichtet der JACK/CoJACK Ansatz bewusst auf einen vollständigen Roundtrip-Ansatz , um nur modellgetriebene Änderungen zuzulassen. Hierbei besteht zur Zeit noch das Problem, dass der CoJACK-Mergeprozess keine Veränderungen von Methoden-signaturen oder Packagenamen erkennt und somit implementierter Code manuell verschoben werden muss.

Die Projekte bestätigen, dass die Genauigkeit und Verwertbarkeit der Anforderungs-analyse stark verbessert wurde. Gleichzeitig wird bemängelt, dass die Integration der Mitarbeiter aus den Fachbereichen deutlich erschwert wurde. In der herkömmlichen Methode arbeiteten die Fachbereiche nur mit Worddokumenten, heute müssen Sie mit den Analytikern UML Modelle diskutieren. Da im Vorfeld die fachlichen Anforderungen für die einzelnen Projektteilnehmer nicht hinreichend bekannt waren, wurde zum Teil zu spät mit den entsprechenden Ausbildungsmaßnahmen begonnen. Vergleichbare Probleme gab es auch bei der Einbindung externer Mitarbeiter in den Entwicklungsprozess.

Externe Partner arbeiten zum Teil nach eigenen Vorgehensmodellen. Hier mussten teilweise langwierige Harmonisierungen der Prozesse herbeigeführt werden. Besondere Probleme gab es bei Partnern, die zu Festpreisen arbeiten. Sie empfanden die Reglementierungen des MDA-Ansatzes teilweise als aufwandserhöhend. Hier kolli-dieren unterschiedliche Ansprüche an Kosten und Qualität.