Diplomarbeit im Studiengang Medieninformatik

Generierung einer MVC-Präsentationsschicht für eine J2EE-Applikation unter Einsatz eines MDA-konformen metamodellbasierten Generator-Frameworks und eines opensource Frameworks für Web-Anwendungen

vorgelegt von Joachim Hengge

an der Fachhochschule Stuttgart – Hochschule der Medien

am 29. Oktober 2004

1. Prüfer: Herr Prof. Dr. Edmund Ihler, Hochschule der Medien

2. Prüfer: Herr Dipl.-Inf. (FH) Klaus Banzer, Softlab GmbH

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst habe und keine anderen als die angegebenen Hilfsmittel verwendet habe. Die Arbeit wurde in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde zur Erlangung eines akademischen Grades vorgelegt.

München, den 25. Oktober 2004

_____

Joachim Hengge

Danksagung

An dieser Stelle möchte ich mich bei all jenen bedanken, die durch Ihre fachliche und persönliche Unterstützung zum Gelingen der Diplomarbeit beigetragen haben.

Mein Dank gilt der Softlab GmbH und Michele Siegler, der mir ermöglicht hat, dieses interessante Thema im Rahmen der Diplomarbeit bearbeiten zu können und mich auch in meiner weiteren Entwicklung unterstützt hat. Des weiteren möchte ich mich bei meinem Betreuer Klaus Banzer bedanken, der sich immer Zeit für anstehende Diskussionen genommen hat und durch seine Anregungen sehr zur Verbesserung der Arbeit beigetragen hat.

Vielen Dank an Prof. Dr. Edmund Ihler, der die Arbeit von Seiten der Hochschule betreut hat und durch seine Vorlesungen und verschiedene MDA-Workshops das theoretische Grundwissen und Interesse geschaffen für die Bearbeitung dieser Diplomarbeit geschaffen hat. An dieser Stelle möchte ich mich auch bei Prof. Walter Kriha bedanken, der durch seine hervorragenden praxisnahen Vorlesungen, den Studenten ermöglicht, interessante Themengebiete im Rahmen des Studiums zu bearbeiten.

Abstract

Die vorliegende Diplomarbeit beschäftigt sich mit der Generierung der Präsentationsschicht einer J2EE-Applikation unter Einsatz von Apache Struts. Der vieldiskutierte Ansatz der Object Management Group (OMG) - Model Driven Architecture - wird vorgestellt und unter Zuhilfenahme von open ArchitectureWare, einem metamodellbasierten Generator Framework, wird aufgezeigt, wie sich modellbasierte Entwicklung von der herkömmlichen Softwareentwicklung unterscheidet und welche Vor- und Nachteile sich daraus ergeben können. Im Bezug auf die technische Realisierung der MDA werden zum einen die verschiedenen Vorgehensweisen bei der Entwicklung von Modellen und deren Transformationen nach der MDA Spezifikation der OMG erläutert, und zum anderen werden die eingesetzten Basistechnologien, wie zum Beispiel UML, inklusive der Erweiterungsmöglichkeit über UML-Profile, Meta Object Facility (MOF) und XML Metadata Interchange (XMI), vorgestellt. In diesem Zusammenhang werden auch mögliche pragmatische Interpretationen der MDA diskutiert. Darüber hinaus werden die Technologien, die bei der Entwicklung der Präsentationsschicht im Einsatz sind, Apache Struts und Tiles, näher betrachtet. Ein zentraler Bestandteil der Diplomarbeit besteht aus der Beschreibung der Funktionsweise von open ArchitectureWare, dem opensource Generator Framework, und dem dazugehörigen Vorgehensmodell der generativen Entwicklung. Schlussendlich werden die einzelnen Schritte, die während der Umsetzung vonnöten waren, anhand der Generierung der Präsentationsschicht einer dreischichtigen J2EE-Anwendung zur Verwaltung von Stammdaten, exemplarisch erläutert. Wesentliche Punkte während der Entwicklung, wie zum Beispiel Metamodellierung, plattformunabhängiges Design oder Template-Entwicklung werden am konkreten Projektbeispiel aufgezeigt. Nach erfolgreicher Generierung der Stammdatenanwendung wurde das Ergebnis der Diplomarbeit an einem weiteren Projekt namens BONSAI (Bonus Applikation International) erprobt. Im letzten Teil der Diplomarbeit werden die gewonnenen Erfahrungen bezüglich der aktuellen MDA-Entwicklung zusammengetragen und bewertet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

1 Einleitung

Code-Generierung und automatisierte Erzeugung einzelner System-Komponenten sind keine grundlegende Neuerung. Kaum ein Projekt kommt heute ohne gewisse generative Anteile aus. Häufig wird mithilfe von Templates oder Transformationsskripten die statische Persistenzschicht einer Anwendung anhand der Abhängigkeiten der einzelnen Tabellen einer Datenbank generiert. Auch viele der gängigen CASE-Tools bieten den Entwicklern elementare Schablonen zur Code-Generierung aus dem UML-Modell. Zusätzlich können Technologien wie XDoclet[1], sofern sinnvoll eingesetzt, den Entwicklern sich wiederholende und somit wenig interessante Aufgaben abnehmen.

Modellgetriebene Entwicklung ist im Grunde nichts anderes als die konsequente Fortführung des bisher eingeschlagenen Weges auf einem höheren Abstraktionslevel. Spätestens seit der Veröffentlichung des MDA Guide [MDA00] durch die Object Management Group (OMG) im Jahre 2001, wird das Konzept metamodellbasierter Architekturen kontrovers auf verschiedensten Konferenzen und in einschlägigen Fachmagazinen diskutiert. Mit UML (Unified Modelling Language) steht im Portfolio der OMG eine ausgereifte Sprache zur Verfügung, mit der sich nicht nur statische Zusammenhänge, sondern auch Ablaufbeschreibungen von Applikationen modellieren lassen. In einem UML-Modell hat man die Möglichkeit, verglichen zum Beispiel mit einer Datenbank, weitaus mehr Informationen abzubilden. Aus der Diskussion und aufgrund der Tatsache, dass die Spezifikation der MDA nach OMG, Spielraum für Interpretationen lässt, entstanden im Laufe der Zeit unterschiedliche, sowohl kommerzielle als auch frei erhältliche Produkte, die das Konzept der modellbasierten Entwicklung in verschiedensten Ausprägungen umsetzen. Obwohl sich die einzelnen Ansätze teilweise grundlegend unterscheiden, so haben doch alle zum Ziel, dem Entwickler wiederkehrende und fehleranfällige Aufgaben abzunehmen und automatisiert auszuführen. Mittlerweile stehem dem Anwender sowohl ausgereifte kommerzielle Produkte wie Compuwares OptimalJ[2] oder ArcStyler von Interactive Objects Software[3], als auch mächtige opensource Lösungen, wie zum Beispiel AndroMDA[4] oder open ArchitectureWare zur Verfügung. Der Trend geht in die Richtung der Interpretation der MDA-Spezifikation für den Praxiseinsatz im Projekt. MDSD (Model Driven Software Development) und MDSE (Model Driven Software Engineering) sind zwei der verschiedenen Strömungen, die sich im Laufe der Diskussion entwickelt haben. Die vorliegende Diplomarbeit zur Generierung einer Präsentationsschicht nutzt anhand des opensource Frameworks open ArchitectureWare eine MDA-Interpretation und generiert aus dem plattformunabhängigen Design (PIM) ohne den Zwischenschritt der Modelltransformation den Quellcode der Anwendung.

1 XDoclet - http://xdoclet.sourceforge.net (Stand: 28.09.2004)
2 OptimalJ – http://www.optimalj.com (Stand: 28.09.2004)
3 Interactive Objects Software – http://www.io-software.com (Stand: 28.09.2004)
4 AndroMDA – http://www.andromda.org (Stand: 28.09.2004)

2 Softlab GmbH

Die neu formierte Softlab Gruppe besteht aus den drei eigenständigen Unternehmen Softlab, Nexolab und Axentiv. Das Beratungshaus Axentiv gehört seit Sommer 2004 zur Softlab Gruppe und ergänzt das Angebot von Softlab und Nexolab durch geballte SAP-Kompetenz. Zusätzlich ist Softlab zu 50% an F.A.S.T. beteiligt.

Softlab ist eine hundertprozentige BMW-Group Company mit zur Zeit ca. 1200 Mitarbeitern am Hauptsitz in München und den weiteren Standorten in Deutschland. Zusätzlich zu den Standorten in Deutschland existieren weitere Tochterunternehmen in Österreich, Schweiz und England.

Fokus ist die Entwicklung kundenspezifischer Individual-Lösungen auf Basis neuer Technologien. Im Vordergrund steht die Betreuung von Kunden aus Industrie, Automotive, Finanzdienstleister und Handel während des gesamten Prozesses von der Konzeption über die Integration bis zur Unterstützung im laufenden Betrieb. Das Leistungsangebot umfasst dabei die klassischen IT-Schwerpunkte: Beratung – Implementierung – Betrieb (plan – build – run), d.h. Betreuung der Kunden von der Konzeption über die Systemintegration bis zur Übernahme eines IT-Projektes.

Eckdaten:

- Geschäftssegmente:
  - IT-Consulting
  - Customer Relationship Management (CRM)
  - Supply Chain Management (SCM)
  - Enterprise Application Services (EAS)
  - Cross functional Services (XFS)

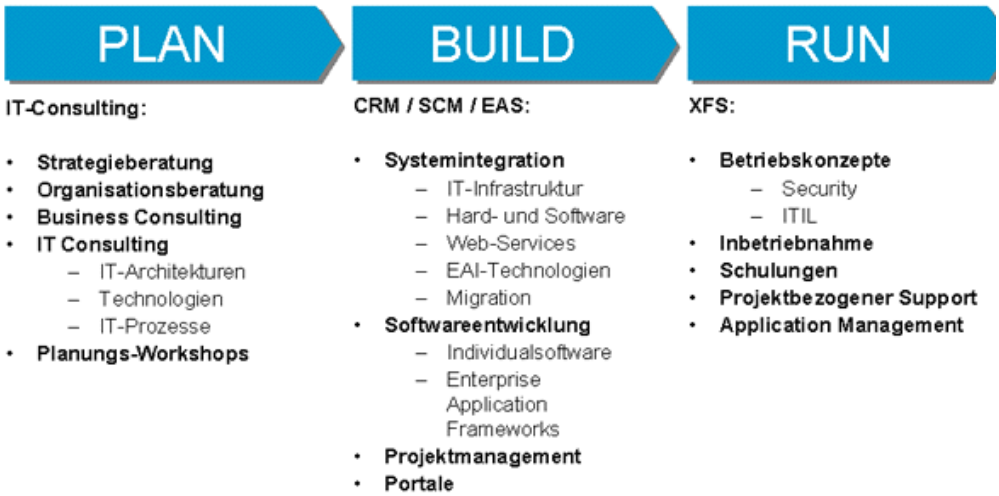


Abbildung 2-1: Geschäftssegmente der Softlab GmbH

- Standort München: ca. 800 Mitarbeiter
- 1971: Gründung
- 1992: BMW Group Company

Anschrift

Softlab GmbH
Zamdorfer Strasse 120
81677 München

Internet: http://www.softlab.de
Email: info@softlab.de

3 Diplomarbeit

3.1 Basis

Eine bereits abgeschlossene Arbeit [Mär01] mit dem Thema „Erweiterung eines metamodell-basierten Generator-Frameworks zur Erzeugung von Quellcode für eine mehrschichtige Client/Server Architektur am Beispiel von J2EE“ bildet den Ausgangspunkt für die vorliegende Arbeit. Einerseits baut die Diplomarbeit auf [Mär01] auf, um, nach Abschluss der beiden Arbeiten, eine lauffähige dreischichtige Applikation generieren zu können, andererseits soll die Generierung der Präsentationsschicht vom Backend entkoppelt sein, um keine Abhängigkeiten und Einschränkungen bei der Generierung berücksichtigen zu müssen.

Als Grundlage während der Entwicklung der Diplomarbeit dient das Projekt `International Vehicle System - Reengineering (IVS-R)`, das ein neues web-basiertes Online Bestellsystem für die BMW Gruppe zur Verfügung stellt. Das System stellt Funktionalität bereit, die von BMW Händlern dazu benutzt wird, Fahrzeuge für ihre Kunden bei der BMW AG zu bestellen. Alle Bestellungen werden online ausgeführt und erhalten eine direkte Bestätigung über die Machbarkeit und das Produktionsdatum aus der verantwortlichen Fabrik. Die Anwendung wurde für die Java 2 Plattform Enterprise Edition (J2EE) realisiert, das Backend mit Enterprise Java Bean Komponenten, die auf BEA Weblogic Application Server angewendet werden. Anhand einer Teilfunktionalität von IVS-R - Pflege der Stammdaten von BMW Händlern, die auf etwa zehn bis fünfzehn Datenbanktabellen verteilt sind (vgl. [Mär01]) – werden in der vorliegenden Arbeit die einzelnen Schritte zur Generierung der Präsentationsschicht aufgezeigt.

Aufgrund der Vorleistung wurde während der Entwicklung, die bereits in der abgeschlossenen Diplomarbeit bestehende Systemlandschaft weiter verwendet und auf die Bedürfnisse der Generierung einer Präsentationsschicht angepasst.

3.2 Ziele

Ausgehend von der oben beschriebenen Diplomarbeit zur Generierung eines Backends für eine J2EE-Applikation, war der Fokus der Aufgabenstellung bei der jetzigen Diplomarbeit auf das Frontend gerichtet. Das User Interface soll anhand eines oder mehrerer UML Interaktionsdiagramme, die den dynamischen Ablauf der Applikation und das Zusammenspiel von Anzeige und Aktion definieren, modelliert werden. Um den Mehrwert, der durch die modellgetriebene Entwicklung entsteht, nutzen zu können, soll keine an die Generierung angepasste Modellierung erforderlich sein. Vielmehr wird durch Anreichern und Verfeinern bereits bestehender Diagramme ein plattformunabhängiges Modell der Applikation erstellt. In Kombination mit der Diplomarbeit zur Generierung der Persistenzschicht [Mär01] soll es nach Abschluss der Diplomarbeit möglich sein, eine lauffähige dreischichtige J2EE-Applikation generieren zu können, die anschließend manuell um fachliche Logikteile erweitert werden kann. Durch eine lose Kopplung von Front- und Backend soll es aber trotzdem möglich sein, die einzelnen Komponenten unabhängig voneinander generieren zu können.

4 Model Driven Architecture (MDA)

4.1 MDA nach OMG

4.1.1. Einführung

Die Object Management Group (OMG) [OMG00] ist eine nicht-kommerzielle Organisation, die sich dem Erstellen und Verwalten von Computer Industrie-Standards für offene Unternehmensapplikationen verschrieben hat. Zu den bekanntesten Spezifikationen der OMG zählen unter anderem CORBA[1], OMA[2] und UML [UML00].

Im Jahr 2001 hat die OMG ein neues Entwicklungsmodell, die Model Driven Architecture (MDA), eingeführt. MDA ist, im Vergleich zu CORBA und OMA, kein Framework für die Implementierung von verteilten Systemen, sondern der Ansatz, Software aus UML-Modellen zu generieren. Im Juni 2003 wurde die erste MDA Spezifikation in Form des MDA Guides [MDA00] veröffentlicht.

Grundsätzlich verfolgt die Model Driven Architecture bekannte und bewährte Konzepte, wie die Trennung der Spezifikation eines Systems von den Details der plattformspezifischen Umsetzung. Die MDA bietet einen Ansatz und die dazugehörigen Hilfsmittel, um ein System unabhängig von der zu unterstützenden Plattform zu spezifizieren und diese Systemspezifikation an die jeweils gewünschte Plattform anzupassen.

Die drei Hauptziele der MDA, Übertragbarkeit, Interoperabilität und Wiederverwendung, sollen hierdurch erreicht werden. Darüber hinaus soll durch das Anheben des Abstraktionslevels und der weitgehenden Generierung eine deutliche Steigerung der Entwicklungsgeschwindigkeit und dadurch eine Einsparung von Zeit und Kosten erreicht werden. Außerdem wird durch die Einhaltung von Design-Richtlinien im Modell und der Generierung von Code, die Anzahl der Fehler, im Gegensatz zu manuell erstelltem Code, reduziert und somit die Code-Qualität erheblich optimiert.

4.1.2. Grundprinzip

Die untenstehende Abbildung stellt das Grundprinzip der Model Driven Architecture dar: Software wird in einem oder mehreren Modellen abstrakt spezifiziert und durch sukzessives Verfeinern und Transformieren schlussendlich zu einem ausführbaren Softwaresystem auf einer bestimmten Plattform zugeschnitten. Ausgangspunkt ist das sogenannte Computation Independent Model (CIM), das per Modell-Modell Transformation in ein plattformunabhängiges Modell (Platform Independent Model – PIM) überführt wird. Das PIM enthält keine technischen Details über die Zielplattform oder die technische Umsetzung des Systems. Aus dem PIM können wiederum per Modell-Modell Transformation verschiedene plattformspezifische Modelle (Platform Specific Model – PSM) generiert werden, die auf den nichttechnischen, fachlichen Anforderungen der Modelle aus den höheren Ebenen basieren. Das PSM kann je nach Bedarf noch um technische Modellierungsdetails erweitert werden, um dann in einer Model-Code Transformation den Sourcecode für die im PSM definierte

1 Common Object Request Broker Architecture – http://www.omg.org/corba (Stand: 25.08.2004)
2 Object Management Architecture™ - http://www.omg.org/oma (Stand: 25.08.2004)

Technologie zu generieren.

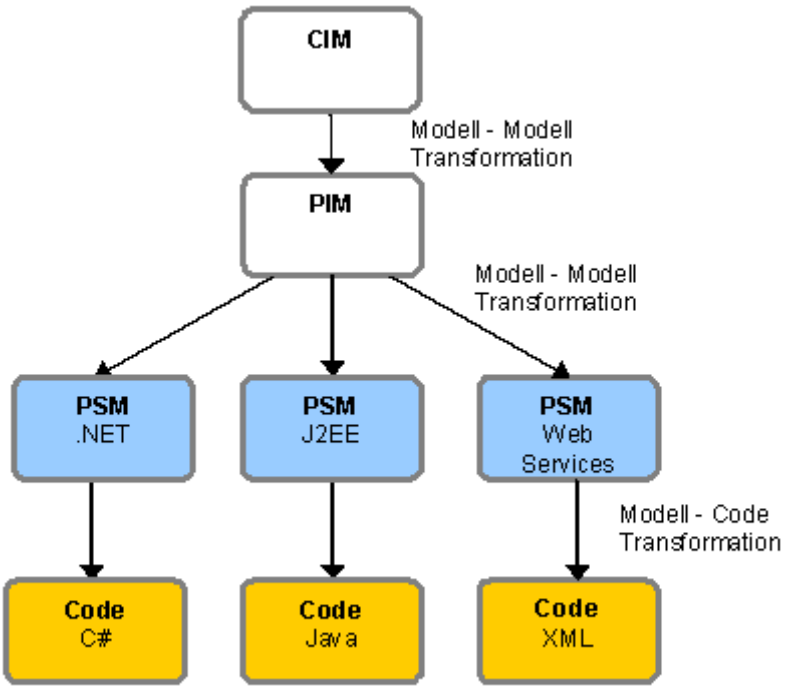


Abbildung 4-1 - MDA nach OMG

Darüber hinaus spielt der Begriff der Abstraktion eine bedeutende Rolle. Man spricht bei der Modellierung von verschiedenen Stufen der Abstraktion (levels of abstraction), die jeweils zu einem gewissen Grad die Komplexität des Systems verstecken. Die Verfechter der Model Driven Architecture argumentieren, dass die MDA durch die mehrstufige Verfeinerung der Modelle, das Level der Abstraktion anhebt und somit die Diskrepanz, die oft zwischen fachlichen und technischen Abteilungen besteht, verringert.

Die einzelnen Bestandteile und Modellarten der MDA werden im Anschluss im Kapitel 4.1.3 noch detaillierter behandelt.

4.1.3. Basiskonzepte der MDA

Im Folgenden werden die einzelnen Bestandteile und Konzepte der Model Driven Architecture laut [MDA00] beschrieben:

4.1.3.1. Modell

Ein Modell ist eine abstrakte Beschreibung oder Spezifikation eines Systems, das einem bestimmten Zweck dient. Es besteht oft aus einer Kombination von Zeichnung und Text. Modelle in der MDA werden gewöhnlich in UML erstellt, der Text kann auch in einer natürlichen Sprache verfasst sein.

4.1.3.2. Modellgetriebene Entwicklung

Die Bedeutung von Modellen erhöht sich bei der Software-Entwicklung nach der MDA signifikant. Modellgetriebene Entwicklung bedeutet, dass Änderungen am System im Modell erfasst werden und Modelle so Verständnis, Design, Erstellung, Auslieferung, Verwaltung und

Abänderung des Systems bestimmen.

4.1.3.3. Plattform

Eine Plattform wird in der MDA-Spezifikation folgendermaßen definiert:
Eine Plattform ist eine Menge von Subsystemen und Technologien, die eine kohärente Ansammlung von Funktionalität, in Form von Schnittstellen und bestimmten Anwendungsmustern bereitgestellt. Jede Applikation, die von der Plattform unterstützt wird, kann diese Funktionalität nutzen, ohne sich um deren Implementierung kümmern zu müssen. [MDA00, Kapitel 2.2.7]

Beispiele für Plattformen sind unter anderem J2EE, CORBA und .NET.

4.1.3.4. Computation Independent Model (CIM)

Die Vorgaben für ein System und die Umgebung, in der das System genutzt wird, werden im Computation Independent Model beschrieben [MDA00, Kapitel 3.1], das teilweise auch Domänen- oder Geschäftsmodell genannt wird. Diese Kapselung in einem Modell, hilft nicht nur beim Verständnis eines Problems, sondern bietet auch ein gemeinsames Vokabular, das in verwandten Problemfällen oder bei der Spezifikationsentwicklung im Team sehr hilfreich sein kann. Die Details der Struktur und der Implementierung des Systems sind nicht sichtbar oder noch gar nicht definiert, im PIM und PSM sollten die Bestandteile und Vorgaben, die im CIM modelliert sind, aber wieder auffindbar sein. Das CIM ist somit ein wichtiger Bestandteil im Bemühen die Lücke zwischen den fachlichen Experten der Domäne auf der einen Seite und den Spezialisten für Design und Entwicklung auf der anderen Seite, zu schließen.

4.1.3.5. Platform Independent Model (PIM)

Ein plattformunabhängiges Modell beschreibt die Funktion eines Systems, ohne auf die technischen Details einer bestimmten Plattform einzugehen und zeigt so den Teil der kompletten Spezifikation, der sich beim Wechsel auf eine andere Plattform nicht ändert. Je nach Verwendungszweck kann das PIM unterschiedliche Level an Plattformunabhängigkeit besitzen, der dem Einsatz von verschiedenen Plattformen ähnlichen Typs, angepasst wird. Ein PIM wird entweder in einer universellen Modellierungssprache angefertigt oder einer spezifischen Sprache für das Anwendungsgebiet, in dem das System genutzt wird.
Um eine gewisse Plattformunabhängigkeit zu erreichen, werden Modelle oftmals auch bezüglich einer technologieunabhängigen virtuellen Maschine (VM) erstellt.

4.1.3.6. Platform Specific Model (PSM)

Das PSM ist die plattformspezifische Sicht auf ein System. Es kombiniert die plattformunabhängige Sichtweise des Systems aus dem PIM mit einem zusätzlichen Fokus auf die Details der Verwendung einer speziellen Plattform. Im PSM werden somit plattformspezifische Details, wie zum Beispiel der Einsatz von Struts für eine J2EE-Plattform oder ASP.NET für eine .Net-Plattform, definiert.

4.1.3.7. Modell-Transformationen

Modell-Transformation sind der Prozess der Überführung eines Modells in ein anderes Modell des gleichen Systems [MDA00, Kap 2.2.15]

Die untenstehende Abbildung verdeutlicht das MDA Pattern, durch das ein plattformunabhängiges Modell in ein plattformspezifisches Modell konvertiert wird.

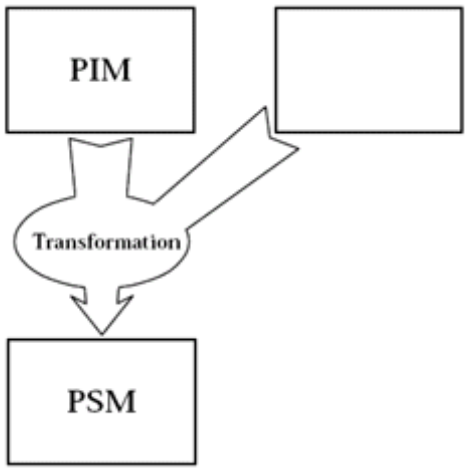


Abbildung 4-2 - Modell Transformation

Das plattformunabhängige Modell und zusätzliche Informationen (z.B. Markierungen, Mapping-Informationen) werden in der Transformation kombiniert, um daraus ein plattformspezifisches Modell zu generieren. Die Abbildung stellt den Transformationsprozess generisch dar, d.h. es gibt viele Wege, wie eine solche Transformation umgesetzt werden kann (siehe Kapitel 4.1.4).

4.1.4. Anwendung der MDA

Nach der Beschreibung der einzelnen Bestandteile und Konzepte der Model Driven Architecture, wird nun auf die Anwendung und Funktionsweise der MDA eingegangen.

4.1.4.1. Mapping

Ein MDA Mapping beschreibt detailliert, wie ein PIM in ein PSM für bestimmte Plattform überführt wird. In der MDA Spezifikation [MDA00] sind verschiedene Möglichkeiten des Mappings definiert, die auch kombiniert werden können. Im Anschluss werden die Wichtigsten vorgestellt:

Modelltyp Mapping

Beim Modelltyp Mapping werden bestimmte Modelltypen im PIM, gekennzeichnet durch die Modellierungs-Sprache (z.B. Stereotypen in UML), auf bestimmte Typen im PSM gemappt. Ein spezielles Beispiel für ein Modelltyp Mapping ist das sogenannte Metamodell Mapping. Hierbei werden die Stereotypen der Modellelemente im PIM und PSM durch Klassen im Metamodell repräsentiert. Somit kann das Verhalten und die Zusammenhänge der verschiedenen Modelltypen beschrieben werden.

Modellinstanz Mapping

Eine andere Herangehensweise beschreibt das Modellinstanz Mapping. Bestimmte Modellelemente, die für die Transformation in Frage kommen, werden mit Markierungen (engl. marks) versehen. Die Markierungen sind plattformspezifisch, d.h. der Architekt versieht für jede Ziel-Plattform das PIM mit speziellen Markierungen, die dann für die jeweilige Transformation in das PSM herangezogen werden.

Templates

Templates bilden eine Art Schablone für die Transformation der einzelnen Elemente des PIM in das PSM. Sie gleichen einem Entwurfsmuster, können aber schon sehr viel mehr technische Information enthalten. Beispiel: Ein mit „Entity“ markiertes Modellelement im PIM wird während des Transformationsschrittes mithilfe eines Templates in Local-, Remote Interfaces und EntityBean für das PSM einer J2EE-Plattform expandiert.

4.1.4.2. Methoden der Modelltransformation

Wie in Kapitel 4.1.3.7 beschrieben, bilden (Modell-)Transformationen einen wesentlichen Bestandteil im Konzept der Model Driven Architecture. Die MDA Spezifikation [MDA00] definiert keinen allgemeingültigen Weg, Transformationen durchzuführen, sondern zeigt verschiedene Möglichkeiten auf, von denen die Wesentlichen im Folgenden vorgestellt werden. Die verschiedenen Ansätze sind nicht klar voneinander abgrenzbar, es ist auch durchaus eine Kombination der Ansätze möglich.

Markierungen

Abbildung 4-3 beschreibt einen möglichen Weg, ein PIM in ein PSM zu überführen. Für eine

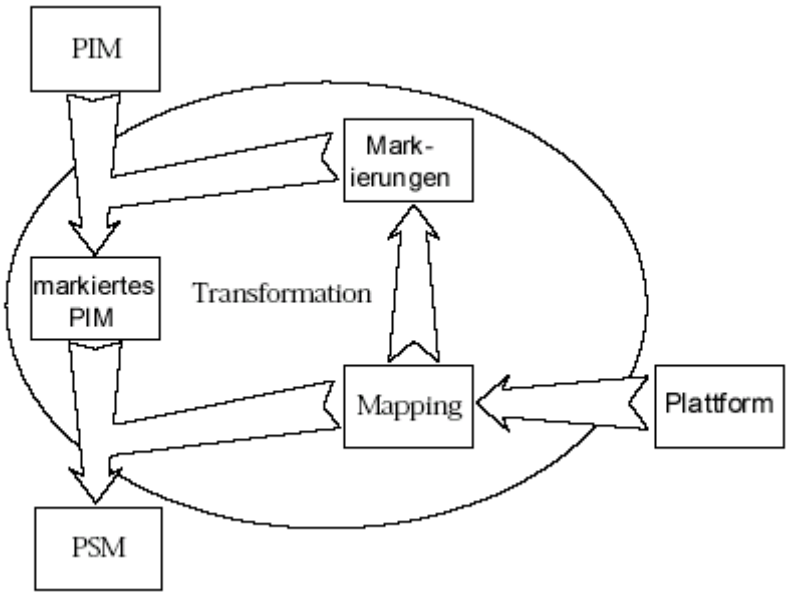


Abbildung 4-3 - Modelltransformation mithilfe von Markierungen

ausgewählte Plattform wird ein Mapping definiert, um die Modelltransformation durchzuführen. Bestandteil dieses Mappings ist eine Menge von Markierungen, die wie oben beschrieben, diejenigen Modellelemente, die für die Transformation in Frage kommen, markieren und so die Transformation steuern.

Metamodell Transformation

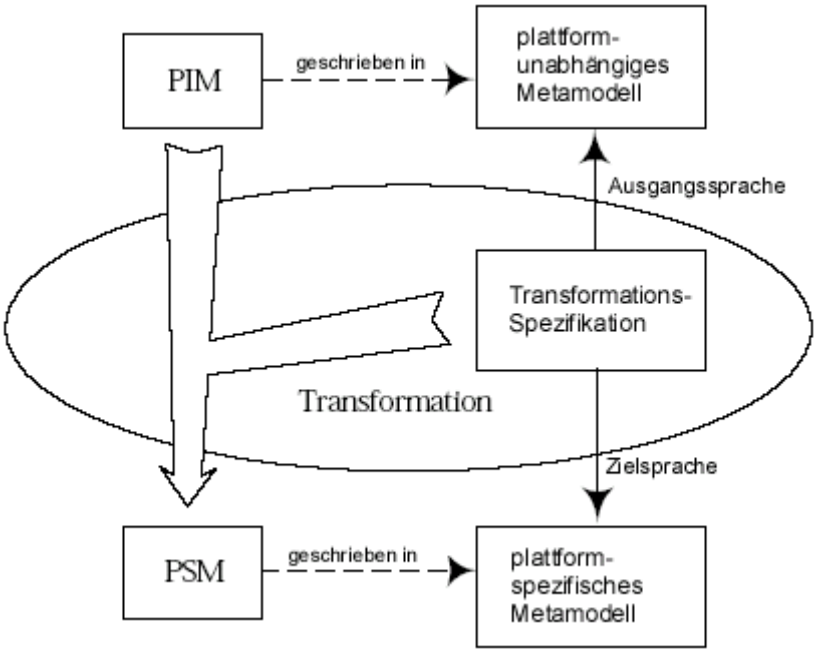


Abbildung 4-4 - Modelltransformation mithilfe von Metamodellen

In Abbildung 4-4 wird die Modelltransformation anhand von Metamodellen aufgezeigt. Ein Metamodell beschreibt das Verhalten und die Beziehungen eines Modells. Um eine vollständig automatisierte Generierung zu erreichen, sollten beide Metamodelle auf einem gemeinsamen Meta-MetaModell (MOF) basieren, das die Metamodelle abstrakt beschreibt. Zusätzlich wird eine Plattform ausgewählt und eine Transformationsspezifikation erstellt. Diese Transformationsspezifikation steuert die Abbildung des PIMs auf das PSM anhand von Mapping-Regeln zwischen den beiden Metamodellen. Ein MDA-Mapping beschreibt detailliert, wie ein PIM in ein PSM für eine bestimmte Plattform überführt wird. In der MDA Spezifikation [MDA00] sind verschiedene Möglichkeiten des Mappings definiert, die auch kombiniert werden können.

4.1.4.3. Queries Views Transformations (QVT)

Wie aus den oben beschriebenen Transformationsmöglichkeiten deutlich wird, fehlt der MDA noch eine eindeutige standardisierte Beschreibungssprache für Transformationen. Die Sprache QVT [QVT00] wird von der OMG als neuer Standard für Modelltransformationen propagiert, liegt jedoch zur Erstellungszeit dieser Diplomarbeit nur als „Request For Proposal (RFP)“ vor.

QVT setzt sich laut [QVT00] aus den folgenden Bestandteilen zusammen:

Queries
haben ein Modell als Vorgabe und wählen spezifische Elemente dieses Modells aus.
Views
Sind von anderen Modellen abgeleitet

Transformations
sind entweder Spezifikationen oder Implementierungen die ein Modell als Vorgabe haben und dieses auf ein anderes Modell beziehen oder jeweils ein neues Modell erstellen.

4.2 MDA Technologieüberblick

4.2.1.1. Unified Modeling Language (UML)

Die Unified Modeling Language [UML00] ist eine grafische Modellierungssprache zur Visualisierung, Spezifizierung, zum Bau und zur Dokumentation der Artefakte eines Softwaresystems. Nicht nur die konkrete Umsetzung eines Systems, wie Ausdrücke in Programmiersprachen, Datenbankschemata und Softwarekomponenten, sondern auch konzeptionelle Entwürfe, wie zum Beispiel Geschäftsprozesse und Systemfunktionen und -begrenzungen, lassen sich abbilden. Die grafische Modellierung in UML wird in Diagrammen festgehalten, von denen verschiedene Ausprägungen (z.B. statische, Interaktions-, Anwendungsfall-Diagramme) zur Verfügung stehen.

Mit der UML Spezifikation Version 1.4 lässt sich jedoch nur sehr bedingt das Verhalten und die Logik einer Applikation grafisch abbilden, dies wird in der neuesten UML Spezifikation (Version 2.0) adressiert, die zum Zeitpunkt der Erstellung der Diplomarbeit aber noch nicht endgültig verabschiedet war.

Im Rahmen dieser Diplomarbeit wird aufgrund der Komplexität, des Umfangs und der hohen Verbreitung nicht näher auf die einzelnen Bestandteile der UML eingegangen.

UML Profile
UML-Profile sind der Standardmechanismus zur Erweiterung des Sprachumfangs der UML. Durch die Entwicklung eines UML-Profils kann man sich gegebenen Plattformen oder Domänen anpassen.

Aus folgenden Sprachkonzepten kann sich ein UML-Profil zusammensetzen:

- Basis UML-Konstrukte
  Verwendung bestimmter Klassen, Beziehungen und Attribute aus dem UML Sprachumfang.
- Stereotypen
  Stereotypen dienen der Erweiterung vorhandener UML Modellelemente, genauer, des Metamodells. Sie basieren auf bestehenden Typen oder Klassen im Metamodell, erweitern diese, verändern aber nicht die Struktur der vordefinierten Klassen und Typen. Ein Stereotyp beeinflusst die Semantik eines Modellelements, auf das es angewendet wird und definiert so mögliche Verwendungszusammenhänge.
- Tagged Values (Eigenschaftswerte)
  Tagged Values sind benutzerdefinierte Schlüssel/Wert Paare, die die Semantik einzelner Modellelemente um spezielle Eigenschaften erweitern. Im Unterschied zu den Stereotypen wird das Metamodell nicht um ein neues Element erweitert, sondern einzelne Ausprägungen bestehender Modellelemente (z.B. eine Operation) um bestimmte Eigenschaften erweitert.

- Constraints (Einschränkungen)
  Unter einem Constraint versteht man einen Ausdruck, der mögliche Inhalte, Zustände oder die Semantik eines Modellelements einschränkt und der immer erfüllt sein muss. Ein Constraint kann unter anderem mithilfe der Object Constraint Language (OCL) ausgerückt werden. OCL ist eine Sprache zur Beschreibung von Zusicherungen, Vor- und Nachbedingungen und Navigation innerhalb von UML-Modellen.

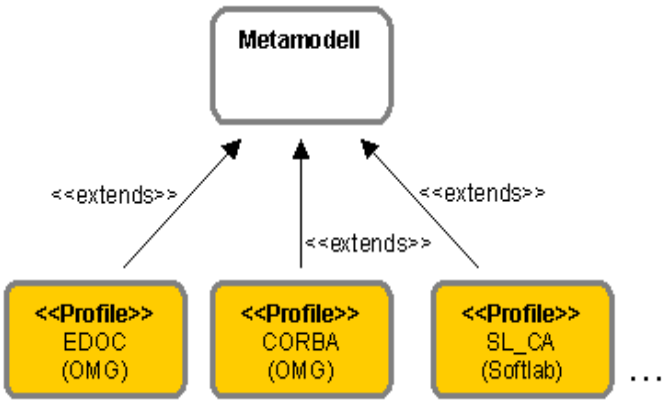


Abbildung 4-5 - UML Profile

Wie in Abbildung 4-5 deutlicht wird, gibt es verschiedene von der OMG vordefinierte UML Profile. Das UML Profil für Enterprise Distributed Object Computing (EDOC), zum Beispiel, legt fest, wie Standard UML-Modelle für Verteilte Anwendungen erweitert werden können.

4.2.1.2. Meta Object Facility (MOF)

Wie oben beschrieben, wird die UML dazu verwendet, Modelle in der Model Driven Architecture (PIM, PSM) zu notieren. Syntax und Semantik der UML werden im UML-Metamodell festgelegt, das, vergleichbar mit einer Document Type Definition (DTD) in XML, Beziehungen und Verhalten der UML-Modellierungselemente auf einer abstrakteren Ebene definiert. Sämtliche Modellarten der UML, wie z.B. Klassen-, Aktivitäts- oder Zustandsdiagramme, sind dementsprechend im UML-Metamodell beschreiben.

Die OMG stellt mit der Meta Object Facility (MOF) eine abstrakte Sprache zur Verfügung, mit der Metamodelle im allgemeinen (u.a. auch das UML-Metamodell) und Ihre Abhängigkeiten untereinander definiert werden können (Meta-Metamodell). Metamodelle, die mithilfe der MOF beschrieben sind (z.B. das UML-Metamodell), sind Instanzen der MOF. MOF definiert die Syntax und somit die Struktur von Metamodellen. Folglich können alle Metamodelle, die MOF instanziieren, auf die gleiche Art verarbeitet werden, da sie auf demselben Basis-Metamodell aufbauen und dadurch die gleichen Konstrukte verwenden.

MOF spielt eine zentrale Rolle bezüglich der MDA. Durch den Einsatz von Metamodellen hat man die Möglichkeit, Informationen über die Struktur eines Modells zur Laufzeit abzufragen und auf diese Weise über das Modell zu navigieren, um anschließend die Modell-Transformation oder die Transformation in Programm-Code zu beeinflussen.

Die 4-schichtige Architektur der MOF:

Mit dem 4-Schichtenmodell der OMG wird eine Modellhierarchie beschrieben, welche sukzessive die verschiedenen Modellebenen definiert. Der UML-Anwender befindet sich üblicherweise auf der Modell-Schicht M1 und begnügt sich mit dem Erstellen von UML-Modellen.

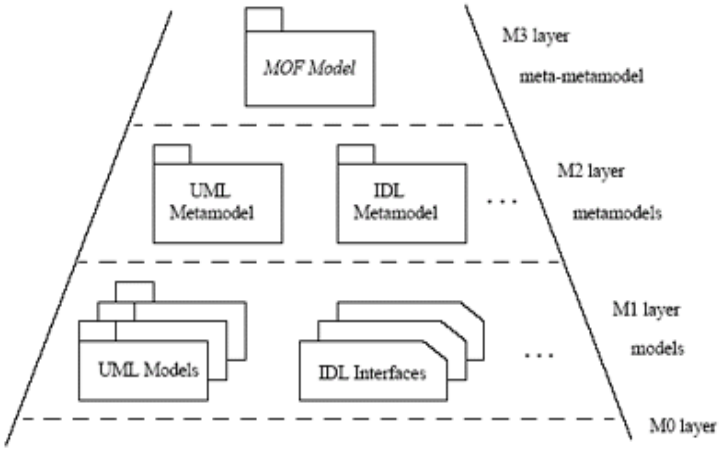


Abbildung 4-6 - Die 4 Schichten der MOF

Die 4-Schichtenarchitektur der MOF setzt sich aus folgenden Elementen zusammen:

M0 Objekte:

Objekte sind Instanzen des Modells und beschreiben die Ausprägungen einer bestimmten Domäne.

M1 Modell:

Modelle sind Instanzen von Metamodellen. Zu ihnen gehören die bekannten UML-Modelle, wie z. B. Klassendiagramme.

M2 UML-Metamodell:

Metamodelle sind Instanzen von Meta-Metamodellen und definieren die Sprachelemente für Modelle. Auf dieser Ebene befinden sich z.B. die Sprachdefinitionen für UML und IDL1. Darüber hinaus befinden sich hier die sprachspezifischen UML-Profile, da sie Erweiterungen bezüglich der Metamodelle darstellen.

M3 Meta-Metamodell:

Auf der Ebene M3 ist die gemeinsame Basis für alle weiteren Metamodelle angesiedelt. Durch die MOF werden auf dieser Schicht die Sprachelemente zur Spezifikation von Metamodellen spezifiziert (z. B. ModelElement, Namespace, Classifier, etc.).

1 Interface DefinitionLanguage - http://www.omg.org/docs/formal/02-06-07.pdf

Abbildung 4-7 verdeutlicht das Schichtenmodell der MOF nochmals an einem Beispiel.

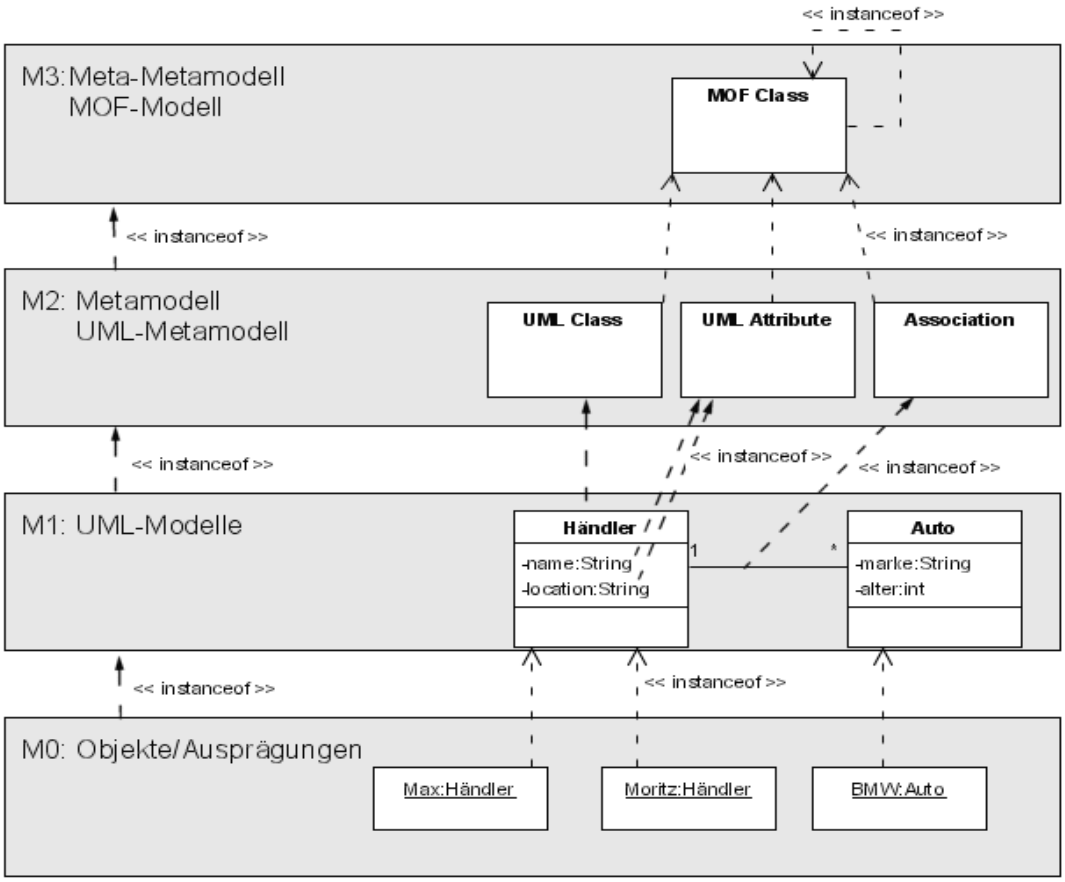


Abbildung 4-7 - Beispiel für das Schichtenmodell der MOF

4.2.1.3. XML Metadata Interchange (XMI)

Die Zielsetzung von XML Metadata Interchange [XMI00] ist, einen einfachen Austausch von Metadaten zwischen verschiedenen Modellierungswerkzeugen (basierend auf UML) und Metadaten-Repositories (basierend auf MOF) in verteilten heterogenen Umgebungen zu ermöglichen. XMI ist eine Spezifikation der OMG, die auf drei grundlegenden Industrie-Standards aufbaut:

- XML – eXtensible Markup Language (W3C[1])
- UML – Unified Modeling Language (OMG)
- MOF – Meta Object Facility (OMG)

1 World Wide Web Consortium - http://www.w3.org (Stand: 02.09.2004)

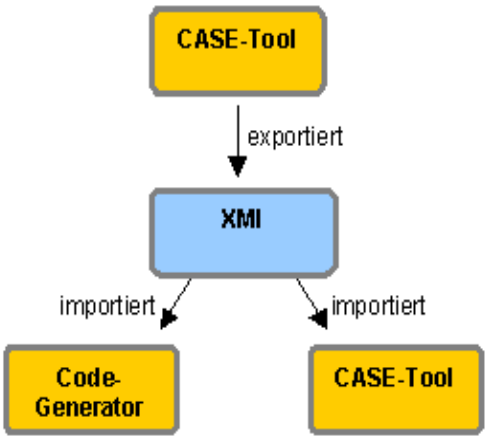


Abbildung 4-8 - TemplateManager der GenFW Utils

Durch die Verschmelzung bewährter Technologien der OMG und des W3C bezüglich Metadaten und Modellierung, bietet XMI die Funktionalität der textuellen Beschreibung, der mit UML graphisch formulierten Modelle. XMI ist aber nicht nur als Modellaustauschformat im Einsatz, sondern bietet darüber hinaus auch in den Bereichen der Modellvalidierung, der toolunabhängigen Code-Generierung und der eindeutigen textuellen Beschreibung von Metamodellen äusserst hilfreiche Dienste (vgl. Abbildung 4-8).

4.3 MDA light – pragmatischer Ansatz

4.3.1. Einführung

Seit der Einführung der Model Driven Architecture vor wenigen Jahren und der Veröffentlichung des MDA Guides im Juni 2003, hat die modellgetriebene Softwareentwicklung einen enormen Aufschwung erlebt. Viele Firmen haben das Potential, das in der generativen Entwicklung und MDA liegt, erkannt. Aufgrund der abstrakten und teilweise nicht sehr konkreten Definition der MDA, dreht sich die Diskussion in der Praxis häufig um pragmatische Ansätze und Interpretationen der Model Driven Architecture: Aus einem plattformunabhängigen Modell wird unmittelbar Sourcecode generiert, ohne den Zwischenschritt der Modelltransformation zum PSM durchzuführen. Der generierte Sourcecode enthält alle schematischen Elemente der Architektur und wird vom Entwickler um individuellen fachlichen Code ergänzt. Oft werden pragmatische Ansätze wie dieser, als „MDA light“ bezeichnet. Projekte wie AndroMDA[1] und das in der Diplomarbeit verwendete open ArchitectureWare [oAW00] verfolgen diesen pragmatischen Ansatz mit zunehmendem Erfolg und steigender Akzeptanz. Oft werden auch die Technologievorgaben der OMG für MDA interpretiert. So werden zum Teil Metamodelle nicht in UML spezifiziert, sondern deklarativ, z.B. mit der Templatesprache Velocity[2] der Apache Foundation, oder domänenspezifischen, von der UML abgeleiteten, Modellierungssprachen. Die Ansätze, die

1 AndroMDA - http://www.andromda.org (Stand: 07.09.04)
2 Velocity Template Engine - http://jakarta.apache.org/velocity (Stand:08.09.04)

von der Technologie und der Umsetzung der Generierung einen anderen Weg gehen, als von der OMG vorgegeben, werden als Model Driven Software Development (MDSD) bezeichnet und erfreuen sich zunehmend großer Beliebtheit.

Überwiegend an die MDA-Spezifikation der Modelltransformation halten sich die kommerziellen Anbieter von integrierten MDA-Entwicklungstools, allen voran ArcStyler[3] von Interactive Objects Software und Compuwares OptimalJ[4].

4.3.2. Konzepte MDA light

Der pragmatische Ansatz „MDA light“ basiert, wie oben beschrieben, auf der Generierung von Code direkt aus dem plattformunabhängigen Modell (siehe untenstehende Abbildung).

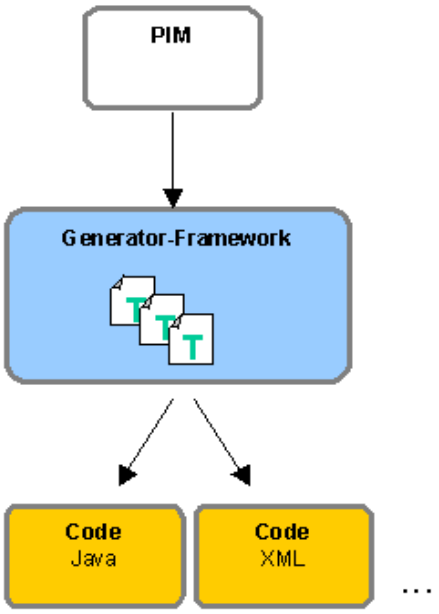


Abbildung 4-9 - Transformation nach pragmatischem Ansatz

In einer Reihe von Whitepapern ([Bet01]) und Artikeln ([Völ01], [Sta01], [Neu01]) in bekannten Java- und OO-Magazinen ist zu lesen, wie sich der pragmatische Ansatz der MDA entwickelt hat. Aufgrund der abstrakten Spezifikation und der fehlenden Definition eines Entwicklungsprozesses der OMG war man gezwungen, eigene Lösungen zu finden, um die Vorteile des modellbasierten Ansatzes trotzdem zu nutzen. Obwohl sich dieser pragmatische Ansatz doch erheblich von der OMG-Spezifikation der MDA unterscheidet, ist die direkte Generierung von Code aus dem plattformunabhängigen Modell im MDA Guide erwähnt (vgl [MDA00] Kap. 3.7 Direct Transformation to Code).

Nachfolgend werden die grundlegenden Konzepte des pragmatischen Ansatzes vorgestellt und die Unterschiede zur MDA nach OMG aufgezeigt:

3 Interactive Objects Software GmbH - http://www.io-software.com (Stand: 07.09.04)
4 Compuware Corp. - http://www.optimalj.com (Stand: 07.09.04)

Architekturzentriertes Design:
Durch den Verzicht auf das plattformspezifische Modell und die Konzentration auf ein plattformunabhängiges architekturzentriertes Design, umgeht man die Probleme des mehrstufigen Modelltransformationsprozesses, wie zum Beispiel Konsistenzprobleme bei manuellen Änderungen im Code und aufwändige Metamodellierung auf jeder Abstraktionsebene.

Forward Engineering:
Forward Engineering bedeutet, dass manuelle Änderungen im Code nicht automatisiert im Modell widergespiegelt werden. Die Generierung erfolgt nur in einer Richtung und zwar vom Modell zum Code. Bewusst wird auf ein Roundtrip-Engineering verzichtet, mit der Begründung, dass in den Modellen echte Abstraktion vorgenommen wird, und daher Roundtrip-Engineering nicht sinnvoll erscheint.

Transformation mit Schablonen:
Anstatt der Beschreibung von Abbildungen zwischen Metamodellen als Vorschrift für Modelltransformationen wählt MDA light den Weg über Schablonen. Der Gebrauch von Generatorschablonen hat sich in der Praxis bewährt und ist vielen Entwicklern schon von Code-Wizards oder der Generierung von Code aus einer Datenbank bekannt.

Generierung eines Architekturrahmens:
Mit den gegenwärtig zur Verfügung stehenden Technologien ist eine hundertprozentige Generierung des Systems in der Regel nicht möglich. Generiert werden die schematischen bzw. technischen Aspekte einer Anwendung. Die individuellen fachlichen Konzepte werden nach der Generierung manuell hinzugefügt. Die Einführung der UML 2.0 mit ihren erweiterten Konzepten (z.B. Bedingungsprüfung, Schleifen und zeitliche Verläufe) könnte den Grad der Generierung erheblich steigern. Die Auswirkungen sind zur Zeit aber noch schwer abzuschätzen, da es im Moment noch kaum praktische Beispiele zur Anwendung der UML 2.0 gibt.

Das Vorgehensmodell bei der Arbeit mit open ArchitectureWare wird im Kapitel 5 detailliert beschrieben.

5 Open ArchitectureWare

5.1 Einführung

Open ArchitectureWare ist ein flexibles template-basiertes Generator-Framework, das auf die architekturzentrierte Softwareentwicklung gemäß der Model Driven Architecture ausgelegt ist. Ursprünglich als Generator mit Metamodell und einer einfachen Transformationssprache Ende der 90er Jahre von der b+m Informatik AG[1] entwickelt, hat sich das Framework über die Jahre hinweg in verschiedenen Projekten bewährt und wurde stetig weiterentwickelt. Im September 2003 wurde die damals aktuelle Version als opensource Software unter der LGPL (Gnu Lesser General Public License) auf der opensource-Webplattform SourceForge.net[2] veröffentlicht und erfreut sich seitdem zunehmender Beliebtheit. Mangels eines Vorgehensmodells der MDA, wurde parallel zum Generator auch der sogenannte Generative Development Process (GDP) entwickelt, der auf die Arbeit mit dem open ArchitectureWare abgestimmt ist.

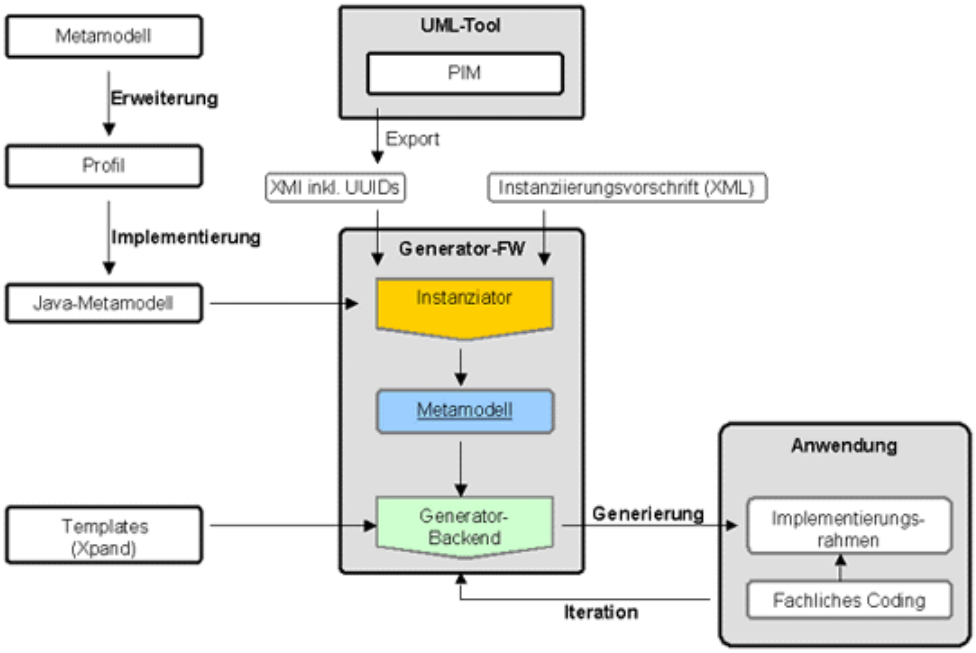


Abbildung 5-1 - Funktionsweise open ArchitectureWare

5.2 Funktionsweise

Open ArchitectureWare ist keine integrierte MDA-Entwicklungsumgebung, in der man vom PIM-Design, über Transformationen, bis zur Code-Bearbeitung, alle Arbeitsschritte in einem Werkzeug abwickelt. Das ganze Framework besteht grundsätzlich aus zwei Java Archiven (jar-Dateien), die über einen Build-Prozess (z.B. Apache Ant) in die gewünschte Entwicklungsumgebung eingebunden werden.

[1] b+m Informatik AG – http://www.bmiag.de (Strand: 19.10.04)
[2] Open Source Software Plattform: http://www.sourceforge.net (Stand: 19.10.04)

Die obenstehende Abbildung verdeutlicht die generelle Funktionsweise des Generators. Das Generator Framework lässt sich durch seinen modularen Aufbau relativ flexibel einsetzen. Verschiedene Quellen dienen dem Generator dabei als Input:

Open ArchitectureWare besitzt eine flexible Import-Schnittstelle, und ist somit unabhängig von der Modellierungssprache und dem Werkzeug, in dem das plattformunabhängige Design erstellt wird. Einzige Anforderung ist, das Modell in dem von der OMG spezifizierten Austauschformat XMI, mit global eindeutigen Identifikatoren für die Modellelemente (UUIDs), vorzuhalten. Open ArchitectureWare unterstützt somit unterschiedlichste Modellierungsformen und UML-Werkzeuge (u.a. Rational Rose XDE, TogetherJ, Poseidon). Da die Struktur des XMI-Exports der unterschiedlichen Tool-Hersteller sehr stark variiert, wird die einheitliche Repräsentation innerhalb des Generators über Mapping-Dateien, die für die jeweiligen Werkzeuge zur Verfügung stehen, gewährleistet.

Zusätzlich zum plattformunabhängigen Design wird eine Implementierung des Metamodells in Java erwartet. Das sogenannte Java-Metamodell wird vom Generator eingelesen und zur Instanziierung des technischen Anwendungsdesigns (PIM) verwendet. Während des Instanziierungsprozesses wird das Anwendungsdesign nach den Regeln, die in der Instanziierungsvorschrift festgelegt sind, auf das Instanzgeflecht des Java-Metamodells abgebildet. Die Instanziierungsvorschriften sind in einer XML-Datei (metamappings.xml) abgelegt und legen die Zuordnung der Stereotypen auf existierende Metamodellklassen fest. Das daraus resultierende instanziierte Java-Metamodell repräsentiert das Anwendungsdesign. Auf diese Weise können beliebige Diagrammarten unterstützt werden. Somit lassen sich im plattformunabhängigen Design nicht nur statische Aspekte des zu generierenden Systems, sondern auch Interaktionen und Abläufe darstellen, was gerade im Bereich der Frontend-Generierung enorm wichtig ist. Open ArchitectureWare hebt sich dadurch von vielen anderen MDA-konformen Werkzeugen ab, die nur statische Diagrammarten (meist Klassendiagramme) unterstützt werden.

Eine vereinfachte Java-Implemetierung des UML-Metamodells für grundlegende Diagramme (Klassen, Aktivitäts, Zustandsdiagramme) ist im Lieferumfang des open GeneratorFrameworks enthalten. Die Unterstützung weiterer Diagrammarten ist jederzeit möglich und erfordert lediglich die Implementierung der jeweiligen Klassen im Java-Metamodell, das vom Generator eingelesen wird. Der Generator ist somit völlig unabhängig von der Sprache in der das Metamodell definiert ist, solange eine Implementierung des Metamodells in Java vorliegt.

Über die von der b+m Informatik AG entwickelte Template-Sprache Xpand wird die Sourcecode-Expansion vorgenommen. Die Templates werden dynamisch an das Metamodell gebunden und erlauben so Navigation über das instanziierte Objektgeflecht des Anwendungsdesigns zur Laufzeit. Bei der Entwicklung der Templatesprache wurde auf eine einfache und auf die Code-Generierung zugeschnittene Syntax geachtet, die die Einarbeitungszeit in Xpand gering hält.

In den Templates lassen sich geschützte Bereiche definieren, die bei einem nachfolgenden Generierungsdurchlauf nicht überschrieben werden. Dies ist insofern wichtig, da sich im PIM ein System nicht vollständig abbilden lässt und Fachlogik nach der Generierung von Hand in die geschützten Bereiche eingefügt werden muss.

Im Anschluss werden die wesentlichen Bestandteile des Generierungsprozesses noch einmal separat nach [b+m04] beschrieben (vgl. obenstehende Abbildung):

Instanziator:
Der Instanziator liest den XMI-Export des UML-Werkzeugs ein und instanziiert anhand des Mappings, das in der Instanziierungsvorschrift definiert ist, das Java-Metamodell.

Instanziierungsvorschrift:
Die Instanziierungvorschrift ist eine XML-Datei und definiert die Zuordnung von XMI (d.h. der Modellierungselemente, die im PIM mit entsprechenden Stereotypen versehen wurden) auf Klassen im Java-Metamodell.

Java-Metamodell:
Eine in Java vorliegende Implementierung des Metamodells, das vom Generator zur Generierungszeit eingelesen wird. Das Metamodell selber kann dabei in beliebiger Form vorliegen.

Templates:
Schablonen zur Sourcecode-Generierung, in denen die architekturspezifischen Aspekte festgelegt werden. Zur Laufzeit werden die Templates an das instanziierte Java-Metamodell gebunden und haben Zugriff auf Methoden und Abhängigkeiten des Metamodells.

Instanziiertes Java-Metamodell:
Das instanziierte Java-Metamodell wird während des Instanziierungsprozeses erzeut und repräsentiert die Abbildung des Anwendungsdesigns auf das Instanzengeflecht des Metamodells.

Generator-Backend:
Das Generator-Backend bindet die Templates an das Java-Metamodell, kapselt den Interpreter für die Template-Sprache und ist somit für die Erzeugung des Sourcecodes verantwortlich.

Geschützte Bereiche:
Die geschützten Bereiche markieren die Stellen im generierten Sourcecode, an denen nach dem Generierungsdurchlauf, manuell Fachlogik hinzugefügt werden kann. Sie sind durch eine ID eindeutig gekennzeichnet und werden bei einem erneuten Generierungdurchlauf nicht überschrieben.

Implementierungsrahmen:
Das Generat wird bei der Arbeit mit open ArchitectureWare als Implementierungsrahmen bezeichnet, da architekturspezifische Details zwar generiert werden können, aufgrund von modellierungstechnischen Beschränkungen aber Logik nicht im Modell abgebildet werden kann und später manuell eingefügt werden muss.

5.2.1. Verwendung

Wie oben beschrieben besteht open ArchitectureWare grundsätzlich aus zwei Java Archive-Dateien (jar), die die kompilierten Quelldateien des Generators enthalten. Gestartet wird der Generator über den Aufruf der Klasse `de.bmiag.genfw.Generator`. Der Generator benötigt hierbei einige Parameter, die die Ausführung des Generierungsprozesses konfigurieren. Open ArchitectureWare kann zum einen textzeilenbasiert gestartet werden, zum anderen steht eine einfache Java Swing-GUI zur Verfügung, die den Aufbau des Designs aus den verschiedenen UML-Elementen als Baum visualisiert.

Da der Generierungsprozess weitgehend automatisiert werden sollte, kommt in der Diplomarbeit das javabasierte Build-Werkzeug Ant[1] der Apache Software Foundation zum Einsatz. Konfigurationsparameter lassen sich dabei in einer gesonderten Properties-Datei in Form von Schlüssel/Wert-Paaren auslagern und im Build-Skript verwenden.

Hier eine Auflistung der wichtigsten Parameter:

- `de.bmiag.genfw.instantiator.design:`
  Pfad zum XMI-Export des Anwendungsdesigns (PIM)
- `de.bmiag.genfw.instantiator.xmlmap:`
  Pfad zur XML-Mapping-Datei, die den XMI-Export des jeweiligen UML-Werkzeugs auf die interne XMI-Repräsentation des Generators mappt.
- `de.bmiag.genfw.instantiator.tooladapter.class:`
  Java-Klasse, die für die Konvertierung der oben definierten XMI-Datei für das entsprechende UML-Werkzeug verantwortlich ist.
- `de.bmiag.genfw.xpand.path:`
  Pfad zum Verzeichnis, in dem sich die Templates zur Code-Generierung befinden.
- `de.bmiag.genfw.instantiator.metamap:`
  Pfad zu den Instanziierungsvorschriften, die das Mapping des PIM (bzw. der Stereotypen des PIM) auf die Klassen des Metamodells beschreiben.
- `de.bmiag.genfw.baseuml.identifier.class:`
  Name der Identifier-Klasse des Metamodells. Eine Default-Implementierung wird vom Framework mitgeliefert (`de.bmiag.genfw.meta.core.Identifier`), kann aber vom Anwender durch Ableitung erweitert werden.
- `de.bmiag.genfw.prresolver.path:`
  Definiert den Ort, an den diejenigen Dateien, die geschützte Bereiche enthalten, kopiert werden, um bei einem erneuten Generator-Durchlauf eingefügte Fachlogik nicht zu überschreiben.
- `de.bmiag.genfw.filewriter.path:`
  Pfad, in dem die generierten Dateien gespeichert werden.

1 Apache Ant - http://ant.apache.org (Stand: 13.09.2004)

- `de.bmiag.genfw.textui.check:`
  Boolescher Wert, der definiert, ob das PIM vor der Generierung validiert werden soll.
- `de.bmiag.genfw.textui.select:`
  Definiert das Modellelement, von dem die Generierung ausgehen soll (üblicherweise eine Komponente, die weitere Modellelemente enthält).
- `de.bmiag.genfw.ui.class:`
  Definiert, in welchem Modus der Generator gestartet wird. Für die kommandozeilenbasierte Version kommt die Klasse `de.bmiag.genfw.ui.text.TextUI` zum Einsatz.

5.2.2. Metamodellierung

5.2.2.1. Modellvalidierung

Durch den modellbasierten Ansatz verschiebt sich auch der Fokus der Entwicklung. Die Zeitspanne, in der manuell programmiert wird, verringert sich signifikant, da sämtliche sich wiederholenden Anteile generiert werden können und nur die Fachlogik ausprogrammiert werden muss. Dadurch erhält das Modell einen viel höheren Stellenwert als bisher. Gleichzeitig sollen aber zur Designzeit im Modell dieselben Werkzeuge wie bei der konventionellen Softwareentwicklung (Debugging, Syntaxüberprüfung, Validierung, ...) zur Verfügung stehen, um weiterhin effektiv entwickeln zu können. In diesem Gebiet ist das Entwicklungspotential der MDA-Werkzeuge und -Frameworks noch am Größten.

Auch Open ArchitectureWare adressiert diese Problematik:
Die Validierung des Anwendungsdesigns findet mit Hilfe von Constraints statt. Diese sind nicht in OCL (Object Constraint Language) definiert, sondern in Form einer Methode (`checkConstraints()`) in derjenigen Metamodell-Klasse, die validiert werden soll. In der Validierungs-Methode können beliebige Einschränkungen definiert sein, wie zum Beispiel erlaubte Beziehungen zu anderen Modellierungselementen oder Zugehörigkeit zu einer Komponente. Während des Generierungsprozesses wird die Modellvalidierung durchgeführt und erst nach erfolgreicher Validierung wird die eigentliche Code-Generierung gestartet. Tritt ein Fehler im PIM auf, so wird ein sogenannter `DesignError` geworfen und der Generierungsprozess wird abgebrochen.

Die Problemstelllungen des Modelldebugging und der Syntaxüberprüfung zur Designzeit sind offene Punkte, die in openArchitectureWare noch nicht implementiert sind. Da das Opensource-Projekt aber stetig weiterentwickelt wird, besteht die Chance und die Hoffnung, dass sich auch im Bezug auf diese offenen Punkte in Zukunft einiges ändern wird.

5.2.2.2. Metamodellinstanziierung

```
<?xml version="1.0"?>
<!DOCTYPE MetaMap PUBLIC
    "-//b+m Informatik AG//DTD b+m Generator FrameWork MetaMappings 2.1//EN"
    "http://www.bmiag.com/dtds/metamappings_2_1.dtd">
<MetaMap>
  <Mapping>
    <Map>Activity</Map>
    <To>de.softlab.mda.metamodel.Activity</To>
  </Mapping>
  ...
```

Listing 5-1 - Beispiel einer Instanziierungsvorschrift

Das Generator Framework nutzt die in XML vorliegenden Instanziierungsvorschriften, um das Anwendungsdesign (in XMI) zur Laufzeit in ein instanziiertes Java-Metamodell zu überführen. Ergebnis der Instanziierung ist ein internes Instanzengeflecht, auf das anschließend das Generator-Backend in Form der Templates zugreifen kann. Aus objektorientierter Sicht bietet ein solches Netz aus Instanzen einige Vorteile gegenüber einem XML-Dokumentenbaum, hinsichtlich Navigierbarkeit und Einfachheit. Die obenstehende Abbildung zeigt einen Ausschnitt aus einer Instanziierungsvorschrift.

5.2.2.3. Erweiterbarkeit

Im Lieferumfang von Open ArchitectureWare ist eine vereinfachte Version des UML-Metamodells der OMG enthalten. Aufgrund der unabhängigen Schnittstelle des Generators (liest eine Java-Implementation des Metamodells ein), ist der Anwender jedoch nicht auf UML festgelegt, sondern kann das Metamodell auf beliebige Art und Weise spezifizieren, solange eine Implementierung des Metamodells in Java vorliegt. Für viele Projekte mag das mitgelieferte UML-Metamodell ausreichen, für eine detaillierte Beschreibung der Domäne und Ihrer Meta-Ebene besteht zusätzlich die Möglichkeit, das Metamodell zu erweitern:

- Metamodell-Erweiterung (UML-Profil)
  Open ArchitectureWare unterstützt standardmäßig Klassen, Zustands- und Aktivitätsdiagramme. Durch die Offenheit und Flexibilität des Frameworks ist es möglich weitere Diagrammarten der UML einzubinden. Erforderlich ist lediglich die Implementierung von Metamodell-Klassen der gewünschten Erweiterung und die Einbindung in die Instanziierungsvorschriften, die das Mapping der Stereotypen im PIM auf die Metamodell-Klassen festlegt. Dieser modulare Erweiterungsmechanismus ermöglicht es, das Metamodell auf gegebene Designrichtlinien oder Architekturvorgaben auszurichten, ohne dabei auf technische Details eingehen zu müssen. Ein im PIM als `<<Entity>>` gekennzeichnetes Modellelement kann dann im Generierungsprozess, je nach Template-Implementierung z.B. als Entity Bean (EJB) oder Java Data Object (JDO) erzeugt werden.
- XML Metadata Interchange (XMI)
  Das Konzept, das ursprünglich mit XMI verfolgt wurde (generisches Austauschformat für Modelle) hat sich in der Praxis bisher nicht bewährt. Unter der teilweise mangelhaften Umsetzung der UML Spezifikation der Modellierungswerkzeuge leidet auch XMI. Das

XMI verschiedener Toolhersteller unterscheidet sich daher teilweise deutlich in Struktur und Inhalt.
Das Generator Framework muss somit in der Lage sein, unterschiedliche XMI-Versionen einzulesen und auf eine interne Repräsentation abzubilden. Überflüssige Information, die für den Austausch der Modelle zwischen verschiedenen Werkzeugen wichtig sind (z.B. die graphische Anordnung der Modellierungselemente) sollen herausgefiltert werden, da sie für die Code-Generierung keine Bedeutung haben.
Open ArchitectureWare adressiert diese Problemstellung in Form einer generischen Adapterschnittstelle, die je nach UML-Werkzeug eine spezielle XML-Mapping-Datei einliest. Diese Datei wird in der Konfiguration vor dem Start des Generators (z.B. im Ant Build-Skript) angegeben und mappt das jeweilige XMI-Derivat auf die interne XMI-Repräsentation des Generators.
Die Inkompatibilität des XMI Exports soll sich mit der Einführung der UML 2.0 Spezifikation ändern, da dort die Struktur des XMI im Metamodell festgelegt ist und somit der Weg für ein einheitliches Austauschformat geebnet ist.

5.2.3. Templates – Skriptsprache Xpand

5.2.3.1. Einführung

Die Code-Generierung erfolgt bei open ArchitectureWare über Templates, die in Textdateien mit der Endung „.tpl“ angelegt werden. Templates stellen die konkrete Plattformbindung einer Anwendung dar. Erst bei der Templateentwicklung legt man der generischen Architektur im PIM eine bestimmte Technologie zugrunde, d.h. die Metaprogrammierung ist auch programmiersprachlich von der Anwendungsentwicklung getrennt. Zum einen besteht so die Möglichkeit aus einem PIM mit entsprechendem Metamodell, auf beliebige Plattformen generieren zu können und zum anderen kann man schnell auf etwaige Technologieänderungen reagieren. Beispielsweise hat ein Versionswechsel des Applikationsservers mit veränderter Struktur der Deploymentdeskriptoren, nur Änderungen im Template, das den Deploymentdeskriptor generiert, zur Folge. Oder aber man hat die Möglichkeit aus dem gleichen PIM, unter Einsatz unterschiedlicher Templates, sowohl ein Frontend auf Basis von Struts, als auch auf Basis von JavaServer Faces[1] zu generieren.

5.2.3.2. Syntax und Entwicklung

Ein Template besteht aus generischen Anweisungen der Skriptsprache Xpand und statischen Teilen der zu generierenden Programmiersprache. Xpand ist eine Entwicklung der b+m Informatik AG, deren Syntax auf das Anwendungsgebiet der Code-Generieung zugeschnitten ist. Im Gegensatz zu anderen templatebasierten Generatoren, die Skriptsprachen wie z.B. Velocity[2] einsetzen, ist Xpand frei von überflüssigen Sprachelementen, und besitzt ausschließlich Konstrukte (bedingte Anweisungen, Schleifen, Operatoren, ...), um auf einzelne Modellelemente zuzugreifen und die erforderlichen Codefragmente zu generieren. Auch Polymorphie wird von Xpand unterstützt. Die einzelnen Template-Dateien stellen Namensräume dar und dürfen weitgehend frei benannt werden. Zur Kennzeichnung der syntaktischen Elemente werden im open Generator Framework die Zeichen « und »

[1] JavaServer Faces - http://java.sun.com/j2ee/javaserverfaces/index.jsp (Stand: 13.09.04)
[2] Velocity Template Engine - http://jakarta.apache.org/velocity (Stand: 25.09.04)

verwendet.

Definieren der Templates

```
«DEFINE <TemplateName> [(<ParameterList>)]FOR <MetaClass>»
        <Sequence>
«ENDDEFINE»
```

Listing 5-2 - Syntax der DEFINE-Anweisung

Xpand-Anweisungen sind geschlossene Einheiten, die von doppelspitzen Klammern, sowie einem Anfangs- und Endschlüsselwort (vgl. Abbildung: `DEFINE`, `ENDDEFINE`) umschlossen sind. Die Besonderheit von Xpand ist der Zugriff auf das Anwendungsdesign (PIM), in Form des instanziierten Java-Metamodells, von einem Template aus. Mithilfe des Schlüsselworts `DEFINE` werden die Templates definiert und dynamisch während der Laufzeit an eine Instanz der angegebenen Metamodellklasse gebunden. Dadurch wird der Zugriff auf Methoden und Beziehungen zwischen den Metamodellklassen möglich.

Struktur

Wie oben erwähnt, können die Template-Dateien weitgehend frei benannt werden, die Dateinamen `Root.tpl` und `Check.tpl` bilden hierbei die einzigen Ausnahmen. Die in der Datei `Root.tpl` definierten Schablonen stellen für den Generator den Ausgangspunkt für die Code-Generierung dar.

Als Beispiel sieht man in der untenstehenden Abbildung das Root-Template für ein System (Metaklasse `System`), das zwei Komponenten (`UIComponent`, `BusinessComponent`) enthält.

```
«REM --- SYSTEM ---»
«DEFINE Root FOR System»
        «EXPAND Root FOREACH BusinessComponent»
        «EXPAND Root FOREACH UIComponent»
«ENDDEFINE»

«REM --- UICOMPONENT ---»
«DEFINE Root FOR UIComponent»
        «EXPAND UIComponent::UIComponent»
«ENDDEFINE»

«REM --- BUSINESSCOMPONENT ---»
«DEFINE Root FOR BusinessComponent»
        «EXPAND BusinessComponent::BusinessComponent»
«ENDDEFINE»
```

Listing 5-3 - Root-Template: Einstiegspunkt in die Generierung

Mithilfe der Datei der Datei `Check.tpl` wird die initiale Syntaxüberprüfung gesteuert, d.h. `Check.tpl` wird vor der eigentlichen Generierung expandiert, um bei Fehlern im Design, die Code-Generierung nicht starten zu müssen.

Expansion

Die Anweisung `EXPAND` realisiert den Aufruf eines (Sub-)Templates und ist mit einem Methodenaufruf in Java zu vergleichen. Falls der Templatename nicht explizit einen Namensraum vorgibt (andere Template-Datei), wird der aktuelle Namensraum (aktuelle Template-Datei) verwendet. Darüber hinaus können Templates mit Parametern aufgerufen werden.

Im obenstehenden Beispiel werden im Roote-Template `UIComponent` und `BusinessComponent` aus `System` expandiert, und expandieren dann in Templates verschiedener Namensräume (`BusinessComponent::BusinessComponent`, `UIComponent::UIComponent`).

Im Kapitel 7 wird ausführlicher auf die Templateentwicklung am konkreten Projekt eingegangen, ausserdem findet man in [b+m02] die komplette Spezifikation und Grammatik der Templatesprache Xpand.

5.2.4. Geschützte Code-Bereiche

Da bei der generativen Entwicklung mit open ArchitectureWare hauptsächlich der Implementierungsrahmen generiert wird, benötigt man einen Mechanismus, der die manuell in den Code eingefügte Fachlogik bei einer erneuten Generierung nicht überschreibt. Diese Problematik ist im open Generator Framework durch den Einsatz von sogenannten geschützten Bereichen (engl. protected regions) gelöst. Ein geschützter Bereich muss als solcher im Code, sowohl für den Generator als auch für den Entwickler, eindeutig identifizierbar sein. Der Generator überschreibt den Code, der innerhalb eines geschützten Bereichs steht, nicht, der Programmierer darf ausschließlich Code-Änderungen innerhalb geschützter Bereiche vornehmen.

```
«PROTECT CSTART <String> CEND <String> ID <Id>»
// TODO: enter your own code here...
«ENDPROTECT»
```

Listing 5-4 - Xpand-Anweisung zur Erstellung eines geschützten Bereichs

Für die Erstellung der geschützten Bereiche stellt Xpand eine gesonderte Anweisung zur Verfügung. `<String>` wird durch die sprachabhängige Kommentar-Notation der jeweiligen Zielsprache ersetzt und `<Id>` steht entweder für eine manuell eingefügte oder generierte eindeutige Identifikationsnummer. Der Generator identifiziert den Kommentarbereich anhand der eindeutigen ID als geschützten Bereich und überschreibt die Inhalte dieses Bereichs nicht. Der Entwickler darf unter keinen Umständen eine Änderung an der generierten Notation des geschützten Bereichs vornehmen, da bei einem erneuten Generierungsdurchlauf der Code in diesem Bereich gelöscht werden würde.

5.3 GDP – Generative Development Process

5.3.1. Einführung

Der Trend der architekturzentrierten Entwicklung setzt sich immer weiter fort. Den Firmen fehlt aber oft der Prozess und das Vorgehensmodell, um das Potential der modellbasierten Ansätze ganz auszuschöpfen.

Um diese Problematik zu adressieren und die Anwender bei der generativen Entwicklung zu unterstützen, hat die b+m Informatik AG parallel zur Veröffentlichung des open Generator FrameWorks auch ein passendes architekturzentriertes Vorgehensmodell vorgestellt:
Der Generative Development Process (GDP).
Laut [b+m03] steht der GDP nicht in Konkurrenz zu herkömmlichen, in Firmen etablierten, objektorientierten Entwicklungsprozessen, sondern ist vielmehr als Verfeinerung im Hinblick auf die Anwendung architekturzentrierter und generativer Technologien zu sehen. Der GDP soll hauptsächlich die Entwickler bei der Arbeit mit dem open Generator FrameWork unterstützten und dazu beitragen, Entwicklungszeit und -aufwand zu reduzieren und die Qualität der Software zu steigern.

In den folgenden Kapiteln werden die Grundzüge und wichtigsten Bestandteile des Generative Development Process hinsichtlich Architekturzentrierung und Erstellung von Anwendungsfamilien kurz vorgestellt. Wie bei allen Vorgehensmodellen, liegt die Umsetzung in den Händen der Anwender, was auch eine leichtgewichtige Anwendung des Prozesses zulässt. Eine derartige Interpretation des GDP wurde auch im Rahmen der Diplomarbeit eingesetzt und wird im Folgenden beschrieben.

5.3.2. Grundlagen

GDP ist laut [b+m03] ein iterativer und inkrementeller Prozess, d.h. die einzelnen Teilprozesse werden immer wieder durchlaufen und stellen so wiederverwendbare Bausteine des GDP dar.

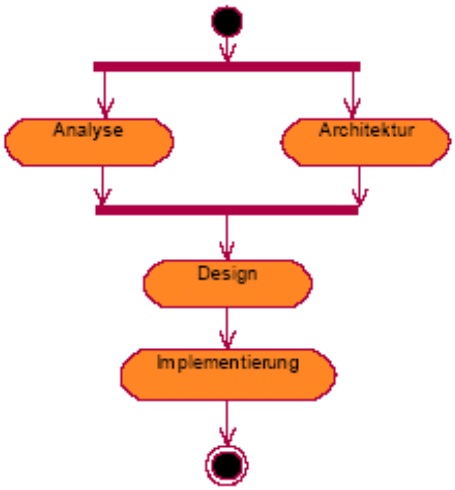


Abbildung 5-2 - Grundprozess des GDP

Fachliche und technische Modellierung sind klar voneinander abgetrennt und können parallel erfolgen. Im Design wird die Fachlichkeit mit der Technik verknüpft, d.h. die Fachlichkeit wird in der Designsprache ausgedrückt. Im Anschluß wird das Design mithilfe des Generators in

eine Implementierung überführt

5.3.3. Architektur

In der untenstehenden Abbildung werden die Aktivitäten des Architektur-Teilprozesses deutlich. Einzelne Bestandteile können bei Projektbeginn schon aus vorangegangenen Projekten vorhanden sein oder werden im Projektverlauf erstellt, wenn es sich um eine Neuentwicklung handelt.

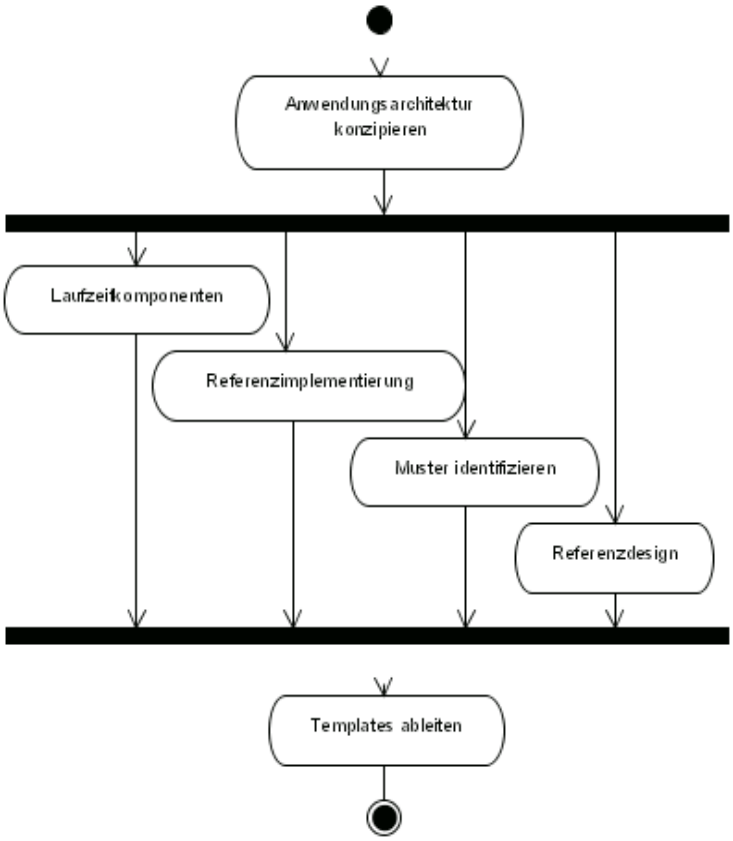


Abbildung 5-3 - Teilprozess des GDP

Anwendungsarchitektur konzipieren

Während der Konzeption der Anwendungsarchitektur werden typische Architekturkonzepte wie zum Beispiel die technische Schichtung, Kommunikation zwischen den Schichten oder die verwendeten Protokolle festgelegt. Dabei ist vor allem auf die strikte Trennung zwischen Fachlichkeit und Technologie zu achten, um die Möglichkeit, das Konzept auch für fachlich unterschiedlich ausgerichtete Projekte verwenden zu können, zu wahren.

Laufzeitkomponenten erstellen

Der Begriff Laufzeitkomponenten beinhaltet Software-Bausteine, wie Frameworks, Basisklassen, Bibliotheken. Komponenten, die nicht in unterschiedlichen fachlichen Ausprägungen bestehen, sondern rein technisch motiviert sind und in der Regel auch nicht generiert werden.

Referenzimplementierung erstellen
Die Referenzimplementierung ist ein integraler Bestandteil der generativen Entwicklung mit open ArchitectureWare. Sie ist als Prototyp für die Generierung zu verstehen und wird manuell anhand eines oder mehrerer relevanter UseCases erstellt. Laut [b+m03] darf die Referenz-Implementierung nicht als einfaches, isoliertes Beispiel missverstanden werden, aus dem man bei Bedarf Anregungen für die Implementierung bezieht. Sie zeigt vielmehr die Anwendung und Umsetzung der Spezifikationen der Anwendung.

Der Umfang der Referenzimplementierung hängt davon ab, ob in dem Technologieumfeld, in dem das System erstellt werden soll, bereits Erfahrung vorliegt. Bei einer neu zu erstellenden Anwendungsfamilie wird die Referenzimplementierung zunächst komplett von Hand erstellt. In einem späteren Schritt werden aus der Referenzimplementierung die Generator-Templates abgeleitet.

Designmuster identifizieren
Die Identifizierung der Designmuster ist der wesentliche Schritt der Abstraktion. Im Metamodell und dem plattformunabhängigen Modell werden die identifizierten generischen Anteile der Applikation in Form von Stereotypen, Tagged Values und Metamodell-Klassen wiederzufinden sein.

Referenzdesign erstellen
Laut [b+m03] zeigen Referenzdesign und Referenzimplementierung zusammen, exemplarisch die Syntax und Semantik der Anwendung bis auf die Implementierungsebene und konkretisieren damit das Architekturkonzept im Detail.

Generator-Templates ableiten
Die Generator-Templates werden aus der Referenzimplementierung mit Hilfe des Referenzdesigns abgeleitet. Sämtliche generischen Anteile der Anwendung werden in den Templates mit Hilfe der Skriptsprache Xpand ausgedrückt. Da, wie weiter oben schon erwähnt, nicht die gesamte Fachlogik generiert werden kann, werden in den Templates sogenannte geschützte Bereiche angelegt, die von den Anwendungsentwicklern durch manuelles Codieren ausgefüllt werden können.

5.4 Generator Framework Utilities

5.4.1. Überblick

Die Generator FrameWork Utilities (GenFW Utils) sind eine Sammlung von Hilfsmitteln, die die Arbeit mit dem Generator erleichtern und teilweise den Funktionalitätsumfang erweitern. In einer späteren Version sollen die Utilities, die sich zum Teil noch in der Entwicklungsphase befinden, in den Generatorkern integriert werden und nicht mehr über ein separates Java Archiv eingebunden werden müssen.

Folgende Teile der GenFW Utils sind in der Diplomarbeit im Einsatz:

- Ant Task
  Die Utilities stellen einen Ant Task (generate), der die verschiedenen Start-Parameter und Eigenschaften des Generators kapselt, zur Verfügung
- Plugin-Mechanismus
  Plugins können modular über den Ant Task in das Generator Framework eingebunden

werden. Es besteht die Möglichkeit, verschiedene Modi für die Generierung festzulegen, und jedem einzelnen Modus unterschiedliche Plugins zuzuordnen. So wird zum Beispiel dem Validierungs-Modus, der das PIM in Bezug auf das Metamodell validiert, das Metamodell-Plugin zugeordnet, das den Pfad zum XMI des PIM und das Mapping auf das werkzeugabhängige XMI festlegt. Dem Generierungsmodus wird, zusätzlich zum Metamodell-Plugin, das Code-Generierungsplugin zugewiesen, das den Pfad zu den Templates beschreibt. Auf diese Weise kann man modular die Konfiguration des Generierungsdurchlaufs vornehmen.

- TemplateManager
  Der Templatemanager bietet die Möglichkeit, Templates logisch in Verzeichnisse aufzuteilen. Der Pfad zu den Templates wird beim Aufruf Teil des Namensraums des Templates (siehe Abbildung)

```
<<DEFINE Root FOR System>>
  <<EXPAND buildtime/API::Root>>
  <<EXPAND composition/MW::Root>>
  <<EXPAND util/Build::BuildFile>>
  <<EXPAND util/Launch::LaunchFile>>
  <<EXPAND util/WindowsActivator::BatchFile
        FOREACH { Container AS c | c.Type == "java"
<<ENDDEFINE>>
```

Listing 5-5 - TemplateManager der GenFW Utils

- MMUtils (MetaModel Utils)
  MMUtils sind eine Menge von Hilfsklassen, die den Entwickler bei der Metamodellerstellung unterstützen sollen. So werden zum Beispiel technologieabhängige Methoden (z.B. Ausgabe von Package-Namen in Java oder Header-Dateien in C) in Packages gekapselt, ohne am Aufruf der Methode im Template etwas ändern zu müssen. Auf diese Weise kann die Implementierung eines technologieunabhängigen Metamodells sichergestellt werden.

6 Apache Struts Framework

6.1 Einführung

6.1.1. Model-View-Controller Design Pattern

Das Model-View-Controller Design Pattern ist aus dem Smalltalk MVC Framework entstanden und wird häufig bei der Entwicklung von komplexen Web-Applikationen eingesetzt. Der Kontrollfluss der Applikation wird von einem zentralen Controller gesteuert. Dieser delegiert eingehende Anfragen (im Fall von Struts: HTTP-Requests) an Aktionen, die an ein sogenanntes Model geknüpft sind. Der Controller hat somit die Rolle eines Adapters zwischen Request und Model. Das Model repräsentiert und kapselt die Geschäftslogik und den Zustand der Applikation. Nachdem die passenden Aktionen für die jeweilige Anfrage auf dem Model ausgeführt wurden, wird der Kontrollfluss vom Controller an die passende View weitergeleitet. Das Weiterleiten kann durch Abbildungsregeln, die üblicherweise aus einer Datenbank oder einer Property-Datei geladen werden, festgelegt werden. Dies resultiert in einer losen Kopplung zwischen Model und View, was die Applikation bedeutend wartbarer und übersichtlicher macht.

6.1.2. Model 2 Architektur

Ausgehend von der Web-Entwicklung mit JavaServer Pages (Model 1 Architektur) hat sich durch eine Kombination von JSPs und Java Servlets die Model 2 Architektur [Sun01] entwickelt. Die Konzentration der Stärken der einzelnen Technologien, Steuerung des Kontrollflusses und serverseitige Verarbeitung in den Servlets und Generierung der Präsentationsschicht in JSPs, hat sich mittlerweile bewährt und viele Web-Anwendungen und etliche Web-Frameworks werden unter Zuhilfenahme der Model 2 Architektur implementiert. Vom grundlegenden Aufbau her orientiert sich die JSP Model 2 Architektur an den Vorgaben des oben beschriebenen Model-View-Controller Design Patterns.

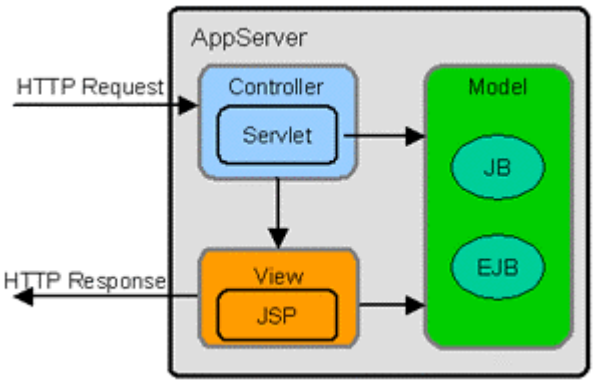


Abbildung 6-1 - Model 2 Architektur

Das Ziel der JSP Model 2 Architektur besteht darin, eine Trennung der Verarbeitung eines Anwendungsfalls von dessen Präsentation zu erreichen.

- Das Model beschreibt die Komponente der Anwendungsdaten und deren Verhalten. Neben der Repräsentation der Anwendungsdaten und deren Verwaltung führt es zusätzlich sämtliche Änderungen auf den Daten aus.
- Ein View dient der Präsentation von Informationen eines Models. Ein Model kann hierbei durch mehrere Views, die unterschiedliche Sichtweisen des Models darstellen, visualisiert werden.
- Der Controller ist verantwortlich für die Interaktion des Benutzers mit der Anwendung, übernimmt die Steuerung der Verarbeitung und stellt den Kontext, in dem verschiedene Views und Models zueinander stehen, dar. Das MVC-Konzept erlaubt somit eine klare Schichtentrennung zwischen den Anwendungsdaten, den Sichten auf die Anwendungsdaten und der Benutzerschnittstelle

Durch die Trennung von Logik, Datenhaltung und Darstellung erreicht man eine klare Abgrenzung der Funktionalität und somit einen hohen Grad an Wiederverwendung.

6.2 Apache Struts 1.2

6.2.1. Überblick

Das opensource Projekt Struts [Apa02] besteht aus einem Framework zur Erstellung von Web-Applikationen und wurde im Mai 2000 der Apache Software Foundation [Apa01] übergeben, um der Java Community ein MVC Framework zur Entwicklung von komplexen Web-Anwendungen zur Verfügung zu stellen.

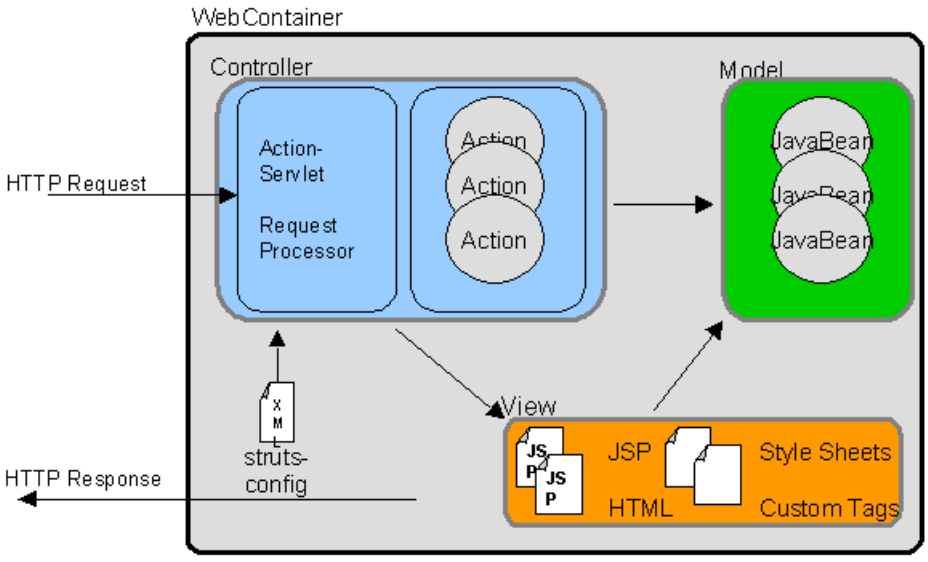


Abbildung 6-2 - Apache Struts - Übersicht

Struts bietet eine große Variabilität in der Umsetzung der Web-Anwendung. So ist der Entwickler zum Beispiel nicht auf eine bestimmte Model-Implementierung festgelegt, sondern

kann variabel verschiedene Technologien (z.B. JavaBeans, EJB, Hibernate, ...) einsetzen. Auch bei der Entwicklung der Präsentationsschicht stehen dem Entwickler verschiedene Möglichkeiten, von JSP bis XML/XSLT oder JavaServer Faces zur Verfügung. Das Framework selbst basiert auf offenen und bewährten Standards, wie zum Beispiel Java Servlets, XML und Java.

Im Folgenden wird näher auf die einzelnen Elemente der Architektur und Ihre Umsetzung im Struts Framework, auch im Hinblick auf die Umsetzung der Model 2 Architektur, eingegangen.

6.2.2. Controller

Grundsätzlich ist der Controller eine Implementierung des Command-Patterns, die es erlaubt, einkommende Anfragen (typischerweise von einem Benutzer einer Web-Anwendung) entgegenzunehmen, die entsprechende Funktion der Geschäftslogik aufzurufen und anschließend die Kontrolle an die passende View-Komponente zu delegieren. Der Controller definiert den Ablauf der Anwendung und ist Vermittler zwischen der View- und der Modelkomponente. Er besteht hauptsächlich aus drei Grundkomponenten, die zum Teil vom Framework mitgeliefert werden:

- ActionServlet
- RequestProcessor
- Action Klassen

Allgemein kann sich eine Struts Web-Applikation aus mehreren Modulen, denen jeweils eine XML-Konfigurationsdatei (struts-config.xml) zugeordnet ist, zusammensetzen. Für eine Applikation besteht eine Instanz der Klasse ActionServlet, für jedes Modul, innerhalb einer Applikation, ein RequestProcessor. Vom Framework werden sowohl für das ActionServlet als auch für den RequestProcessor Implementierungen bereitgestellt, die bei Bedarf jederzeit erweitert werden können.

Das ActionServlet liest die XML-Konfigurationsdatei ein, in der unter anderem das sogenannte ActionMapping festgelegt wird. Dort sind verschiedene Pfade definiert, die mit der URI der Anfrage abgeglichen werden und üblicherweise auf einen qualifizierten Klassennamen einer Action Klasse verweisen. Das ActionServlet nimmt die eingehenden HTTP-Requests an und delegiert die Anfragen an den RequestProcessor des für die Anfrage passenden Moduls. Zuvor werden die vom Anwender übermittelten Daten vom ActionServlet aus den Request Parametern ausgelesen und z.B. in JavaBeans gespeichert, um beim Ausführen der Geschäftslogik wieder auf die übermittelten Daten zugreifen zu können. Über den RequestProcessor laufen alle Anfragen für das jeweilige Modul, er entscheidet, aufgrund des vordefinierten ActionMappings, welche Action-Klasse aufgerufen werden soll. Die Action-Klassen interagieren typischerweise mit der Geschäftslogik, führen Operationen auf den Daten des Models aus, mappen die Modelldaten in den scope (request, session, etc.) und geben einen ActionForward an den RequestController zurück. Der ActionForward repräsentiert den View oder die Action, an die der RequestProcessor den Kontrollfluss übergibt.

6.2.2.1. Struts Konfiguration

Nachstehend werden die einzelnen Elemente der zentralen Konfigurationsdatei des Struts Frameworks (`struts-config.xml`) kurz beschrieben.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
          "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">
<struts-config>

<!-- ========== Data Source Configuration =============================== -->
    1   <data-sources />

<!-- ========== Action Forms ============================================ -->
    2   <form-beans>
                <form-bean name="GetGroupDealerForm"
                           type="de.softlab.ivsr.struts.form.GetGroupDealerForm"/>
        </form-beans>

<!-- ========== Global Exceptions ======================================= -->
    3   <global-exceptions>
                <exception key="system.error.fatal"
                           path="/SystemError"
                           type="java.lang.Throwable" />
        </global-exceptions>

<!-- ========== Global Forwards ========================================= -->
    4   <global-forwards>
                <forward name="index" path="/actions/home.do" />
        </global-forwards>

<!-- ========== Action Mappings ========================================= -->
    5   <action-mappings>
                <action path="/actions/GetGroupDealerAction"
                        name="GetGroupDealerForm"
                        input="GetGroupDealerTDef"
                        attribute="GetGroupDealer">
                        <forward name="forward" path="ListGroupDealerTDef" />
                </action>
        </action-mappings>

<!-- ========== Message Resources ======================================= -->
    6   <message-resources parameter="resources.ApplicationResources" null="false"/>

<!-- ========== Plug Ins Configuration ================================== -->
    7   <plug-in className="org.apache.struts.tiles.TilesPlugin">
                <set-property property="definitions-config"
                              value="/WEB-INF/struts/tiles-definitions.xml" />
                <set-property property="moduleAware" value="true" />
                <set-property property="definitions-parser-validate" value="true" />
        </plug-in>
</struts-config>
```

Abbildung 6-3 - Zentrale Konfigurationsdatei des Struts Frameworks

1. <data-sources>:
Anhand des Elements <data-sources> kann eine beliebige Menge von Datenquellen (javax.sql.DataSource) definiert werden, auf die anschließend in der Web-Applikation zugegriffen werden kann. Ein typisches Beispiel für eine Datenquelle ist ein Connection Pool

zu einer Datenbank oder ähnliche persistente Speicherquellen. Falls die Applikation bereits eine dedizierte Persistenzschicht (z.B. EJB) zur Verfügung stellt, ist die Definition einer Datenquelle in der Konfigurationsdatei hinfällig.

2. <form-beans>:
<form-bean> Elemente sind Deskriptoren für ActionForm-Klassen, die ein HTML-Formular mit den vom Anwender eingegebenen Daten im Struts Framework abbildet. Anhand der Deskriptoren werden die Instanzen der ActionForms zur Laufzeit erzeugt.

3. <global-exceptions>:
Im Bereich der Global Exceptions hat man die Möglichkeit globale Ausnahmebehandlung im Struts Framework zu definieren. Bei Auftreten der definierten Exceptions wird der Kontrollfluss der Anwendung auf die in der Konfigurationsdatei angegebene Seite delegiert.

4. <global-forwards>:
Ähnlich wie bei den oben beschriebenen Global Exceptions, kann man anhand der Global Forwards, je nach definiertem Muster, zentral den Kontrollfluss der Anwendung auf die spezifizierte Seite leiten.

5. <action-mappings>:
ActionMappings definieren die manuell erstellten Action-Klassen. Nach Auftreten eines im Attribut path angegebenen URL-Musters, wird die spezifizierte Action-Klasse aufgerufen, führt die implementierten Aktion aus und leitet an eine, im Element `<forward>` definierte, Seite oder andere Action-Klasse weiter.

6. <message-resources>:
Im Struts Framework besteht die Möglichkeit, eine oder mehrere Property-Dateien anhand von <message-resources> Elementen zu definieren. Durch Einsatz dieser Property-Dateien hat man die Möglichkeit, eine Applikation für verschiedene Sprachen bereitzustellen. Sämtlicher Fliesstext einer JavaServer Page, wird in der jeweiligen Sprache in die dazu passende Property-Datei ausgelagert und über Schlüssel zur Laufzeit angezeigt.

8. <plug-in>:
Struts Plugins (vgl. Kapitel 6.2.5) werden über das <plug-in> Element in der Struts Konfiguration konfiguriert und beim Start der Applikation geladen. Beispiele für Plugins in Struts sind Tiles (vgl. Kapitel 6.2.5.1) und Validator (vgl. Kapitel 6.2.5.2).

6.2.3. Model

Das Model repräsentiert die Anwendungsdaten und die Geschäftslogik der Anwendung, die bestimmte Aktionen und Änderungen auf den Anwendungsdaten ausführt. Es sollte generell unabhängig sein und vom Framework (in diesem Fall Struts) und der Controller-Schicht losgelöst sein.

Das Struts Framework an sich stellt keine Model Klassen zur Verfügung, es liegt in der Hand der Entwickler, die Persistenz und die zugehörige Logik zu implementieren. Das Model wird spezifisch an die Applikation und die Anforderungen angepasst, Technologien wie JavaBeans, Enterprise JavaBeans oder Java Data Objects kommen hierbei zum Einsatz. Auch eine große Menge an Design Patterns hat sich bei der Implementierung der Modelkomponente bewährt. Beispiele hierfür sind unter anderem Service Locator, Business Delegate, Session Facade Pattern und Front Controller, alle Bestandteile der Core J2EE

Patterns [Sun02].

6.2.4. View

Die View Komponente visualisiert Teile des Models für den Anwender. Die Repräsentation des Models kann, je nach Anforderung (web client, rich GUI client, etc.), auf unterschiedliche Art und Weise erfolgen. Die View Komponente sollte weder direkt Model-Daten bearbeiten, noch direkt mit der Geschäftslogik des Models interagieren.

Das Struts Framework spezifiziert keine spezielle Technologie für die Präsentationsschicht, stellt aber für die Entwicklung mit JavaServer Pages Tag Bibliotheken, die den Entwickler bei der Erstellung der JavaServer Pages unterstützen, zur Verfügung. Da Struts jedoch generell komplett losgelöst von der Technologie der View-Komponente ist, kann das Frontend auch problemlos z.B. mit JavaServer Faces oder XSLT entwickelt werden.

6.2.4.1. Tag Libraries

Häufig wird in JavaServer Pages die Präsentation (HTML) mit der Implementierung (Java) vermischt. Dies kann eine Vielzahl an Problemen aufwerfen. Zum einen wird die Instandhaltung der JSPs sehr aufwendig, da Designer und Java-Entwickler an den gleichen Dateien arbeiten müssen, zum anderen wird eine JSP sehr schnell unübersichtlich, wenn die in Java programmierten Teile in der JSP (sog. Scriptlets) überhand nehmen. Da man aber auch einfachste Präsentationslogik, wie zum Beispiel Iteration, nicht ohne Sciptlets darstellen kann, gibt es verschiedene Tag-Bibliotheken, die den Entwicklern das Leben erleichtern sollen.

- Struts Tag Bibliothek
  Obwohl das gesamte Struts Framework weitgehend unabhängig von der Präsentationsschicht ist, stellt es eine Sammlung von Tags zur Verfügung, die Model Objekte der Applikation auf der Präsentationsschicht abbilden.
    - HTML Tag Bibliothek
      Die HTML Tag Library dient hauptsächlich dazu, komfortabel HTML-Elemente und vor allem Formulare zu erstellen, die mit dem Struts Framework interagieren.
    - Logic Tags Bibliothek
      Logic Tags kommen bei bedingten Ausgaben auf der Präsentationsschicht zum Einsatz, iterieren zum Beispiel über eine Liste von Objekten und regeln den Kontrollfluss der Applikation.
    - Bean Tags Bibliothek
      Die Bean Tag Bibliothek stellt nützliche Funktionen zur Verfügung, mit denen Eigenschaften von JavaBeans ausgelesen, aber auch neue JavaBeans erstellt werden können.
    - Nested Tags Bibliothek
      Die Nested Tags stellen Funktionalitäten zur Erweiterung vorhandener Tags im Bezug auf Verschachtelung zur Verfügung. Man hat die Möglichkeit ein verschachteltes Objektmodell zu definieren, darzustellen und dieses Model auch zu verändern.

Allerdings empfehlen die Entwickler des Struts Framework mittlerweile den Einsatz der JSP Standard Tag Library, da die Schnittmenge der beiden Bibliotheken in vielen Bereichen recht groß ist und man auf einen gemeinsamen Standard in der Web-Entwicklung setzten will. Somit werden die Struts-spezifischen Tag Bibliotheken in den nächsten Versionen des Frameworks nicht mehr weiterentwickelt werden, beziehungsweise nicht mehr vorhanden sein.

- JSTL – JSP Standard Tag Library
  JSTL [Sun03] ist eine Ansammlung verschiedener Tag Bibliotheken und kapselt die Grundfunktionalitäten vieler Web-Anwendungen in einfachen Tags. Die Bibliothek liefert Unterstützung im Bereich der Präsentationslogik (Iteration und Bedingungen) und Formatierung, kann mit XML umgehen und auch Internationalisierung und Datenbankzugriff sind integriert. Wie oben schon erwähnt liegt in der, an kein Framework gebundenen JSP Standard Tag Library, die Zukunft der JSP-Entwicklung auch über die Grenzen von Frameworks hinweg, da allgemeine Funktionalitäten, die in sehr vielen Web-Anwendungen zum Einsatz kommen, zur Verfügung gestellt werden.
- Struts-EL Tags
  Dieses Sub-Projekt ist eine Erweiterung der Struts Tag Bibliothek. Jeder Struts-EL Tag erweitert einen bereits vorhandenen Struts Tags, durch Subklassenbildung. Kernstück der Bibliothek ist die im Jakarta Taglibs Projekt [Jak00] entwickelte Rendering-Engine, die die Attribut-Werte evaluiert, im Gegensatz zu dem Arttribut `rtexprvalues`, das bisher in den Struts Tags eingesetzt wurde. Einige Struts Tags wurden nicht für diese Bibliothek implementiert, da deren Funktionalität in der JSTL bereits zur Verfügung gestellt wird.

6.2.4.2. Internationalisierung (I18N)

Der Trend, eine Anwendung generisch und universell einsetzbar zu machen, unabhängig von speziellen Sprache und Kulturen, ist auch in Struts zu erkennen. Basierend auf den Java-Klassen `java.util.Locale` und `java.util.ResourceBundle`, implementiert Struts nicht nur Unterstützung für verschiedene Sprachen, sondern auch länderspezifische Unterschiede in der Darstellung zum Beispiel der Zeit und des Datums oder der Währung. Dies geschieht durch verschiedene, den Ländern zugeordnete Property-Dateien, die abhängig von der jeweiligen Länderauswahl geladen werden.

6.2.5. Struts Plugins

Ein Plugin im Sinne von Struts kapselt Konfigurationen für eine modulspezifische Resource oder einen Service, der bei Inbetriebnahme oder Beenden einer Applikation verständigt werden muss. Plugins werden in der zentralen Konfigurationsdatei struts-config.xml für das jeweilige Modul definiert. Sie sind integraler Bestandteil von Struts seit Veröffentlichung der Struts Versions 1.1.

6.2.5.1. Tiles Plugin

Seit Struts 1.1 gehören Tiles (wörtlich übersetzt: „Kacheln“) zum Kern des Struts Frameworks. Der Fokus der Tiles liegt auf der View-Komponente innerhalb des Model-View-Controller Paradigmas. Tiles baut auf der Include-Funktionalität auf, die in der JSP Spezifikation [Sun04] definiert wird und stellt ein Framework zur Verfügung, mit dem die Präsentationsschicht aus wiederverwendbaren, klar voneinander abgetrennten Teilen schematisch zusammengefügt wird. Dies verringert die Menge an JSP/HTML-Code merklich und führt zu einer besseren Übersichtlichkeit und Wartbarkeit der Applikation, auch im Hinblick auf Designänderungen. Jedes Tile ist eine JSP-Seite, die sich wiederum aus Tiles zusammensetzen kann, und kann beliebig oft in der Applikation wiederverwendet werden. In einer Konfigurationsdatei (`tiles-definitions.xml`) wird definiert, aus welchen Tiles sich die einzelnen Seiten zusammensetzen.

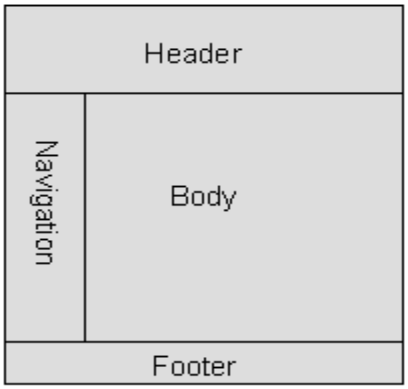


Abbildung 6-4 - beispielhafte Zusammensetzung von Tiles

Es besteht die Möglichkeit ein Basis-Layout anzulegen, das ein Grundschema der Applikation festlegt (siehe Grafik). Alle Bildschirmseiten der Applikation erweitern dann dieses Basislayout und ändern nur noch die auf den jeweiligen Seiten dynamischen Anteile. Statischen Teilen einer Seite, wie zum Beispiel einer Fußzeile, kann im Basis-Layout eine JSP zugewiesen werden und muss daher auf den einzelnen Seiten nicht mehr spezifiziert werden. Das Tiles Framework stellt zusätzlich eine Tag-Bibliothek zur Verfügung, die den Entwickler bei der Seiten-Definition und dem Einfügen der Tiles unterstützt. Dass Tiles und Struts sich sehr gut ergänzen, erkennt man auch an der Internationalisierungs-Unterstützung von Tiles. Für verschiedene Ländereinstellungen können unterschiedliche Konfigurationsdateien definiert und somit auch jeweils spezielle Tiles geladen werden.

6.2.5.2. Validator Plugin

Seit der Struts Version 0.5 ist die Validierung von Anwender-Eingaben verfügbar. Die Validierung wurde zunächst in das Jakarta Commons Projekt [Jak01] ausgelagert, bis in der Struts Version 1.1 eine spezielle Erweiterung des Validierungs-Frameworks integriert wurde. Das Validator Framework erleichtert den Umgang mit Muss-Eingabefeldern, Überprüfen von Eingaben auf reguläre Ausdrücke, Email-, Kreditkarten- und Datumsvalidierung. Auch serverseitige Typüberprüfung der Eingaben des Anwenders wird bereitgestellt. Man unterscheidet generell zwischen client- und serverseitiger Validierung. Bei der clientseitigen

Validierung wird dynamisch eine Javascript-Abfrage generiert, die z.B. abprüft, ob bestimmte Felder gesetzt sind. Vorteil hierbei ist, dass man für einfache Validierungen die Anzahl der Anfragen zum Server verringern kann, jedoch keine Validierungssicherheit hat, da der Anwender sehr einfach den Einsatz von Javascript im Browser deaktivieren kann. Deshalb ist die clientseitige Validierung optional und das Framework garantiert die Validierung auf der Serverseite.

Das Validator Framework basiert auf zwei XML-Konfigurationsdateien, die die Validierung definieren. In `validator-rules.xml` vom Framework vorgefertigte Standard-Validatoren bereitgestellt:

- required
  prüft, ob die Pflichtfelder eines Formulars ausgefüllt sind.
- validwhen
  Validator, der ein Feld in Abhängigkeit eines anderen überprüft.
- minlength
  überprüft die Eingabedaten auf eine Mindestlänge.
- mask
  validiert die Eingabe auf einen angegeben regulären Ausdruck
- ...

Trotz der vorkonfigurierten Regeln ist das Framework sehr flexibel und erweiterbar. Der Anwender kann jederzeit eigene auf die Applikation zugeschnittene Regeln definieren und diese in `validator-rules.xml` festlegen.

Konfiguriert wird die Validierung über die zweite Konfigurationsdatei `validation.xml`, aus der die Validierungsregeln für die einzelnen Felder der Eingabeform generiert werden. Hierbei wird für jede Form in der Präsentationsschicht angegeben, ob und wenn ja, welche Felder mit welchem Validator geprüft werden sollen.

Der Einsatz des Validator Frameworks bringt erhebliche Vorteile mit sich. Durch die lose Kopplung des Frameworks an die Applikation, kann man die Validierungslogik ändern, ohne JSP- oder Javacode bearbeiten zu müssen. Auch die Integration in Struts und somit die gemeinsame Nutzung von Property-Dateien, z.B. zum Anzeigen von Fehlermeldungen, erhöht das Maß an Effektivität bei der Entwicklung einer Applikation.

7 Realisierung

7.1 Aufgabenstellung

Die Generierung einer MVC-Präsentationsschicht sollte anhand einer bereits bestehenden und im Einsatz befindlichen Applikation durchgeführt werden. Die Aufgabenstellung stammt aus dem Projekt IVS-R (International Vehicle System Reengineering) der BMW AG. Softlab GmbH war im Rahmen dieses Projekts unter anderem für die Entwicklung von IVS-R DealerMasterdata, einer Anwendung zur Pflege von Stammdaten mit typischer CRUD-Funktionalität (create, read, update, delete), verantwortlich.

Stammdaten sind Daten, die über längere Zeit nicht verändert und bei der Abwicklung von Geschäftsvorfällen verwendet werden. Sie sind von großer Bedeutung für Unternehmen, da sich in den Stammdaten die Kerninformationen von Unternehmen wiederfinden und diese in zahlreichen Anwendungen und diversen Informationssystemen in unterschiedlichem Kontext wiederverwendet werden können.

Folgende Anforderungen werden an das System gestellt:

- Anlegen, Editieren und Löschen von Datensätzen
- Suchen von Datensätzen
- Web-Oberfläche als Frontend
- Validierung der Eingabedaten

7.2 Zielarchitektur

Abbildung 7-1 zeigt die Zielarchitektur des zu generierenden Systems. Das Struts-Framework nimmt die Anfragen des Clients entgegen und delegiert auf entsprechende Aktionen in der Geschäftslogik. Der Zugriff von der Präsentationsschicht auf die Geschäftslogik wird durch die J2EE Design Patterns [Sun02] Business Delegate und Service Locator gekapselt.

Es handelt sich um eine dreischichtige Client-Server Architektur auf Basis der Java 2 Enterprise Edition (J2EE). Das Backend implementiert die BMW Component Architecture und wurde im Rahmen der Diplomarbeit [Mär01] generiert. Die BMW Component Architecture (CA) definiert die System-Architektur bei BMW und gibt ein einheitliches Konzept für verteilte Anwendungen vor. Das in der Diplomarbeit generierte Frontend knüpft nahtlos an die bestehende Architektur an, so dass am Ende der beiden Diplomarbeiten eine komplette dreischichtige J2EE-Anwendung generiert werden kann. Auf der ursprünglichen IVS-R-Clientanwendung kommen JavaServer Pages in Kombination mit Java Servlets zum Einsatz, diese werden im Generat durch das Apache Struts Framework abgelöst. Die relativ simplen und wiederkehrenden Anwendungsfälle, wie das Anlegen, Lesen und Löschen von Stammdaten, sprechen zusätzlich für den Einsatz eines Generators.

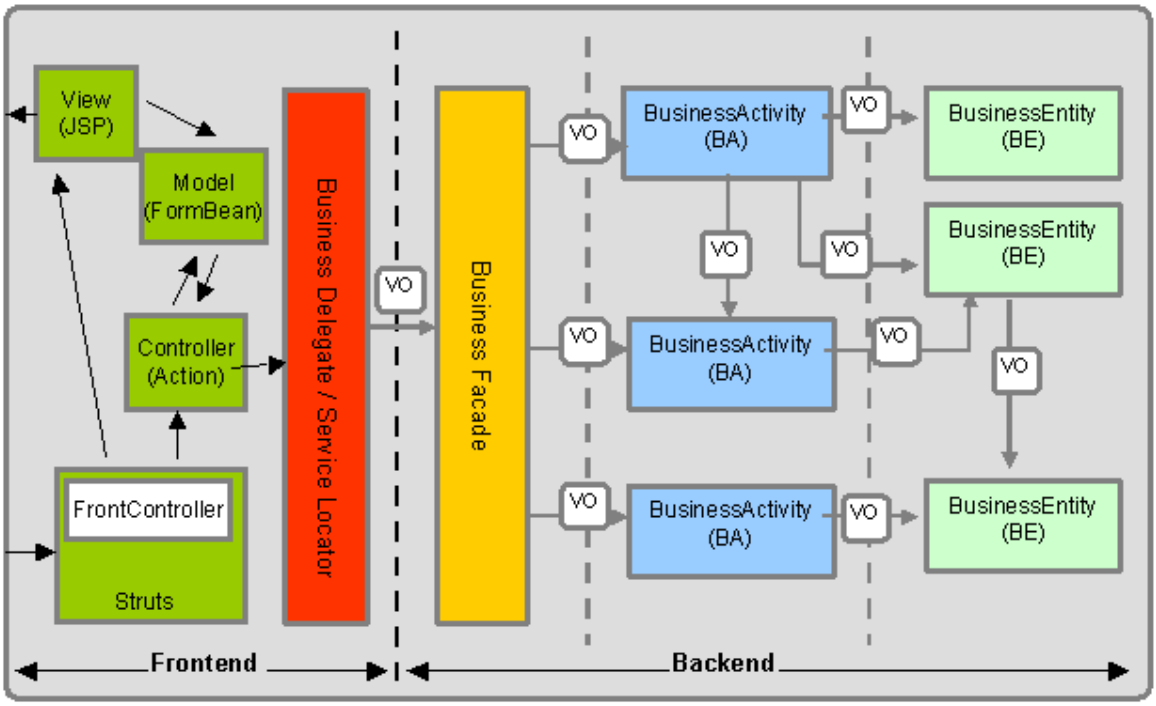


Abbildung 7-1 - Architektur der zu generierenden Anwendung

Aus obenstehender Abbildung lassen sich die einzelnen zu generierenden Artefakte ableiten:

- Action Klassen
- FormBeans
- JavaServer Pages
- BusinessDelegate
- ServiceLocator
- Konfigurationsdateien (nicht in der Abbildung sichtbar):
  - struts-config.xml
  - tiles-definitions.xml
  - validation.xml
  - validator-rules.xml
  - web.xml

Der Kern eines User Interface, das mit dem Struts Framework realisiert ist, ist die Konfigurationsdatei struts-config.xml. Die Ablaufsteuerung wird deklarativ in XML festgelegt, beim Start der Web-Anwendung eingelesen und anschliessend als Mapping-Tabelle im Speicher gehalten. Somit ist der Ablauf der Anwendung komplett unabhängig vom Sourcecode, definiert in einer XML-Datei.

Da mit zunehmender Größe der Anwendung auch die Komplexität der zentralen Konfigurationsdatei erheblich ansteigt, ist eine elementare Anforderung an den

Generierungsmechanismus, struts-config.xml komplett zu generieren. Auch die restlichen Klassen und Konfigurationsdateien sollen zu einem möglichst großen Anteil generiert werden.

Nachfolgend wird die pragmatische Umsetzung von MDA light im Projekt, die Arbeit mit dem Generator und der Projektaufbau detailliert beschrieben.

7.3 Laufzeitumgebung

Auf den Aufbau der Laufzeitumgebung wird im Folgenden nur kurz eingegangen, da er in [Mär01] recht ausführlich dokumentiert ist.

Als Resultat der zwei Diplomarbeiten entsteht ein Generat, in Form einer lauffähigen J2EE-Anwendung. Aus diesem Grund bleibt die Auswahl der Laufzeitumgebung, im Vergleich zur Diplomarbeit mit dem Thema der Generierung des Backends, gleich.

Als Application-Server kommt der Weblogic Application Server (WAS) Version 8.1 der Firma Bea Systems Inc. zum Einsatz. Der WAS ist eine kommerzielle komponentenbasierte Plattform in einer serverzentrierten Architektur. Bei der Erstellung Client-Server Architektur in J2EE-Anwendungen kommen häufig mehrschichtige Architekturen zum Einsatz, der WAS bietet hierbei sowohl einen Web-Container für die Präsentations-Schicht, als auch einen EJB-Container für die Persistenz-Schicht.

Die Datenbank persistiert relevante Daten, die vom Anwender der Applikation auf der Seite des User Interface eingegeben werden. Bei der Diplomarbeit kommt das relationale Datenbanksystem DB2 Version 7 der IBM Corporation zum Einsatz.

Beide Produkte, Application Server und Datenbanksystem sind weit verbreitet bei der BMW AG im Einsatz, auch beim Projekt IVS-R, in dem das zu generierende Stammdatenverwaltungssystem ursprünglich implementiert wurde.

7.4 Entwicklungsansatz

7.4.1. Werkzeuge

Eclipse IDE

Als Plattform für die Anwendungsentwicklung und Projektstrukturierung wurde die integrierte Umgebung Eclipse in der Version 2.1 eingesetzt. Eclipse (entwickelt als opensource-Projekt unter der Führung der IBM Corporation) ist eine offene, erweiterbare Entwicklungsplattform, die es über einen ausgereiften Erweiterungsmechanismus erlaubt, Werkzeuge aller Art und technologiespezifische Erweiterungen einzubinden. Das Werkzeug kann so über frei erhältliche oder eigenentwickelte Plugins an die spezifischen Bedürfnisse des Projekts optimal angepasst werden. Eclipse bietet volle Unterstützung für die Entwicklung in Java, inklusive ausgereiftem Editor und Debugger, sowie integrierte Build- und Testunterstützung (Ant, Junit).

Poseidon For UML

Poseidon For UML ist ein vollwertiges Werkzeug der Firma Gentleware AG[1] aus Hamburg zur Erstellung von UML-Diagrammen. Bei der Entwicklung des Anwendungsdesigns kam die

1 Gentleware AG - http://www.gentleware.com (Stand: 20.09.2004)

Version 2.4.1 zum Einsatz. Das Werkzeug unterstützt die Diagramme der UML-Spezifikation 1.4 und bietet die Möglichkeit, das Design in Form von XMI zu exportieren. Leider sind bei der Arbeit mit der frei erhältlichen Community-Edition einige Schwierigkeiten in Zusammenhang mit der Umsetzung der UML-Spezifikation im Werkzeug und der grafischen Modellierung aufgetreten.

open ArchitectureWare

Wie in Kapitel 5 beschrieben, ist open ArchitectureWare ein offenes metamodellbasiertes Generator-Framework zur Generierung von Code direkt aus dem plattformunabhängigen Design. Aufgrund der stetigen Weiterentwicklung des Generator Frameworks wechselte die verwendete Version mehrmals, bei Abschluss der Arbeit befindet sich die Version 3.0 Release Candidate 1 im Einsatz. Das Projekt wird auf der OpenSource-Plattform SourceForge unter der LGPL[1] veröffentlicht.

7.4.2. Analyse und Lösungskonzept

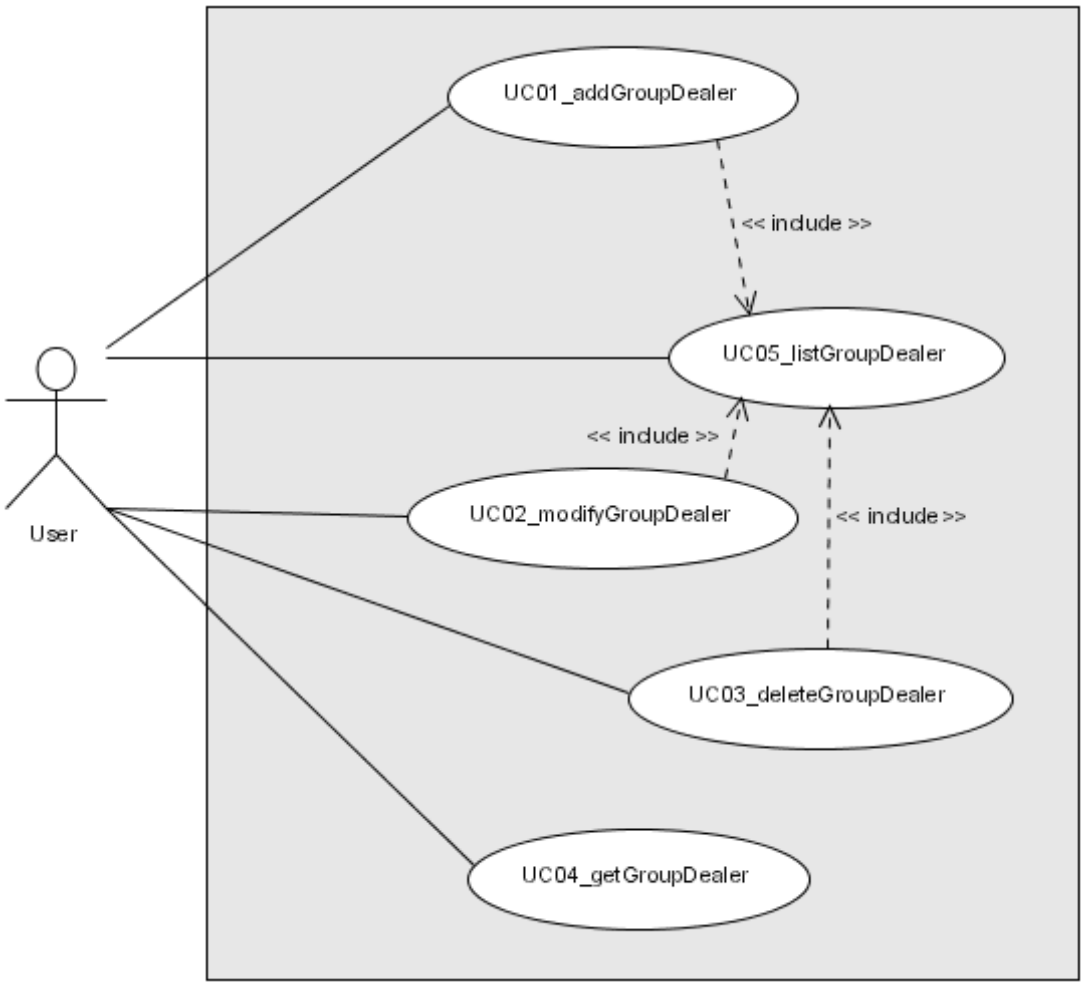


Abbildung 7-2 - UseCase Diagramm der Stammdatenanwendung

[1] Lesser General Public License - http://www.gnu.org/copyleft/lesser.html (Stand: 20.09.2004)

Die obenstehende Abbildung 7-2 zeigt einen Ausschnitt des UseCase-Modells der Stammdaten-Anwendung, die in der Diplomarbeit generiert werden soll. Beziehungen zwischen Akteuren und Anwendungsfällen werden aus Sicht der Anwender dargestellt. UseCase-Diagramme repräsentieren vorwiegend die Darstellung der Anforderungen und dienen weniger der Verhaltensbeschreibung und dem Systemdesign.

Die für eine Stammdatenanwendung typische CRUD-Funktionalität (create, read, update, delete) ist schon im Usecase-Diagramm zu erkennen.

Im Laufe der Entwicklung des Systems, haben sich zwei verschiedene Lösungsansätze zur Generierung des User Interface durchgesetzt, die im Folgenden vorgestellt werden:

1. Aktivitäts-/Zustandsdiagramm:

Wie oben erwähnt, wird das System in der Analysephase anhand von Anwendungsfällen (UseCases) und deren Abhängigkeiten untereinander beschrieben. Da es die Syntax eines UseCase-Diagramms nicht erlaubt, Ablaufbeschreibungen zu modellieren, werden die einzelnen Abläufe und Übergänge zwischen den Anwendungsfällen in einem Aktivitätsdiagramm notiert.

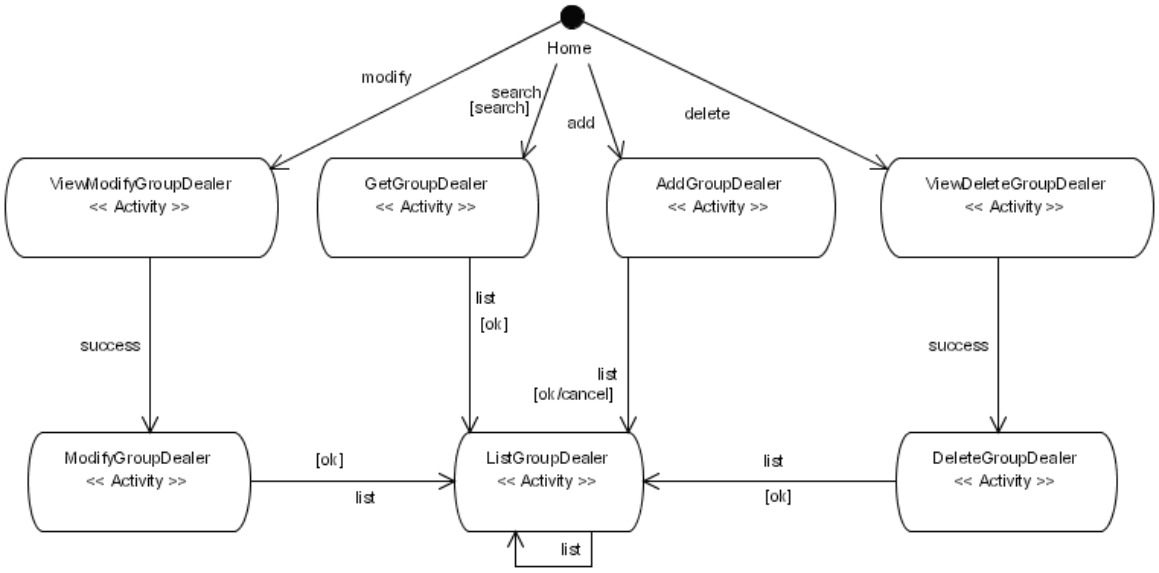


Abbildung 7-3 - Ablaufbeschreibung in Form eines Aktivitätsdiagramms

Die statische Sicht, die in einem UseCase-Diagramm modelliert wird, wird durch ein Aktivitätsdiagramm verfeinert und Interaktionen und Übergänge zwischen den einzelnen Anwendungsfällen modelliert. Aktivitätsdiagramme beschreiben Ablaufmöglichkeiten, die aus einzelnen Aktivitäten (Schritten) bestehen.

Da sich eine einzelne Aktivität potentiell aus mehreren Aktionen zusammensetzen kann, hat man zusätzlich die Möglichkeit, eine einzelne Aktivität, über ein Zustandsdiagramm, im Detail zu spezifizieren. Die Verknüpfung zwischen Aktivität und Zustandsdiagramm erfolgt über den Namensraum. In einem Zustandsdiagramm hat man die Möglichkeit, mehrere Aktionen, aus denen sich die Aktivität zusammensetzen kann, zu spezifizieren. Auch Details wie Transitionen zwischen den Aktionen und verschiedene Endpunkte lassen sich in einem

Zustandsautomaten detailliert beschreiben. Zusätzlich werden Eigenschaften, die für die anschließende Codegenerierung von Bedeutung sind (z.B. ob aus einer Aktivität eine Struts Action generiert werden soll) im Modell in Form von Tagged Values angegeben.

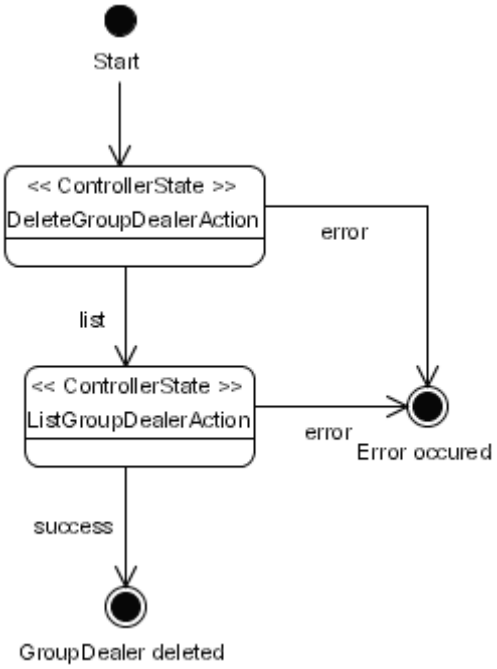


Abbildung 7-4 - Zustandsdiagramm zur detaillierten Beschreibung der Aktivität 'DeleteGroupDealer'

Im folgenden Kapitel wird detailliert auf das Design des plattformunabhängigen Modells für die Code-Generierung eingegangen.

2. Zustandsdiagramm:

Nachdem mit der oben beschriebenen Lösung die Anwendung bereits vollständig generiert werden konnte, richtete sich der Fokus der Entwicklung auf die Modellierung des Ablaufs der Applikation in einem Diagramm. Die Konzepte, die bisher in einem Aktivitätsdiagramm und mehreren Zustandsdiagrammen definiert waren, sollten in einem einzelnen Interaktionsdiagramm untergebracht werden, um den Designaufwand zu verringern und die Entwicklung des PIM intuitiver zu gestalten.

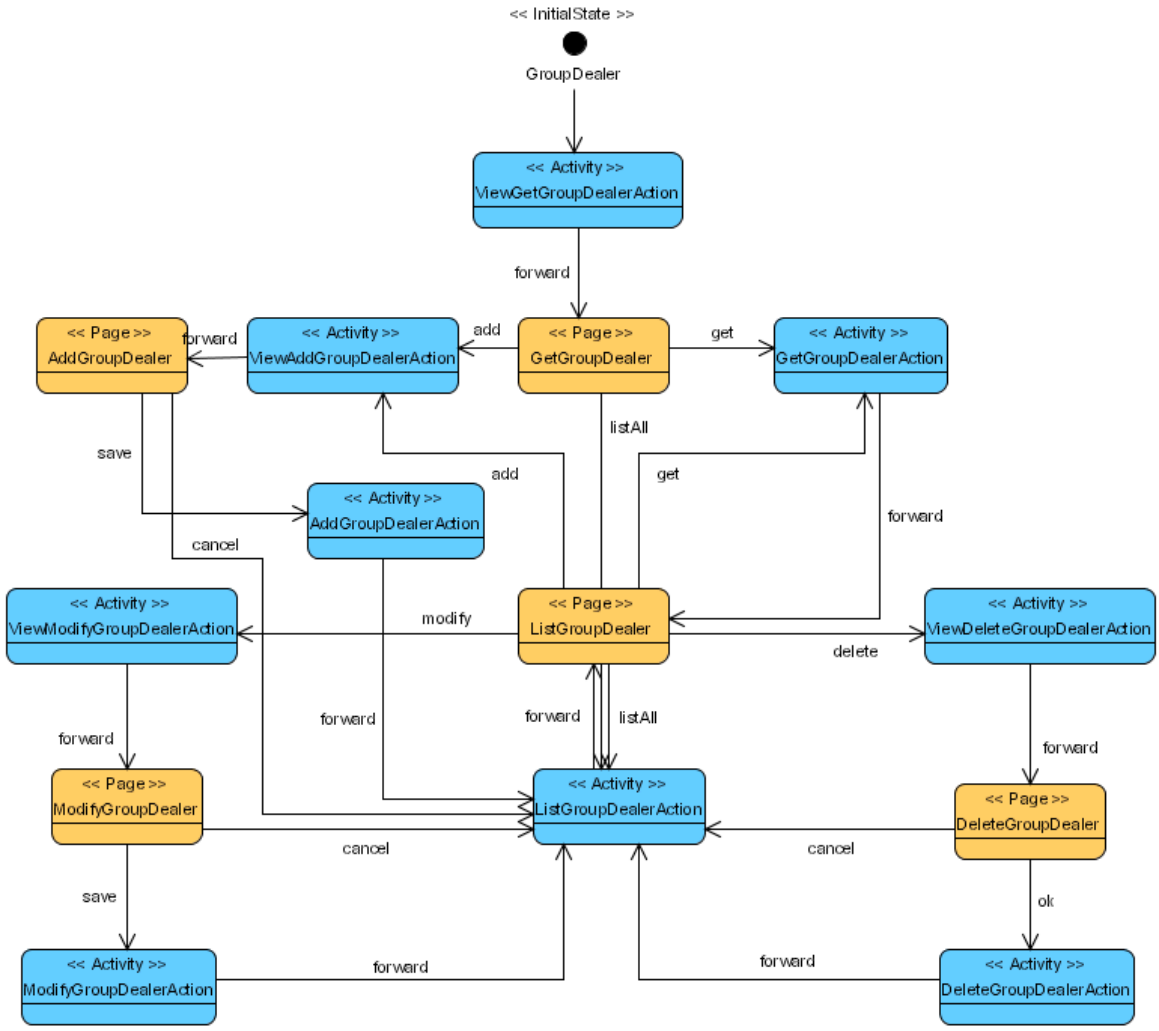


Abbildung 7-5 - Zustandsdiagramm zur Ablaufbeschreibung der Anwendung

In einem Zustandsdiagramm (siehe Abbildung 7-5) werden Bildschirmseiten und Aktivitäten dargestellt und stellen somit eine intuitive Sicht auf das System dar. Der Ablauf einer Präsentationsschicht kann anhand eines Zustandsautomaten modelliert werden, der auf Ausführung der Aktivitäten, teilweise getriggert durch User-Eingaben, reagiert und daraufhin seinen Zustand ändert. Obenstehende Abbildung stellt den Zustandsautomaten für die zu generierende Stammdatenanwendung dar. Sowohl `Pages`, als auch `Activities` sind in Form von Zuständen modelliert. Aus den Übergängen, die die einzelnen Zustände miteinander verbinden wird der Ablauf der Webapplikation definiert. Daraus werden dann im Generierungsschritt unter anderem sowohl die `struts-config.xml`, als auch Hyperlinks, die die einzelnen JavaServer Pages miteinander verbinden, generiert. In einem Komponenten Diagramm, das mit dem Zustandsautomaten verknüpft ist, wird detailliert der Zusammenhang und die einzelnen Felder der JSPs, ActionForms und Actions, im Falle von Struts modelliert (siehe Kapitel 7.7)

3. Zusammenfassung:

Die zwei oben vorgestellten Lösungsansätze gehen beide in das während der Diplomarbeit erarbeitete Endergebnis ein. Die intuitive und relativ einfache Modellierung eines User Interface als Zustandsautomat, wie im zweiten Beispiel beschrieben, erscheint für die Generierung einer Präsentationsschicht besser geeignet, da alle Informationen in einem Diagramm vorliegen und auch der Ablauf der Applikation, die Übergänge zwischen den einzelnen Bildschirmseiten klar zu erkennen ist. Der Umfang, der in der Diplomarbeit generierten Anwendung, ist, verglichen mit komplexen Geschäfts-Anwendungen, recht gering. Da eine Anwendung größeren Ausmaßes mit komplexer Fachlichkeit nicht, oder nur sehr schwer, in einem einzelnen Zustandsdiagramm abbildbar ist, kommt hier die im ersten Beispiel beschriebene Lösung zum Einsatz. Mithilfe einer oder mehrerer Aktivitätsdiagramme hat man die Möglichkeit, Fachlichkeiten und Prozesse, des Problemfeldes abzubilden. Durch die Verbindung der einzelnen Aktivitäten mit Zustandsdiagrammen (Zoom In) über den Namensraum kann so eine relativ grobgranulare Sicht auf das System im Aktivitätsdiagramm, durch geschickte Beschreibung mithilfe einzelner voneinander losgelöster Zustandsdiagramme, die ihrerseits ein ganzes Modul beschreiben, verfeinert werden.

Die Komplexität der in der Diplomarbeit zu generierenden Stammdatenanwendung lässt sich in einem einzigen Zustandsdiagramm abbilden, somit wird im Folgenden hauptsächlich der zweite Lösungsansatz beschrieben. Eine Umsetzung der oben beschriebenen Zusammenfassung der beiden Lösungsansätze ist jedoch relativ problemlos möglich, da die Metamodell-Klassen im Rahmen der Lösungsfindung bereits entwickelt wurden.

7.5 Referenzimplementierung

Das in Kapitel 5.3 beschriebene Vorgehensmodell GDP - Generative Development Process - wurde während der generativen Entwicklung der Stammdatenanwendung eingesetzt. Die Referenzimplementierung setzt sich aus zwei unterschiedlichen Teilen zusammen, zum einen die Entwicklung eines statischen HTML-Modells, zum anderen die Umsetzung des HTML-Modells in Struts.

7.5.1. Statisches HTML-Modell

Um den Funktionsumfang und das Look-and-Feel der Präsentationsschicht zu definieren; wurde im ersten Schritt ein statisches HTML-Modell der Anwendung erstellt. Eine Reihe verlinkter HTML-Seiten, gefüllt mit Testdaten, zeigen den Ablauf der Anwendung auf und definieren den grundlegenden Aufbau einer Bildschirmseite. Anhand dieses Modells wurden Schlüsselelemente einer Benutzeroberfläche, wie zum Beispiel Navigation oder Seitengestaltung, definiert. Das HTML-Modell verdeutlicht das Aussehen der Anwendung und wird im nächsten Schritt in die Zieltechnologie (JSP, JSF, ...) überführt. Abbildung 7-6 zeigt den grundsätzlichen Aufbau einer Seite der Anwendung.

Bewusst wurde während der Entwicklung des HTML-Modells, auf den Einsatz von Standard-Technologien geachtet, so wurden zum Beispiel der Einsatz von Javascript (deaktivierbar) oder Frames (verkompliziert die Übertragung von Formulardaten über Frame-Grenzen hinweg) vermieden, um die Applikation nicht von Browsereinstellungen oder firmeninternen Sicherheitsrichtlinien abhängig zu machen.

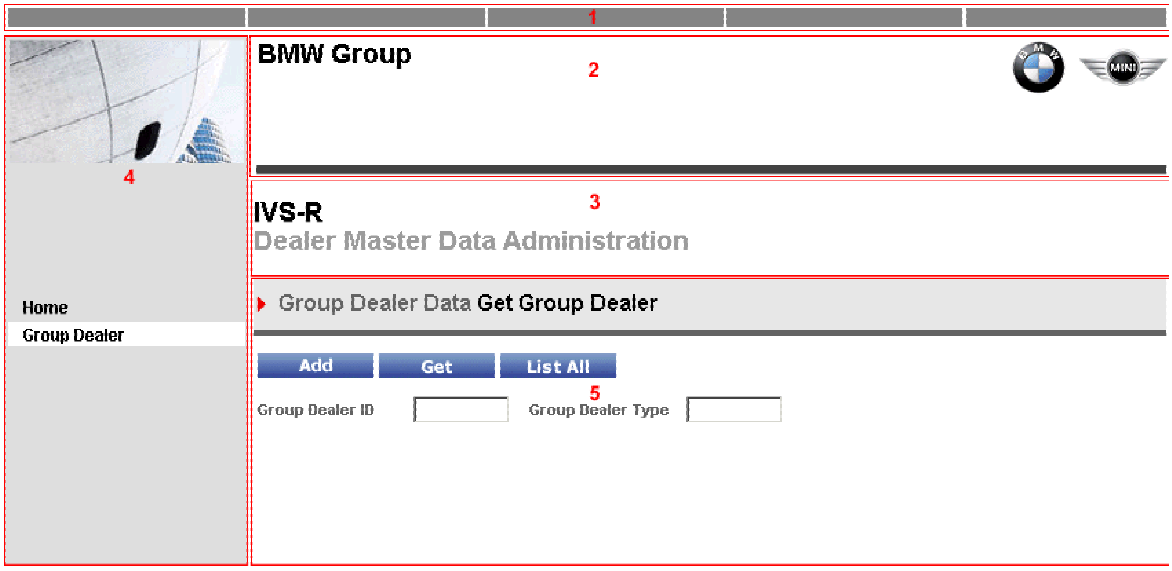


Abbildung 7-6 - Eine Bildschirmseite des statischen HTML-Modells der Anwendung

Die Benutzeroberfläche ist grundsätzlich in fünf verschiedene Bereiche aufgeteilt:
(vgl. Nummerierung in Abbildung 7-6):

1. Meta-Navigation
Kann optional mit applikations- oder kontextspezifischen Links in den einzelnen Kacheln befüllt werden.

2. Titel
Bezeichnet den Titel, der nicht zwingend erforderlich applikationsspezifisch sein muss.

3. Subtitel
Enthält den Titel der Applikation.

4. Navigation
Applikationsspezifische Navigationsleiste.

5. Inhalt
Enthält sowohl eine kontextspezifische Überschrift als auch eine Leiste mit Schaltflächen, die, ebenfalls kontextspezifisch, verschiedene Schaltflächen enthält. Darüber hinaus wird unterhalb Buttonleiste der eigentliche Inhalt der Anwendung dargestellt.

7.5.2. Umsetzung in Struts

Auf die Erstellung des statischen HTML-Modells und die damit verbundene Definition des Look-and-Feel der Präsentationsschicht, folgt die exemplarische Umsetzung des HTML-Modells unter Einsatz des Struts Frameworks.

Das Frontend setzt auf die bereits bestehende Geschäftslogik auf. Bei der Implementierung wurde verstärkt auf eine lose Kopplung zwischen Präsentationsschicht und Backend geachtet. Dies wurde durch den Einsatz der J2EE Patterns BusinessDelegate und ServiceLocator erreicht. Das BusinessDelegate Pattern kapselt die zugrundeliegende Implementierung der Geschäftslogik, zum Beispiel die Implementierungsdetails der EJB Architektur. Ein ServiceLocator Objekt abstrahiert den Einsatz des Java Naming Directory

and Interface (JNDI) und verbirgt die Komplexität, die beim Erzeugen eines Kontexts und Auffinden von EJB Objekten schnell entsteht. Darüber hinaus wurde bei der Umsetzung des HTML-Modells auf eine strikte Einhaltung des Model-View-Controller Paradigmas geachtet (vgl. Kapitel 6.1).
Abbildung 7-7 zeigt die einzelnen Java Pakete der Struts-Anwendung.

Abbildung 7-7 - Paktierung der Struts Anwendung

- `de.softlab.ivsr.struts.BD`:
  Enthält Klassen, die die J2EE Patterns ServiceLocator und BusinessDelegate implementieren.
- `de.softlab.ivsr.struts.action`:
  Enthält die Action-Klassen, die Teil des Controllers im MVC-Paradigma sind. Alle Action-Klassen erben von einer Basis-Klasse (`BaseAction.java`), die den Action-Klassen potentiell allgemeine Funktionalität zur Verfügung stellen kann. Zusätzlich enthält die Basisklasse zwei Methoden (protected), die unmittelbar vor und nach der eigentlichen Aktion ausgeführt werden und von den erbenden Klassen implementiert werden können.
  Listing zeigt die zentrale Methode execute, die in der Basisklasse abstrakt implementiert ist und somit von allen erbenden Klassen implementiert werden muss.

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response)
throws Exception;
```

Listing 7-1 - zentrale Methode der Action-Klasse

  Aufgabe einer Action-Klasse ist, eine eingehende Anfrage über die `execute` Methode zu bearbeiten und ein `ActionForward` Objekt, das den Kontrollfluss der Applikation steuert (Delegation an eine JSP, Tiles Definition, andere Action Klasse), zurückzugeben. Die Action-Klassen lesen die Formulardaten zur weiteren Verarbeitung aus der ActionForm und rufen die jeweiligen Geschäftsprozesse (z.B. addGroupDealer, deleteGroupDealer) über die BusinessDelegate-Klasse auf.
- `de.softlab.ivsr.struts.form`:
  Die Daten, die vom Anwender in ein HTML-Formular eingegeben werden (zum Beispiel 'User anlegen'), werden in einer Struts `ActionForm` gespeichert. Eine `ActionForm` ist gewöhnlich in Form einer JavaBean realisiert, d.h. die Klasse stellt Methoden zur

Verfügung, die den Zustand des Formulars halten und schreibenden sowie lesenden Zugriff auf die Werte erlauben. `ActionForms` können sowohl im Anwendungsbereich der Session als auch im Request gehalten werden. Das Struts Framework belegt die Werte einer `ActionForm` aus den Request Parametern und schickt das Formular an die in der Konfigurationsdatei definierte Action-Klasse, die die erforderliche Aktivität ausführt.

Zusätzlich zu dieser Basisfunktionalität der ActionForms besteht die Möglichkeit einer Validierung der vom Anwender eingegebenen Formulardaten, unabhängig vom im Kapitel 6.2.5.2 beschriebenen Funktionalität des Validator Plugins:

```
validate(ActionMapping mapping,
HttpServletRequest request);
```

Listing 7-2 - Validierungs-Methode der ActionForm-Klasse

Die `validate` Methode wird nach der Bestückung der Felder der ActionForm und vor der Ausführung der `execute` Methode der ActionKlasse vom Controller Servlet aufgerufen. Die Methode greift auf die eingegebenen Werte zu und führt die manuell erstellten Prüfungen durch.

- `de.softlab.ivsr.struts.util:`
  Enthält Utility-Klassen, wie zum Beispiel eine Factory-Klasse, die eine Instanz des BusinessDelegate erzeugt, oder eine Klasse, die einen spezifischen Tag zur Anzeige von Texteingabefehlern in einer JavaServer Page definiert.
- `de.softlab.ivsr.struts.vo:`
  Enthält Klassen, die Value Objekte auf der Präsentationsebene definieren, um unabhängig vom zugrundeliegenden Geschäftsmodell, eventuelle Operationen auf den vom Anwender eingegebenen Daten ausführen zu können. (Aufgrund der Fachlichkeit der Stammdatenanwendung sind die ValueObjects auf der Präsentationsebene nur exemplarisch realisiert und finden in der Endfassung der Applikation keine Verwendung).
- `Resources:`
  Enthält eine Properties-Datei (`ApplicationResources.properties`), die die in Kapitel 6.2.4.2 beschriebene Funktionalität der Mehrsprachigkeit durch Auslagerung sämtlicher Fließtexte einer JavaServer Page in eine Property-Datei anwendet. In einer JSP wird über feste Schlüssel auf die Werte der Property-Datei, die `in struts-config.xml` definiert ist, zugegriffen.

Validierung:
Die Validierung der vom Anwender eingegebenen Daten erfolgt mithilfe des Struts Validator Plugins (vgl. Kapitel 6.2.5.2). Auf eine clientseitige Validierung per JavaScript, die das Validator Plugin unter anderem zur Verfügung stellt, wird aufgrund der oben beschriebenen Unabhängigkeit der Applikation von werkzeugabhängigen Einstellungen und Richtlinien verzichtet.

Seitenaufbau:
Der im HTML-Modell definierte Aufbau der Bildschirmseiten wird bei der Realisierung der Applikation mit dem Struts Framework ohne Einschränkungen umgesetzt. Zum Einsatz

kommt hierbei die in Kapitel 6.2.5.1 beschriebene Tiles Technologie. Tiles wird über ein Struts Plugin, das in struts-config.xml definiert wird, beim Start der Anwendung geladen. Anhand einer XML-Konfigurationsdatei (`tiles-definitions.xml`) wird die Zusammensetzung der einzelnen Bildschirmseiten deklarativ beschrieben. Man hat die Möglichkeit, ein Basislayout zu definieren, von dem die Bildschirmseiten erben können. Auf den einzelnen Seiten muss somit nur der jeweils variable Anteil neu definiert werden.

Listing 7-3 zeigt einen Ausschnitt der `tiles-definitions.xml` der Stammdatenanwendung

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tiles-definitions PUBLIC
        "-//Apache Software Foundation//DTD Tiles Configuration 1.1//EN"
        "http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">

<tiles-definitions>
        <definition name="baseDefinition" page="/view/layout/layout.jsp">
                <put name="titleText" value="IVS-R Dealer Master Data"/>
                <put name="meta-nav" value="/view/tiles/base/meta-nav.jsp"/>
                <put name="title" value="/view/tiles/base/title.jsp"/>
                <put name="subtitle" value="/view/tiles/base/subtitle.jsp"/>
                <put name="nav" value=""/>
                <put name="content" value=""/>
        </definition>

        <definition name="Error" extends="baseDefinition">
                <put name="nav" value="/view/tiles/Error/navigation.jsp"/>
                <put name="content" value="/view/tiles/Error/content.jsp"/>
        </definition>

        <definition name="Home" extends="baseDefinition">
                <put name="nav" value="/view/tiles/Home/navigation.jsp"/>
                <put name="content" value="/view/tiles/Home/content.jsp"/>
        </definition>

        <definition name="GetGroupDealerTDef" extends="baseDefinition">
                <put name="nav" value="/view/tiles/GetGroupDealer/navigation.jsp"/>
                <put name="content" value="/view/tiles/GetGroupDealer/content.jsp"/>
        </definition>

        ...
```

Listing 7-3 - Auszug aus der Tiles Konfigurationsdatei

Im Bereich der `baseDefinition` werden einzelnen Bereiche der Bildschirmseite, die im HTML-Modell angelegt wurden, definiert. Über das Schlüsselwort `extends` übernehmen, alle vom Basislayout abgeleiteten Definition den grundlegenden Seitenaufbau und ergänzen diesen nur noch um die auf jeder Seite variablen Anteile. Im Fall der Stammdatenanwendung werden somit Meta-Navigation, Titel und Subtitel im Basislayout festgelegt, Inhalt und Navigation werden im Definitionsbereich der einzelnen Seiten geregelt.

7.6 Metamodellierung

7.6.1. Java Metamodell

Algorithmische Anteile, die zur Generierung notwendig sind, werden beim Einsatz von open ArchitectureWare in der Programmiersprache Java im Metamodell programmiert. Die Metaprogrammierung (Template-Sprache + Java-Metamodell) ist damit insbesondere auch programmiersprachlich klar von der Anwendungsentwicklung getrennt. Dadurch lassen sich beliebige Zielsprachen unterstützen. Die Fachlogik wird von den Entwicklern manuell ergänzt und bleibt bei erneuter Generierung erhalten.

Nach Fertigstellung der Referenzimplementierung in der gewählten Anwendungssprache, unter Einsatz der erforderlichen Frameworks, gilt es nun im nächsten Schritt, wiederholt auftretende Muster in der Referenzimplementierung zu identifizieren und im darüberliegenden Metamodell zu abstrahieren.

Wie in Kapitel 5.2 bereits beschrieben, wird vom Generator Framework eine Java-Implementierung des Metamodells eingelesen. Im Lieferumfang von open ArchitectureWare befindet sich bereits eine Implementierung des Metamodells der UML 1.4 in Java. Open ArchitectureWare schreibt keine spezielle Metamodell-Implementierung vor, was dem Entwickler eine freie Definition des Metamodells erlaubt, lediglich eine Umsetzung des Metamodells in Java ist erforderlich. Eine Erweiterung des Metamodells, d.h. die Erstellung eines UML Profils, für die jeweilige Problemdomäne ist daher einfach durch Erzeugen von Java-Klassen, die das mitgelieferte Java-Metamodell erweitern, zu bewerkstelligen. Das Java Metamodell orientiert sich an der OMG Spezifikation zu UML [OMG01], wurde aber vereinfacht, da nicht alle Modellierungselemente für den praktischen Einsatz notwendig sind.

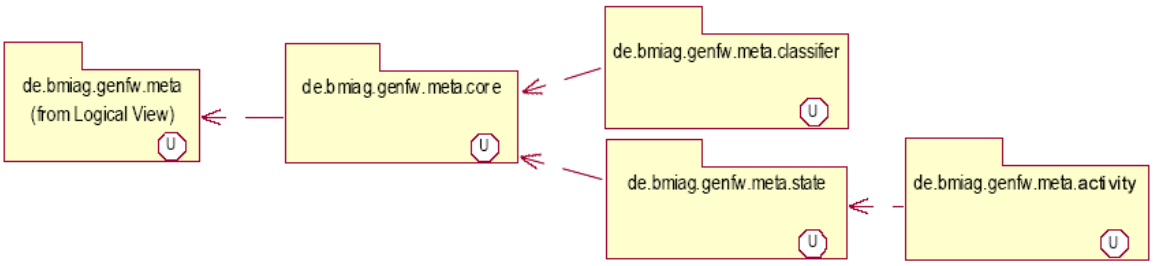


Abbildung 7-8 - Die fünf Basis Java-Pakete der Metamodell-Implementierung

Abbildung 7-8 zeigt die vier grundlegenden Java-Pakete und Ihre Abhängigkeiten laut [b+m02]:

- `de.bmiag.genfw.meta`:
  Enthält die Basistypen aller Metamodell-Klassen und stellt das Typsystem des Generator Frameworks dar.
- `de.bmiag.genfw.core`:
  Das core-Package enthält die UML-Basisstrukturen und den Identifier-Mechanismus, der die Unterstützung von Namenskonventionen in Form von Name- und Documentation-Properties bereitstellt.

- `de.bmiag.genfw.classifier:`
  Enthält die Basis-Elemente, die für die Unterstützung von Klasssendiagrammen in open ArchitectureWare nötig sind.
- `de.bmiag.genfw.state:`
  Enthält das mit dem Generator ausgelieferte Metamodell für die in Zustandsdiagrammen verwendeten Automaten.
- `de.bmiag.genfw.activity:`
  Enthält Elemente, die Unterstützung bei der Generierung aus Aktivitätsdiagrammen bietet. Aktivitätsdiagramme sind, wie auch im Diagramm zu erkennen, in der UML 1.4 eine Spezialisierung von Zustandsdiagrammen.

Abbildung 7-9 zeigt den Aufbau des Java Metamodells für Zustandsdiagramme:

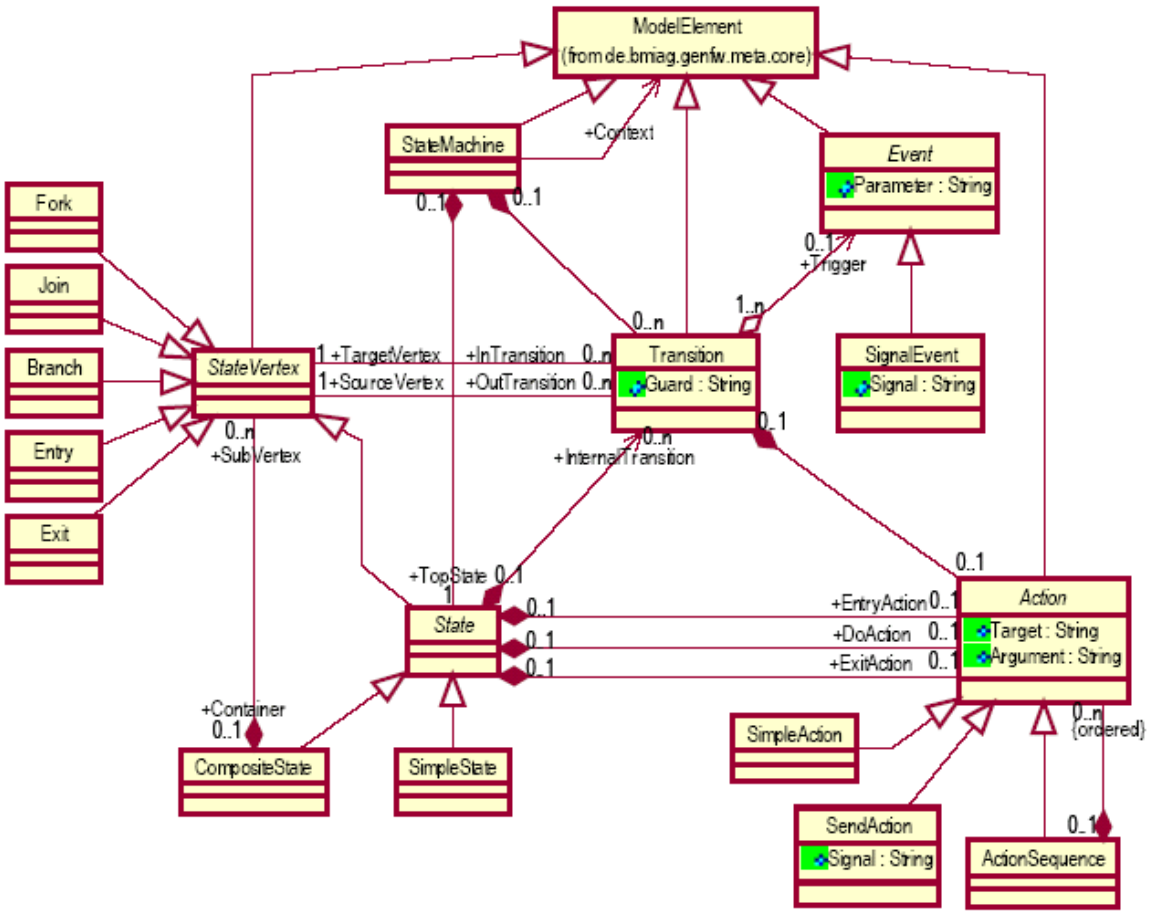


Abbildung 7-9 - Ausschnitt des Metamodells (Zustandsdiagramm)

7.6.2. Erweiterung des Java Metamodells

7.6.2.1. Fachliche Erweiterungen

Für die Generierung des Frontends wird das oben beschriebene Java Metamodell (Java-Implementierung eines vereinfachten UML-Metamodells) um kontextspezifische Elemente bzw. Klassen erweitert, es wird ein sogenanntes UML Profil erstellt. Die in Kapitel 4.2.1.1 beschriebenen UML Erweiterungsmechanismen kommen hierbei zum Einsatz.

Folgende Erweiterungen des Metamodells liegen als Java-Klassen vor und kommen im plattformunabhängigen Design in Form von Stereotypen zum Einsatz(siehe folgendes Kapitel 7.7):

- `System`:
  Erweitert `de.bmiag.genfw.meta.classifier.Component` und bildet die Grundkomponente bei der Modellierung und Generierung der mehrschichtigen J2EE-Applikation. Die mit dem Stereotyp «`System`» gekennzeichnete Komponente stellt den Gesamtüberblick über die Applikation dar und enthält weitere Komponenten vom Typ «`UIComponent`» (repräsentiert die Präsentationsschicht) und «`BusinessComponent`» (repräsentiert die Schicht der Geschäftslogik).

  `BusinessComponent()`: Gibt eine Menge von BusinessComponents zurück, die in der System-Komponente enthalten sind.

  `UIComponent()`: Gibt eine Menge von UIComponents zurück, die in der System-Komponente enthalten sind.

- `UIComponent`:
  Erweitert `de.bmiag.genfw.meta.classifier.Component` im Metamodell und repräsentiert die statische Sicht auf das Frontend in Form einer UML-Komponente.

  `SystemController()`: Gibt eine Menge von SystemControllern zurück, die in UIComponent enthalten sind.

  `Controller()`: Gibt eine Menge von Controllern zurück, die in UIComponent enthalten sind.

  `Presentation()`: Gibt eine Menge von Presentation-Objekten zurück, die in UIComponent enthalten sind.

  `View()`: Gibt eine Menge von View-Objekten zurück, die in UIComponent enthalten sind.

- `SystemController`:
  Die Klasse `SystemController` ist die Implementierung des gleichnamigen Stereotypen. Sie ist Bestandteil der Komponente «`UIComponent`» im Komponentendiagramm. Über den Namensraum eines SystemControllers wird die Verbindung zwischen der statischen Sicht im Komponentendiagramm und der dynamischen Ablaufbeschreibung im Zustandsdiagramms modelliert. Die einzelnen

Modellelemente (Zustände) aus dem Zustandsautomaten sind dementsprechend im Komponentendiagramm in Form von Klassen für die Generierung modelliert.

- `Controller`:
  Die Klasse Controller repräsentiert den gleichnamigen Stereotypen im Java Metamodell. Ein Controller in der Komponente `UIComponent` ist das Gegenstück zu einem Zustand mit dem Stereotyp «`Activity`», modelliert im Zustandsdiagramm. Der Controller steht, wie der Name deutlich macht, für den Kontrollteil des MVC-Paradigma im Komponentendiagramm.

  `Activity()`: Die Methode `Activity()` iteriert über die modellierten Elemente der Meta-Umgebung und liefert den modellierten gleichnamigen Zustand mit dem Stereotyp `Activity`.

- `Presentation`:
  Der Tagged Value 'PresentationClass' einer Activity im Zustandsdiagramm markiert, ob der entsprechenden Aktivität ein FormBean zur Speicherung der vom Anwender eingegebenen Daten, existiert. Falls dies zutrifft, so wird im Komponentendiagramm eine Klasse mit dem Stereotyp Presentation modelliert, die als Datenspeicher von Formulardaten fungiert. Zusätzlich hat man die Möglichkeit in einer Presentation-Klasse im Komponentendiagramm, wiederum in Form eines Tagged Values (BEClass), anzugeben, ob die einzelnen Felder der generierten Klasse, den Einträgen einer Entität aus der Datenbank entsprechen und daraus generiert werden sollen, oder, ob die Attribute im Diagramm modelliert und daraus generiert werden.

  `BusinessEntity()`: Gibt, falls der Tagged Value BEClass gesetzt ist, die entsprechende Business-Entität zurück.

- `View`:
  Die Klasse `View` entspricht einer Bildschirmseite und repräsentiert somit einen Zustand `Page`, der im Zustandsdiagramm modelliert ist. Attribute einer View-Klasse werden auf der Bildschirmseite in Form von Textfeldern eines HTML-Formulars dargestellt. Views können Abhängigkeiten zu Presentation-Klassen besitzen. Ist eine Dependency modelliert, so werden die Attribute der Presentation-Klasse auf der Bildschirmseite übernommen, alternativ können auch eigene Attribute und somit Textfelder gesetzt werden.

  `Page()`: Iteriert über sämtliche Modellelemente und gibt in Abhängigkeit des View-Namens den entsprechenden Zustand 'Page' zurück.

  `InitialState()`: Liefert den InitialState (siehe unten) eines Zustandsdiagramms.

  `Presentation()`: Gibt diejenige Presentation-Klasse zurück, die eine Abhängigkeit zu der View-Klasse im Komponentendiagramm besitzt.

- `Activity`:
  Ein mit dem Stereotyp Activity ausgezeichneter Zustand im Zustandsdiagramm steht für eine auszuführende Aktion. Meist ist die Ausführung einer Aktion getriggert über

Eingaben des Anwenders (z.B. Mausklick), zur Änderung des applikationsinternen Zustands.

Es gibt verschiedene Arten von Aktivitäten: Ein Aktivitäts-Zustand repräsentiert entweder eine physikalische Java-Klasse (z.B. Action-Klasse in Struts) oder dient der Ablaufbeschreibung der Anwendung, d.h. die Aktivität ist in der Konfiguration der Anwendung enthalten (z.B. struts-config.xml), es wird jedoch keine Klasse generiert. Diese Kategorisierung der Aktivitäten wird über Tagged Values vorgenommen. Ist der Tagged Value 'ControllerClass' gesetzt, so wird die der Aktivität entsprechende Controller-Klasse im Komponenten-Diagramm modelliert, generiert und in der Ablaufbeschreibung eingetragen. Bleibt der Tagged Value ungesetzt, so wird keine Controller-Klasse modelliert, die Aktivität erscheint somit nur in der Ablaufbeschreibung der Anwendung.
Des Weiteren hat man die Möglichkeit zu spezifizieren, ob der Aktivität eine Presentation-Klasse (z.B. FormBean in Struts) zugeordnet wird. Auch diese Zuordnung erfolgt über einen Tagged Value. 'PresentationClass' definiert, ob dem Aktivitätszustand eine Präsentations-Klasse zugeordnet wird, oder nicht.

`Presentation()`: Gibt, abhängig vom Wert des Tagged Values 'PresentationClass', die entsprechende Präsentationsklasse zurück. Namenskonventionen spielen hierbei eine große Rolle, der Wert des Tagged Values und die Präsentationsklasse müssen denselben Namen tragen.

`Controller()`: Gibt, abhängig vom Wert des Tagged Values 'ControllerClass', die entsprechende Controller-Klasse zurück. Namenskonventionen spielen hierbei eine große Rolle, der Wert des Tagged Values und die Controller-Klasse müssen denselben Namen tragen.

- `Page`:
  Ein Zustand mit dem Stereotyp Page repräsentiert eine Bildschirmseite (z.B. JavaServer Page) im Zustandsautomat der Anwendung. Für die Generierung der Anwendung werden die Seiten per Tagged Values ('Category') in verschiedene Kategorien eingeteilt.
  - input:
    Generierte Seite enthält Textfelder innerhalb eines HTML-Formulars, das es dem Anwender ermöglicht, Daten einzugeben bzw. zu bearbeiten. Die Seiten der Anwendungsfälle „AddGroupDealer“ und „ModifyGroupDealer“ sind beispielsweise mit der Kategorie input gekennzeichnet.
  - list:
    Generierte Seite stellt Daten, zumeist aus der Datenbank, in einer Listenform auf der Bildschirmseite dar. Der UseCase „ListGroupDealer“ ist zum Beispiel mit der Kategorie 'list' gekennzeichnet.
  - select:
    Die generierte Seite ermöglicht die Darstellung eines Datensatzes, der zuvor in der Listenform z.B. per RadioButton ausgewählt wurde. Diese Kategorie kommt bei den Anwendungsfällen „DeleteGroupDealer“ und

„ModifyGroupDealer“ zum Einsatz.

`View()`: Liefert die dem Page-Zustand zugeordnete View-Klasse als Rückgabewert.

- `InitialState`:
  Die Klasse `InitialState` repräsentiert den Startpunkt eines Zustandsautomaten, der als PseudoState im UML-Metamodell realisiert ist.

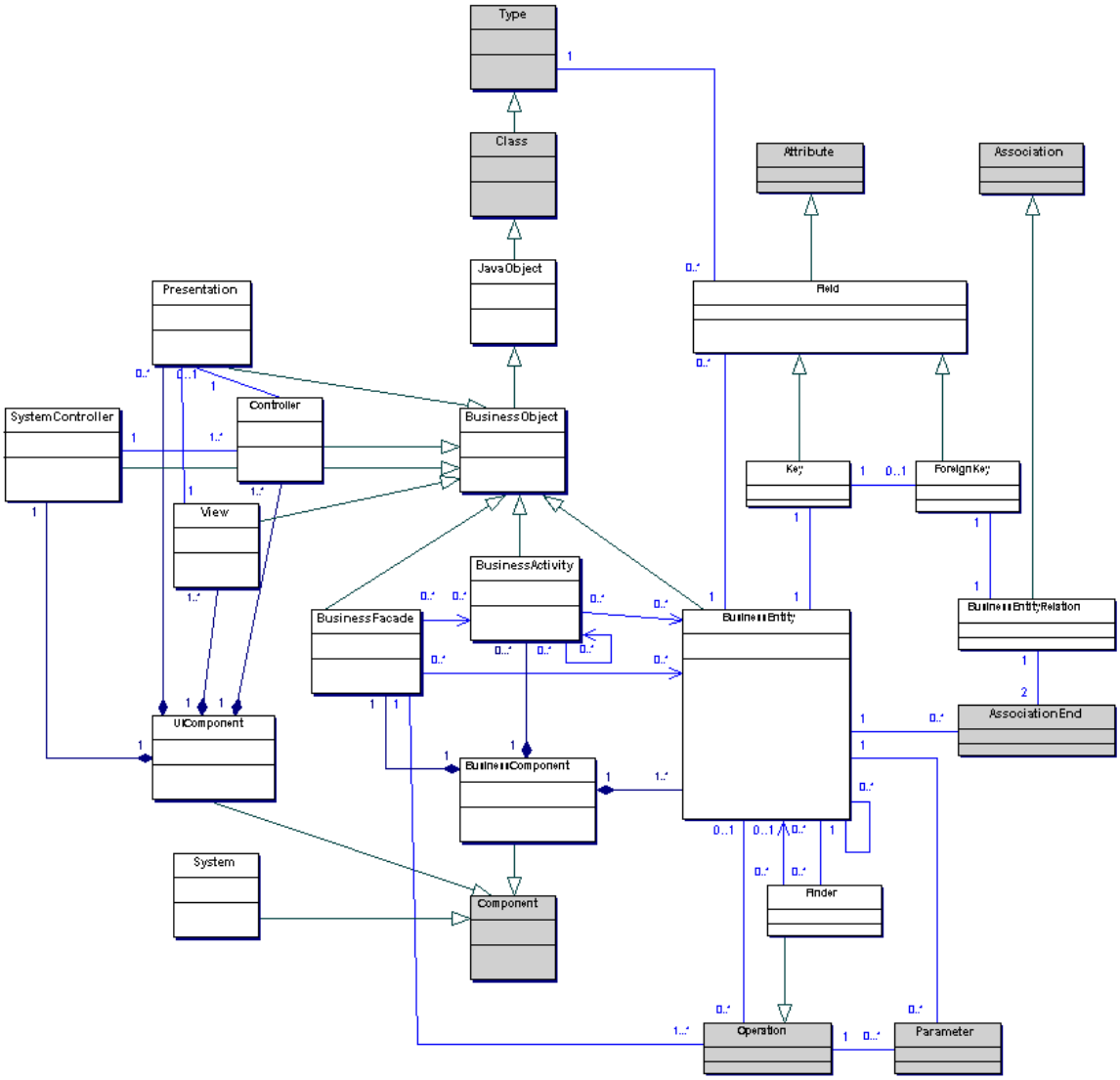


Abbildung 7-10 - Erweitertes Metamodell für den Bereich der Klassendiagramme

7.6.2.2. Technische Erweiterungen

`JavaObject`:
Erweitert `de.bmiag.genfw.meta.classifier.Class` im Metamodell. JavaObject kapselt sprachspezifisches Verhalten der Programmiersprache Java und stellt Methoden zur

Verfügung, die in den Templates aufgerufen werden können:

- `count()`: Zähler, der ein Inkrement zurückgibt.
- `resetCounter()`: Setzt den Zähler auf den Ausgangswert zurück
- `PackageName()`: Gibt den Paket-Namen der aktuellen Klasse zurück
- `getUid()/setUid()`: Erlaubt das Speichern einer UID, die ein JavaObject eindeutig identifiziert.

`JavaPackage`:
JavaPackage ist die programmatische Umsetzung eines Packages in Java. Die Klasse stellt verschiedene Methoden zur Verfügung, die aus den Templates aufgerufen werden können:

- `FullPackageName()`: Gibt den gesamten Java Paket-Pfad ab dem Root-Paket aus.
- `FullPathName()`: Gibt den Paket-Pfad als Verzeichnis-Pfad zurück.

`Identifier`:
Erweitert `de.bmiag.genfw.meta.core.Identifier` im Metamodell und bietet verschiedene Hilfsfunktionen, um die textuelle Repräsentation der Modellelemente von den Templates aus zu formatieren. Die Erweiterung der Klasse Identifier kann dem Generator als Kommandozeilenparameter angegeben werden, um die Default-Implementierung zu ersetzen.

Beispiele:

- `asVAR()`: Der Klassenname wird als Variablenname zurückgegeben, z.B. GroupDealer -> groupDealer
- `asJavaType()`: Ein im PIM technisch unabhängiger Typ wird als Java Typ gerendert, z.B. String -> java.lang.String
- `asPACKAGE()`: Der Klassenname wird als Package-Name gerendert, z.B. GroupDealer -> groupdealer

7.6.2.3. Erweiterungen für Generator Utilities

In Kapitel 5.4 wird der Plugin-Mechanismus, der im Zuge der Einführung der Generator Framework Utilities, in den Generierungsmechanismus integriert wird, erläutert. In der Diplomarbeit kommt zum einen das Codegenerierungs-Plugin zum Einsatz, das den Generator für die Codegenerierung konfiguriert, und zum anderen das Metamodell-Plugin, das den XMI-Instantiator des Frameworks mit den benötigten Eigenschaften konfiguriert.

Folgende Klassen erweitern daher das Metamodell der Anwendung:

- `CodeGenPlugin`:
  Konfiguriert das Framework für den Generierungsprozess.
  - `contributeTemplates()`: Über den in Kapitel 7.8 vorgestellten Template-Mechanismus wird die eigentliche Code-Generierung gesteuert. In dieser Methode wird das Template-Verzeichnis und der Einstiegspunkt für die Generierung definiert.
  - `contributeInvokers()`: Falls in einer Template-Datei eine Eigenschaft

der Metaklasse abgefragt wird, die entsprechende Metamodell-Klasse aber nicht auffindbar ist, läuft der Invoker-Mechanismus an, der versucht, die aufgerufene Eigenschaft in den angegebenen Invokern zu finden. Bei Erfolg wird das Ergebnis an das Template zurückgeliefert, Misserfolg führt zu einer `PropertyNotFoundException`.

Folgende Invoker werden zur Generierung der Stammdatenanwendung eingesetzt:

- `UtilInvoker`: Ermöglicht den Einsatz von Utility-Funktionen in den Templates, wie z.B. `«UpperCaseName»`
- `CounterInvoker`: Stellt in den Template-Dateien eine Zähler-Variable zur Verfügung. (`«CounterReset»`, `«CounterValue»`, `«CounterInc»`, `«CounterNext»`)
- `SubpackageInvoker`: Mit der Einführung des Subpackage-Invoker werden plattform- und anwendungsspezifische Methoden, wie zum Beispiel `JavaPackageName()` oder `CHeaderName()` aus dem Kern des Metamodells entfernt. Mithilfe des Subpackage-Invokers werden diese plattformspezifischen Details in Subklassen für die jeweilige Technologie ausgelagert, ohne, dass sich der Aufruf aus der jeweiligen Template-Datei ändert.
  Beispiel:
  Anlegen eines Subpackage-Invoker für Java:
  `new SubpackageInvoker("javaspecific", "Java")`

- `MetamodelPlugin`:
  Konfiguriert den XMI-Instantiator des Frameworks.
  - `contributeInstantiators()`: Ein Instanziator bildet gewissermaßen das Frontend des Generators. Er liest Modelle in Form von XMI und erstellt die instanziierte Darstellung der Modelle im Speicher. Die Methode `contributeInstantiators()` definiert eine Reihe von Instanziatoren und gibt sie in Form einer Liste zurück.

7.7 Design

Das plattformunabhängige Design (PIM) setzt sich aus mehreren verschiedenen Diagrammen zusammen, der Begriff PIM kann demnach als Oberbegriff für verschiedene Diagramme, die eine Anwendung plattformunabhängig beschreiben, gesehen werden.

Bei der Erstellung eines plattformunabhängigen Designs für die Business- und Persistenzschicht einer Anwendung, kommen im Wesentlichen statische Strukturdiagramme, die die einzelnen Artefakte in Form von Klassen und deren Abhängigkeiten untereinander beschreiben, zum Einsatz (vgl.[Mär01]). Um eine modulare Zuordnung von Klassen und Abhängigkeiten untereinander zu erreichen, werden Komponentendiagramme eingesetzt, die den modularen Systemteil mit transparenter Kapselung darstellen.

Die Dynamik einer Präsentationsschicht stellt an das plattformunabhängige Design einer Anwendung weitergehende Anforderungen. So wird, wie schon mehrmals erwähnt, zur

Beschreibung der Interaktion der Frontend-Applikation und zur Modellierung der einzelnen Bildschirmseiten und Aktivitäten, ein Zustandsdiagramm eingesetzt. In einem zusätzlichen Komponentendiagramm, das mit dem Zustandsautomaten über den Namenraum verknüpft ist, hat man die Möglichkeit, detailliert die Eigenschaften und Abhängigkeiten der einzelnen zu generierenden Klassen (im Falle von Struts: ActionForms, Actions, JSPs) zu definieren. Aus den im Zustandsdiagramm modellierten Klassen entsteht schlussendlich das Generat.

Im Folgenden wird die in der Diplomarbeit erarbeitete Vorgehensweise zur Entwicklung eines plattformunabhängigen Design (PIM) als Generierungsgrundlage beschrieben.

7.7.1. PIM Design

Die Beschreibung eines Systems in Form eines plattformunabhängigen Designs, erfolgt sehr modular, um die einzelnen Diagramme der Päsentations- und Geschäftslogik explizit voneinander getrennt zu halten und damit auch unabhängig voneinander einsetzen zu können.

System-Übersicht:

Abbildung 7-11 beschreibt zeigt die System-Komponente (ausgezeichnet mit dem Stereotyp `<<System>>`), die das gesamte zu generierende System anhand der einzelnen Komponenten der Applikation beschreibt und somit sowohl einen Überblick über die Applikation gibt, als auch modular diejenigen Komponenten festlegt, die generiert werden sollen.

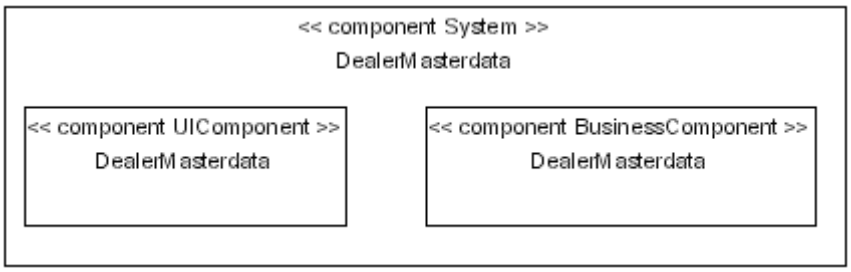


Abbildung 7-11 - System Komponente

Darüber hinaus bildet die System-Komponente den Einstiegspunkt für die Generierung der Applikation über den in Kapitel 7.8 beschriebenen Template-Mechanismus. Die einzelnen Komponenten des Systems sind mit verschiedenen Stereotyopen, die das Verhalten des Modellelements beschreiben, ausgezeichnet. `<<UIComponent>>` repräsentiert ein User Interface, `<<BusinessComponent>>` steht für die Geschäftslogik und Persistenzschicht. Beide Komponenten jeweils werden durch weitere detaillierte Komponentendiagramme beschrieben.

Die verschiedenen statischen und dynamischen Aspekte einer Präsentationsschicht werden mithilfe verschiedener Diagrammarten ausgedrückt:

- Der dynamische Ablauf des User Interface wird anhand eines Zustrandsautomaten modelliert.

- Die feingranularen Zusammenhänge und Eigenschaften der einzelnen zu generierenden Klassen wird in Form eines Komponentendiagramms ausgedrückt.

Ablaufbeschreibung der Präsentationsschicht:

Abbildung 7-12 zeigt das UML-Zustandsdiagramm für die Stammdatenanwendung:

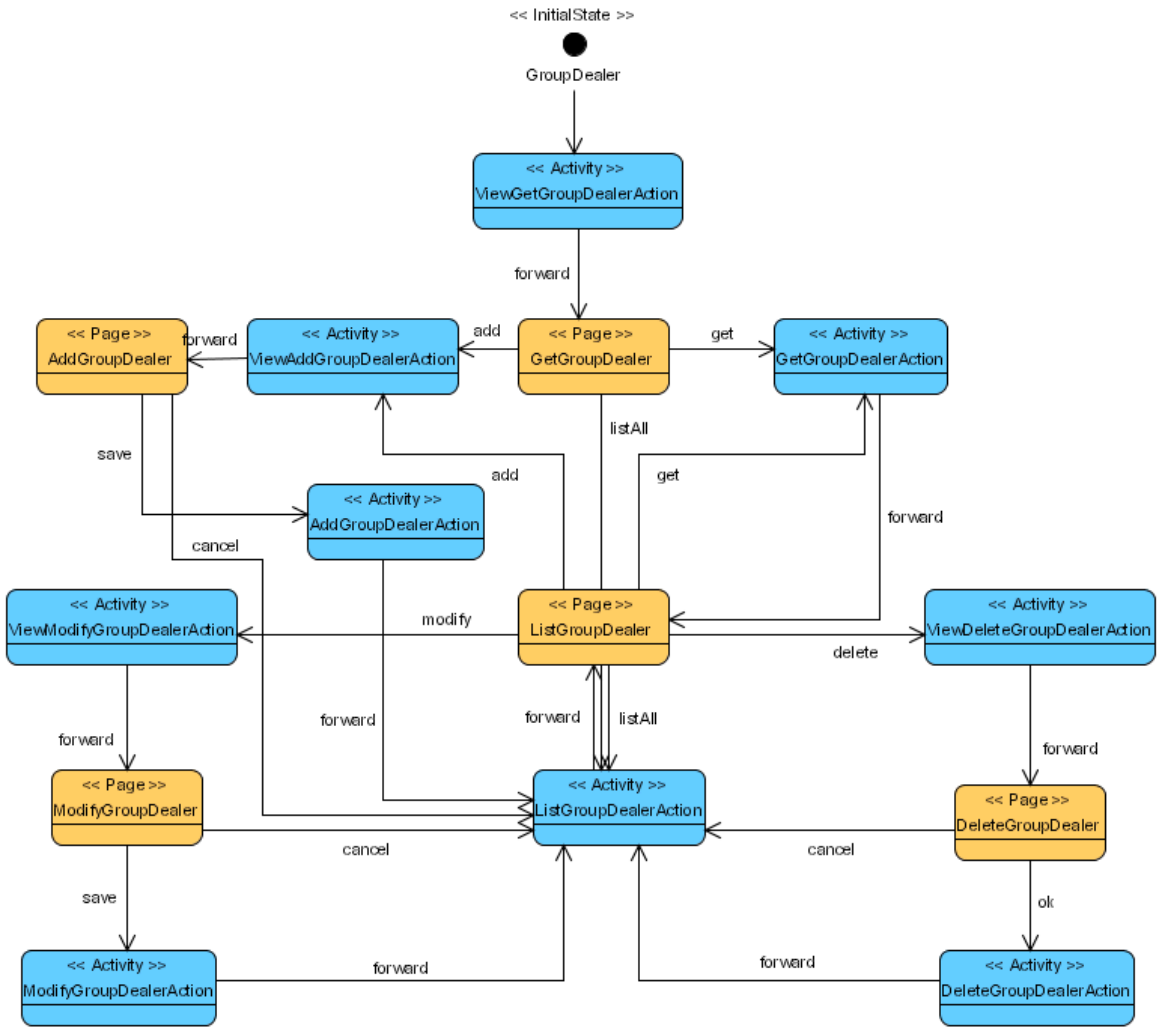


Abbildung 7-12 - Zustandsautomat der Stammdatenanwendung

Grundsätzlich wird bei der Systembeschreibung mithilfe eines Zustandsdiagramms zwischen zwei verschiedenen Arten von Zuständen unterschieden:

1. Activity:

Eine Aktivität repräsentiert eine Aktion, die auf der Benutzeroberfläche vom Anwender ausgelöst wird. Das System reagiert auf die Aktion in der modellierten Art und Weise. Aktivitäten sind mit dem Stereotyp `<<Activity>>` ausgezeichnet. Zum einen kann aus einem Aktivitätszustand eine Klasse des Präsentationsframework (z.B. Struts Action-Klasse) generiert werden, zum anderen kann eine Aktivität für die Ablaufbeschreibung des Systems (`struts-config.xml`) modelliert sein, d.h. die modellierte Aktivität erscheint zwar in der Konfigurationsdatei, eine Klasse wird aber nicht generiert. Des weiteren spielen ausgehende

Transitionen eines Aktivitäts-Zustands bei der Ablaufbeschreibung eine große Rolle. Der Name der Transition und der Zielzustand definieren in der Konfigurationsdatei den weiteren Ablauf der Applikation (Struts `ActionForward`). Eine Aktivität kann sowohl ausgehende Transitionen zu weiteren Aktivitäten (Struts Action Chaining), als auch zu Pages besitzen.

Eine Aktivität besitzt verschiedene Eigenschaften, die über Tagged Values modelliert werden:

- `ControllerClass`:
  Ist der Tagged Value `ControllerClass` gesetzt, so muss eine modellierte Klasse gleichen Namens im Komponentendiagramm (z.B. Struts Action-Klasse) existieren. Namenskonventionen spielen hierbei eine große Rolle, der Name der Klasse im Komponentendiagramm muss mit dem Wert des Tagged Value übereinstimmen.
- `PresentationClass`:
  Der TaggedValue `PresentationClass` definiert, ob zu der Aktivität ein Daten-Container existiert, der die vom Anwender eingegebenen Formulardaten einer HTML-Seite zur weiteren Verarbeitung speichert (z.B: Struts `ActionForm`, JavaBean). Ist der Tagged Value gesetzt, so muss eine entsprechende Klasse gleichen Namens im KomponentenDiagramm vorhanden sein.

2. Page

Ein Zustand, der im Modell mit dem Stereotyp `<<Page>>` ausgezeichnet ist, repräsentiert eine Bildschirmseite (z.B. JavaServer Page) im Modell der zu generierenden Applikation. Für jeden Zustand mit dem Stereotyp `Page`, muss eine entsprechende Klasse im Komponentendiagramm modelliert sein, um feingranular und unabhängig die Eigenschaften der einzelnen Seiten definieren zu können. Wie schon bei dem Aktivitäts-Zustand spielen die ausgehenden Transitionen eine große Rolle. Ein Page-Zustand kann nur ausgehende Transitionen zu Aktivitäten besitzen, d.h. auf eine Bildschirmseite folgt zwangsläufig eine Aktivität. Der Name der Transition wird bei der Generierung unter anderem zur Erzeugung von Hyperlinks und zur Benennung von Schaltfläche herangezogen.

Um die Generierung der verschieden gearteten Bildschirmseiten zu ermöglichen, erfolgt eine Kategorisierung über einen Tagged Value, d.h. der Wert des Tagged Value `Category` definiert das Erscheinungsbild der generierten Seite. Im Rahmen der Diplomarbeit wurden drei verschiedene Arten von Kategorien festgelegt:

- input:
  Ist der Tagged Value `Category` mit dem Wert `input` ausgezeichnet, so wird eine Seite generiert, die es dem Anwender erlaubt, Daten zu erfassen. D.h. es werden HTML-Textfelder zur Eingabe von Daten generiert. Typische Beispiele für diese Kategorie sind die Anwendungsfälle „AddGroupDealer“ oder „ModifyGroupDealer“.

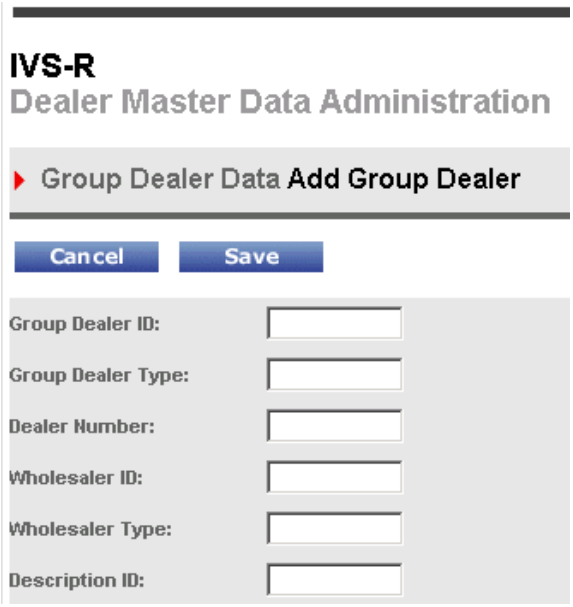


Abbildung 7-13 - Inhalt der Seite AddGroupDealer

- select:
  Die Kategorie `select` definiert die Möglichkeit einen der aufgelisteten Datensätze per Selektion auf der Bildschirmseite (RadioButton, CheckBox,...) auszuwählen und den ausgewählten Datensatz auf der generierten Bildschirmseite darzustellen, um Aktionen darauf ausführen zu können. Der UseCase „DeleteGroupDealer“, bei dem über einen RadioButton ein zu löschender Datensatz ausgewählt und angezeigt wird, ist ein Einsatzbeispiel für diese Kategorie.

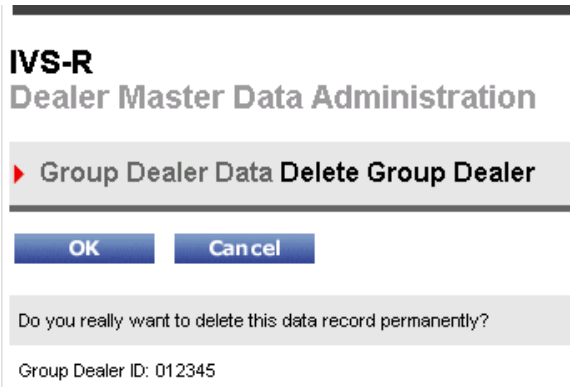


Abbildung 7-14 - Inhalt der Seite DeleteGroupDealer

- list:
  Die Kategorie `list` definiert eine Listenanzeige von Daten, die entweder aus der Datenbank stammen oder vom Anwender eingegeben wurden. Wahlweise werden die einzelnen Felder eines oder mehrerer Datensätze angezeigt. Die Kategorie list kommt zum Beispiel bei den Anwendungsfällen „GetGroupDealer“ (ein Datensatz) und „ListGroupDealer“ (mehrere Datensätze) zum Einsatz.

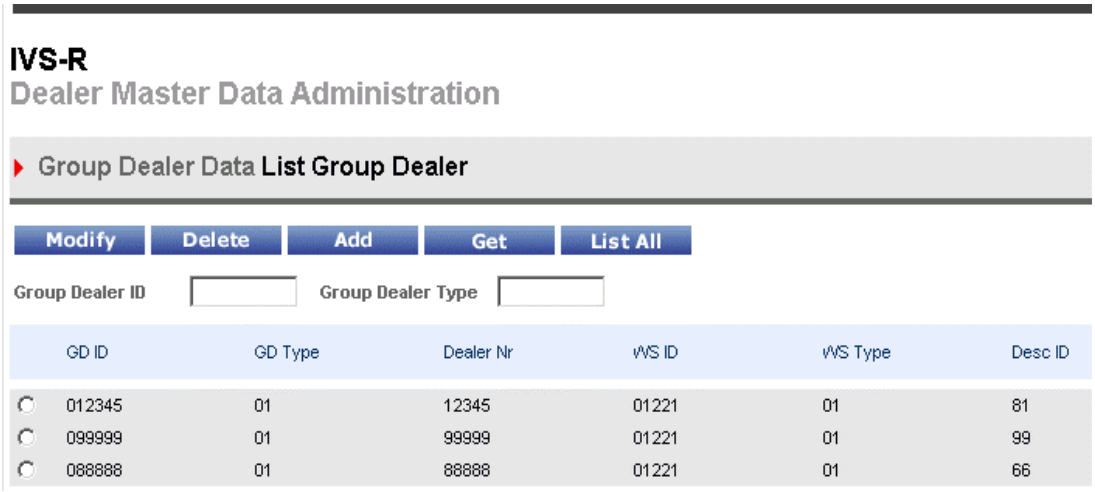


Abbildung 7-15 - Inhalt der Seite ListGroupDealer

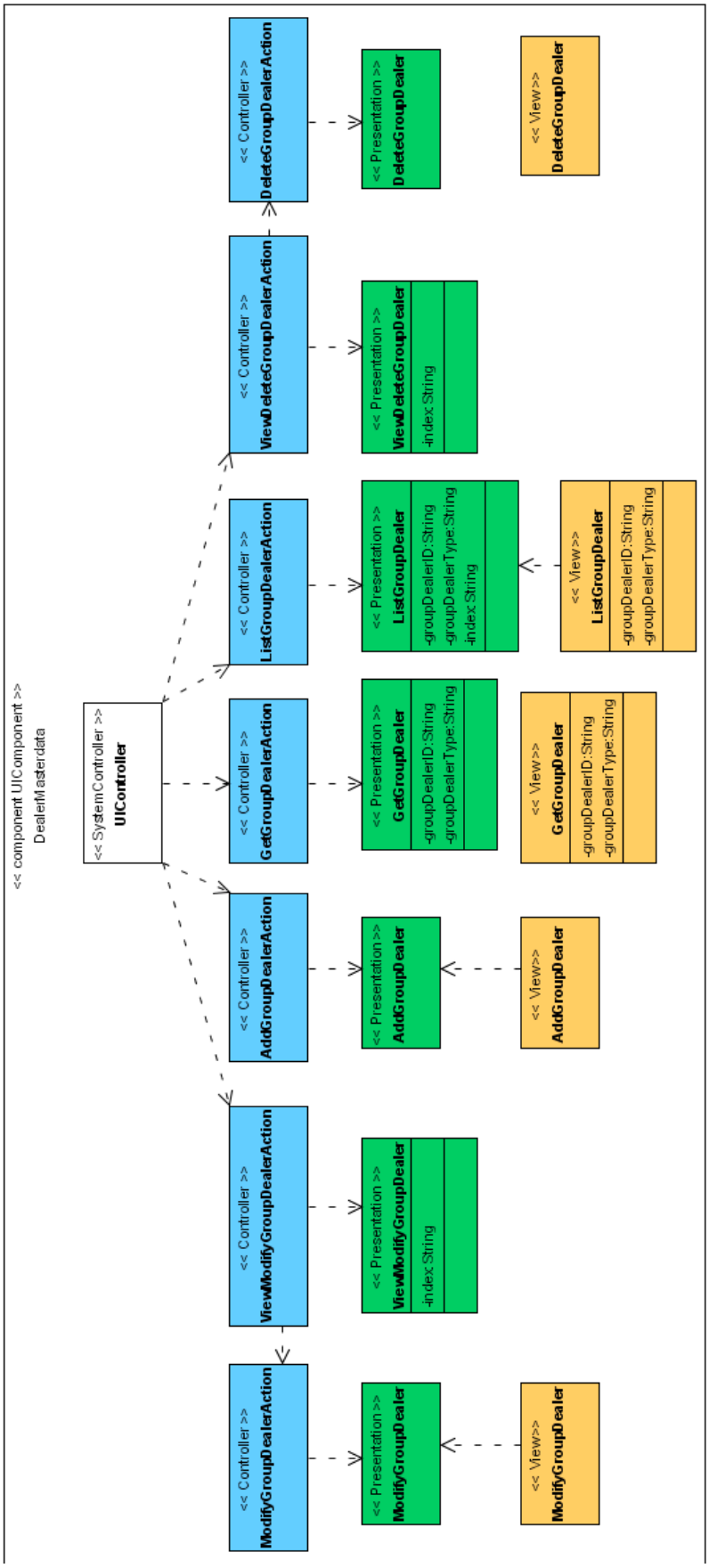
Statische Beschreibung der Präsentationsschicht:

Im Zustandsdiagramm wird anhand von Tagged Values spezifiziert, welche Artefakte schlussendlich als Java-Klassen generiert werden sollen und dementsprechend im Komponentendiagramm modelliert werden müssen.

Im Komponentendiagramm wird die dynamische Sicht auf die Applikation in Form einer feingranularen Definition der einzelnen zu generierenden Klassen modelliert. Hierbei besteht die Möglichkeit im Detail die Eigenschaften der jeweiligen Modellierungselemente zu definieren.

Untenstehende Abbildung 7-16 zeigt das Komponentendiagramm der Stammdatenanwendung. Die einzelnen Bestandteile des Komponentendiagramms werden in Form von Klassen modelliert, die mit verschiedenen Stereotypen ausgezeichnet sind:

- `SystemController`
  Eine mit dem Stereotyp `<<SystemController>>` ausgezeichnete Klasse stellt über den Namensraum die Verbindung vom Komponentendiagramm zum Zustandsautomaten her und verknüpft so den dynamischen Teil der Modellierung mit dem statischen. In einer `UIComponent` darf jeweils nur ein `SystemController` modelliert sein.
- Controller
  Für jeden Zustand, der im Zustandsdiagramm mit dem Stereotyp `<<Activity>>` markiert ist und einen Wert im Tagged Value `ControllerClass` besitzt, wird eine Klasse modelliert, deren Namen dem Wert des Tagged Values im Zustandsdiagramm entsprechen muss. Diese Klasse wird mit dem Stereotyp `<<Controller>>` ausgezeichnet. Bei der Umsetzung des Frontends mit dem Struts Frameworks zum Beispiel, wird bei der Generierung für einen Controller eine Struts Action-Klasse angelegt. Eine modellierte Controller-Klasse darf ausschließlich Abhängigkeiten zu anderen Controller-Klassen oder Klassen mit dem Stereotyp `<<Presentation>>` besitzen.


<< component UIComponent >>
DealerMasterdata
<< SystemController >>
UIController
<< Controller >>
DeleteGroupDealerAction
<< Presentation >>
DeleteGroupDealer
<< View >>
DeleteGroupDealer
<< Controller >>
ViewDeleteGroupDealerAction
<< Presentation >>
ViewDeleteGroupDealer
-index:String
<< Controller >>
ListGroupDealerAction
<< Presentation >>
ListGroupDealer
-groupDealerID:String
-groupDealerType:String
-index:String
<< View >>
ListGroupDealer
-groupDealerID:String
-groupDealerType:String
<< Controller >>
GetGroupDealerAction
<< Presentation >>
GetGroupDealer
-groupDealerID:String
-groupDealerType:String
<< View >>
GetGroupDealer
-groupDealerID:String
-groupDealerType:String
<< Controller >>
AddGroupDealerAction
<< Presentation >>
AddGroupDealer
<< View >>
AddGroupDealer
<< Controller >>
ViewModifyGroupDealerAction
<< Presentation >>
ViewModifyGroupDealer
-index:String
<< Controller >>
ModifyGroupDealerAction
<< Presentation >>
ModifyGroupDealer
<< View >>
ModifyGroupDealer

- `Presentation`
  Für jeden Zustand, der im Zustandsdiagramm mit dem Stereotyp `<<Activity>>` markiert ist und einen Wert im Tagged Value `PresentationClass` besitzt, wird eine Klasse modelliert, deren Namen dem Wert des Tagged Values im Zustandsdiagramm entsprechen muss. Diese Klasse wird mit dem Stereotyp `<<Presentation>>` ausgezeichnet und darf keine Abhängigkeiten zu anderen Klassen besitzen. Eine Presentation-Klasse repräsentiert einen Datencontainer (z.B. JavaBean oder Struts `ActionForm`), der die vom User auf der Bildschirmseite eingegebenen Daten zur weiteren Verarbeitung speichert und Methoden zum Lesen der Daten zur Verfügung stellt. Je nach Design der Applikation können die einzelnen Attribute des Datencontainers entweder einer persistenten Entität des Backends entsprechen oder im Komponentendiagramm in Form von Attributen der Presentation-Klasse modelliert werden, wobei die modellierten Attribute auf jeden Fall generiert werden. Die Generierung der Felder der Presentation-Klasse wird über den Tagged Value `BEClass` gesteuert.
    - `BEClass`
      Falls die Attribute des Datencontainers aus einer Entität generiert werden sollen, so muss der Wert des Tagged Values `BEClass` dem Namen einer Entität aus der Persistenzschicht der Anwendung entsprechen. Bleibt der Tagged Value leer, so werden nur die modellierten Attribute generiert.
- `View`
  Für jeden Zustand, der im Zustandsdiagramm mit dem Stereotyp `<<Page>>` markiert ist, wird eine Klasse gleichen Namens im Komponentendigramm modelliert. Diese Klasse wird mit dem Stereotype `<<View>>` ausgezeichnet und repräsentiert eine Bildschirmseite. Auf einer Bildschirmseite muss grundsätzlich zwischen zwei verschiedenen Arten von Feldern unterschieden werden. Zum einen werden Felder für die Eingabe von Daten (z.B. HTML Textfeld) benötigt, zum anderen muss die Bildschirmseite in der Lage sein, über die Felder eines Datencontainers, der nach Ausführung einer Aktion vom Backend gelieferten wird, zu iterieren und die entsprechenden Daten darstellen zu können. Die Modellierung der zur Eingabe von Daten erforderlichen Felder wurde bereits in der Presentation-Klasse vorgenommen, zur Generierungszeit können diese über eine Abhängigkeit (UML Dependency) zwischen View- und Presentationklasse im Modell ausgelesen und generiert werden. Diejenigen Felder, die für die Anzeige von Daten erforderlich sind, werden in der View-Klasse in Form von Attributen modelliert.

7.7.2. Design Constraints

Wie in Kapitel 5.2.2.1 beschrieben, hat man im Rahmen der Metamodell-Entwicklung bei open ArchitectureWare, die Möglichkeit, Einschränkungen am plattformunabhängigen Design, über Methoden in den Metamodell-Klassen, vorzunehmen. Die Methode `CheckConstraints()` der Framework-Klasse `Element`, von der alle Modellierungselemente erben, wird in den jeweiligen Metamodell-Klassen überschrieben und an die speziellen Bedürfnisse angepasst.

Zusätzlich besteht die Möglichkeit, den Generator im Validierungs-Modus zu starten, d.h. es wird kein Code generiert, sondern lediglich eine Modellvalidierung vorgenommen, bei der die jeweilige Implementierung der CheckConstraints-Methode auf das modellierte PIM angewendet wird und bei Designfehlern je nach Umsetzung ein DesignError geworfen wird.

Listung 7-4 zeigt die Implementierung der Modellierungseinschränkungen für die Metamodell-Klasse `Activity`.

```
/**
 * Checks, if the ModelElements in the StateChart Diagram are of type
 * <ul>
 * <li>{@link Page}</li>
 * <li>{@link Activity}</li>
 * </ul>
 * Otherwise a {@link DesignError} will be thrown. <br/>
 *
 * @see de.bmiag.genfw.meta.Element#CheckConstraints()
 * @return An empty dummy string.
 */
public String CheckConstraints() {
    Iterator iter = this.OutTransition().iterator();
    while(iter.hasNext()){
        Transition t = (Transition) iter.next();
        StateVertex st = (StateVertex)t.TargetVertex();
        if(!(st instanceof Page || st instanceof Activity)){
            throw new DesignError("State " + st.Name().toString()
                    + " has invalid stereotype in StateChart Diagram ");
        }
    }
    return "";
}
```

Listing 7-4 - CheckConstraints-Methode der MM-Klasse Activity

Darüber hinaus wurde die Modellvalidierung in der Metamodell-Klasse `BusinessObject`, die die abstrakte Basisklasse für alle in `UIComponent` und `BusinessComponent` modellierten Klassen darstellt, weiter unterteilt. `BusinessObject` definiert drei abstrakte Methoden, die aus `CheckConstraints` aufgerufen werden, und von allen erbenden Klassen implementiert werden müssen:

- `checkAttributes()`
- `checkOperations()`
- `checkDependencies()`

So lässt sich eine feingranulare Modellvalidierung programmatisch in den Metamodell-Klassen definieren und dementsprechend klare Fehlermeldungen können bei Designfehlern ausgegeben werden.

7.8 Template-Entwicklung

Für die Generierung des Quellcodes der Zielplattform wird der Template-Mechanismus von open ArchitectureWare eingesetzt. Das Generator-Backend greift über Templates auf das nach der Instanziierung vorliegende instanziierte Java-Metamodell zu.

Wie in Kapitel 5.2.3 beschrieben ist das Root-Template in der Datei `Root.tpl` der Ausgangspunkt der Generierung. Ausgehend vom Root-Template werden die einzelnen.

Abbildung 7-17 illustriert die Zusammenhänge zwischen den Templates:

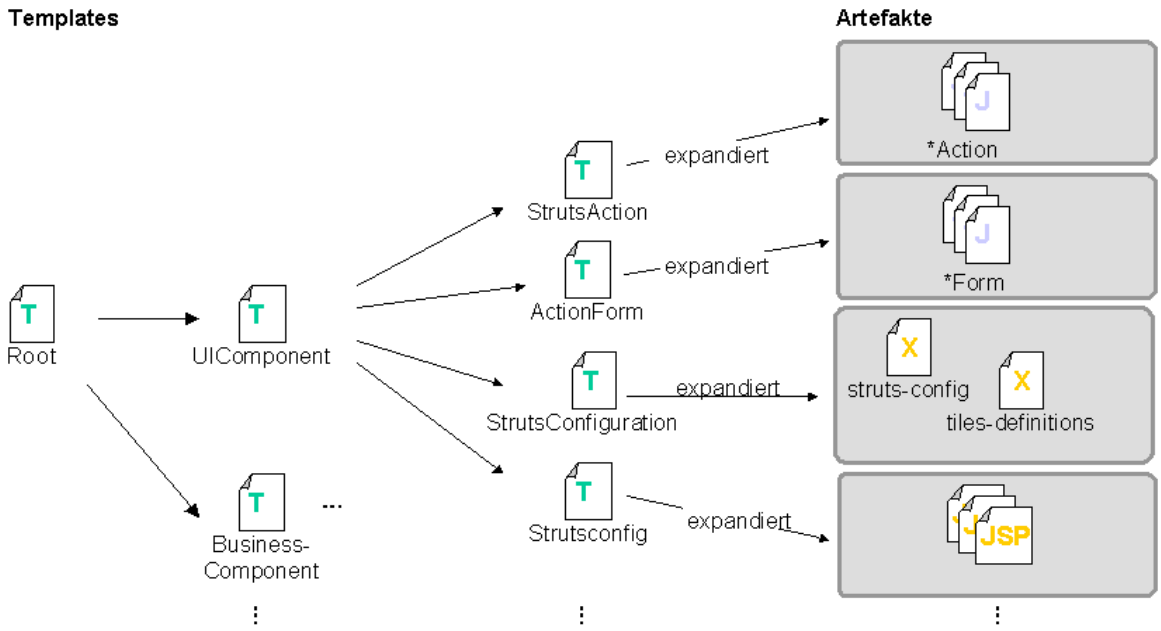


Abbildung 7-17 – Überblick über die Template-Dateien

Eine Template-Datei bildet den Namensraum, in dem mehrere Templates definiert werden können. Logisch zusammengehörige Klassen und Interfaces werden daher innerhalb eines Namensraums definiert. Bei der modellbasierten Generierung ist die 1:1 Zuordnung von Template und generierter Datei somit nicht sinnvoll.

In der folgenden Tabelle werden die für die Generierung entscheidenden Artefakte und Ihre Zuordnung zu den Templates aufgelistet:

Namensraum	***Templates***	***Artefakte***
Root.tpl	Root (System) Root (UIComponent) Root (BusinessComponent)	-

Namensraum	*Templates*	*Artefakte*
UIComponent.tpl	UIComponent Root (Controller) Root (Presentation) Root (View)	-
StrutsAction.tpl	StrutsAction	*Action.java
ActionForm.tpl	StrutsActionForm ActionFormFromBE (BE) ActionFormFromModel	*Form.java
StrutsConfiguration.tpl	StrutsConfigXML formbean ActionMapping Forward(curState, curTransition) TilesDef tiles	struts-config.xml tiles-definitions.xml
View.tpl	Content Navigation taglibs list CategoryListPropertiesFromModel input CategoryInputPropertiesFromModel select contentFixPart	*.jsp
BusinessDelegate.tpl	BusinessDelegate	DealerMasterdataBD.java
ServiceLocator.tpl	ServiceLocator	ServiceLocator.java

Tabelle 1 - Überblick über Namensräume, Templates und Artefakte

Die Umsetzung der wichtigsten Templates wird in den folgenden Kapiteln kurz vorgestellt:

7.8.1. Root Templates

Die Templates, die im Namensraum Root.tpl definiert sind, bilden den Einstiegspunkt in die Generierung. Im Kapitel 5.3.2 sind die einzelnen Templates des Root-Namensraums abgebildet. Folgende Einstiegspunkte sind definiert:

- System:
  Greift auf die im PIM modellierte System-Komponente zu und expandiert die modellierten Komponenten UIComponent und BusinessComponent.

- UIComponent:
  Die Komponente UIComponent repräsentiert die Präsentationsschicht im Modell und expandiert in das Template 'UIComponent' im Namensraum UIComponent.tpl
- BusinessComponent:
  Die Komponente BusinessComponent repräsentiert die Geschäftslogik und Persistenzschicht im Modell und expandiert in das Template 'BusinessComponent' im Namensraum BusinessComponent.tpl

7.8.2. UIComponent Templates

Im Template UIComponent werden für die einzelnen Bestandteile der Präsentationsschicht, modular weitere Templates in verschiedenen Namensräumen aufgerufen.

Listing 7-5 verdeutlicht die Modularität der Template-Expansion:

```
«DEFINE UIComponent FOR UIComponent»
    «EXPAND StrutsConfiguration::StrutsConfigXml»
    «EXPAND StrutsConfiguration::TilesDef»

    «EXPAND Root FOREACH Controller»

    «EXPAND Root FOREACH Presentation»

    «EXPAND Root FOREACH View»
«ENDDEFINE»

«DEFINE Root FOR Controller»
    «EXPAND StrutsAction::StrutsAction»
«ENDDEFINE»

«DEFINE Root FOR Presentation»
    «EXPAND ActionForm::StrutsActionForm»
«ENDDEFINE»

«DEFINE Root FOR View»
    «EXPAND View::Content»
    «EXPAND View::Navigation»
«ENDDEFINE»
```

Listing 7-5 - UIComponent.tpl

7.8.3. StrutsAction Template

Im Namensraum StrutsAction ist ein gleichnamiges Template definiert, das für die Generierung der Struts Action-Klassen zuständig ist.

Nach Erzeugung des Klassenrahmens und der Import-Statements (teils generiert, teils statisch) wird im Template über alle im Zustandsautomat modellierten Activities iteriert. Falls der Aktivität im Modell per Tagged Value eine Presentation-Klasse (Struts ActionForm) zugeordnet ist und somit vom Anwender eingegebene Daten gespeichert, werden diese

ausgelesen und in einem ValueObject zur weiteren Verarbeitung im Backend gespeichert. Zusätzlich wird eine Instanz des BusinessDelegate erzeugt, über diese die einzelnen Methoden der Anwendungsfälle aufgerufen werden. Über verschiedene per Xpand eingefügte geschützte Regionen wird sichergestellt, dass die Entwickler zu einem späteren Zeitpunkt die Klassen um manuellen Code erweitern können, ohne dass dieser bei einem erneuten Generatordurchlauf überschrieben wird.

7.8.4. ActionForm Templates

Die Templates im Namensraum ActionForm sind für die Generierung der Struts ActionForms, die die vom Anwender eingegebenen Daten speichern, zuständig.

Nach Erzeugung des Klassenrahmens und der Import-Statements (teils generiert, teils statisch) werden im Template für jede Presentation-Klasse entweder die modellierten Felder oder die Felder einer per Tagged Value (’BEClass’) zugeordneten Entität ausgelesen und Getter- bzw. Setter-Methoden für den Zugriff auf die Daten modelliert.

7.8.5. StrutsConfiguration Templates

Im Namensraum StrutsConfiguration wird sowohl die zentrale XML-Konfigurationsdatei des Struts Framework (struts-config.xml), als auch die Konfigurationsdatei für die Tiles-Technologie (tiles-definitions.xml) generiert.

StrutsConfigXML:
Das Hauptaugenmerk bei der Genenrierung der struts-config.xml liegt im Wesentlichen auf dem Teil der ActionForm- und ActionMapping-Einträge.

Zur Generierung der einzelnen <form-bean> Elemente (ActionForm), wird über die modellierten Activities aus dem Zutandsautomat iteriert. Für jede Aktivität, der eine Presentation-Klasse zugeordnet ist, wird ein <form-bean> Eintrag generiert.

Bei der Generierung des ActionMappings spielt der modellierte Zustandsautomat eine große Rolle. Alle Zustände, die mit dem Stereotyp `<<Activity>>` ausgezeichnet sind, resultieren in einem Action-Mapping Eintrag in der Struts-Konfigurationsdatei. Die Form des Eintrags hängt vom Wert des Tagged Values 'ControllerClass' ab. Ist der Wert gesetzt, so besteht eine Java-Klasse für die Aktivität und der ActionMapping-Eintrag definiert z.B. die Zuordnung der ActionForm zur Action-Klasse, Validierung, Laden bestimmter Tiles, usw. Ist der Wert des Tagged Values nicht gesetzt, so enthält die Aktivität kein applikationsspezifisches Verhalten, es wird keine Action-Klasse generiert und der Eintrag in der Konfigurationsdatei beschreibt eine statische Weiterleitung zwischen zwei Bildschirmseiten. In beiden Fällen wird über den Zielzustand der ausgehenden Transition definiert, an welche Seite oder Action der Kontrollfluss weitergeleitet wird.

TilesDef:
Für jeden Zustand, der im Zustandsdiagramm mit dem Stereotyp `<<Page>>` ausgezeichnet ist, wird ein Eintrag in der Tiles-Konfigurationsdatei, zusätzlich zu den fest eingetragenen Definitionen, generiert.

7.8.6. View Templates

Die Templates im Namensraum View sind für die Generierung der JavaServer Pages zuständig. Für die drei unterschiedlichen Kategorien (list, input, select), in die die Page-Zustände per Tagged Value eingeteilt sind, bestehen verschiedene Templates. Ein gewisser Inhaltsteil, der auf jeder Seite gleich ist, wird in ein gesondertes Template (contentFixPart) ausgelagert und wird, wie auch die Taglib-Definitionen, von jedem der einzelnen Kategorie-Templates aufgerufen. Grundsätzlich werden die Namen der ausgehenden Transitionen der Page-Zustände zur Generierung von Hyperlinks und Schaltflächen auf den resultierenden Bildschirmseiten verwendet. Bei der Generierung muss zwischen Feldern, die eine User-Eingabe erwarten (HTML Textfeld) und Feldern, die für die Anzeige der vom Backend gelieferten Daten verantwortlich sind, unterschieden werden.

list:
Bei der Kategorie 'list' erfolgt die Generierung der Felder zur Anzeige der vom Backend gelieferten Daten anhand der im Komponentendiagramm modellierten Attribute der View-Klassen.

input:
Bei der Kategorie ’input’ handelt es sich um Bildschirmseiten, die ausschließlich Felder zur Dateneingabe bereitstellen. Die Generierung der Felder erfolgt zum einen über die modellierten Attribute der zugeordneten Presentation-Klasse, zum anderen über den potentiell gesetzten Tagged Value 'BEClass'

select:
Bei der Kategorie 'select' handelt es sich um eine relativ einfache Seite, die einen einzelnen per Radiobutton ausgewählten Datensatz anzeigt, d.h. das Formbean, das zuvor in der Action-Klasse in Form einer Variable in der Session gespeichert wurde, wird ausgelesen und angezeigt.

7.9 Konfiguration des Generierungsprozess

Der größte Teil an Informationen, der für die Generierung von Quellcode benötigt wird, stammt aus dem PIM und den darin modellierten Elementen. Um das PIM möglichst plattformunabhängig zu halten, müssen gewisse technische Informationen ausgelagert werden und in einer externen Konfigurationsdatei (config.properties), die beim Start des Generator gelesen wird, definiert werden. Das Einlesen der Datei config.properties wird über die Singleton-Klasse `de.softlab.metamodel.PropertyProvider` gesteuert. Diese verwaltet die Properties-Datei und stellt eine Methode `getConfigProperty(String name)` zur Verfügung, über die aus dem Metamodell heraus Konfigurations-Einträge ausgelesen werden können.

Beispiel: Bestimmte Arten von Exception-Klassen werden in Java-Paketen gekapselt und bei der Generierung der Struts-Klassen im Template hartcodiert eingefügt. Um zu vermeiden, bei jeder Exception den gesamten Packagepfad im Template aufführen zu müssen, wird der Pfad in der externen Konfigurationsdatei gespeichert und per Xpand-Statement dynamisch eingefügt: `«UIComponent.CommonExceptionPackage».` Die Metamodell-Methode delegiert den Aufruf an den PropertyProvider und liefert über einen entsprechenden Schlüssel den Package-Namen zurück.

Darüber hinaus werden bestimmte nicht generierbare Teile, wie zum Beispiel Basisklassen, Taglib Definitionen, Logfile Beschreibungen, etc. über einen Ant Task in die Projektstruktur des Generats kopiert.

7.10 Build- und Deployment Prozess

Verschiedene unterschiedliche Arbeitsschritte sind zur Generierung der Präsentationsschicht und der Auslieferung an den Applikationsserver nötig. Um diese einzelnen Schritte nicht manuell ausführen zu müssen, wird die Steuerung und Konfiguration, sowie die Paketierung und das Deployment, in einem Skript gekapselt. Hierbei kommt das auf Java basierende Build-Tool Apache Ant[1] zum Einsatz.

7.10.1. Build Prozess

Abbildung 7-18 bietet einen Überblick über die verschiedenen Ant Tasks zur Generierung und zum Deployment der J2EE-Stammdatenanwendung in der Datei build.xml und ihre Abhängigkeiten untereinander:

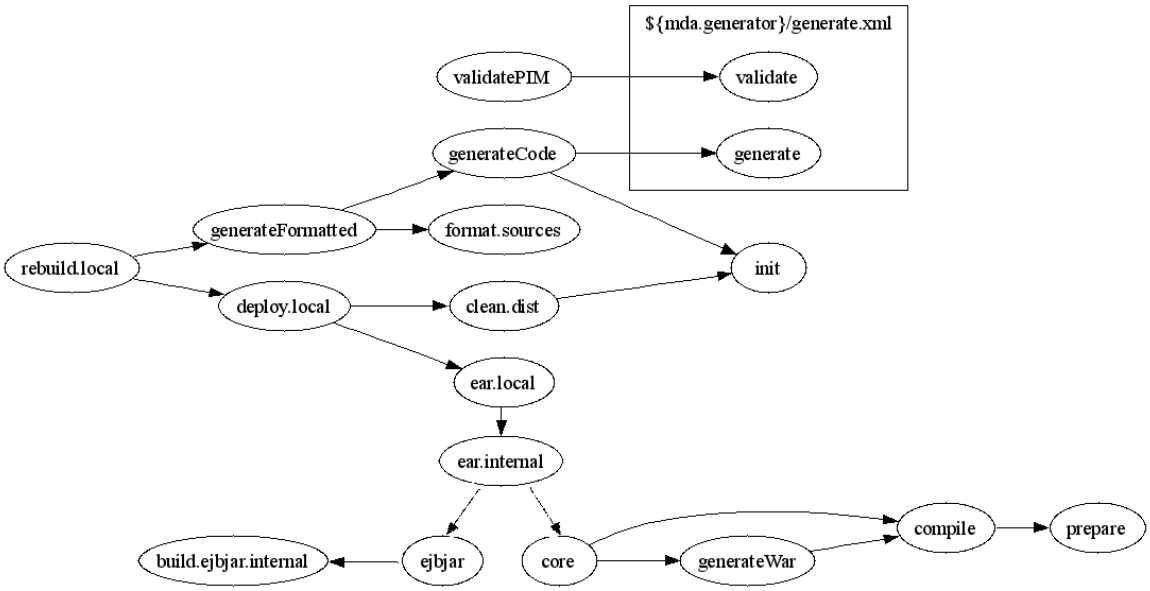


Abbildung 7-18 - Abhängigkeiten der einzelnen Ant Tasks

Folgende Arbeitschritte sind für die Generierung, Paketierung und das Deployment der Präsentationsschicht erforderlich:

- PIM Design
- Export des PIM nach XMI
- Code-Generierung / PIM Validierung
- Code Formatierung
- Kompilierung

1 Apache Ant – Java Build Werkzeug - http://ant.apache.org/

- Paketierung
- Auslieferung

Die in Abbildung 7-19 dargestellte Ant-Datei generate.xml ist für die Konfiguration und den Start des Generierungsprozess zuständig

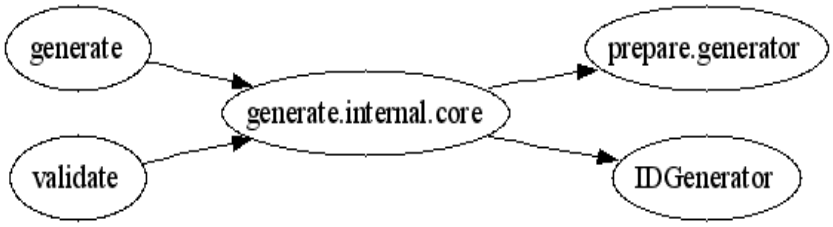


Abbildung 7-19 - Visualisierung von generate.xml

Im Folgenden werden die für die Generierung zuständigen Ant-Targets vorgestellt:

- generate
  Das Ant Target generate ist für den Aufruf des unten beschriebenen Targets generate.internal.core im Modus 'generate' verantwortlich, wodurch die Code-Generierung gestartet wird.
- validate
  Das Ant Target validate ist für den Aufruf des unten beschriebenen Targets generate.internal.core im Modus 'validate' verantwortlich, wodurch die Validierung des plattformunabhängigen Design angestoßen wird.
- prepare.generator
  Das Target prepare.generator ist für die Vorbereitung des Generierungsprozesses verantwortlich. Zum einen legt es die Verzeichnisse an, die für die Generierung benötigt werden (src, backup) und kopiert alle Source-Dateien vor der Generierung in das Backup-Verzeichnis, zum anderen werden sämtliche nicht generierbaren Anteile in das Zielprojekt kopiert.
- generate.internal.core
  Untenstehendes Listing zeigt das Ant Target generate.internal.core, das den im Kapitel Generator Utilities beschrieben Ant Task generate enthält. Der Task generate kann, wie oben beschrieben in den Modi generate und validate gestartet werden. Die verschiedenen Modi definieren bestimmte im Task ebenfalls konfigurierte Plugins, die für die Ausführung der geforderten Funktionalität erforderlich sind. Des weiteren muss der Generierungsprozess über verschiedene Attribute des Tasks generate, wie zum Beispiel Source-Verzeichnis, Instanziierungsvorschrift oder Tool-Adapter für das verwendete CASE-Tool, konfiguriert werden.

```
<target name="generate.internal.core" depends="prepare.generator,IDGenerator">
    <generate log4JPropertiesFile="${mda.generator}/log4j.properties"
            templateEvalCallbackClassname="genfwutil.templateEvalCallback.CallTreeTemplateEvalCallback"
            toolAdapter="${tooladapter}"
            xmlMap="NOT_USED"
            metaMap="${metamap}"
            outDir="${generated}/src"
            srcDir="${generated}/backup"
            logLevel="3"
            mode="${GENERATOR.MODE}">
        <modeConfig name="validate" plugins="mm"/>
            <modeConfig name="generate" plugins="mm, gen"/>
            <plugin name="mm" classname="de.softlab.mda.generator.plugin.MetamodelPlugin">
                <property name="MODEL" value="${design}/${design.name}"/>
                <property name="XMLMAP" value="D:/Projects/MDA-Generator/config/poseidon20_xmi12_all.xml"/>
            </plugin>
            <plugin name="gen" classname="de.softlab.mda.generator.plugin.CodeGenPlugin">
                <property name="TEMPLATES.ROOT" value="${templates}"/>
            </plugin>
    </generate>
</target>
```

Listing 7-6 - Ant Target generate.internal.core

7.10.2. Deployment Prozess

Für das Deployment einer Web-Applikation werden alle erforderlichen Komponenten in einem sogenannten Web ARchive (WAR) gekapselt und an einen Applikationsserver ausgeliefert. Das Erstellen der WAR-Datei wird über einen Ant Core-Task im Build Skript automatisiert.

Verzeichnisstruktur des Web-Archivs:

name.war

/classes: Enthält sämtliche Java Class-Dateien der Anwendung

/lib: Enthält für die benötigte Bibliotheken, wie z.B. struts.jar, commons-validator.jar

/struts: Enthält Struts-Konfigurationsdateien: struts-config.xml, tiles-definitions.xml, validation.xml, validator-rules.xml

/tld: Enthält die Struts Tag-Bibliotheken zur Anwendung in den JavaServer Pages

web.xml: Deployment Deskriptor für die Web-Anwendung

8 Projektbeispiele

Die Umsetzung der Generierung einer Präsentationsschicht aus einem plattformunabhängigen Modell unter Einsatz von open ArchitectureWare wurde exemplarisch anhand des Softlab Projektes IVS-R DealerMasterdata Administration realisiert.

Im Folgenden wird das Projekt noch einmal kurz vorgestellt und die Erprobung der Diplomarbeit im Praxiseinsatz, anhand eines weiteren Softlab Projekts beschrieben.

8.1 IVS-R DealerMasterdata Administration

8.1.1. Überblick

Der Stammdatenbestand der BMW AG besteht laut [Märk01] unter anderem aus Informationen über die Struktur und die Adressen des Vertragshändlernetzes. Zur Pflege dieser unternehmenskritischen Daten wurde im Rahmen des Projekts IVS-R (International Vehicle System – Reengineering) ein System zur Stammdatenverwaltung (DealerMasterdata Administration) entwickelt.

Bei der zu generierenden Stammdatenanwendung, handelt es sich um eine dreischichtige Client/Server Architektur auf Basis von Java 2 Enterprise Edition (J2EE). Für die Persistenzschicht werden Enterprise Java Beans über Application Server verwendet. Die auf dem Client eingesetzten JSPs und Java Servlets sollen durch Einsatz des Struts Frameworks abgelöst werden. Die relativ simplen und wiederkehrenden UseCases, wie zum Beispiel Anlegen, Lesen oder Löschen von Stammdaten sprechen für den Einsatz eines Generators.

Die Vorgehensweise bei der Generierung der Anwendung im Einzelnen, wurde ausführlich im Kapitel 7 behandelt, deshalb soll an dieser Stelle nicht näher darauf eingegangen werden.

8.2 VERA/FCS BONSAI

8.2.1. Überblick

Eine Generierung der Präsentationsschicht ist in vielen Projekten mit modularen Anwendungsfällen denkbar. Ziel der Diplomarbeit war unter anderem, die Generierungsumgebung, das erweiterte Metamodell und die Templates für die Stammdatenanwendung, generisch zu gestalten, damit der Einsatz des Generator Frameworks zur Generierung einer Präsentationsschicht in weiteren Softlab-Projekten ermöglicht wird. Im Rahmen des Projekts VERA/FCS (Vertriebs-Auswertung – Field Communication System) wurde das erarbeitete Ergebnis der Diplomarbeit exemplarisch anhand des Moduls BONSAI (**Bo**nu**s** **A**pplikation **I**nternational) erprobt. BONSAI adressiert laut [Ban01] die Bonifizierung von Fahrzeugen in den Importeursmärkten der neuen EU-Beitrittsländer im Rahmen der EU-Osterweiterung, welche zum 01.05.2004 in Kraft tritt. Innerhalb des BONSAI-Moduls soll der Bereich der Grosskundenbonifizierung generiert werden.

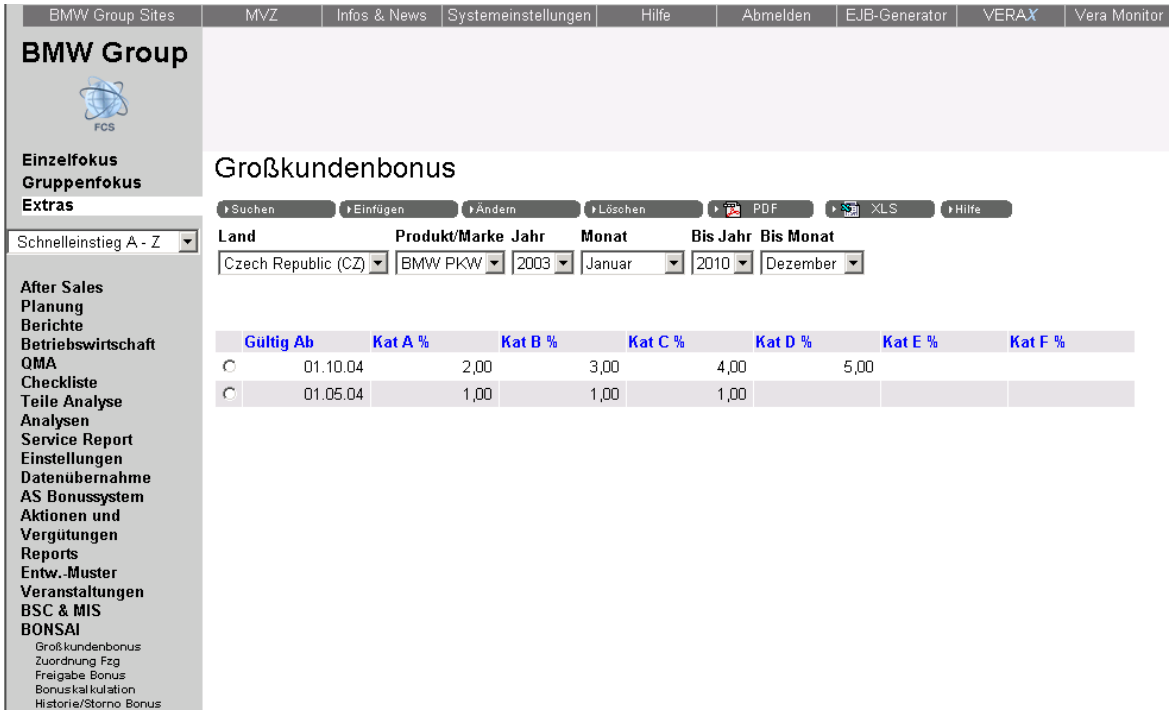


Abbildung 8-1 - VERA/FCS BONSAI Großkundenbonus

Der Dialog zeigt alle hinterlegten Generationen des Grosskundenbonus mit dem jeweiligen Gültig-Ab Datum an. Über die Funktionsbuttons <Einfügen>, <Ändern> und <Löschen> kann eine Generation eines Grosskundenbonus hinzugefügt, geändert und gelöscht werden. Für die Funktionen <Ändern> und <Löschen> kann genau eine Generation des Grosskundenbonus markiert werden, für die die Bearbeitung durchgeführt wird [Ban01]. Über den Funktionsbutton <Suchen> kann nach bestimmten Auswahlkriterien, die anhand von Auswahlboxen festgelegt werden gesucht werden.

Die Präsentationsschicht wird auf Basis von Apache Struts und der Tiles Technologie umgesetzt.

8.2.2. Metamodellierung

Aufgund der Charakteristika und Funktionsweise von open ArchitectureWare ist es möglich ein nahezu technologiefreies Metamodell zu definieren. Die einzelnen Metamodell-Klassen beschreiben Verhalten und Abhängigkeiten zu anderen Klassen und ermöglichen dadurch den Entwurf eines generischen Metamodells, das in verschiedensten Projekten eingesetzt werden kann. Der Vorteil der Technologieunabhängigkeit und der damit verbundenen Kapselung der eingesetzten Technologie in den Templates wurde auch in der Erprobung des Ergebnisses der Diplomarbeit im Projekt VERA/FCS BONSAI Großkundenbonus deutlich. Für die Generierung des Moduls Großkundenbonus war keine Änderung an den bereits vorhandenen Metamodellklassen der Stammdatenanwendung vonnöten, d.h. nach Erstellung eines generischen Metamodells beschränkt sich die Entwicklung einer Anwendung mit oAW auf das Design der plattformunabhängigen Modelle und die Implementierung der einzelnen Templates.

8.2.3. PIM

8.2.3.1. Überblick

Analog zum plattformunabhängigen Design, das während der Entwicklung der IVS-R DealerMasterdata Administration Anwendung entwickelt wurde, werden die einzelnen logischen Bestandteile des Systems in Form von Komponenten dargestellt.

Abbildung 8-2 zeigt die System-Komponente, die eine weitere Komponente <<UIComponent>> enthält, d.h. für das Modul BONSAI Großkunden wird die Präsentationsschicht generiert, die Geschäfts- und Persistenzschicht wird bei der Generierung nicht berücksichtigt.

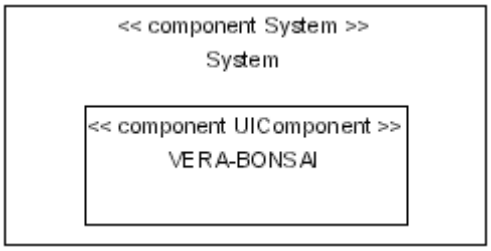


Abbildung 8-2 - Systemüberblick der Anwendung BONSAI Großkunden

8.2.3.2. Zustandsdiagramm

Der Ablauf der Applikation und die Übergänge zwischen den einzelnen Bildschirmseiten (Zustand mit Stereotyp `<<Page>>`) und Aktivitäten (Zustand mit Stereotyp `<<Activity>>`) werden in Form eines Zustandsdiagramms modelliert. Die einzelnen Zustände werden, ihrer Funktionalität entsprechend, jeweils mit den in Kapitel 7 beschriebenen Stereotypen und Tagged Values ausgezeichnet. So entsteht eine Ablaufbeschreibung des Moduls, die die Basis für die Generierung bildet. Durch das unten beschriebene Komponentendiagramm, das über den Namensraum mit dem Zustandsautomaten verknüpft ist, werden die einzelnen zu generierenden Bestandteile des Struts Frameworks (Action, ActionForms, JSPs,...) noch detaillierter über weitere Stereotypisierung und Tagged Values spezifiziert.

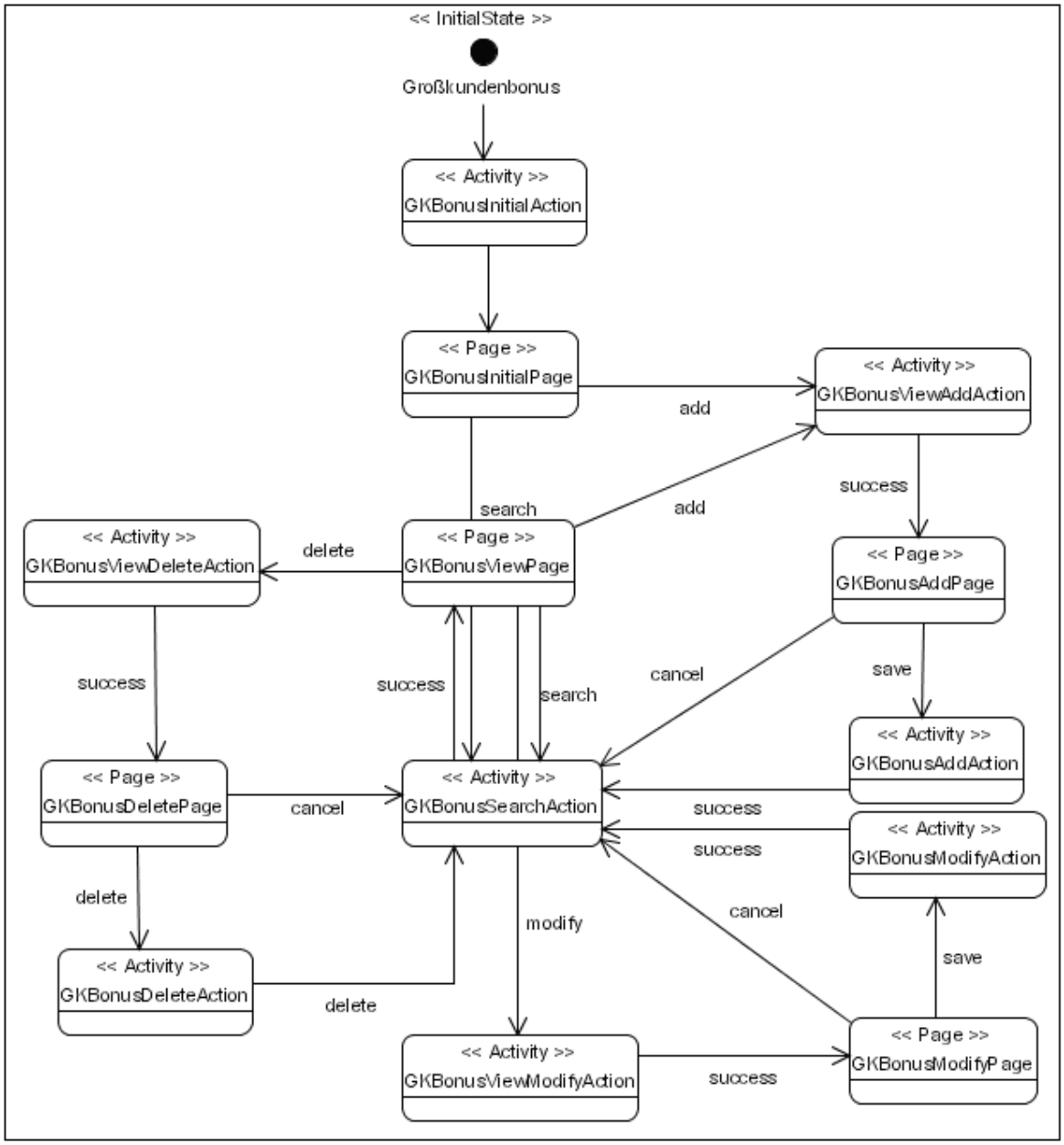


Abbildung 8-3 - BONSAI Zustandsautomat zur Ablaufbeschreibung

8.2.3.3. Komponentendiagramm

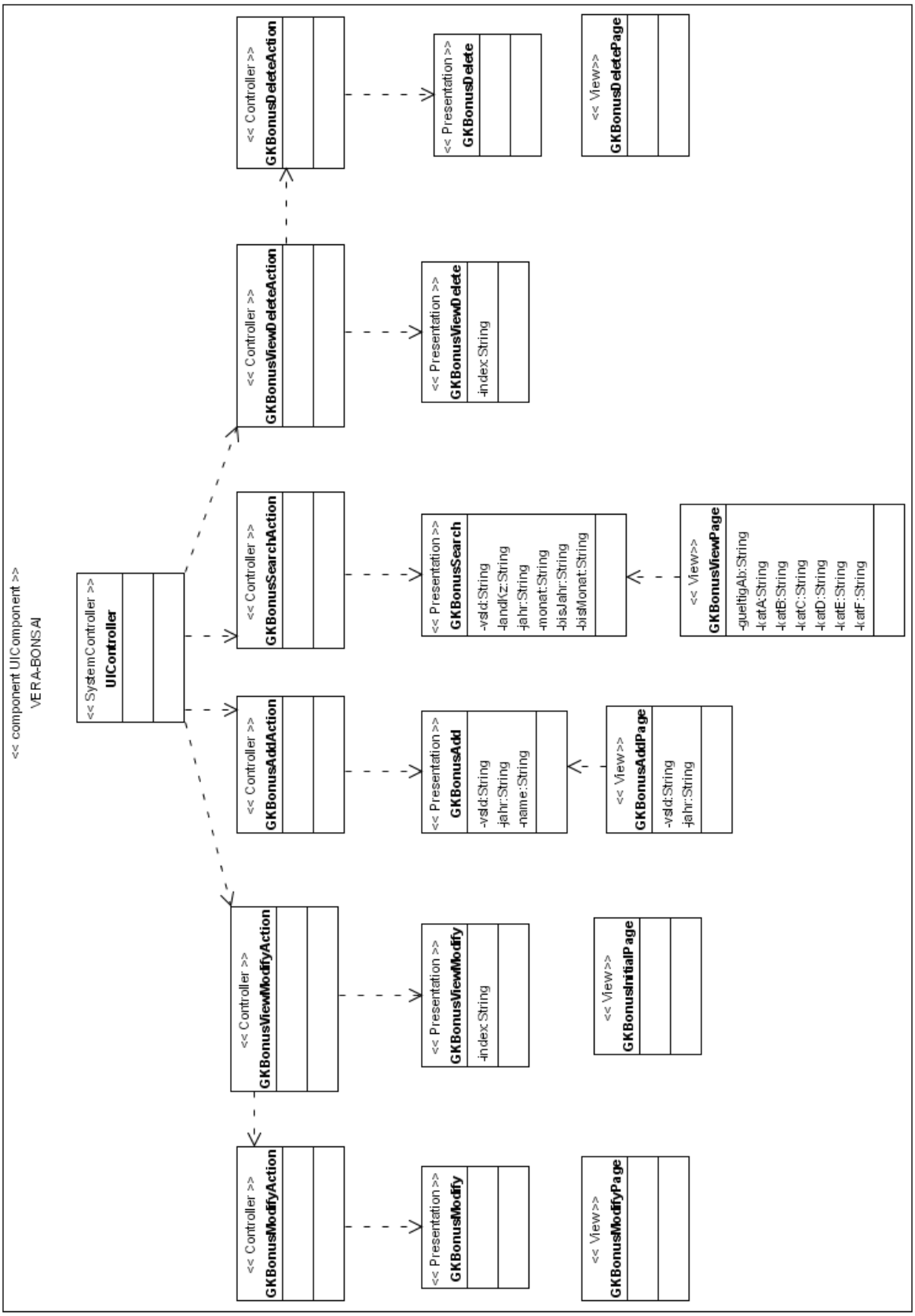


Abbildung 8-4 - BONSAI Komponentendiagramm

Das obenstehende Komponentendiagramm definiert feingranular die Attribute und Abhängigkeiten der einzelnen modellierten und somit zu generierenden Klassen. Die Klasse `UIController`, die mit dem Stereotyp `<<SystemController>>` ausgezeichnet ist, stellt über den Namensraum die Verbindung zwischen dem statischen Komponentendiagramm und der dynamischen Ablaufbeschreibung im Zustandsautomaten her.

8.2.4. Templates

Die Präsentationsschicht von BONSAI Großkundenbonus baut auf den gleichen Technologien (Apache Struts, Tiles, MVC Paradigma), wie das zuvor generierte Frontend, auf. Insofern war der Änderungsaufwand an den Templates sehr gering, nur einige applikationsspezifische Details mussten angepasst werden. Falls die Präsentationsschicht auf eine vollkommen andere Technologiebasis (z.B. JavaServer Faces) umgestellt werden sollte, müssen die Template-Dateien neu entwickelt werden, das plattformunabhängige Design, sowie das bereits modellierte Metamodell bleiben jedoch von einer Technologieänderung gänzlich unberührt.

9 Bewertung

Der Mehrwert, der bei der Generierung einer Präsentationsschicht, inklusive der View Komponente (hier: JavaServer Pages), entsteht, wird bei näherer Betrachtung besonders deutlich. Die Einführung der MDA und die daraus resultierenden pragmatischen Interpretationen des OMG-Standards erfordern neue Konzepte und damit eine erhöhte Flexibilität der Entwickler, da die Modellzentrierung auch Veränderungen im Softwareentwicklungsprozess mit sich bringt. Schon in der fachlichen Konzeptionsphase sollte mit der PIM-Modellierung begonnen werden, um zu einem möglichst frühen Zeitpunkt in der Implementierungsphase, in der Lage zu sein, Code generieren zu können. Die eigentliche Entwicklung wird auf eine höhere Abstraktionsstufe („Modell wird zum Code“) verschoben. Viele Anbieter von Werkzeugen zur modellbasierten Entwicklung empfehlen daher einen Softwareentwicklungsprozess, der an das Werkzeug und die dazugehörigen Konzepte angepasst ist.

Bisher war der Fokus der Code-Generierung zumeist auf die statischen Backend-Komponenten (z.B. Enterprise JavaBeans) gerichtet, da die Generierung einer Präsentationsschicht ohne den Ansatz der modellgetriebenen Entwicklung nur rudimentär möglich war. Bei näherer Betrachtung einer Benutzeroberfläche erkennt man schnell, dass auf Seiten der Präsentationsschicht ähnlich viele schematischen Anteile bestehen, wie zum Beispiel bei einer Persistenzschicht. Allein durch die Verwendung des Web-Frameworks Apache Struts erhöht sich der Anteil der sich wiederholenden Elemente (Struts Action Klassen, ActionForms) im User Interface signifikant. Hinzu kommt, dass mit der UML eine Sprache zur Verfügung steht, mit der man nicht nur statische Anwendungen in Form von Klassen- oder Komponentendiagrammen modellieren kann, sondern auch dynamische Abläufe anhand von Interaktionsdiagrammen, wie z.B. dem Zustands- oder Aktivitätsdiagramm, darstellen kann. Anhand dieser Interaktionsdiagramme lassen sich in Verbindung mit statischen Modellelementen (z.B. Klassen), aus denen schlussendlich Code generiert wird, auch größere komplexe Web-Anwendungen beschreiben und generieren. Auch die auf den ersten Blick eventuell uneffizient erscheinende Generierung der View-Komponente (JavaServer Pages) macht anhand der im Modell vorliegenden Informationen Sinn. Anhand der Kategorisierung der Bildschirmseiten in Form von Tagged Values und der Ablaufbeschreibung im Zustandsautomat kann ein Großteil der JSP, inklusive Hyperlinks und Beschriftungen, generiert werden.

Trotz des positiven Ergebnisses bei der Generierung der Stammdatenanwendung und in der Projekterprobung, sollte man sich im Klaren darüber sein, dass die modellgetriebene Entwicklung mit oAW nicht in jedem Projekt zu bewerkstelligen ist. Gerade im Bezug auf die Präsentationsschicht bestehen in Projekten oft spezielle Anforderungen an Bildschirmmasken und Funktionalität, die sich nur schwer oder gar nicht in ein definiertes Schema gießen lassen. Alternativ hat man jedoch immer die Möglichkeit, nur einzelne Teile einer Präsentationsschicht zu generieren und bei komplexen Anforderungen zum Beispiel die Generierung der View-Komponente auszulassen. Auch die Erfahrung im Umgang mit MDA und dem passenden Entwicklungsprozess sollte in der Entscheidungsphase berücksichtigt werden, da ein modellzentrierter Ansatz für die Entwickler eine Umstellung mit sich bringt. Die Lernkurve bei der anfänglichen Arbeit mit open ArchitectureWare ist durch die erhöhte Komplexität, die die Architektur des Generator Frameworks mit sich bringt, sehr steil, nach

einer gewissen Einarbeitungszeit überwiegen jedoch die Vorteile, die durch die modulare Architektur von oAW entstehen. Ein weiterer offener Punkt, der erst durch den Projekteinsatz in größeren Teams geklärt werden kann, ist die Teamfähigkeit eines modellzentrierten Ansatzes. Fragen der Versionskontrolle von Modellen, Rollenverteilung innerhalb der Projektteams oder Definition der verschiedenen Entwicklungsphasen sind Punkte, die in dieser Arbeit nicht behandelt werden, aber vor dem Einsatz in einem kommerziellen Projekt geklärt werden müssen.

Open ArchitectureWare im Speziellen wurde im letzten halben Jahr während der Arbeit an der Diplomarbeit signifikant weiterentwickelt. Der Entwickler wird durch verschiedene Werkzeuge bei der Arbeit unterstützt. Zum Beispiel erlaubt ein Template-Editor, der in Form eines Eclipse-Plugins realisiert ist, komfortable Entwicklung der Template-Dateien, unter anderem inklusive Code-Ergänzung und farblicher Syntax-Markierung. Auch die in Kapitel 5.4 beschriebenen Generator Utilities, die in einer späteren Version dem Generator-Kern hinzugefügt werden sollen, haben erheblich zu einer Effizienzsteigerung bei der Entwicklung beigetragen. Ein Wehmutstropfen, speziell zu Beginn, während der Einarbeitungszeit, ist die entweder veraltete oder nur spärlich vorhandene Dokumentation des Generator Frameworks. An dieser Stelle soll aber auch auf das Internet-Forum hingewiesen werden, in dem unter anderen, die Entwickler des Generators in kürzester Zeit kompetente Antworten auf Fragen jeglicher Art liefern.

10 Ausblick

Oft wird in der Diskussion über Model Driven Architecture angeführt, dass der Hype, der um MDA entsteht, mit dem der CASE-Tools Anfang der 90er Jahre zu vergleichen ist. Im gleichen Atemzug wird erwähnt, dass die Konzepte der modellgetriebenen Entwicklung alt sind und in wenigen Jahren niemand mehr über diesen kurzfristigen Hype reden wird. Demgegenüber stehen die Verfechter der Meinung, dass MDA und deren praktische Anwendung die Softwareentwicklung und -Konzepte revolutionieren.

Die Wahrheit liegt sicher irgendwo zwischen diesen überspitzt formulierten Meinungen. Auf der einen Seite sollte der Einsatz modellzentrierter Entwicklung gezielt und überlegt, den Projektanforderungen angepasst, erfolgen, auf der anderen Seite haben Projekte, die schon mit MDA arbeiten, in den letzten Jahren erstaunliche Erfolge verzeichnet. Auch aus Artikeln in verschiedenen einschlägigen Fachmagazinen oder Konferenz-Agendas der letzten zwei Jahre lässt sich ablesen, dass der am Anfang als Hype bezeichnete MDA-Ansatz den Kinderschuhen entwachsen ist.

Einige Beispiele:

- Das Eclipse Tools Projekt entwickelt ein Subprojekt namens EMF[1] (Eclipse Modelling Framework), ein Java/XML Framework, mit dem, auf Basis von einfachen Klassen-Modellen, Werkzeuge und Applikationen generiert werden können.
- BMW entwickelt im Rahmen einer neuen Komponentenarchitektur (Component Architecture 2.0) einen MDA-konformen Transformator, der die gesamte Persistenzschicht generiert.
- Auf einer Konferenz Anfang 2005 in München werden erstaunlich viele Vorträge und Erfahrungsberichte -fünfzehn an der Zahl -, teilweise renommierter Firmen, zum Thema modellgetriebener Entwicklung zu sehen sein.
- Sowohl die Hersteller integrierter MDA-Werkzeuge, als auch verschiedenste opensource Projekte, entwickeln Ihre Produktpalette kontinuierlich weiter, um dem Einsteiger den Zugang zur MDA zu erleichtern und möglichst viel Funktionalität zu liefern.

Auch die Zukunft von open ArchitectureWare bringt einiges an Neuheiten, die in der vorliegenden Diplomarbeit noch nicht eingesetzt werden. Die Weiterentwicklung des Frameworks beinhaltet sowohl Verbesserung bestehender Komponenten, wie zum Beispiel das Eclipse-Template-Plugin oder die Generator Utilities, als auch die Entwicklung neuer Komponenten, beispielsweise ein Metamodell-Generator, der die grafische Modellierung des Metamodells in einem CASE-Tool ermöglicht und anschließend die Java-Implementierung des Metamodells generiert. Des weiteren stellt sich auch die Frage nach der Integration des generierten Quellcodes. Zum einen hat man die Möglichkeit, die vom Generator angesprochenen Geschützten Bereiche in Anspruch zu nehmen, zum anderen wird die Integration auch häufig über Ableitung bestehender Klassen und Interface-Programmierung gelöst.

1 EMF – Eclipse Modelling Framework - http://www.eclipse.org/emf (Stand: 17.10.04)

Diese und die weiter oben angesprochenen offenen Punkte bei der modellgetriebenen Entwicklung lassen sich nur durch mutigen Einsatz in Projekten und den damit hoffentlich verbundenen Erfolgsgeschichten lösen. Zusätzlich zum Projekteinsatz birgt auch das Ergebnis der zwei abgeschlossenen Diplomarbeiten weiteres Potential. Beispielsweise könnte in weiteren Arbeiten die Generierung auf unterschiedliche Plattform aus dem gleichen PIM oder eine Erzeugung von Anwendungen unter Einsatz unterschiedlicher Technologien auf der gleichen Plattform (thin/fat client, Struts/JSF,...) erfolgen.

Gespannt und aufmerksam sollte man die weitere Entwicklung abwarten und den Weg der modellgetriebenen Entwicklung vom scheinbaren Hype zum anerkannten Entwicklungsmodell mitverfolgen.

A Glossar

CIM (Computation Independent Model)

Ein CIM beschreibt Geschäftsmodelle und verfügt daher über ein spezielles Vokabular, das es erlaubt, Geschäftsprozesse, Aktionäre, Abteilungen, Abhängigkeiten zwischen Prozessen, usw. abzubilden.

EJB (Enterprise Java Beans)

Serverseitige Komponentenarchitektur zur Erstellung von verteilten, transaktionsorientierten Anwendungen mit der Programmiersprache Java.

Generator-Backend

Das Generator-Backend interpretiert die Templates, bindet sie an das Java-Metamodell und nimmt die Sourcecode-Generierung vor.

Geschützte Bereiche (Protected Regions)

Geschützte Bereiche kennzeichnen die Stellen in der generierten Anwendung, an denen Fachlogik manuell implementiert werden kann. Bei einem erneuten Generierungsdurchlauf werden diese Bereiche nicht überschrieben.

HTML (Hypertext Markup Language)

Eine SGML-konforme Beschreibungssprache, zur Entwicklung von Internetseiten

IDE (Integrated Development Environment)

Entwicklungsumgebung, die im Bereich der Softwareentwicklung eingesetzt wird. Eine IDE stellt Funktionen zur Verfügung, um Quellcode editieren, kompilieren und debuggen zu können.

Instanziator

Der Instanziator parst den XMI-Export des plattformunabhängigen Designs und instanziiert unter Zuhilfename der Instanziierungsvorschrift das Java-Metamodell.

Instanziierungsvorschrift

Die Instanziierungsvorschrift beschreibt das Mapping der mit Stereotypen ausgezeichneten und nach XMI exportierten Modellelemente des PIM, auf die Klassen des Java-Metamodells.

J2EE (Java 2 Enterprise Edition)

Klassen, Schnittstellen, Konzepte und Technologien für die Realisierung verteilter, unternehmenskritischer Softwaresysteme auf Basis der Java 2 Plattform.

Java-Metamodell

Das Java-Metamodell ist die programmatische Umsetzung des UML-Metamodells in Form Java-Klassen.

JSP (Java Server Page)

Spezifikation, die das Java Servlet API um die Generierung von Internetseiten erweitert.

JSTL (JSP Standard Tag Library)

Eine von Sun Microsystems entwickelte Tag-Bibliothek zur Unterstützung bei der JSP-Entwicklung.

MOF (Meta Object Facility)

Abstrakte Sprache, mit der Metamodelle im allgemeinen und Ihre Abhängigkeiten untereinander definiert werden können. Metamodelle, die mithilfe der MOF beschrieben sind (z.B. das UML-Metamodell), sind Instanzen der MOF.

Open ArchitectureWarwe (oAW)

Generator Framework, das ursprünglich von der b+m Informatik AG entwickelt wurde und seit September 2004 als opensource Projekt auf www.sourceforge.net zur Verfügung steht.

PIM (Platform Independent Model)

Das PIM definiert ein plattformunabhängiges Modell eines Softwaresystems, das in weiteren Transformationen auf konkrete Technologien abgebildet wird.

PSM (Platform Specific Model)

Das PSM ist ein mit plattformspezifischen Details angereichertes Modell eines Softwaresystems, das per Modelltransformation aus dem PIM entsteht. Es dient in der MDA als Vorstufe zur Code-Generierung.

QVT (Queries, Views, Transformations)

Sprache zur Beschreibung von Modelltransformationen. Zum Zeitpunkt der Erstellung der Diplomarbeit noch nicht endgültig von der OMG verabschiedet.

Tag Bibliothek

Eine Tag-Bibliothek erweitert die Tags einer JSP mit eigenen Implementierungen. Ziel ist es, über Tag-Bibliotheken oft benutzte Konstrukte oder Geschäftslogik aus einer JSP-Seite in Java-Dateien auszugliedern.

Templates

Die in der Referenzimplementierung identifizierten Architekturaspekte werden in Generator-Templates gekapselt. Diese werden zur Generierung dynamisch an die Klassen des Metamodells gebunden und steuern die Sourcecode-Generierung.

UML (Unified Modelling Language)

Grafische Modellierungssprache zur Visualisierung, Spezifizierung und Dokumentation der Artefakte eines Softwaresystems.

UML Profile

Erweiterungsmechanismus der UML um domänen-, fach- oder projektspezifische UML-Diagramme zu erstellen.

XMI (XML Metadata Interchange)

Von der OMG standardisiertes Format, das auf XML (extended Markup Language) basiert. XMI dient der textuellen Beschreibung der in UML grafisch modellierten Elemente und

ermöglicht so den Austausch von Modellen zwischen Modellierungswerkzeugen.

Xpand

Poprietäre (b+m Informatik AG) Skriptsprache zur Entwicklung der Generator Templates.

B Quellen- und Literaturverzeichnis

[Apa01] Apache Software Foundation
http://apache.org

[Apa02] Apache Software Foundation
Struts Project
http://struts.apache.org
Stand: 19.08.2004

[Jak00] Apache Software Foundation
Jakarta Taglibs Project
http://jakarta.apache.org/taglibs
Stand: 23.08.2004

[Jak01] Apache Software Foundation
Jakarta Commons Project
http://jakarta.apache.org/commons
Stand: 24.08.2004

[Sun01] Sun Microsystems Inc.
Java BluePrints – MVC Design Pattern
http://java.sun.com/blueprints/patterns/MVC-detailed.html
Stand: 19.08.2004

[Sun02] Sun Microsystems Inc.
Core J2EE Patterns
http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html
Stand:20.08.2004

[Sun03] Sun Microsystems Inc.
JavaServer Pages Standard Tag Library
http://java.sun.com/products/jsp/jstl/index.jsp
Stand: 23.08.2004

[Sun04] Sun Microsystems Inc.
JavaServer Pages Technology
http://java.sun.com/products/jsp
Stand: 23.08.2004

[OMG00] Object Management Group
http://www.omg.org
Stand: 25.08.2004

[MDA00] Model Driven Architecture Guide 1.0.1
Object Management Group
http://www.omg.org/docs/omg/03-06-01.pdf
Stand: 25.08.2004

[QVT00] Queries Views Transformations RFP
Object Management Group
http://www.qvtp.org/downloads/1.1/qvtpartners1.1.pdf
Stand: 31.08.2004

[XMI00] XML Metadata Interchange
Object Management Group
http://www.omg.org/cgi-bin/apps/doc?formal/02-01-01.pdf
Stand: 01.09.2004

[MOF00] Meta Object Facility
Object Management Group
http://www.omg.org/docs/formal/00-04-03.pdf
Stand: 03.09.2004

[oAW00] open ArchitectureWare
Opensource Projekt
http://sourceforge.net/projects/architecturware
Stand: 07.09.2004

[Neu01] Wolfgang Neuhaus, Carsten Robitzki
Eine praktische Interpretation
Javamagazin 09/03

[Sta01] Thomas Stahl, Martin Schepe, Carsten Robitzki
Model Driven Architect
Javamagazin 10/03

[Völ01] Markus Völter
Modellgetriebene Softwareentwicklung
Objektspektrum Nr.4 07/04

[Völ02] Markus Völter
Metamodellbasierte Codegenerierung
Javaspektrum Nr. 5 09/03

[Bet01] Jorn Bettin
Model-Driven Software Development
http://www.softmetaware.com/mdsd-and-isad.pdf
Stand: 07.09.2004

[Bet02] Jorn Bettin
Model-Driven Software Development Teams
http://www.softmetaware.com/distributed-software-product-development.pdf
Stand: 07.09.2004

[b+m01] Generative Devellopment Process - GDP
b+m Informatik AG
Stand: 09/04

[b+m02] open Generator Framework Referenz
b+m Informatik AG
Stand: 10/03

[b+m03] b+m Generative Development Process
b+m Informatik AG
Stand: 01/03

[b+m04] Funktionsweise b+m Generator FrameWork
b+m Informatik AG
Stand: 01/03

[Mär01] Diplomarbeit Christian Märkle
Thema: Erweiterung eines metamodell-basierten Generator-Frameworks zur Erzeugung
von Quellcode für eine mehrschichtige Client/Server Architektur am Beispiel von J2EE

[Ban01] Klaus Banzer, Rosa Rottenfußer
VERA/FCS Fachkonzept BONSAI

CD-ROM zur vorliegenden Arbeit

Inhalte:

- Diplomarbeit (PDF)
- erweiterter Generator, inkl. Templates und Metamodell (Eclipse-Projekt)
- Projektbeispiel „DealerMasterdata Administration“ (Eclipse-Projekt)
- Projektbeispiel „VERA/FCS BONSAI“ (Eclipse-Projekt)
- Javadoc des erweiterten Metamodells