FACHHOCHSCHULE
GIESSEN
FRIEDBERG

Diplomarbeit

zum Thema

Evaluierung der Model Driven Architecture

Möglichkeiten für die Umsetzung von Managementarchitekturen am Beispiel der Verwaltung von Bankperipheriegeräten

---

zur Erlangung des akademischen Grades
Diplom-Informatikerin (FH)

vorgelegt am
Fachbereich Mathematik, Naturwissenschaften und Informatik der Fachhochschule Gießen-Friedberg

von **Stephanie Weg**

Referent: **Prof. Dr. A. H. Kaufmann**

Korreferent: **Prof. Dr. O. Hoffmann**

Abgabe: **September 2003**

Danksagung

Ich möchte an dieser Stelle den Personen danken, die mich während der Erstellung meiner Diplomarbeit unterstützt haben. Vor allem in Bezug auf die MDA-Entwicklungsumgebungen war der persönliche Kontakt und Austausch mit den Mitarbeitern der Firmen sehr hilfreich.

Vielen Dank an meine Betreuer Roger Zacharias und Yvonne Stöckle der Wincor Nixdorf GmbH & Co. KG, die stets gesprächsbereit waren und sich immer Zeit genommen haben, wenn ich fachliche Fragen hatte.

Ebenso möchte ich meinen Betreuern Prof. Dr. Kaufmann und Prof. Dr. Hoffmann von der Fachhochschule Gießen-Friedberg danken, die mir ermöglicht haben, diese Arbeit zu verfassen und jederzeit für Rückfragen zur Verfügung standen.

Danke auch an alle, die meine Diplomarbeit korrekturgelesen haben und sie durch zahlreiche Verbesserungsvorschläge aufgewertet haben.

Mein ganz besonderer Dank geht an Joachim Rehm, der mir durch großes Interesse am Thema geholfen hat, einen Gesprächspartner zu finden, mit dem ich mich austauschen konnte.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ABD	Analysis-by-Design
ANSI	American National Standards Institute
API	Application Programming Interface
B2B	Business-to-Business
B2C	Business-to-Consumer
CA	Convergent Architecture
CARAT	Cartridge Architecture
C-BOM	Convergent Business Object Modeler
CCM	Change and Configuration Management
C-GEN	Convergent Translative Generator
C-GEN-IDE	Convergent Generator IDE
CIM	Computation Independent Model
C-IX	Convergent Implement, Deploy and Test Environment
CM	Code Model
C-MOD	Federated UML/XML Repository
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CP	Client Personality
C-RAS	Convergent Pattern Refinement Assistant
CRC	Class Responsibility Collaboration
C-REF	Convergent UML Refinement Assistant
CWM	Common Warehouse Metamodel
DBMS	Database Management System
EAR	Enterprise Application Archive
EJB	Enterprise Java Bean
EPM	Evolutionary Project Management
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IP	Internet Protocol
IT	Information Technology

J/XFS	J/Extensions for Financial Services
J2EE	Java 2 Enterprise Edition
JAR	Java Archive
JMI	Java Metadata Interface
JMX	Java Management Extensions
JSP	Java Server Pages
JVM	Java Virtual Machine
MBean	Managed Bean
MDA	Model Driven Architecture
M-let	Management Applet
MOF	Meta Object Facility
OCL	Object Constraint Language
OMG	Object Management Group
OPR	Object Process Resource
ORB	Object Request Broker
PA	Protected Area
PIM	Platform Independent Model
PSM	Platform Specific Model
RASC	Reduced Abstraction Set Computing
RMI	Remote Method Invocation
ROI	Return On Investment
RUP	Rational Unified Process
SFD	software factory definition file
SOAP	Simple Object Access Protocol
SP	Server Personality
TCP	Transmission Control Protocol
TPC	Technology Projection Component
TPL	Template Pattern Language
UML	Unified Modeling Language
VM	Virtual Machine
WAR	Web Application Archive
WOSA	Windows Open Services Architecture for Financial Services
WWW	World Wide Web
XMI	XML Metadata Interchange
XML	Extensible Markup Language

Abbildungsverzeichnis

Never send a man to do a machine’s job.

(Agent Smith, Matrix, The Wachowski Brothers 1999)

1 Einführung

1.1 Einleitung

"Bridges generally work reliably. Large software systems generally don't. The essential difference is in design complexity, and in our inability to tame it."[1]

Barry Morris, CEO von IONA Technologies, fasst in diesem Satz sehr treffend eines der Kernprobleme aktueller Software und damit einen Ansatzpunkt von MDA zusammen.

Software ist heutzutage ein fester Bestandteil des Alltages. Vom Content Management System über elektronische Verkehrssteuerung bis hin zum Notensatz: fast alles wird heute mit Hilfe von Computern erledigt. Um so wichtiger ist die Zuverlässigkeit der eingesetzten Software. Hier sind jedoch häufig Defizite vorhanden. Des weiteren fällt in der Softwareentwicklung mangelnde Wiederverwendung, schlechte Kommunikation in interdisziplinären Teams sowie die wenig wertvolle ad-hoc-Qualität der entwickelten Produkte negativ auf. Hinzu kommt die rasante Entwicklung neuer Technologien, die immer neue Möglichkeiten eröffnen und deren Vorteile genutzt werden sollen. Ein Anpassen der Software auf die neue Technologie ist jedoch meist mit sehr großem Aufwand, häufig aber auch mit einer Neuentwicklung der Software verbunden. Wie aber kommt es zu diesen Defiziten?

Da die Informatik eine vergleichsweise junge Technologie ist, wird hier noch anders gearbeitet als in vielen anderen Bereichen der Industrie. Kein Maschinenbauingenieur käme auf die Idee, ein komplexes Gerät zu entwickeln, ohne auf bereits bestehende Baupläne, technische Ansätze und Anleitungen zurückzugreifen. In der Softwareentwicklung ist die Lage anders: Hier wird oft „from scratch" entwickelt, viele Projekte erfinden das Rad neu, ohne auf in anderen Projekten bereits gewonnene Erkenntnisse zurückzugreifen. Die Folgen sind hoher Zeit- und Geldaufwand für qualitativ mäßige Software. Wenn Flugzeug- oder Automobilhersteller so arbeiten würden wie Softwareentwickler, wären die Auswirkungen katastrophal. Es wäre lebensgefährlich, Transportmittel zu benutzen, die sich ähnlich unzuverlässig verhalten und schlecht zu warten sind wie Software. Langfristig ist ein solches Vorgehen in der Softwareentwicklung nicht akzeptabel.

Als ein Weg aus diesem Dilemma hat sich in den letzten Jahren Model Driven Architecture (MDA, modellgetriebene Architektur) entwickelt. Spezifiziert von der OMG[2] stellt die neue

[1] Aus [CONARC].

Technologie dem Softwareentwickler in Aussicht, durch den Einsatz von MDA „Mehr Software schneller mit mehr Qualität“[3] zu entwickeln. Der Lösungsansatz, um diesen Erfolg zu erzielen, lautet im Wesentlichen: Sorgfältig auf hohem Abstraktionslevel modellieren, dann einen Großteil des Codes spezifisch für die gewünschte Plattform generieren lassen. So soll es möglich sein, Anwendungen[4] oder Teile von Anwendungen wieder zu verwenden. Weiterhin verspricht das Generieren von Quellcode, also das Erledigen von Standardaufgaben durch Maschinen, eine niedrigere Fehlerquote, höhere Produktivität sowie niedrigere Kosten.

1.2 Zielsetzung

Ziel dieser Diplomarbeit ist es, zu evaluieren ob und in welchem Rahmen sich der Einsatz der Model Driven Architecture für die Wincor Nixdorf GmbH & Co. KG im Bereich Managementsysteme lohnt.
Wincor Nixdorf ist einer der weltweit führenden Anbieter von IT-Lösungen und -Produkten im Bereich Handel und Finanzen. Durch den Einsatz der Anwendungen bei vielen verschiedenen Kunden wird die Firma immer wieder mit der Problematik unterschiedlicher Software- und Hardware-Voraussetzungen für die gleiche Anwendung konfrontiert. Hier könnte durch den Ansatz der Model Driven Architecture eine bedeutende Erleichterung geboten werden.
Bei der Evaluierung der Technologie wird folgendermaßen vorgegangen: Zunächst erfolgt ein theoretischer Überblick über die Model Driven Architecture. Es werden Technologien vorgestellt, die entweder für die MDA oder für die Architektur der Beispielanwendung, die später umgesetzt wird, von Bedeutung sind. Zusätzlich werden drei ausgewählte Werkzeuge für die Softwareentwicklung mit MDA vorgestellt und ihre Funktionalität erläutert.
Darauf folgend wird die Anwendung der Werkzeuge in der Praxis vorgestellt sowie Stärken und Schwächen hervorgehoben. Dies geschieht im Rahmen der prototypischen Realisierung einer (in etwas umfangreicheren Form) bei Wincor Nixdorf eingesetzten Managementarchitektur mit jedem der drei vorgestellten Werkzeuge.
Abschließend wird eine Analyse der Ergebnisse vorgenommen, in welcher der Einsatz von MDA im Allgemeinen und die einzelnen Werkzeuge im Besonderen auf ihren Nutzen für

---

[2] Die Object Management Group Inc. ist eine unabhängige, nicht gewinn-orientierte Organisation, welche von mehreren großen internationalen Firmen gegründet wurde um Standards für die Software-Industrie zu entwickeln (vgl. www.omg.org).
[3] Vgl. [BITP02a].
[4] Im Folgenden auch „Systeme“.

Wincor Nixdorf geprüft werden. Die Ergebnisse werden in einem Überblick dargestellt und in einem Fazit zusammengefasst.

1.3 Vorgehensweise und Aufbau der Arbeit

Diese Arbeit ist in fünf Kapitel eingeteilt. Nach Erläuterung der Fragestellung und des Zieles der Arbeit (Kapitel 1) folgt der in einen theoretischen und einen praktischen Abschnitt gegliederte Hauptteil. Im theoretischen Teil (Kapitel 2) werden Paradigmen, Technologien und drei ausgewählte Entwicklungsumgebungen im Zusammenhang mit Model Driven Architecture erläutert. Weiterhin werden die Basistechnologien einer Managementarchitektur vorgestellt. Diese Managementarchitektur wird im praktischen Teil (Kapitel 3) prototypisch mit MDA umgesetzt. Hierbei wird die Umsetzung mit jeder der drei vorgestellten Entwicklungsumgebungen individuell dargestellt, da nur so den besonderen Eigenschaften und Fähigkeiten Rechnung getragen werden kann. In Kapitel 4 wird eine ausführliche Bewertung der Ergebnisse des Einsatzes von Model Driven Architecture und der einzelnen Werkzeuge vorgenommen, welche dann in Kapitel 5 zu einem Ergebnis im Hinblick auf die Fragestellung aus Kapitel 1 zusammengefasst wird.

An dieser Stelle soll darauf hingewiesen werden, dass zu dem Thema sehr wenig gedruckte Literatur existiert und dass der größte Teil in englischer Sprache verfasst ist. Dies führte vor allem bei der Verwendung von Fachbegriffen zu Problemen, da für diese Begriffe zum großen Teil noch keine deutschen Synonyme existieren. Daher wird folgende Vorgehensweise angewandt: Im theoretischen Teil steht bei der ersten Verwendung eines Begriffes die englische Version in Klammern hinter der deutschen Übersetzung. Um ein einheitliches Sprachbild zu erreichen, wird danach nur noch der deutsche Begriff verwendet. Dies gilt nicht für den praktischen Teil. Begriffe aus den MDA-Entwicklungsumgebungen werden in der Originalfassung verwendet, um die Orientierung in den Abbildungen zu erleichtern.

2 Technologien, Tools und Architekturen

Model Driven Architecture ist kein alleinstehendes Konzept, sondern fasst einige bewährte Technologien und deren Anwendung in einem Standard zusammen. Im Folgenden wird ein Überblick über die Konzepte von MDA gegeben. Die wesentlichen Technologien, die beim Einsatz von MDA zum Tragen kommen, werden im Detail vorgestellt und, soweit möglich, bewertet (UML, MOF, CWM). Es wird ein Überblick über die Convergent Architecture als Architekturstil für MDA gegeben und eine Bewertung vorgenommen. Weiterhin wird eine Einführung in drei MDA-Tools/Entwicklungsumgebungen gegeben, um einen praktischen Eindruck der heute für MDA existierenden Software zu geben. Mit diesen Tools wird später die praktische Umsetzung eines Managementsystems realisiert. Dieses System nutzt die Technologien JMX und J/XFS – sie werden am Ende von Kapitel zwei vorgestellt.

2.1 MDA und verwandte Technologien

2.1.1 Konzepte der Model Driven Architecture

2.1.1.1 Problemstellung

Wie in der Einleitung bereits dargelegt, befindet sich die Softwarebranche in einer Krise. In [FRAN03], S. 26 ff. werden die wesentlichen Anforderungen und Problembereiche in einzelne Aspekte untergliedert:

- Komplexität

  Im Gegensatz zur letzten Generation der Informationstechnologie, als unternehmensinterne Datenverarbeitung in einem einheitlichen System vollkommen ausreichend war, müssen heute vielfältige Endbenutzer-Systeme und -Geräte unterstützt werden. Dies sind beispielsweise Fat Clients, Web Clients und Handhelds, wobei jede Art von Geräten eine individuelle Behandlung erfordert. In diesem Umfeld sollen Funktionen nicht nur innerhalb des eigenen Unternehmens sondern auch für B2B- und B2C-Beziehungen realisiert werden. Sowohl die Anforderungen an die Funktionalität als auch an die Kompatibilität der Software steigen. Dies führt zu einer höheren Komplexität der Systeme.

- Produktionskosten

  Je komplexer Systeme werden, desto aufwändiger ist die Realisierung. Zum einen werden mehr Entwickler benötigt, zum anderen wird die Software in ihrer Gesamtheit schwerer zu erfassen, zu verwalten und zu warten sein. Dies erzeugt einen Mehraufwand, der letzten Endes finanziell zu Buche schlägt.

- Qualität

  Durch Voraussetzungen wie erhöhte Komplexität, knappe Budgets und enge Zeitrahmen werden während der Softwareentwicklung häufig Einschränkungen bei der Qualität gemacht. Die Entwickler haben nicht die Möglichkeit, Systeme genau zu planen und zu strukturieren, wie es in anderen Ingenieursdisziplinen der Fall ist. Es wird ohne umfassenden Plan mit der Implementierung begonnen: Das Resultat ist das Scheitern des Projekts oder qualitativ minderwertige Software, die im täglichen Gebrauch nicht zuverlässig arbeitet.

- Langlebigkeit

  Durch die erhöhte Bedeutung von Software werden auch die Innovationsbemühungen verstärkt. Bestehende Technologien werden verbessert bzw. ändern sich signifikant und neue Technologien entstehen. In der Konsequenz werden manche ältere Technologien nicht mehr unterstützt, der Support wird eingestellt und die Kompatibilität zu neuen Entwicklungen oder Fremdsystemen lässt zu wünschen übrig. Ein Softwaresystem, welches zwei Jahre alt ist und mit großem Aufwand vorbildlich entwickelt wurde, kann durch die Wahl der „falschen“ Technologie wertlos werden.

Wie aus diesen Punkten zu erkennen ist, bestehen vielfältige und gegensätzliche Anforderungen, die mit den altgedienten Methoden der Softwareentwicklung nicht erfüllt werden können. Neue Konzepte, wie bspw. Design Patterns und Middleware wurden entwickelt um Abhilfe zu schaffen. Sie sind jedoch auf bestimmte, abgegrenzte Bereiche beschränkt.
Am 08. März 2001 stellte die OMG eine neue Technologie vor, die viele der beschriebenen Probleme durch ein Zusammenfassen dieser Konzepte behandelt: *Model Driven Architecture.*

2.1.1.2 Überblick

Die Model Driven Architecture ist eine offene und herstellerunabhängige Technologie. Sie umfasst den kompletten Lebenszyklus der Software von technologie-unabhängiger Model-

lierung über Entwicklung und Integration auf fast jeder Plattform bis hin zu Deployment und Wartung[5]. Damit trägt sie den im vorigen Kapitel genannten Problemen Rechnung.
Wichtig ist die Tatsache, dass die Model Driven Architecture selbst keine vollständig neue Technologie ist, sondern vielmehr die Integration vieler bereits bestehender Standards und Technologien, die so optimal genutzt werden können.
In [OMG 03c] wird zusammengefasst:

> *"MDA provides an approach for, and enables tools to be provided for:*
> - *specifying a system independently of the platform that supports it,*
> - *specifying platforms,*
> - *choosing a particular platform for the system, and*
> - *transforming the system specification into one for a particular platform."*

In Abbildung 1 wird ein theoretischer Überblick über MDA und die damit verbundenen Technologien gegeben.
Den Kern der Architektur bilden die drei OMG-Modellierungsstandards *Unified Modeling Language (UML), Meta Object Facility (MOF)* und *Common Warehouse Metamodel (CWM)*. Auf diese Standards und ihre Eigenschaften wird in den Kapiteln 2.1.2 bis 2.1.4 näher eingegangen. Mit ihrer Hilfe wird ein *datenverarbeitungsunabhängiges Modell (CIM[6])* und ein *plattformunabhängiges Modell (PIM)* der zu entwickelnden Software erstellt. Dieses Modell wird in ein *plattformspezifisches Modell (PSM)* transformiert[7]. Mögliche Zielplattformen für diese Transformation werden in der obigen Abbildung im inneren Ring dargestellt. Es ist möglich, einen Teil dieser Transformation zu automatisieren, indem Umsetzungsregeln definiert werden. Dieses plattformspezifische Modell kann dann, wieder zum Teil automatisiert, in ein *Code Modell (CM),* also konkrete Artefakte wie Quelltext, Deployment-Informationen u.a. umgesetzt werden (in der Abbildung nicht dargestellt). Die Eigenschaften von PIM, PSM und CM werden in Kapitel 2.1.1.4 genauer erläutert.
Im äußeren Ring der Abbildung sind *Pervasive Services*, wie sie von jedem verteilten System benötigt werden, dargestellt. Unter solche Dienste fallen beispielsweise Verzeichnisdienste, Ereignisbehandlung, Persistenz, Transaktionen und Sicherheit. Diese Services können bei der modellgetriebenen Entwicklung automatisch erzeugt werden.

---

[5] In Anlehnung an [SJSD02].
[6] CIM basiert auf dem englischen Begriff Computation Independent Model.
[7] Eine detaillierte Erklärung der Modelle ist in Kapitel 2.1.1.4 zu finden.

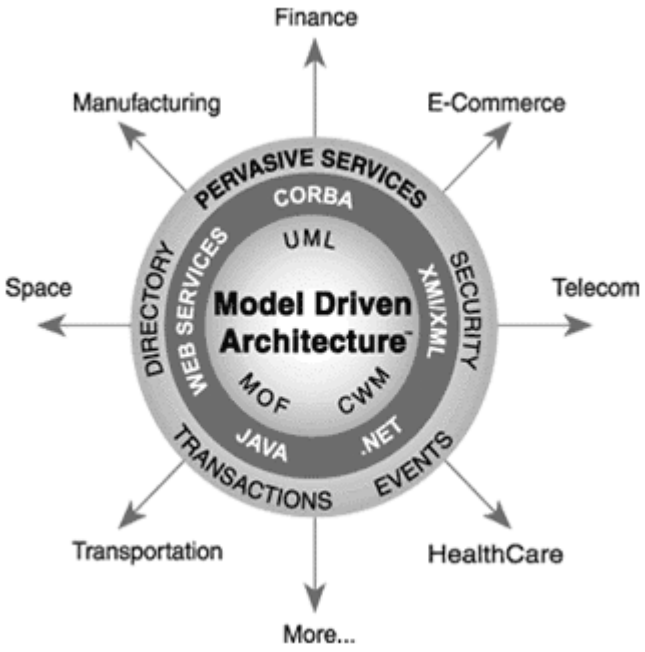


Abbildung 1: MDA und verwandte Technologien[8]

Es ist leicht zu erkennen, dass neue Technologien und Standards ohne Probleme in der entsprechenden Schicht des Modells eingefügt werden können und lediglich die Schnittstelle (oder Transformation) zwischen den Schichten angepasst werden muss, ohne dass die bereits bestehenden Modelle unbrauchbar werden. In ihrer Gesamtheit ist die Model Driven Architecture auf unterschiedlichste Bereiche anwendbar (angedeutet durch die Pfeile und die verschiedenen Kategorien wie Telekommunikation, Finanzen etc.), wobei auch hier Erweiterungsmöglichkeiten zur Verfügung stehen. Mit Hilfe der modellgetriebenen Entwicklung ist eine Einbindung von Legacy-Systemen ebenfalls möglich.

[SJSD02] erklärt:

> *„Any legacy application based on a UML model and a supported middleware platform can be included in a company's circle of MDA interoperability by simply importing its model into MDA tools […]. Lack of a model is not a barrier; tools on the market today reverse-engineer UML models from code, and some even work from executables. Alternatively, stand-alone legacy applications can be wrapped with a layer of code that exposes key functionality to the network on a suitable middleware, and the model for this functionality and its interfaces stored in a library for use by MDA developers."*

Es existieren also vielfältige Möglichkeiten um bestehende Systeme in die durch MDA geschaffene Systemlandschaft zu integrieren.

---

[8] Quelle: [OMGMDA].

2.1.1.3 Plattformunabhängigkeit

Bevor näher auf die verschiedenen plattformunabhängigen und plattformabhängigen Abstraktionsgrade eingegangen wird, muss eine Abgrenzung definiert werden. *Plattformunabhängigkeit* an sich ist ein relatives Konzept[9]: CORBA wird als plattformunabhängig bezeichnet, weil Programmiersprachen und Betriebssysteme für diese Architektur nicht von Bedeutung sind. Betrachtet man aber verschiedene Middleware-Lösungen, so ist ein CORBA-spezifisches Modell sehr wohl plattformabhängig. Es ist also eine Frage des Abstraktionsgrades, ob ein Modell plattformunabhängig ist oder nicht.
[FRAN03] definiert Plattformunabhängigkeit folgendermaßen:

> *„When we use the term platform-independent, I mean independent of the following:*
> - *Information-formating technologies, such as XML DTD and XML Schema*
> - *3GLS and 4GLS, such as Java, C++, C# and Visual Basic*
> - *Distributed component middleware, such as J2EE, CORBA and .NET*
> - *Messaging middleware, such as WebSphere MQ Integrator (MQSeries) and MSMQ"*

Diese Definition gilt für den Rest dieser Arbeit.

2.1.1.4 CIM, PIM, PSM, CM und Transformationen

In [OMG 03c] werden drei verschiedene Abstraktionsebenen bei der modellgetriebenen Entwicklung spezifiziert.

- *datenverarbeitungsunabhängiges Modell*
  (Computation Independent Model, CIM)
  Dieses Modell zeigt das System aus einem Blickwinkel, der sich auf die reine Funktionalität konzentriert. Details in Struktur und Kontrollfluss bleiben verborgen, lediglich die Funktionalität, wie sie ein Experte aus der Fachabteilung beschreiben kann, wird abgebildet. Das CIM dient vor allem der Überbrückung der Kommunikationsbarriere zwischen Anwender und Entwickler: Hier werden in einer implementierungsneutralen Sprache die Anforderungen an das System bestimmt.
- *plattformunabhängiges Modell*
  (Platform Independent Model, PIM)

[9] Vergleiche [FRAN03], S. 48f..

Das PIM setzt die im CIM definierte Funktionalität in Strukturen um. Es werden Objekte und Abhängigkeiten des Systems modelliert, allerdings ohne sich auf eine bestimmte Plattform festzulegen. Der Abstraktionsgrad wird so gewählt, dass keine plattformspezifischen Informationen benötigt werden.

- *plattformspezifisches Modell*

  (Platform Specific Model, PSM)

  Hier wird nun das PSM um plattformspezifische Informationen erweitert. Es wird festgelegt, wie einzelne Elemente oder Abhängigkeiten in einer bestimmten Umgebung umgesetzt werden. Das PSM enthält in Form eines UML Modells die gleichen Informationen wie der fertige Code (CM).

- *Code-Modell*

  (Code Model, CM)

  Zusätzlich zu den von [OMG 03c] spezifizierten Abstraktionsebenen arbeiten die MDA-Werkzeuge mit dem Code-Modell als niedrigstem Abstraktionsgrad. Dieses Modell enthält den (basierend auf dem PSM) generierten Quellcode und andere Artefakte wie z.B. Deployment-Informationen.

Das Konzept von MDA basiert auf modellzentrierter Entwicklung. Dabei wird im Gegensatz zur codezentrierten Entwicklung davon ausgegangen, dass das Modell der Mittelpunkt der Entwicklung ist. Dies äußert sich u.a. darin, dass Änderungen in das Modell fließen und nicht direkt in den Code. Weiterhin sieht das Konzept der modellbasierten Entwicklung mit MDA nur eine Übersetzung von höheren in niedrigere Abstraktionsgrade vor. Ein Reverse-Engineering aus dem CM in ein PSM oder PIM ist mit geeigneten Werkzeugen zwar möglich, entspricht aber nicht dem Gedanken von MDA. Für den Übersetzungsvorgang von einem Modell in ein anderes werden Regeln benötigt, die eine Projektion ermöglichen. Diese Regeln werden *Model Transformations*, oder *Transformationen* genannt. [OMG 03c] nennt als Beispiel für eine Transformationsregel von PIM nach PSM:

> *“If the attribute “sharable” of class “entity” is true for a particular PIM model instance of type entity, then map to an EJB Entity, otherwise map to a Java Class.”*

Diese Regeln werden je nach Hersteller des MDA-Werkzeugs in einer Sprache wie Java oder Jython definiert, aber auch von den Firmen eigens für diesen Zweck entwickelte Template-Sprachen werden verwendet.

2.1.1.5 Stärken und Schwächen

Die Verwendung der modellgetriebenen Architektur bietet eine Reihe von Vorteilen im Vergleich zur herkömmlichen Softwareentwicklung. Durch modellzentrierte Entwicklung statt codezentrierter Entwicklung wird die Software wesentlich kontrollierter entwickelt. Ohne den Einsatz von MDA werden die Modelle von anderen Personen entwickelt als der Code. Programmierer sehen die Modelle meist als Richtlinien oder grobe Vorgaben für die Implementierung an[10], häufig aber entfernt sich mit fortschreitendem Projektverlauf die Implementierung immer weiter vom Modell, spätere Änderungen werden meist nur im Code vorgenommen. Somit verliert das Modell nach und nach an Aktualität und Wert, bis es schließlich kaum noch von Nutzen ist: Die Arbeitszeit, die für die Modellierung aufgewendet wurde, ist verloren. Da bei der modellgetriebenen Entwicklung Änderungen in das Modell eingepflegt werden (aus dem dann neuer Code erzeugt wird), bleibt das Modell stets aktuell. Manche Sachverhalte können nicht modelliert werden und müssen manuell in den Code eingefügt werden. Die Entwicklungsumgebungen schützen diese eingefügten Abschnitte und stellen so sicher, dass diese bei einer erneuten Generierung erhalten bleiben. Der Unterschied bei dieser Vorgehensweise ist, dass Wissen und Arbeit der Entwickler in Modelle fließt, welche auch für Außenstehende leichter zu überblicken sind als Code. Das Wissen wird also von den Entwicklern entkoppelt und das Unternehmen ist nicht mehr auf einzelne Personen angewiesen. Bei codezentrierter Entwicklung hingegen fließt das Wissen in den Code, was eine Einarbeitung ungleich schwerer macht. Weiterhin können bei einem Plattformwechsel die im PIM abgelegten Konzepte auf die neue Plattform umgesetzt werden.

Durch den Einsatz von MDA wird Wiederverwendung gefördert, da für ein neues System bewährte Modelle bereits bestehender Systeme verwendet werden können. Die Qualität der Software steigt, da die Transformationen auf vielfach bewährten Design Patterns basieren, die von den besten Entwicklern erstellt wurden. Die Produktivität wird erhöht: Hersteller von MDA-Werkzeugen geben an, dass ihre Tools je nach Anwendungsbereich 40%-90%[11] des Quellcodes automatisch erstellen können. Zusätzlich unterlaufen Generatoren anders als menschlichen Programmierern keine Flüchtigkeitsfehler, der von ihnen erstellte Code hat eine gleichbleibend hohe Qualität; vor allem Fleißaufgaben wie bspw. die Anbindung an andere Plattformen werden zuverlässig erledigt. Diese Faktoren sowie die Tatsache, dass auch Test- und Deployment-Informationen automatisch erstellt werden

[10] In Anlehnung an [SJSD02].
[11] Aus [OMG 03e].

können, erhöhen die Produktivität und senken die Kosten. Zusammenfassend lässt sich sagen, dass die Model Driven Architecture das Potential hat, die in Kapitel 2.1.1.1 dargestellten Probleme zu mindern. Ob und in welchem Umfang dies in der Realität der Fall ist, wird im praktischen Teil näher diskutiert.

Experten vergleichen den Schritt von der Arbeit mit Programmiersprachen der 3. Generation zur Verwendung der Model Driven Architecture mit der Umstellung von handgeschriebenen Assemblercode auf die Verwendung von Compilern: Die Entwicklung des Systems wird eine Abstraktionsstufe nach oben verlegt. Auch damals wurde die neue Technik zunächst mit Skepsis behandelt, da man sich nicht vorstellen konnte, dass Maschinen eine Arbeit ebenso gut ausführen können wie ein Mensch. In der Tat war der von Entwicklern handgeschriebene Assemblercode effektiver. Aber mit der Zeit setzten sich Compiler durch, weil sie es den Entwicklern erlaubten, auf einem höheren Abstraktionsgrad und somit effektiver und weniger fehleranfällig zu arbeiten. Hinzu kam die Tatsache, dass Rechenleistung preiswerter zur Verfügung stand als menschliche Arbeitszeit, so dass die Einsparung von Entwicklern und den damit verbundenen Kosten zu einem natürlichen Ziel wurde. Betrachtet man den heutigen Stand, so wird klar, dass viele der aktuellen Systeme mit reinem Assemblercode nicht realisierbar wären. Der gleiche Effekt zeichnet sich bei MDA ab. Auch wenn die generierten Anwendungen um einen Bruchteil weniger optimiert sind, so sind die Maschinen doch in der Lage, wesentlich komplexere Strukturen zu entwerfen und umzusetzen. Ein Mitarbeiter der Firma BITPlan berichtete, dass mit MDA ein Algorithmus realisiert wurde, welcher für einen menschlicher Programmierer zu komplex gewesen wäre, um ihn noch überblicken zu können[12].

Es gibt auch Nachteile beim Einsatz von MDA. Der Hauptentwicklungsaufwand verschiebt sich von der Implementierung auf das Design, so dass ein Umdenken in den Köpfen der Entwickler erfolgen muss. Tendiert man heute eher zu Ansätzen wie XP[13], so wird mit MDA ein Schritt in die entgegengesetzte Richtung getan. Ohne ein Entwicklungsteam mit detaillierten Modellierungskenntnissen und hoher Bereitschaft zur modellzentrierten Entwicklung sowie großer Disziplin kann die modellgetriebene Entwicklung nicht erfolgreich sein.

[12] Aus [BITPSK].

[13] Extreme Programming (XP) ist eine Technik, die bei einem Minimum von Design auf eine möglichst schnelle Implementierung abzielt.

Weiterhin ist zu beachten, dass man sich unter Umständen *„die Plattformunabhängigkeit bei Modellierung und eventuell auch der Software durch eine starke Abhängigkeit von einem Werkzeug erkauft.“*[14]

Auch der Zeitpunkt des ROI[15] wird durch erhöhten Aufwand in der ersten Hälfte des Software-Lebenszyklus deutlich nach hinten verschoben – so dass sich laut [OBSP03b] „[...] *die Nutzenaspekte zum großen Teil erst mittel- oder langfristig bemerkbar machen“.*

Modellgetriebene Entwicklung ist also keine Technologie für den ad-hoc-Einsatz. Sowohl wirtschaftlich als auch im Personalbereich muss ihr Einsatz sorgfältig geplant werden.

[14] Aus [OBSP03b], S. 24.
[15] Return-On-Investment.

2.1.2 Unified Modeling Language

2.1.2.1 Überblick

Die Unified Modeling Language (UML) ist eine 1997 von der Object Management Group standardisierte Modellierungssprache, mit deren Hilfe Software-Systeme spezifiziert, visualisiert und dokumentiert werden können. UML (aktuelle Version 1.5) spielt eine bedeutende Rolle im Zusammenhang mit der modellgetriebenen Entwicklung, da diese Sprache zum Spezifizieren der Modelle (CIM, PIM und PSM) verwendet werden kann. Im Rahmen dieser Arbeit sollen keine Grundkonzepte von UML erläutert werden, vielmehr soll auf die für die modellgetriebene Entwicklung besonders wichtigen Eigenschaften von UML eingegangen und ein Ausblick auf die demnächst erscheinende Version 2.0 der Modellierungssprache gegeben werden.

2.1.2.2 UML und MDA – Stärken und Schwächen

Bei der modellbasierten Entwicklung werden die meisten Modelle (PIM und evtl. PSM) mit UML erstellt. Es ist zwar auch möglich andere strukturierte Daten als Quelle zu verwenden, jedoch wird in den meisten Fällen auf UML zurückgegriffen. Besonders wichtig ist dabei die Kopplung von UML und MOF. Diese erlaubt es dem Benutzer, die Modellierungsmöglichkeiten auf immer neue Problemstellungen anzupassen und so neue Technologien einzubinden. Die gängigen MDA-Werkzeuge verwenden vor allem Klassendiagramme, aber auch Aktivitäts- und Zustandsdiagramme kommen zur Anwendung. Dabei stoßen die Entwickler häufig an die Grenzen von UML, und es wird klar, dass bei dem aktuellen Stand der Modellierungsspezifikationen nicht alles modelliert werden kann, was später im Code umgesetzt werden muss. Strukturen und Abhängigkeiten sind leicht umzusetzen, aber bei der Darstellung von Geschäftslogik fehlen häufig die Mittel, um wesentliche Abläufe zu modellieren, was auch im praktischen Teil dieser Arbeit deutlich wird. Bemängelt werden derzeit außer der mangelnden Darstellungsmöglichkeiten von Geschäftslogik laut [FRAN03] Punkte wie zu geringe Modularität, ungenügende Unterstützung verschiedener Blickwinkel, teilweise unklare Definitionen. Auch das Fehlen eines Metamodells für die Object Constraint Language und die damit nicht explizit festgelegte Struktur der Sprache fallen negativ auf. Weiterhin besteht keine klare Trennung oder Schnittstelle zwischen UML und MOF, was ebenfalls ein Änderungsanliegen ist. Mit UML Version 2.0 versucht die OMG diese Mängel zu beheben.

2.1.2.3 UML 2.0

"UML 2.0 is the first major revision to the standard since its inception in 1997."

So beginnt in [IBM03] die Beschreibung des neuen Modellierungsstandards, der Verbesserungen der oben genannten Probleme umsetzen soll. Bei der Spezifikation von UML 2.0 wurde ein besonderes Augenmerk auf die Anforderungen der MDA an die Modellierungssprache gerichtet (siehe [HEIS03]). In [ALEX03] wird allerdings davor gewarnt, revolutionäre Neuerungen zu erwarten. Es gebe allerdings „wesentliche Verbesserungen" der bestehenden Konzepte, von denen viele die modellgetriebene Entwicklung begünstigen. Der Autor fasst zusammen:

„Allgemeines Ziel sind erheblich präzisere und durchgängigere Modellierungsmöglichkeiten sowie eine formale, vollständig definierte Semantik, um die Modelle ausführen und automatisch weiterverarbeiten zu können. Ferner soll die Sprache einfacher und von überflüssigen oder kaum genutzten Konstrukten befreit werden."

Die OMG hat die Änderungsanforderungen in vier Spezifikationspakete aufgeteilt:
Diagram Interchange, Superstructure, Infrastructure und Object Constraint Language (siehe Abbildung 2). Im Folgenden werden diese Pakete kurz beschrieben und die wichtigsten Neuerungen aufgezählt[16].

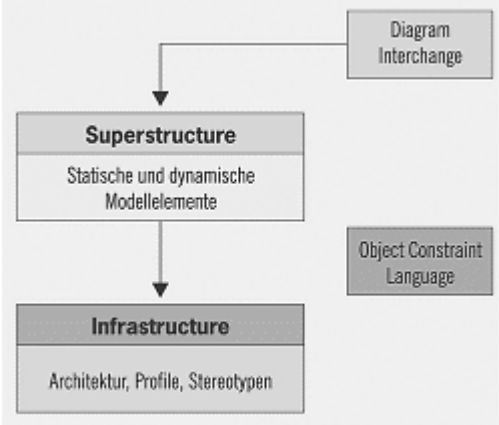


Abbildung 2: Spezifikationspakete für UML 2.0[17]

- Diagram Interchange:
  Hier wird das Format definiert, mit welchem Werkzeuge die Diagramm-Informationen austauschen können.
  - Standardisierung eines einheitlichen und vor allem vollständigen Austauschformats für UML-Diagramme auf der Basis des Standards „XML Metadata Interchange" (XMI), das voraussichtlich auch Layout-Informationen berücksichtigen wird, was bisher nicht der Fall war.
- Superstructure:
  Hier werden statische und dynamische Modellelemente definiert.
  - Sequenzdiagramme können nun in Subroutinen aufgeteilt werden.

[16] In Anlehnung an [ALEX03], [FRAN03] und [OMG 03g].
[17] Quelle: [ALEX03].

  - Aktivitätsdiagramme sind nun keine Spezialform des Zustandsdiagramms mehr, sondern eine eigene Kategorie im Metamodell. Sie erhalten eine große Ähnlichkeit mit Petrinetzen, was ihre Ausdrucksfähigkeit stark erhöht[18].
  - Unterstützung für komponenten-basierte Entwicklung (EJB, CORBA, COM+).
  - Verbesserte Unterstützung für ausführbare Modelle.
  - Konzepte aus dem Real-Time-Object-Oriented Modeling wurden in UML 2.0 realisiert, so z.B. Parts, Connectors und Ports, Timing-Diagramme.
  - Ausführbare Modelle (executable UML) werden unterstützt.
- Infrastructure:

  Hier wird der Kern der Architektur, Profile und Stereotypen in UML behandelt.
  - Das Metamodell wird verbessert und eindeutig definiert, so dass keine Unklarheiten bzgl. der Struktur mehr bleiben.
  - Durch eine stärkere Modularität im Metamodell wird eine Wiederverwendung von einzelnen Modellteilen erleichtert.
  - Verbesserungen des Profiling-Mechanismus erlauben den Anwendern, eigene Metaklassen zu definieren, was es leichter macht, neue Bereiche für das Modellieren zu erschließen.
  - Die unklare Ausrichtung von MOF und UML wird behoben.
- Object Constraint Language

  Die noch wenig bekannte OCL definiert eine Sprache zur Beschreibung von Zusicherungen, Invarianten, Vor- und Nachbedingungen sowie Navigation innerhalb von UML-Modellen.
  - Es wird ein Metamodell für OCL entworfen und in das UML-Metamodell integriert.

In seiner Gesamtheit wird UML 2.0 deutlich mehr Möglichkeiten bieten als die aktuelle Version und wird damit die modellgetriebene Entwicklung sehr viel einfacher machen. Jochen Seemann, Produkt-Manager bei IBM Rational fasst zusammen[19]:

> *"It's not enough to draw some lines and boxes; all the tools must understand the reasons behind the lines and boxes."*

UML 2.0 bietet die Möglichkeiten, diese neue Ebene des Modellierens zu erreichen. Zur Zeit in der Betatest-Phase, wird das Release für Ende 2003 veranschlagt[20].

---

[18] Aus [OEWE03].
[19] In [AMBR03a].
[20] Aus [ILOG03].

2.1.3 Meta Object Facility

2.1.3.1 Überblick

Die Meta Object Facility (MOF), aktuell in der Version 1.5, ist ebenso wie UML ein Standard der OMG. 1997 spezifiziert, wird mit MOF ein Modell-Repository zur Verfügung gestellt, welches die Verwaltung MOF-konformer Metamodelle unterstützt.
In [OMGCWM] wird zusammengefasst:

> *"MOF is an extensible model driven integration framework for defining, manipulating and integrating metadata and data in a platform independent manner."*

Zum Definieren der Syntax entleiht MOF die Objektorientierte Modellierung von UML, so dass MOF-Metamodelle wie UML-Objektdiagramme in Modellierungswerkzeugen bearbeitet werden können. MOF definiert vier Metalevel, welche die (Meta-)Modelle gliedern*: Level M3 – M0*[21], siehe Abbildung 3.

- *Level M3*, die höchste Ebene, ist die Meta Object Facility selbst: Hier definiert ein Metamodell, wie die Struktur der in MOF erstellten Metamodelle organisiert ist.
- *Level M2* enthält Metamodelle, welche in MOF definiert werden. Sie sind Instanzen des M3 Metamodells. Beispiele solcher Metamodelle sind UML[22] (siehe Kapitel 2.1.2), CWM (siehe 2.1.4) oder auch MOF selbst. Diese Metamodelle klassifizieren, wie Modellelemente wie z.B. UML Klasse oder CWM Tabelle aufgebaut sind.
- *Level M1* enthält Instanzen der Metamodelle in M2, also konkrete Modelle. Ein Beispiel für ein solches Modell ist eine in UML modellierte Klasse „Kunde".
- *Level M0*, das Level mit dem niedrigsten Abstraktionsgrad, enthält Instanzen der Modelle aus Level M1. Dies sind konkrete Objekte und Daten, z.B. ein Datensatz in einer Datenbank.

Diese Metalevel helfen, die Modelle und Metamodelle in einer Struktur zu organisieren.

2.1.3.2 MOF und MDA – Stärken und Schwächen

Im Zusammenhang mit Model Driven Architecture ist MOF von noch größerer Bedeutung als UML. Durch die Möglichkeiten, die MOF dem Entwickler an die Hand gibt, können Metamodelle für beliebige Anwendungsbereiche geschaffen werden, die wiederum eine Mo-

[21] In Anlehnung an [FRAN03].
[22] UML 1.4 besteht eigentlich aus zwei Metamodellen, nur eines der beiden ist in MOF definiert. Für nähere Erklärungen siehe [FRAN03], S. 95.

dellierung beliebiger Anwendungen ermöglichen. Gleichzeitig ist es mit MOF möglich, auch eine Vielzahl an Metamodellen zu verwalten und zu pflegen, so dass Wissen, welches bereits in sie geflossen ist, nicht verloren geht sondern im Rahmen der modellgetriebenen Entwicklung wiederverwendet werden kann.

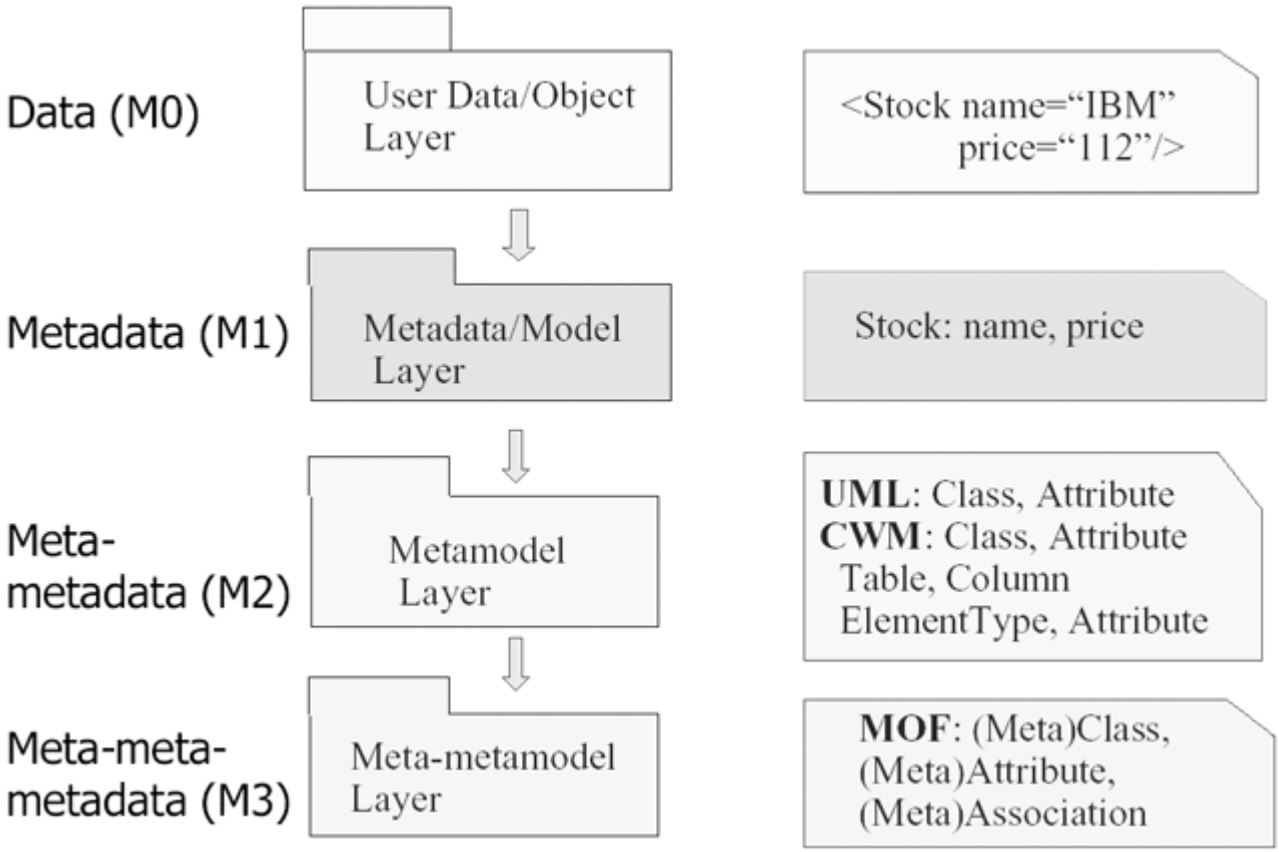


Abbildung 3: MOF-Metalevel[23]

2.1.3.3 MOF 2.0

Auch der MOF-Standard ist nicht ohne Schwachstellen und Fehler. Wie bereits im vorigen Kapitel beschrieben, überschneiden sich die Spezifikationen von UML und MOF teilweise, was viele Unklarheiten und Doppeldeutigkeiten bei der Definition von Modellen bedingt. Laut [FRAN03] existiert keine Definition für die grafische Darstellung von MOF-Modellen. In Anlehnung an UML können zwar Modelle erstellt werden, diese sind aber nicht allgemeingültig. Der Austausch von Modellen ist mit Problemen verbunden, da die Spezifikation teilweise nicht eindeutig ist. Auch eine Möglichkeit, mehrere Versionen zu verwalten, wird vermisst. Diese Mängel sollen mit MOF 2.0 behoben werden.

[23] Quelle: [CHAN01].

2.1.4 Common Warehouse Metamodel

2.1.4.1 Überblick

Das Common Warehouse Metamodel (CWM), zur Zeit in Version 1.1[24], wurde laut [CHAN01] im Jahr 2000 von der Object Management Group als Standard veröffentlicht. Das CWM beschreibt ein MOF-konformes Metamodell des Metalevels *M2* (siehe Kapitel 2.1.3.1). Es standardisiert die Repräsentation von Daten im Rahmen von Datawarehousing, Business Intelligence, Knowledge Management und Portal-Technologies[25].

[OMG 00] beschreibt vier Schichten, in die das CWM eingeteilt ist (siehe Abbildung 4):

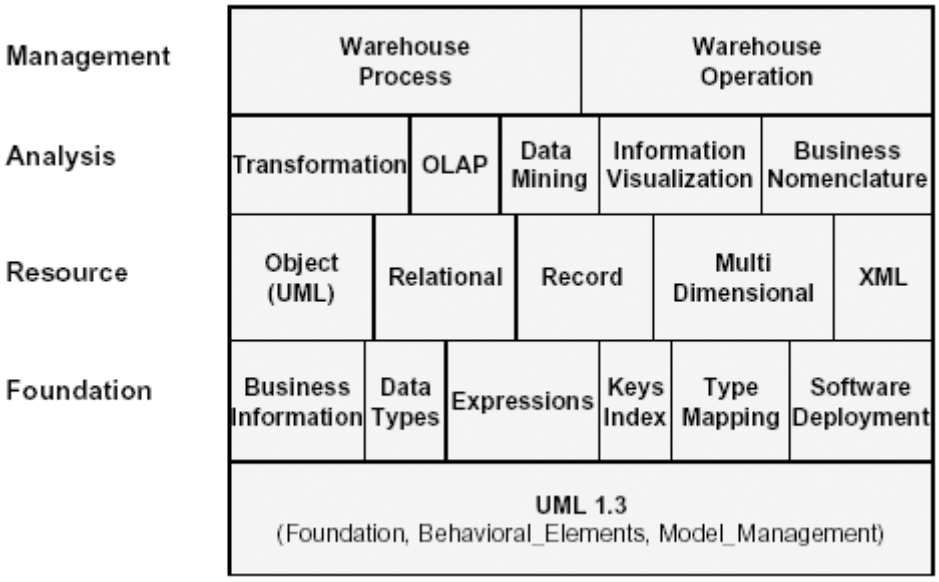


Abbildung 4: Schichten des CWM[26]

- Das *Foundation Layer* basiert auf UML 1.3 und enthält allgemeine Dienste, die von anderen Layern eingebunden werden. CWM verwendet, wann immer es möglich ist, bereits bestehende UML-Elemente, um die Erweiterungen und den damit verbundenen Einarbeitungsaufwand im Rahmen zu halten.
- Im *Resource Layer* liegen die Datenmodelle, welche zur Strukturierung von Datenquellen aller Art verwendet werden und auch für Datawarehousing-Anwendungen benutzt werden.

[24] Laut [OMGSPC].
[25] In Anlehnung an [OMGCWM].
[26] Quelle: [OMG 00].

- Das *Analysis Layer* bietet Metamodelle für logische Dienste an, die auf im Resource Layer definierte Datenquellen angewendet werden können. Ein Beispiel für eine solche Transformation ist die Erstellung eines Datenwürfels aus einer relationalen Datenbank.
- Die im *Management Layer* abgelegten Metamodelle vervollständigen die Funktionalität von Data Warehouses, indem sie die Definition und Verwaltung von Aufgaben wie bspw. Generierung täglicher Reports unterstützen.

2.1.4.2 CWM und MDA

Das CWM baut auf UML/MOF zum Modellieren der Metamodelle auf und verwendet XMI (XML Metadata Interchange) als Standard-Mechanismus zum Austauschen von Metadaten. Ein MOF-IDL[27] Mapping steht als Standardmechanismus für den programmiersprachenunabhängigen Zugriff über APIs auf Metadaten zur Verfügung, ein MOF-Java-Mapping (JMI) ermöglicht einen Zugriff über Java[28].

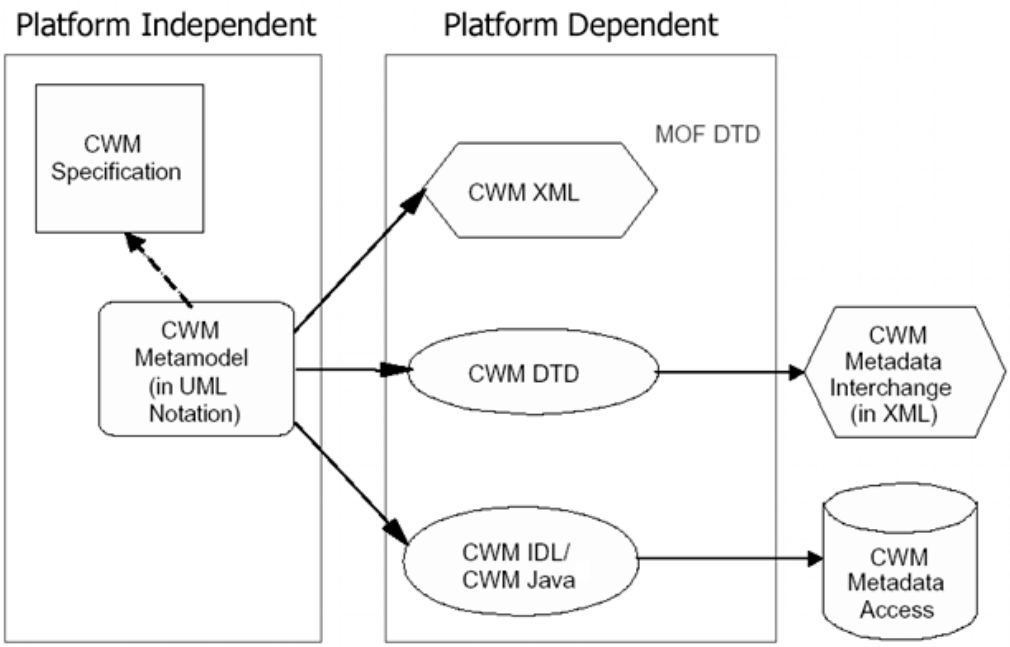


Abbildung 5: Die Rolle von CWM in der MDA[29]

Abbildung 5 zeigt, wie die Struktur des CWM im Rahmen von modellgetriebener Entwicklung von Nutzen ist. Die technologie-unabhängigen Aspekte werden im PIM (siehe Kapitel 2.1.1.4) verwendet. So steht ein fester Rahmen in Form der Spezifikation und des Meta-

[27] Interface Definition Language.
[28] In Anlehnung an [CHAN01].
[29] Quelle: [CHAN01].

modells zur Verfügung; diese liefern die Ressourcen für Plattform-unabhängige Modellierung. Im PSM wird das Austausch- und Mapping-Potential von CWM ausgeschöpft um eine Projektion auf die jeweilige Plattform zu erlangen.

2.2 Convergent Architecture

Es bietet sich an, einen durchgehenden und auf die Model Driven Architecture abgestimmten Architekturstil zu verwenden, um die Möglichkeiten dieser Technologie in vollen Umfang zu nutzen. In seinem Buch „Convergent Architecture“ schlägt Richard Hubert die *Convergent Architecture* (konvergente Architektur) als Referenzarchitekturstil und Engineeringprozess für die Arbeit mit MDA vor[30]. Im Folgenden wird dieser Architekturstil erläutert, der durch einen umfassenden Ansatz und detaillierte Richtlinien die Verwendung der Model Driven Architecture unterstützt.

2.2.1 Konzepte der Convergent Architecture

[BROC02] definiert *konvergent* wie folgt:

„kon|ver|gent <lat.> (sich zuneigend, zusammenlaufend; übereinstimmend)“

In der Softwareentwicklung wird (in Anlehnung an *Convergent Engineering* von David Taylor) mit *konvergenter Architektur* ein Architekturstil bezeichnet, der alle Aspekte der Entwicklung umschließt und diese kontrolliert zu einem Ergebnis zusammenführt. Die konvergente Architektur ist eine wohldefinierte, in sich schlüssige Komplettlösung, welche Austausch mit anderen Anwendern, Wiederverwendung und eine stetige Verbesserung des Stiles ermöglicht. Ziel ist es, Schwächen der bestehenden Strukturen, Vorgehensweisen und somit auch der Software zu beheben. Durch unternehmensweite Konvergenz soll eine leichtere, weniger kostenintensive und qualitativ hochwertigere Softwareentwicklung ermöglicht werden. Konkret setzt Hubert dieses Paradigma um, indem er einen integrierten Architekturstil definiert, der alle Aspekte der Softwareentwicklung abdeckt. Die konvergente Architektur besteht aus vier Komponenten, welche den gesamten Entwicklungsprozess abdecken (siehe Abbildung 6).

- *Das architektonische Metamodell*

  (The Architectural Metamodel)

  Mit der Wahl des architektonischen Metamodells wird eine allgemeine Richtung betreffend der Entwicklung und des Stils der Software vorgegeben[31]. Hier werden die Prinzipien des Architekturstiles definiert. Durch diese wird ein Rahmen für die Softwareentwicklung geschaffen, in dem sich alle Beteiligten bewegen.

[30] Dieses Kapitel basiert auf [HUBE02].
[31] Diese Wahl wird vom leitenden Softwarearchitekten getroffen.

- *Das Lebenszyklus-Entwicklungsmodell*
  (The Full-Lifecycle Development Model)

  Das Lebenszyklus-Entwicklungsmodell bietet die Mittel um die im architektonischen Metamodell festgelegten Prinzipien umzusetzen. Hier werden Elemente wie (Software-)Strukturen, Projektorganisation und Parameter des Entwicklungsprozesses genauer festgelegt. Die Existenz eines architektonischen Metamodells ist eine zwingende Voraussetzung für die Definition des Lebenszyklus-Entwicklungsmodells.

Abbildung 6: Die vier Komponenten der konvergenten Architektur

- *Das umfassende Werkzeugpaket*
  (The Full-Coverage Tool-Suite)

  Basierend auf den im Lebenszyklus-Entwicklungsmodell festgelegten Regeln und Strukturen werden hier konkrete konkrete (Software-)Werkzeuge definiert, die eine Entwicklungszyklus-umfassende, stilkonforme Entwicklung unterstützen.

- *Die formalen Technologieprojektionen* (The Formal Technology Projections)

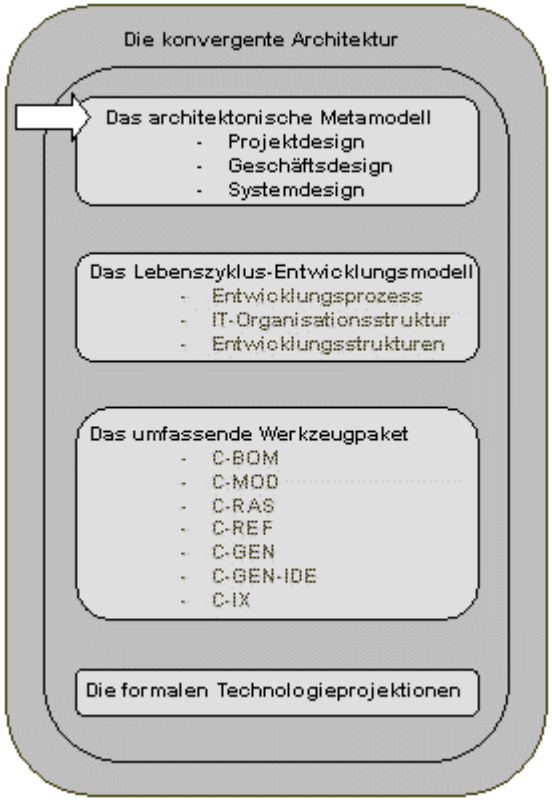
  Die formalen Technologieprojektionen ermöglichen das Umsetzen von abstrakten Modellen in komplette IT-Infrastrukturen. Sie dienen als Regeln beim Generieren von Quellcode, Deployment-Informationen und Test-Umgebungen, und sind die Grundlage für Werkzeuge, die Modelle auf Korrektheit überprüfen.

  Andere Bezeichnungen für formale Technologieprojektionen sind *Cartridges, Transformationen* oder *Generatoren.*

Im Folgenden wird näher auf diese vier Komponenten und ihre Eigenschaften eingegangen.

2.2.2 Das architektonische Metamodell

Das architektonische Metamodell als Basis des Architekturstiles besteht aus drei Säulen. Diese Säulen beschreiben die Bereiche Projekt-, Geschäfts- und Systemdesign:



- Projektdesign/Projektgestaltung
  - Wie werden IT-Organisationen, Projekte und wie wird die Entwicklung von Software eingerichtet, koordiniert und geleitet?
  - Welche Strukturen, Rollen und Verantwortlichkeiten haben IT-Organisation und Entwicklungsteam?
- Geschäftsdesign/Gestaltung
  - Welche Werkzeuge und Vorgehensweisen/Techniken werden benötigt, um die Geschäftsstrategie abzubilden und zu verfeinern?
  - Welche Beziehungen bestehen zwischen Unternehmen und IT-Abteilung?
  - Wie gestaltet sich die Zusammenarbeit von Geschäfts-Dimension und IT-Dimension (siehe Kapitel 2.2.3.3)?
- Systemdesign/Systemgestaltung
  - Wie ist es möglich, das Geschäftmodell auf ein Systemmodell zu übertragen, das die Funktionalität optimal unterstützt?
  - Welche Entwicklungsrichtlinien, technischen Anforderungen (Persistenz, Sicherheit, etc.) und allgemeinen Eigenschaften (Zuverlässigkeit, Performance, etc.) existieren?

Weiterhin zeigt das architektonische Metamodell einige Richtlinien auf, die die Entwicklung vereinfachen und das Fehlerrisiko bzw. das Risiko einer unsauberen Architektur mindern.
Es wird erläutert, wie Geschäfts- und IT-Modelle in ein allgemeines, vereinfachtes Modell umgewandelt werden, um die gewünschte *Konvergenz (convergence and convergent engineering)* in der Architektur zu erhalten.

Für eine vereinfachte Entwicklung der Modelle wird *RASC (Reduced Abstraction Set Computing*[32]*)* eingeführt. Diese abstrakte Modellierungssprache stellt fünf Komponenten für die Modellierung bereit.

Als Basis-Komponenten, sogenannte *OPR-Komponenten*:

- Organisation (organization)
- Prozess (process)
- Ressource (resource)

Als Erweiterungs-Komponenten:

- Zugriffskomponenten[33] (accessors)
- Hilfskomponenten (utility components)

Mit diesen fünf Abstraktionen wird jedes System modelliert. Durch den hohen Abstraktionsgrad ist die Sprache für Experten aus den Fachabteilungen genauso verständlich wie für Entwickler. In Kapitel 2.2.3.3 wird näher auf diese Komponenten eingegangen.

Durch das architektonische Metamodell und dessen Konzepte ist die Basis für eine konvergente, planvolle Entwicklung gelegt.

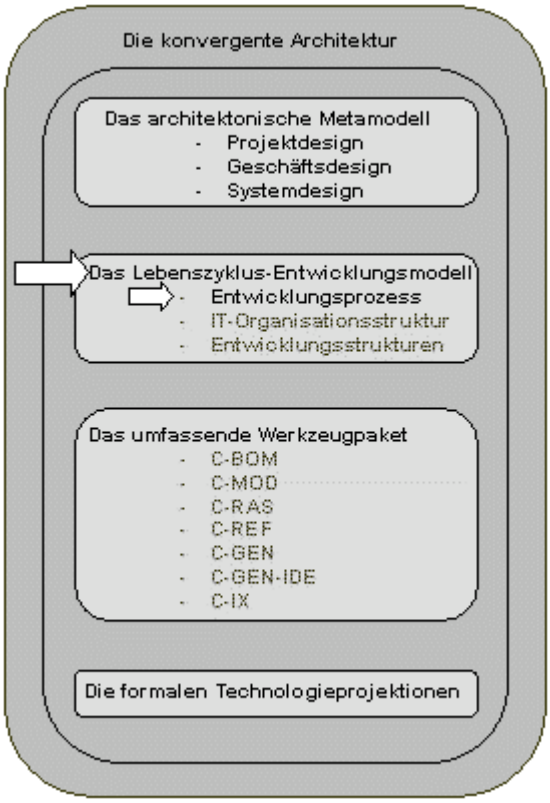
[32] Die Parallele zu RISC (Reduced Instruction Set Computing) ist beabsichtigt. Parallelen sind der kleine „Befehls“ Umfang und die daraus resultierende einfache Verwendbarkeit.

[33] Dies sind Komponenten, welche die Kommunikation eines Systems oder von Teilen eines Systems mit der Außenwelt abwickeln.

2.2.3 Das Lebenszyklus-Entwicklungsmodell

Die Strukturen und organisatorischen Aspekte, welche benötigt werden, um das architektonische Metamodell umzusetzen, sind in drei Bereiche gegliedert. Jeder dieser Bereiche gliedert seine Verantwortlichkeiten in einem eigenen Modell.

2.2.3.1 Der Bereich Entwicklungsprozess



In diesem Bereich liegt der Schwerpunkt auf der Entwicklung und Evolution der Softwareelemente. Es werden die spezifischen Entwicklungsaufgaben definiert und strukturiert. Dies wird von spezialisierten Anwendungen unterstützt. Das dazugehörige *Entwicklungs-Prozess-Modell* (development process model) wird auch als der *konvergente Architekturprozess* (Convergent Architecture (CA) process) bezeichnet

Dieses Modell wird von vielen modernen Technologien beeinflusst, beispielsweise die OPEN Prozess Spezifikation, der Rational Unified Process (RUP) oder die Evolutionary Project Management (EPM) Methode. Der Entwicklungsprozess ist in Workflows organisiert, welche in iterativen Phasen der Entwicklung immer wieder durchlaufen werden. Dabei werden zwei Arten unterschieden[34]:

- *Vorbereitende und projektübergreifende Workflows* (preparatory and cross-project-workflows)

  Diese Workflows existieren außerhalb der einzelnen Projekte und haben die übergreifende Kontrolle über alle Projekte.
- *Projektspezifische Workflows* (canonical project workflows)

Diese Workflows sind einem spezifischen Projekt zugeordnet und dienen seinem Fortgang.

[34] Siehe [HUBE02], S. 145ff..

Vorbereitende und projektübergreifende Workflows

IT-Umgebungs-Workflow (IT-environment workflow)

Dieser Workflow dient dem Initialisieren des Entwicklungsprozesses. Der IT-Organisationsleiter definiert ein spezifisches Modell einer IT-Organisation und besetzt die höchsten Positionen der Unterorganisationen. Die Leiter der Unterorganisationen wiederum organisieren ihre Organisationen, so dass eine komplette Architekturorganisation wie in Kapitel 2.2.3.2 erläutert, entsteht. Ist dieser Vorgang komplett, werden der T-Bar Geschäftsmodellierungs-Workflow und der Anforderungs-Workflow gestartet.

T-Bar Geschäftsmodellierungs- und Anforderungs-Workflow (T-Bar business modeling and requirements workflow)

Dieser Workflow kontrolliert die erste, umfassende Analyse und Optimierung des umzusetzenden Systems. Techniken wie analysis-by-design und class responsibility collaboration (CRC) cards werden verwendet um eine intuitive, übersichtliche Analyse zu ermöglichen. Im Englischen wird der Begriff T-Bar Business Modeling verwendet, wobei der Begriff „T-Bar“, also der Querstrich des T's, verdeutlichen soll, dass hier auf dem höchsten Level der Analyse gearbeitet wird.

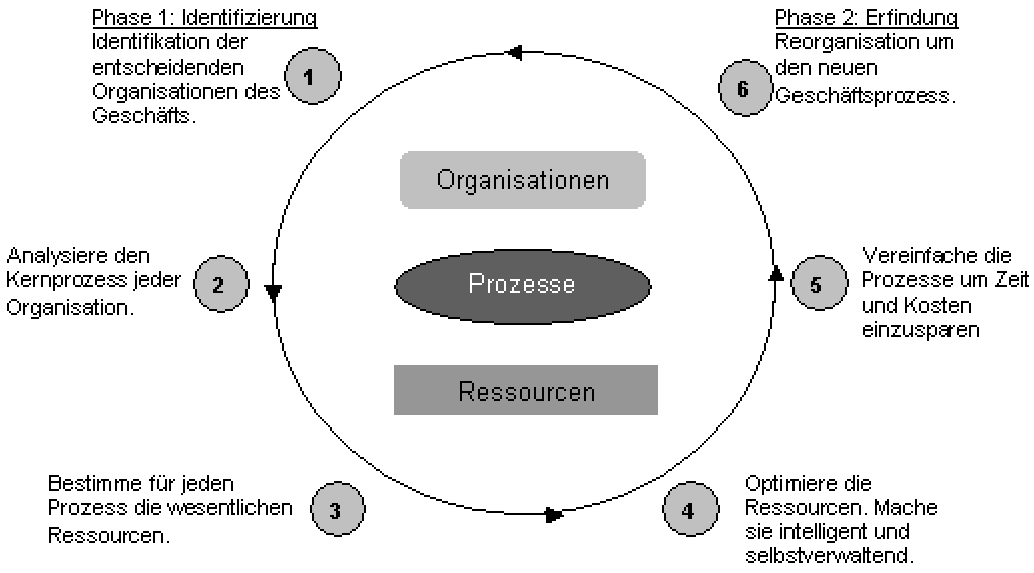


Abbildung 7: Der Analysis-by-Design-Prozess[35]

[35] In Anlehnung an [HUBE02], S. 154.

Die konvergente Architektur sieht einzelne (meist mehrtägige) Sitzungen vor, an denen Experten aus den Fachabteilungen, Entwickler sowie der Anforderungsmanager und der leitende Softwarearchitekt teilnehmen. Diese Sitzungen sind folgendermaßen strukturiert: Vormittags werden zusammen mit den Mitarbeitern aus den Fachabteilungen die fachlichen Anforderungen bestimmt und modelliert. Nachmittags können diese Mitarbeiter in ihre Abteilungen zurückkehren und dort ihrer gewohnten Arbeit nachgehen, während das IT-Team das Modell weiter bearbeitet, verbessert und dokumentiert. Der Verlauf der Sitzungen ist in Abbildung 7 dargestellt. Ergebnis einer Sitzung ist ein übergeordnetes OPR-Modell für den entsprechenden fachlichen Bereich. Diese Modelle werden in einen oder mehrere Vorschläge für die Entwicklung des Systems zusammengefügt. Dieser Workflow wird nach Maßgabe des leitenden Architekten oder des IT-Organisationsleiters mindestens zweimal pro Jahr wiederholt, um eine ständige Aktualität des Modells zu sichern.

Im Anschluss an die Geschäftsmodellierung folgen mehrere Aktivitäten, die jedoch nicht in eigene Workflows gegliedert werden:

Projektbeginn und -lenkung (project initiation and tracking):

Basierend auf den Ergebnissen des T-Bar-Workflows setzt der IT-Organisationsleiter neue Projekte auf. Wenn ein Projekt deutliche Unterschiede zu den Vorschlägen aus der Modellierung aufweist, wird der T-Bar Geschäftsmodellierungs- und Anforderungs-Workflow wiederholt.

Umfassendes Anforderungsmanagement (global requirements management):

Alle Anforderungen, die gestellt werden, um die Entwicklung voranzutreiben, werden vom Anforderungsmanager verwaltet. Er organisiert diese in einem Anforderungspool (bspw. Rational RequisitePro) und sucht für sie eine offizielle Senke. Dies ist eine Person, die bereit ist, offiziell die Verantwortung für diese Anforderung zu übernehmen.

Zu jeder Anforderung müssen mindestens folgende Daten aufgelistet werden:

- Quelle (wer stellt die Anforderung)
- Priorität (elementare Änderungsanforderung oder unwichtige Anregung)
- Senke (wer kann die Anforderung erfüllen)
- Status

Anforderungen, welche der Anforderungsmanager nicht sinnvoll verteilen kann, werden an den IT-Organisationsleiter weitergegeben.

Architektonische Entwicklung (architectural evolution workflow)

In diesem Workflow wird das Modell des Systems weiterentwickelt und angepasst. Es wird darauf geachtet, dass die Richtlinien des Architekturstiles eingehalten werden, und in einem inkrementellen Prozess das Endmodell entwickelt.

Projekt-spezifische Workflows

Projektmanagement-Workflow (project management workflow)

Dieser Workflow verwaltet alle anderen projektspezifischen Workflows und interagiert mit den projektübergreifenden Workflows. Er koordiniert ihren Ablauf und treibt sie voran wo es notwendig ist.

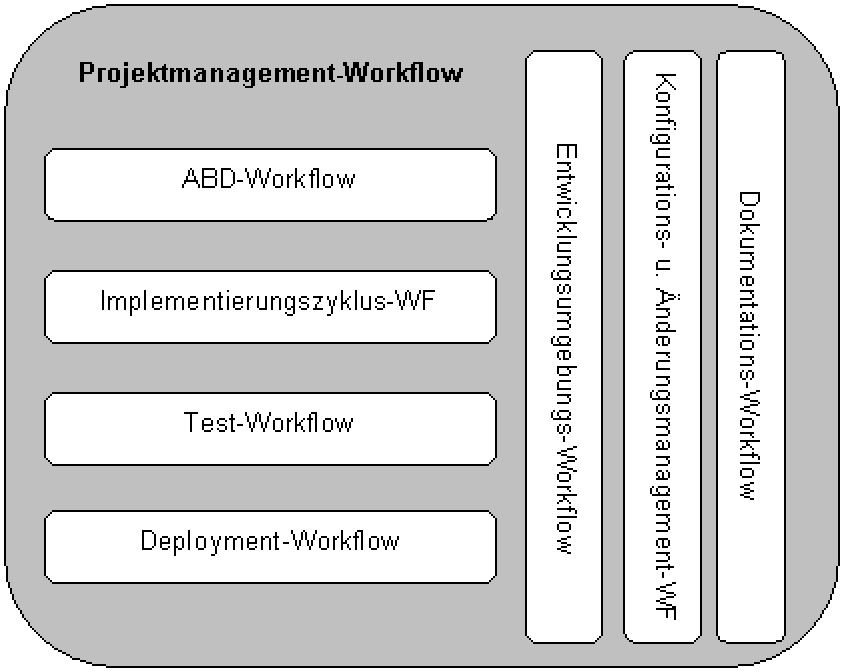


Abbildung 8: Der Projektmanagement-Workflow

Der Projektmanagement-Workflow umschließt die Workflows, welche auf dem kritischen Pfad liegen (siehe Abbildung 8):

- ABD-Workflow[36]
- Implementierungszyklus-Workflow
- Test-Workflow
- Deployment-Workflow

[36] Der ABD(analysis by design)-Workflow entwickelt die Ergebnisse der Geschäftsanalyse weiter, indem ein Design erstellt wird. Eine genauere Erklärung findet sich auf S. 30.

Hinzu kommen begleitende Workflows, welche parallel zu den Aktivitäten des kritischen Pfades ausgeführt werden:

- Entwicklungs-Umgebungs-Workflow
- Konfigurations- und Änderungsmanagement-Workflow
- Dokumentations-Workflow

Der Projektmanagement-Workflow durchläuft den kritischen Pfad während der Entwicklung in mehreren Iterationen. Dabei verlagert sich der Schwerpunkt mit jedem Durchgang von Analyse über Design bis hin zur Implementierung.

Die Iterationen sind als zügige Durchgänge vorgesehen, innerhalb jeder Iteration werden folgende Aktionen durchgeführt:

- Durchsprechen (talk through)
  Hier werden die Anforderungen und elementaren Eigenschaften für die Geschäftslogik und IT-Organisation aufgelistet und priorisiert.
- Zuteilen und abschätzen (allocate and estimate)
  Die am höchsten priorisierten Elemente werden weiter detailliert. Weiterhin erfolgt die Zuteilung von Ressourcen und die Abschätzung des Entwicklungsaufwandes dieser Elemente.
- Durchgehen (walk-through)
  Der Entwurf wird konsolidiert und mit den vorhandenen Einschränkungen (Fähigkeiten, Zeit, usw.) abgeglichen. Schritt 2 wird so lange wiederholt, bis der Entwurf und die Einschränkungen übereinstimmen.
- Durchlaufen (run-through)
  Zum Abschluss werden die Ergebnisse noch einmal zügig durchlaufen und auf Konsistenz und allgemeine Zustimmung überprüft.

Während der Iterationen existiert ein Plan in Form einer Auflistung aller Eigenschaften sowie einer TOMA (task ownership matrix), in welcher für jede Aufgabe (Task) die elementaren Informationen aufgelistet werden (siehe Abbildung 9)

Basierend auf den Ergebnissen dieses Workflows können langfristige Prognosen für das Projekt abgegeben sowie eine Strategie für das Erreichen des Zieles formuliert werden.Im Folgenden werden die in Abbildung 8 dargestellten, dem Projektmanagement-Workflow untergeordneten Workflows beschrieben.

ID	Task	Fortschritt	Besitzer	Dauer	Abh.
1	Diplomarbeit erstellen	100%	Stephanie Weg	5 Monate	0
2	Korrekturlesen	100%	Roger Zacharias	1 Woche	1
3	Korrekturen einpflegen	50%	Stephanie Weg	2 Tage	2
4	Drucken	0%	Stephanie Weg	1 Tag	3
5	Binden	0%	Copyshop	1 Tag	4
6	Benoten	0%	Prof. Kaufmann	1 Woche	5

Abbildung 9: Task Ownership Matrix (TOMA)

Analyse-durch-Design-Workflow (analysis by design/ABD-workflow)

Im ABD-Workflow werden die Ergebnisse der Geschäftsanalyse in drei aufeinander aufbauenden Phasen der Verfeinerung bzw. Anreicherung erweitert. Dabei entstehen Business-Use Case Szenarios, Zugriffskomponenten-Use Case Szenarios und CRC-Cards, welche die Verantwortlichkeiten der Komponenten und die mit ihnen zusammenarbeitenden Komponenten strukturieren. Als Ergebnis dieses Workflows liegt ein verifiziertes, konvergentes und technisch umsetzbares UML-Modell der konvergenten Komponenten vor.

Implementierungszyklus-Workflow (implementation cycle workflow)

Dieser Workflow umschließt das modellgetriebene Verfeinern, Generieren, Editieren, Deployment und Testen. Zu Beginn des Workflows werden von der architektonischen IDE[37] 40-100% des Quellcodes und der Informationen generiert[38], die für Build, Test und Deployment des Systems benötigt werden. Diese generierten Artefakte können sofort getestet werden. In den meisten Fällen fehlt noch Geschäftslogik, welche vom Entwickler in so genannten „geschützten Bereichen“ (protected areas) ergänzt werden muss.

Test Workflow (test workflow)

Im Test-Workflow finden Qualitätsprüfungen und rechtzeitige Diagnosen des entwickelten Systems in jeder Stufe des Entwicklungslebenszyklus statt. Dabei unterscheidet man zwischen Unit/Einheiten-Tests, Komponenten- und Systemtests, Interaktions- und Antworttests sowie Geschäftsfluss- (Business flow) und Konvergenztests.

[37] Eine Alternativbezeichnung ist “Full Coverage Tool Suite”, siehe auch Kapitel 2.2.4.
[38] Laut [OMG 03e].

Deployment und Überwachungs-Workflow (deployment and monitoring workflow)

Als letzter der im kritischen Pfad liegenden Workflows deckt dieser Workflow den Übergang in die Arbeitsumgebung sowie die Überwachung ab. Systementwickler und Deployment-Manager definieren bereits in der späten Entwicklungsphase Installationsanforderungen, Überwachungsmöglichkeiten etc. um einen reibungslosen Übergang zu ermöglichen. Nach der Freigabe der Software tragen der Systemadministrator und der Container[39]-Administrator die Verantwortung dafür, dass das System problemlos läuft. Änderungsanforderungen, die während dieser Phase auftreten, werden an den Anforderungsmanager weitergeleitet. Neben den im kritischen Pfad liegenden Workflows existieren begleitende Workflows, die in Abbildung 8 auf der rechten Seite dargestellt werden.

Entwicklungsumgebungs-Workflow (development environment workflow)

Dieser Workflow unterstützt den korrekten Ablauf der Workflows auf dem kritischen Pfad. Die Umgebung, in welcher die Softwareentwicklung stattfindet, wird konfiguriert und verifiziert. Konkret gehören dazu beispielsweise Tätigkeiten wie Installation, Konfiguration und Wartung der zur Entwicklung verwendeten Hard- und Software.

Konfigurations- und Änderungsmanagement-Workflow (Configuration and change management/CCM-workflow)

In diesem Workflow wird die Verwaltung der erstellten Codeelemente gehandhabt. Dies umschließt Partitionierung, Versionierung und Archivierung aller nicht rein generierten Artefakte sowie eine Versionierung des kompletten Systems am Ende jeder Iteration. Verwaltet werden alle Typen von Daten, beispielsweise XML/XMI-formatierte Daten, Rational Rose-spezifische Daten oder Elemente wie WARs[40], JARs[41] und EARs[42].

Dokumentationsworkflow (documentation workflow)

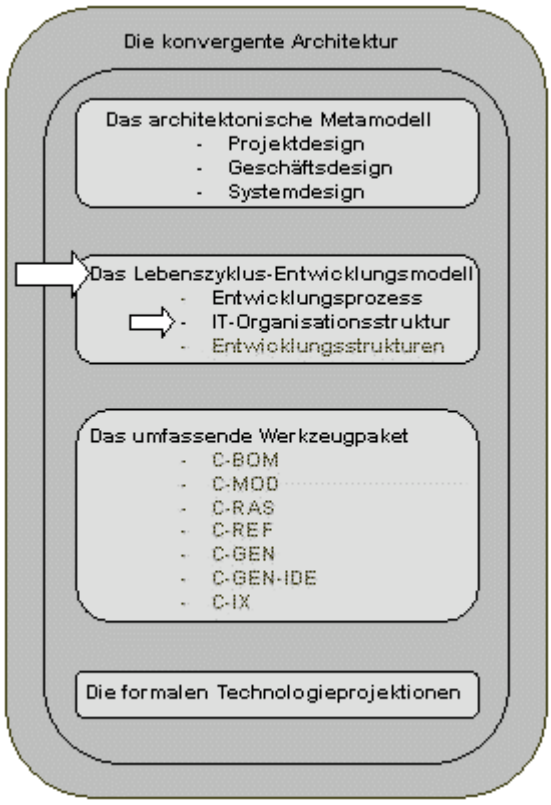
Dieser Workflow beschäftigt sich mit der Erstellung von Dokumentation, Hilfedateien und Trainingsmaterial. Dabei werden verschiedene Blickwinkel wie Dokumentation des Entwurfes, Endbenutzerdokumentation und Hilfesysteme abgedeckt. Im Dokumentationsentwicklungs-Schema (documentation development set) wird ein Dokumentationsstil vorgegeben, so dass ein einheitlicher Stil der Dokumente gewährleistet ist.

---

[39] Ein Container ist eine Ablaufumgebung für Komponenten.
[40] WAR steht für Web Application Archive.
[41] JAR steht für Java Archive.
[42] EAR steht für Enterprise Application Archive.

2.2.3.2 Der Bereich IT-Organisationsstruktur

Dieser Bereich koordiniert Personen, Rollen und Verantwortlichkeiten und wird im *IT-Organisations-Modell* (IT-organization model) abgebildet. Hier wird das gesamte System, welches die Planung, Entwicklung und Wartung eines Softwareproduktes umschließt, dargestellt. Die IT-Organisation wird dabei in mehrere Unterorganisationen gegliedert (siehe Abbildung 10).



- Architektur-Organisation
  (Architecture organization)
- IT-Support-Organisation
  (IT-support organization)
- Systementwicklungs-Organisation
  (System development organization)
- Systemeinsatz-Organisation
  (Operational Systems organization)

Hier wird kurz auf die Funktionsbereiche der einzelnen Unterorganisationen eingegangen und einige Beispiele für Rollen der beteiligten Personen angegeben. Für eine ausführliche Beschreibung der Organisationen empfiehlt sich die Lektüre von [HUBE02].

<u>IT-Organisation</u>

Diese Organisation als Verwalter der Unterorganisationen ist die höchste Entscheidungsebene bezüglich Entwicklung und Betrieb von IT-Systemen. Hier werden Projekte definiert, geplant und verfolgt sowie untergeordnete Organisationen, Anforderungen etc. verwaltet.

Rollen: Projektleiter, Lenkungsausschuss.

Abbildung 10: Die IT-Organisation

Architektur-Organisation

Hier wird die konvergente Architektur definiert und gewartet. Es wird sichergestellt, dass die Architektur korrekt angewendet wird. Dies ist der Einsatzort für das Personal mit dem größten Erfahrungsschatz.
Rollen: Leitender Architekt, Anforderungsmanager.

IT-Support-Organisation

Diese Organisation bietet Support für alle anderen IT-Organisationen. Sie wird in vier Bereiche unterteilt:

- grundlegende Infrastrukturverwaltung (base infrastructure administration)
- Änderungs- und Konfigurationsmanagement (change and configuration management, CCM)
- Projekt-Informationsmanagement (project information management)
- Testcenter-Management (test center management)

Rollen: Systemadministrator, Konfigurationsmanager, Kursleiter, Tester.

Systementwicklungs-Organisation

Hier findet die eigentliche Entwicklung der Software statt. Dabei werden zwei Arten von Teams unterschieden:

- System-Entwicklungsteams (assembly development teams)
- Komponenten-Entwicklungsteams (component development teams)

Die Mitarbeiter des Komponenten-Entwicklungsteams liefern die Elemente, welche von den Mitarbeitern des System-Entwicklungsteams in ein System integriert werden.
Rollen: System-Entwickler, Komponenten-Entwickler, Experten auf Fachgebieten[43].

Systemeinsatz-Organisation

Diese Organisation ist in drei Bereiche untergliedert:

- Transitionsorganisation (transition organization)
  Zuständig für Deployment und reibungslosen Ersteinsatz der entwickelten Software.
- Benutzer-Support-Organisation (user support organization)
  Unterstützung der Benutzer beim Einsatz der Software.
- Infrastruktur und Basissystem-Organisation (infrastructure and base systems organisation)

[43] Bspw. Spezialisten im Umgang mit eingesetzten Technologien.

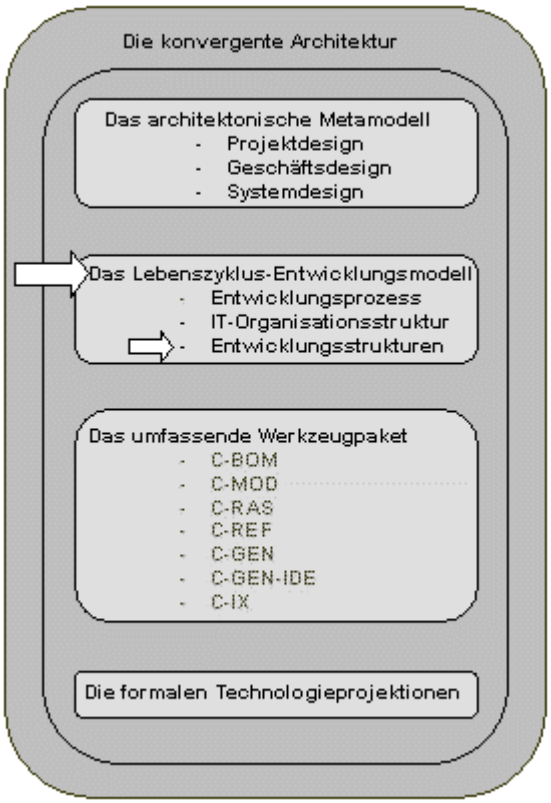
Umsetzung und Wartung des Umfelds für die Software.

Rollen: Deployment Manager, Benutzer-Support-Spezialist, Systemoperator.

Alle Unterorganisationen verwalten OPR-Geschäftskomponenten, auf die im folgenden Kapitel näher eingegangen wird.

2.2.3.3 Der Bereich Entwicklungsstrukturen

Dieser Bereich beschreibt die konkreten Ressourcen und deren Strukturen, welche zum Entwerfen, Implementieren und Anwenden des Systems benutzt werden. Darunter fallen z.B. Architekturschichten und Komponententypen.



Diese werden im *konvergenten Komponenten-Metamodell* (convergent component metamodel) definiert, welches so Projekt-, Geschäfts- und Systemdesign realisiert. Der architektonische Stil wird auf Ebene der Komponenten umgesetzt. Für die Erstellung des Metamodels stehen vier Arten von Komponenten zur Verfügung:

- System (assembly)
- Zugriffskomponente (accessor)
- Geschäft (business)
- Dienst (utility)

Die Hierarchie und Interaktion dieser Komponenten wird in einer vierschichtigen Architektur strukturiert (siehe Abbildung 11). Durch eine derartige Organisation des konvergenten Metamodells werden unerwünschte Bindungen verhindert. Die Verwaltung der Komponenten einer Schicht erfolgt entweder von Komponenten der eigenen oder der nächsthöheren Schicht, so dass eine Kapselung entsteht.

<u>Systemkomponenten-Schicht</u>

Wie in der Abbildung zu erkennen ist, ist die oberste Schicht die *Systemkomponenten-Schicht* (Assembly Component Layer). System-Komponenten sind für die Verwaltung und

Integrität des Systems verantwortlich. Sie verwalten untergeordnete Komponenten, koordinieren Verwendung und Wiederverwendung von Komponenten und kontrollieren somit das System als Ganzes. Hier sind die Komponenten enthalten, aus welchen sich das fertige System zusammensetzt. Diese repräsentieren:

- Menschen, die das System installieren und warten
- Systemelemente, welche für Installation, Updates, Deployment etc. zuständig sind
- Anwendungen zum Anpassen und Überwachen des Systems
- Systemelemente, welche mit anderen Systemen interagieren

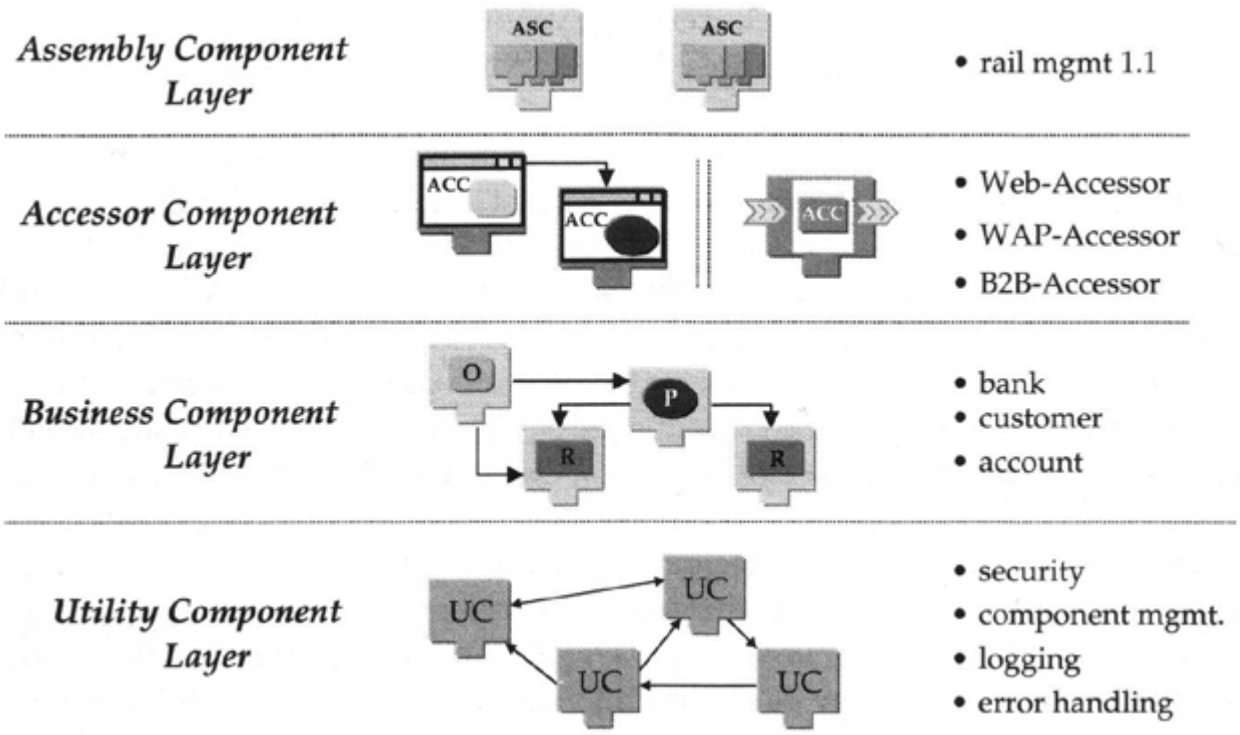


Abbildung 11: Die Schichten des konvergenten Komponenten-Metamodells[44]

Zugriffskomponenten-Schicht

Unter der Systemkomponenten-Schicht liegt die *Zugriffskomponenten-Schicht* (Accessor Component Layer). Diese Komponenten dienen der Kommunikation des Systems mit der Außenwelt. Sie werden untergliedert in System-Interface-Zugriffskomponenten (diese kommunizieren mit externen Systemen) und User-Interface-Zugriffskomponenten (diese kommunizieren mit Benutzern, text-, sprach- oder grafikbasiert). Zugriffskomponenten werden in Containern verwaltet. Dies sind Ablaufumgebungen für Komponenten, die diese verwalten und ihnen Dienste zur Verfügung stellen. Jede Zugriffskomponente besitzt einen

[44] Quelle: [HUBE02], S. 77.

oder mehrere Repräsentanten, die jeweils einen spezifischen Kommunikationskanal repräsentieren, z.B. eine Datenverbindung oder eine GUI. Repräsentanten werden in Repräsentanten-Containern verwaltet, die eine Umgebung für ähnliche Repräsentanten bieten[45]. Die Container können hierarchisch angeordnet werden, wobei der höchste Container vom dazugehörigen Zugriffskomponenten-Container verwaltet wird. Im Hinblick auf das Schichtenmodell kapseln die Zugriffskomponenten die Kommunikation mit OPR-Komponenten der nächstniedrigeren Schicht.

Geschäftskomponenten-Schicht

Die zweitunterste Schicht ist die *Geschäftskomponenten-Schicht* (Business Component Layer), welche die in Kapitel 2.2.2 bereits erwähnten OPR-Komponenten[46] enthält. Diese Komponenten dienen als Basis-Modellierungsbausteine um ein System abzubilden. Sie repräsentieren die reine Geschäftslogik, unabhängig von technologiespezifischen Implementierungsdetails. Durch ihre intuitive Verwendung in der Analyse bilden sie die Kommunikationsschnittestelle zwischen Mitarbeitern aus den Fachabteilungen und IT-Spezialisten. Abbildung 12 zeigt beispielhaft, wie Elemente aus der realen Welt in OPR-Komponenten übertragen werden. Die drei Ausprägungen Organisation, Prozess und Ressource haben dabei verschiedene Eigenschaften.

- *Organisationen* sind für das Managen, Austauschen und Ausführen von OPR-Komponenten zuständig. Sie sind die Knotenpunkte, an denen alle Geschäftskomponenten verwaltet werden. Eine Organisation ist verantwortlich für den Zugriff auf und die Verwendung der Komponenten und überwacht sicherheitsrelevante Aspekte. Benutzer, die Komponenten für einen bestimmten Zweck benötigen, fragen bei Organisationen an.
- *Prozesse* sind Aktivitäten, welche Ressourcen verwenden oder verbrauchen, um diese zu erweitern oder neue Ressourcen zu generieren. Sie können direkt mit Ressourcen verbunden sein und Referenzen auf diese über längere Zeit besitzen. Bei Prozessen werden zwei Granularitäten unterschieden:
  - Workflow (workflow): Langfristig, besteht aus logisch zusammengehörigen Aktivitäten.
  - Aktivität (activity): Gruppen von logisch zusammengehörigen Aufgaben (Tasks[47]).

[45] Ein Beispiel für einen Repräsentanten-Container wäre ein Browser.
[46] Die Abkürzung “OPR“ stammt von der Unterscheidung der Komponenten in Objekt, Prozess und Ressource.
[47] Als Task wird die kleinste identifizierbare und zuweisbare Einheit von Arbeit definiert.

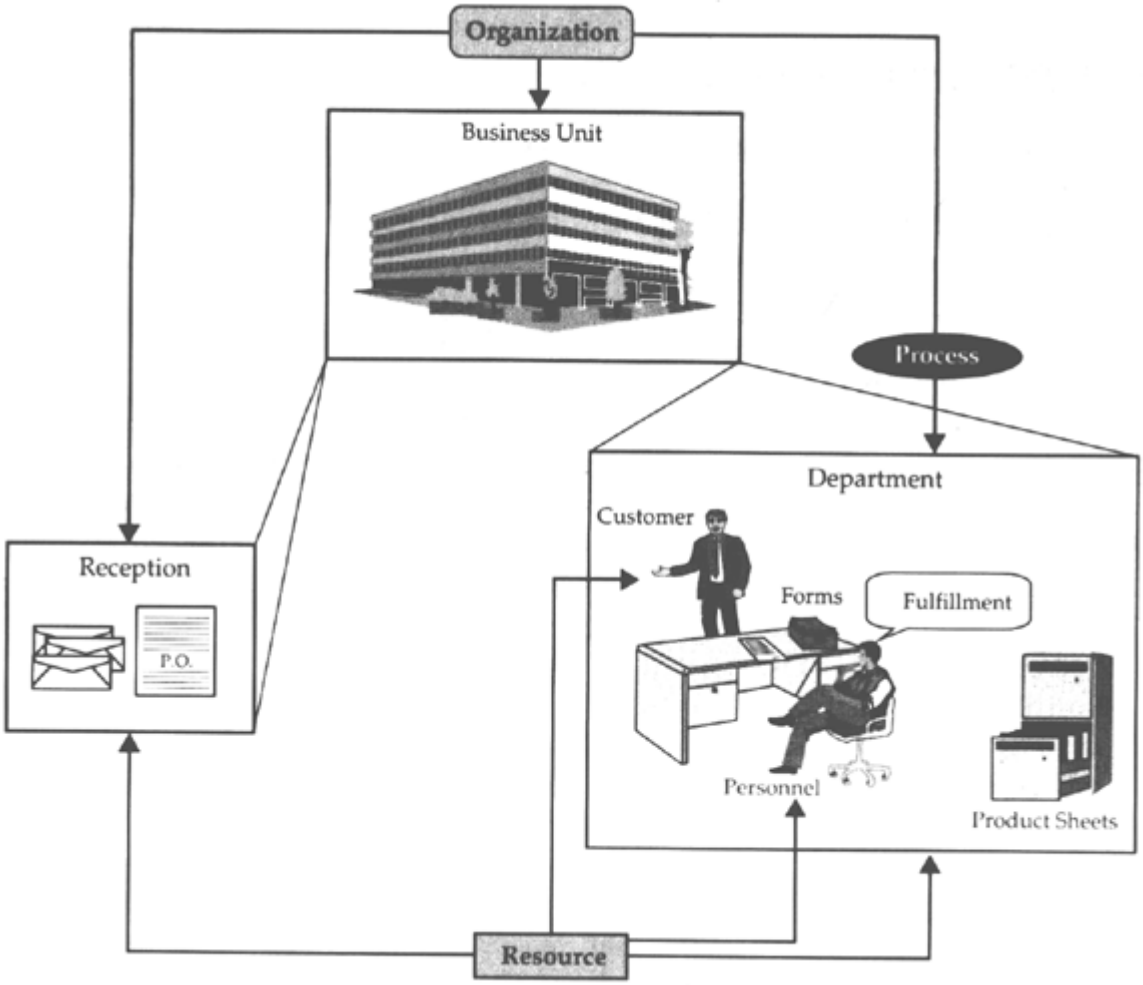


Abbildung 12: Geschäftsmodellierung mit OPR-Komponenten[48]

- *Ressourcen* sind die Elemente einer Organisation, die Wert, Kosten und Arbeitskraft repräsentieren. Es gibt drei Arten von Ressourcen:
  - *Arbeiter* (workers): Menschen, die jeweils mehrere Rollen (worker roles) haben können. Die Summe aller Arbeiter definiert die Verantwortlichkeit der Organisation.
  - *Artefakte und Sätze von Änderungen* (artifacts and change sets): Daten, die während der Entwicklung erstellt oder verwendet werden. Artefakte können in Gruppen versioniert werden und werden dann als Änderungs-Satz bezeichnet.
  - *Technologien* (technologies): Extern entwickelte Technologien, welche als hochspezialisierte Unterstützung zum Entwicklungsprozess hinzugezogen werden.

Ressourcen haben eine enge Bindung an die sie verwaltende Organisation und können in direkter Beziehungen zu anderen Ressourcen stehen.

[48] Quelle: [HUBE02], S. 100.

Bei der Modellierung von OPR-Komponenten wird angestrebt, die komplette Geschäftslogik umzusetzen, ohne eine Bindung an proprietäre Technologien einzugehen. Dies ist mit dem aktuellen Stand der Technik nur eingeschränkt möglich. So entsteht häufig die Notwendigkeit, entweder die modellierte Funktionalität zu vereinfachen oder aber auf proprietäre Erweiterungen zurückzugreifen.

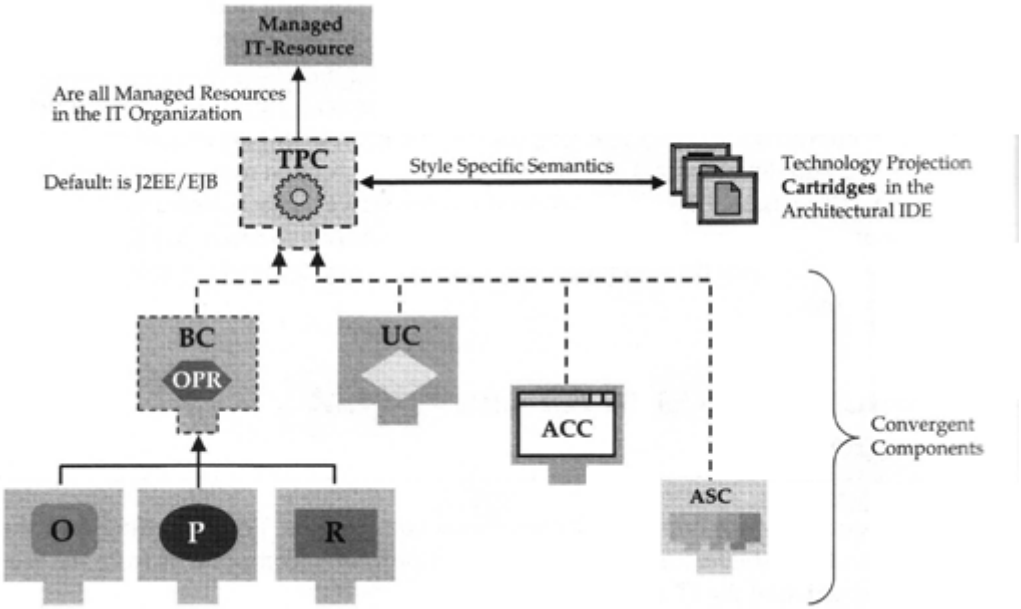


Abbildung 13: Technology Projection Component (TPC)[49]

Dienstkomponenten-Schicht

An unterster Stelle liegt die *Dienstkomponenten-Schicht* (Utility Component Layer). Die hier angesiedelten Komponenten sind das Gegenstück zu den Geschäftskomponenten. Sie haben keinerlei Relevanz für die geschäftlichen Abläufe und bieten technologiespezifische Dienste an. Dienstkomponenten kapseln externe Technologien und können sehr gut wiederverwendet werden. Diese Kapselung trennt den Lebenszyklus des Geschäftsmodells vom Lebenszyklus der Technologien. Die Komponenten dieser Schicht werden von den OPR-Komponenten der nächsthöheren Schicht verwendet.

Die Technologieprojektions-Komponente

Alle Komponenten sind der *Technologieprojektions-Komponente* (Technology Projection Component, TPC) untergeordnet. Sie ist eine Meta-Komponente und repräsentiert die Eigenschaften, die alle Komponenten von ihr erben. Durch die TPC werden Modellierungs-

[49] Quelle: [HUBE02], S. 82.

stil[50], Technologieprojektionen sowie die Beziehung der Komponenten zur restlichen Architektur repräsentiert (siehe Abbildung 13).

Dimensionen und Persönlichkeiten der Komponenten

Jede einzelne Komponente ist weiter strukturiert, um die Modellierung zu erleichtern. Wie in Abbildung 14 zu erkennen ist, ist eine Komponente vertikal in zwei Dimensionen geteilt: Die *Geschäfts-Dimension* und die *IT-Dimension*. Erstere enthält alle inhaltlichen Aspekte des Systems, die Geschäftslogik. In dieser Dimension dürfen keine Informationen zu technischen Aspekten enthalten sein. Diese Aspekte werden durch die IT-Dimension repräsentiert, welche für die Umsetzung der Geschäftsdimension zuständig ist. Hier sind alle technischen Details abgelegt. In den Komponenten, die eine rein technische Bedeutung haben, ist die Geschäfts-Dimension zwar vorhanden, aber leer.

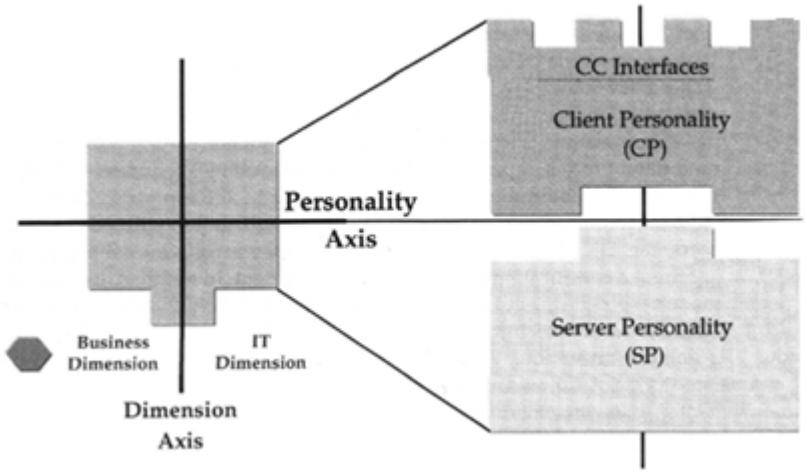


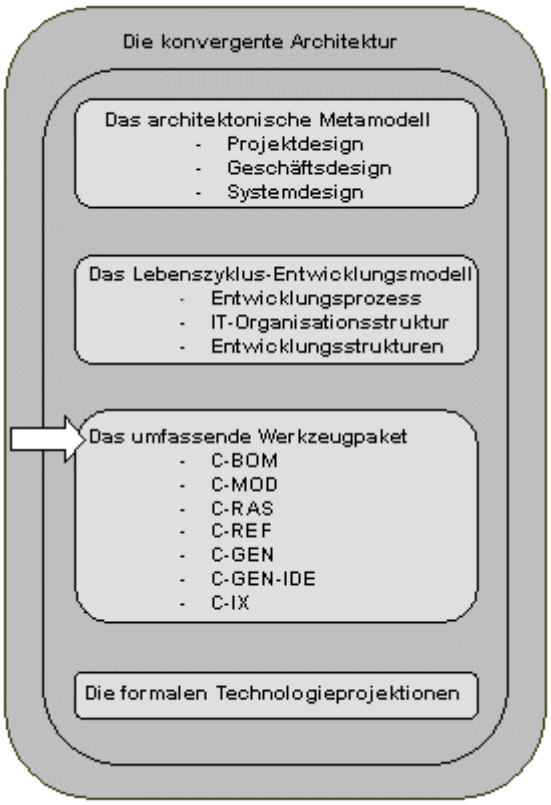
Abbildung 14: Dimensionen und Persönlichkeiten einer konvergenten Komponente[51]

Weiterhin ist jede Komponente horizontal in zwei Persönlichkeiten unterteilt, die Client-Persönlichkeit (client personality, CP) und die Server-Persönlichkeit (server personality, SP). Diese Persönlichkeiten dienen der Kapselung der zwei Aspekte, die in einem verteilten System zwingend unterschieden werden müssen. Auch bei der Entwicklung von nicht-verteilten Systemen wird zwischen CP und SP unterschieden. Dadurch besteht die Möglichkeit, die Komponente in einem verteilten System (wieder) zu verwenden.

[50] Dies geschieht in Form eines UML Profils.
[51] Quelle: [HUBE02], S. 86.

2.2.4 Das umfassende Werkzeugpaket

Das umfassende Werkzeugpaket ist eine Software, die die Unterstützung des modellgetriebenen Entwicklungsprozesses gewährleistet. Genau abgestimmt auf die konvergente Architektur ist beispielsweise die MDA-Entwicklungsumgebung ArcStyler der Firma Interactive Objects Software (siehe Kapitel 2.3.1).



Das umfassende Werkzeugpaket ist in Module gegliedert, die aufeinander aufbauen und so den Benutzer durch den Entwicklungsprozess leiten. Das Zusammenwirken der Module wird in Abbildung 15 verdeutlicht.
Durch die enge Benutzerführung und eine häufige Verifikation des Modells wird sichergestellt, dass der Benutzer konform zum Architekturstil entwickelt.

2.2.4.1 Convergent Business Object Modeler (C-BOM)

In diesem Modul wird mit der Modellierung des Systems begonnen. Durch einen hohen Abstraktionsgrad und die Verwendung intuitiver Verfahren (z.B. CRC-Card-Modellierung) wird eine Zusammenarbeit zwischen Experten aus den Fachabteilungen und Entwicklern erleichtert. In diesem Arbeitsschritt werden die fachlichen Anforderungen und das Geschäftsmodell bestimmt und organisiert. Dabei existieren zwei Modi, die sich zum contract-based Design kombinieren: Die Modellierung der Geschäftsobjekte und die Modellierung von Use-Cases. Es werden vor allem der T-Bar Geschäftsmodellierungs- und Anforderungs-Workflow sowie der Analyse durch Design (ABD) Workflow unterstützt.
Die Ergebnisse werden im Repository (C-MOD) gespeichert.

2.2.4.2 Federated UML/XML Repository (C-MOD)

Das Repository ist eine Java-Implementierung des UML-Metamodells, erweitert durch UML-Profile. Es dient als Schnittstelle zwischen den einzelnen Modulen. Hier werden die

Ergebnisse der verschiedenen Module abgeglichen und koordiniert in einem allgemeinen UML/XML-Modell abgelegt. Das C-MOD ist für den Benutzer weitestgehend unsichtbar, es sei denn, er greift aktiv über Programmier- oder Scripting-Interfaces darauf zu oder nutzt die Möglichkeit des XML-Exports/Imports.

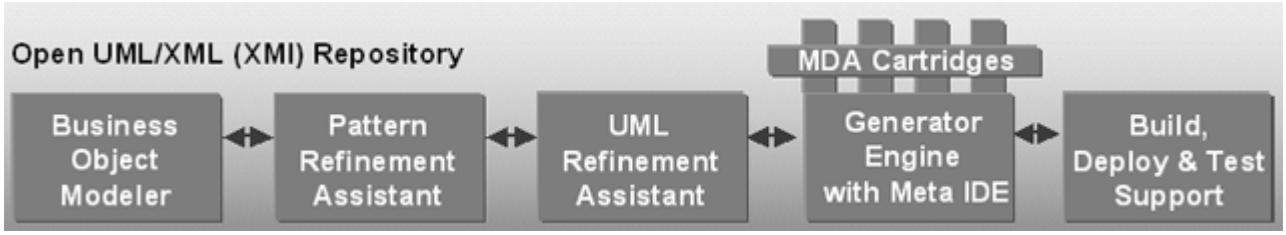


Abbildung 15: Module des umfassenden Werkzeugpaketes[52]

2.2.4.3 Convergent Pattern Refinement Assistant (C-RAS)

Basierend auf den Ergebnissen aus dem C-BOM wird das Modell hier weiter verfeinert. Die vorher modellierten Komponenten werden weiterverwendet und grafikbasiert detailliert. Dazu gehört die Modellierung von Verantwortlichkeiten, Operationen und deren Signaturen. Das Ziel ist, das Geschäftsmodell in ein UML-Komponenten-Modell umzusetzen[53]. Während des gesamten Verfeinerungsprozesses wird der Benutzer durch die Software geleitet, so dass die Konvergenz erhalten bleibt. Hier wird schwerpunktmäßig der oben beschriebene Analyse-durch-Design Workflow unterstützt. Auch hier werden die Ergebnisse im Repository gespeichert.

2.2.4.4 Convergent UML Refinement Assistant (C-REF)

Die Ergebnisse aus dem C-RAS werden geladen und abschließend in Anlehnung an den ausgewählten Modellierungsstil verfeinert. Eigenschaften von Komponenten werden durch die aktivierte Technologieprojektion vorgegeben. Hier können schon sehr viele technologiespezifische Einstellungen vorgenommen werden, wobei es auch möglich ist, mehrere Technologien parallel zu modellieren. Im C-REF Modul können dem Modell neue Komponenten hinzugefügt werden. Diese Änderungen werden automatisch in C-BOM und C-RAS nachgezogen. Je nach Anwendung ist es ebenfalls möglich, die Modellierung direkt im C-REF zu beginnen, z.B. wenn ein fertiges Modell des Systems vorliegt. Des Weiteren lassen sich von diesem Modul aus Aktivitäten wie Verifikation, Generierung u.a. steuern.

---

[52] Quelle: [IOSW03h].
[53] Vgl. [BUTL02], S. 4.

2.2.4.5 Convergent Translative Generator (C-GEN)

Der Generator liest die Modellinformationen aus dem Repository und die Übersetzungsregeln aus der Technologieprojektions-Cartridge. Sowohl Cartridge-Details (z.B. Port-Nummer des Applikationsservers) als auch Eigenschaften des Generators (z.B. Ausgabepfad) können eingestellt werden. Aus den gelesenen Informationen wird Programmcode generiert, der in Verzeichnissen und Dateien an einem angegebenen Pfad abgelegt wird. Beim Generieren werden automatisch „Protected Areas“[54] eingefügt: Dies sind geschützte Bereiche im generierten Code, in denen der Benutzer von Hand Quellcode ergänzen kann. Diese Bereiche bleiben bei einer erneuten Generierung erhalten. Der Generator kann aus dem C-REF oder von der Kommandozeile aus aktiviert werden.

2.2.4.6 Convergent Generator IDE (C-GEN-IDE)

Technologieprojektionen oder auch (Technologieprojektions-) Cartridges sind so genannte Metaprogramme (meist in Form von Templates), die dazu dienen, Programme oder andere Informationen aus dem Modell zu erzeugen. In ihnen sind die Regeln definiert, nach denen die Umsetzung erfolgt. Eine Cartridge akzeptiert als Input ein Modell basierend auf einem UML-Profil und übersetzt es in ein oder mehrere Modelle, Code oder andere Infrastrukturen. Sollte es erforderlich sein, Technologieprojektionen zu ergänzen oder neue Cartridges zu erstellen, so wird dies mit Hilfe der C-GEN-IDE durchgeführt. In dieser IDE ist es möglich, Cartridges auf Codeebene oder in UML-Modellen zu bearbeiten, sie zu debuggen und zu testen.

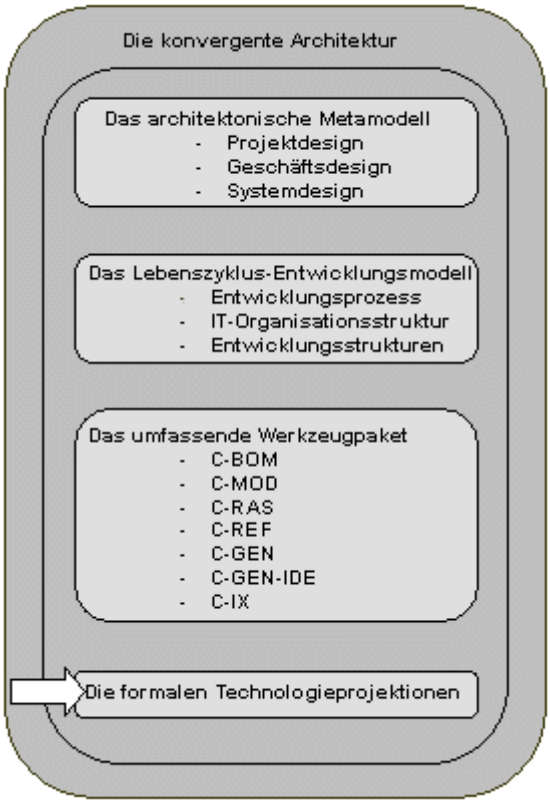
2.2.4.7 Convergent Implement, Deploy & Test Environment (C-IX)

Für den Test der generierten Programme werden in diesem Modul eine IDE sowie Test- und Deploy-Software verwendet. So wird der Übergang von Modellierung und Programmierung bis hin zum Einsatz wesentlich vereinfacht. Mit der Generierung können automatisch Test-Clients erstellt werden, die eine einfache und zeitsparende Überprüfung der Funktionalität erlauben. So ist es bereits in frühen Phasen der Entwicklung möglich, die bereits erstellten Teile des Systems zu testen.
Das Deployment, beispielsweise beim Einsatz eines Applikationsservers, erfolgt ebenfalls weitestgehend automatisiert. Die im C-GEN angegebenen Konfigurationseinstellungen

[54] Eine andere Bezeichnung ist „Free Blocks“.

ermöglichen nun den automatischen Start des Applikationsservers und das selbständige Deployment der Packages.

2.2.5 Die formalen Technologieprojektionen



Die Umsetzung des Modells in Quellcode und weitere Fragmente wie z.B. Deployment- oder Testinformationen wird durch den im Werkzeugpaket integrierten Generator durchgeführt (C-GEN).

Dieser benötigt zum Generieren Umsetzungsregeln, die festlegen, wie Modellelemente behandelt werden sollen. Diese Regeln werden als *formale Technologieprojektionen, Cartridges,* oder *Generatoren* bezeichnet. Je nach Werkzeug sind sie in Java, Jython oder anderen Sprachen definiert. Es ist möglich, Cartridges zu erweitern und selbst zu entwickeln. Auf die spezifischen Eigenschaften und Merkmale, die sich im Umgang mit den formalen Technologieprojektionen ergeben, wird in den Kapiteln über die MDA-Werkzeuge (Kapitel 2.3.1 - 2.3.3) näher eingegangen.

2.2.6 Convergent Architecture und MDA - Stärken und Schwächen

Erfolgt die modellgetriebene Entwicklung unter Verwendung der konvergenten Architektur als Entwicklungsprozessmodell, so lassen sich die Aufgaben und Beiträge der verschiedenen Elemente des Architekturstils dem MDA-Modell von CIM, PIM, PSM und CM zuordnen.

In folgender Matrix wird beispielhaft gezeigt, welche Elemente des Architekturstils in welchem Bereich der MDA (CIM, PIM,PSM,CM) verwendet werden. Diese Verwendung kann je nach Projekt und eingesetztem Werkzeug variieren.

Element	Aufgabe
Das architektonische Metamodell	
RASC-Computing	Erstellung CIM, Weiterentwicklung zum PIM
Das Lebenszyklus-Entwickungsmodell (Entwicklungsprozess)	
T-Bar Geschäftsmodellierungs- und Anforderungs-Workflow	Erstellung des CIM
Architektonische Entwicklung	Weiterentwicklung des CIM
ABD-Workflow	Entwicklung des PIM
Implementierungszyklus-Workflow	Verfeinern des PSM, Generieren des CM, Verfeinern u. Testen des CM
Test-Workflow	Testen des CM
Das Lebenszyklus-Entwickungsmodell (Organisationsstruktur)	
Architektur-Organisation	Definiert und wartet die in CIM und PIM verwendeten Elemente
Systementwicklungs-Organisation	Erstellt CIM, PIM, PSM und CM
Systemeinsatz-Organisation	Wartet CM und dessen Umfeld (u.a.)
Das Lebenszyklus-Entwickungsmodell (Entwicklungsstrukturen)	
Konvergentes Komponenten-Metamodell	Definiert Elemente, welche in CIM und PIM verwendet werden
Das umfassende Werkzeugpaket	
C-BOM	Erstellung CIM
C-RAS	Erstellung PIM
C-REF	Erstellung PSM
C-GEN	Erstellung CM
C-IX	Test und Deployment CM
C-GEN-IDE	Erstellen und Warten der Technologieprojektionen
Die formalen Technologieprojektionen	
Beliebige Projektion	Umsetzung PSM → CM, je nach Werkzeug auch Umsetzung PIM →PSM

Die Verwendung der konvergenten Architektur als Entwicklungsprozessmodell für MDA ist eindeutig zu empfehlen. Der Architekturstil ist an die modellgetriebene Entwicklung angepasst und erlaubt eine optimale Nutzung der Vorteile von MDA. Dies geschieht durch klare Richtlinien und enge Führung der Entwickler. Insgesamt trägt der Architekturstil zu einer planvollen und geordneten Erstellung der Software bei. So wird die Wiederverwendung gefördert und Fehler, die durch unkontrollierte Entwicklung entstehen, gemindert.
Leider unterstützt nur eine der gängigen MDA-Entwicklungsumgebungen die konvergente Architektur. Dennoch bietet sich auch ohne direkte Unterstützung durch eine Software die Verwendung der ersten zwei Elemente (das architektonische Metamodell und das Lebenszyklus-Entwicklungsmodell) an, um so Qualität und Wiederverwendbarkeit der entwickelten Software zu steigern. Ohne die herstellerspezifischen Elemente Werkzeugpaket und Technologieprojektionen ist die konvergente Architektur den aktuellen Entwicklungsprozessmodellen wie RUP sehr ähnlich und bietet neben oben genannten Vorteilen eine gute Vorbereitung für eine spätere Einführung von MDA.
Der Einsatz der konvergenten Architektur ist allerdings nur in Projekten einer gewissen Größe sinnvoll, da sonst der Verwaltungsaufwand unverhältnismäßig hoch ist. Weiterhin erfordert die konvergente Architektur ein sehr diszipliniertes Vorgehen von Seiten der Entwickler.

2.3 MDA-Entwicklungsumgebungen

Bei der modellgetriebenen Softwareentwicklung ist es von großer Bedeutung, welches Tool verwendet wird. Es sind eine Reihe von MDA-Tools am Markt erhältlich, von denen jedoch laut [OBSP03a] nur ein kleiner Teil hundertprozentig MDA-konform ist. Wincor Nixdorf hat für diese Arbeit als Voraussetzung formuliert, dass nur solche Werkzeuge behandelt werden sollen. Diese Werkzeuge unterscheiden sich teilweise sehr in ihren Stärken und Anwendungsgebieten, wie in diesem und dem nächsten Kapitel deutlich wird.
Im Folgenden werden drei ausgewählte MDA-Werkzeuge vorgestellt: ArcStyler der Firma Interactive Objects Software, OptimalJ von Compuware und BITPlan's smartGenerator. Die Wahl fiel auf diese Werkzeuge, da ArcStyler und OptimalJ die umfangreichsten Lösungen sind und in einem Review der Gartner Group als „products at the higher end of the spectrum“[55] hervorgehoben wurden.
smartGenerator wurde stellvertretend für kleinere Werkzeuge ausgewählt, von denen mehrere existieren. Solche Werkzeuge sind kompakter und bieten weniger Benutzerführung sowie eine etwas eingeschränkte Funktionalität[56], sind aber leichter anzuwenden und meist auch einfacher zu erweitern. Kleinere Lösungen sollten trotz des geringeren Funktionsumfanges in Betracht gezogen werden, da hier wesentlich schneller Ergebnisse zu erzielen sind.

[55] Aus [HERR03b] S. 2.
[56] Oft ist nur ein Teil der in Kapitel 2.2.4 vorgestellten Module realisiert.

2.3.1 ArcStyler

2.3.1.1 Überblick

ArcStyler der Interactive Objects Software GmbH ist eine umfassende Lösung für die modellbasierte Softwareentwicklung. Das Werkzeug ist 100% MDA konform und bietet eine integrierte Entwicklungsumgebung für den gesamten Lebenszyklus der Software. Der in Kapitel 2.2 beschriebenen Architekturstil der konvergenten Architektur wird konsequent umgesetzt. Somit ist sichergestellt, dass die Modelle eindeutig sind und der Code entsprechend der Spezifikation generiert wird[57]. Laut [IOSW99] ist dieses Werkzeug seit 2000 auf dem Markt, aktuelles Release ist Version 3.1.

[BUTL02] fasst zusammen:

> *"ArcStyler® is a model-driven architecture and development platform that principally facilitates and supports the entire application lifecycle of analysis, design, development, deployment, and management, based on reusable models of the business, its processes, rules, and logic."*

Optimiert für J2EE und .NET bietet ArcStyler Unterstützung für aktuelle Standards und Technologien wie bspw. Web Services, JSP und XML sowie für gängige Applikationsserver wie z.B. BEA Weblogic, IBM WebSphere, Borland Enterprise Server und JBoss.

Namhafte Unternehmen wie die Deutsche Bank Bauspar[58] oder Austrian National Railroads[59] haben die Architectural IDE[60] nach eigenen Angaben bereits in großen Projekten erfolgreich angewendet.

Obwohl ArcStyler in Java[61] programmiert ist, kann die Software nur auf Windows-Plattformen[62] eingesetzt werden. Grund für diese Beschränkung ist die Bindung an das Modellierungswerkzeug Rational Rose[63], welches auf Windows basiert.

Laut [IOSW03f] benötigt die Software zwischen 50 MB und 200 MB Festplattenspeicher und ist auf einem 300 MHz Rechner mit 128 MB Arbeitsspeicher lauffähig[64]. Auf einem 700 MHz Rechner mit 256 MB RAM ist die Bedienung flüssig und sowohl Reverse Engi-

---

[57] Vgl. [BUTL02] S. 1.
[58] Siehe [IOSWDB] und [SESI03].
[59] Siehe [IOSWAR].
[60] "Architectural IDE" ist die offizielle Produktbezeichnung des Herstellers für ArcStyler, siehe [IOSWWW].
[61] Siehe [BUTL02], S. 7.
[62] Genauer: Windows NT 4.0 (SP 6 und höher), Windows 2000, Windows XP (siehe [BUTL02], S.7).
[63] Rational Rose ist ein fester Bestandteil von ArcStyler.
[64] Dies sind jedoch absolute Mindestvoraussetzungen, die zum komfortablen Arbeiten nicht ausreichen [IOSWSK].

neering als auch Verifikation und Generierung dauern nicht sehr lange (deutlich unter 1 Minute für das später vorgestellte Beispielsystem).

Wie in [HUBE02], S.195 beschrieben, ist ArcStyler die Referenzlösung für das umfassende Werkzeugpaket im Rahmen der konvergenten Architektur. Die einzelnen Module der Software sind vollständig konform zu dem in Kapitel 2.2.4 beschriebenen Werkzeugpaket und werden hier in der Theorie nicht näher beschrieben. Die Module und ihr Zusammenhang werden in Abbildung 16 dargestellt, die Bedienung wird in Kapitel 2.3.1.2 erläutert.

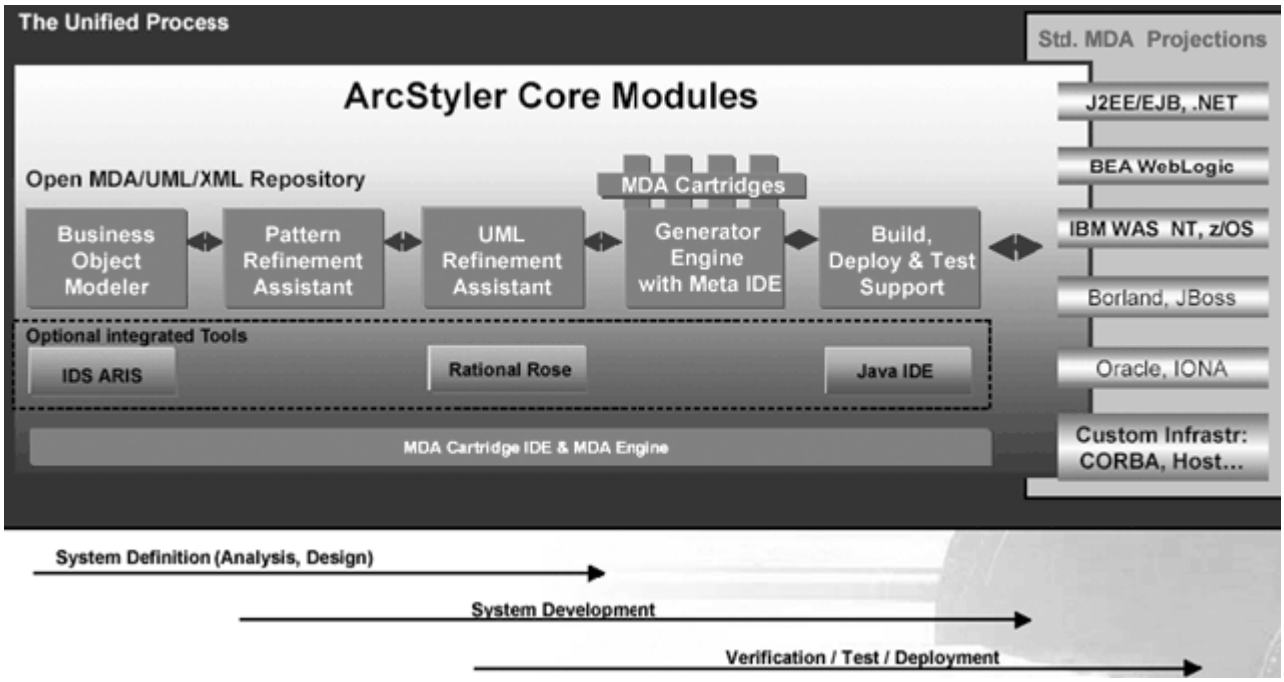


Abbildung 16: ArcStyler - Module und Technologien[65]

Zusätzlich zu den ArcStyler-Modulen ist in der Entwicklungsumgebung *Harvester,* ein Reverse-Engineering-Werkzeug für Java, integriert. Aus dem Quellcode und den Deployment-Deskriptoren wird ein UML-Modell erzeugt, welches in den MDA-Entwicklungsprozess eingebunden werden kann. So ist es möglich, Altsysteme modellgetrieben weiterzuentwickeln, ohne das komplette Modell neu zu erstellen. Es ist jedoch vorgesehen, dass dieses Reverse-Engineering nur einmal durchgeführt wird[66]; danach soll die Entwicklung modell-zentriert[67] weitergeführt werden.

Interactive Objects Software wirbt in ihren „Success Stories“ damit, dass je nach Einsatzgebiet zwischen 40% und 100% des Code generiert werden können[68]. Es ist laut [IOSWSK] auch möglich, Geschäftslogik zu generieren, was allerdings einigen Einschrän-

[65] Quelle: [IOSW02b].
[66] Laut [IOSWSK].
[67] Änderungen dürfen nur Modell vorgenommen werden, der Code wird dann neu generiert.
[68] Entnommen aus [IOSWDB], S. 4.

kungen unterworfen ist; ArcStyler unterstützt lediglich Aktivitätsdiagramme zur Modellierungs von GUIs.

ArcStyler wird in vier Editionen ausgeliefert, die preislich gestaffelt sind[69]:

- ArcStyler Personal Edition
  - Basis Cartridges (Java2, C#, EJB).
  - Kosten: 1890 €

- ArcStyler Enterprise Edition
  - erweiterter Lieferumfang an Cartridges (Java2, C#, EJB, BEA WebLogic, Web Accessors).
  - Kosten: 2900€/3900€/4900€ (<40 /<80 / bel. viele Modellkomponenten)

- ArcStyler Architect Edition
  - Mit C-GEN-IDE
  - Quellcode aller Cartridges
  - Kosten: 9800 €

- ArcStyler Community Architect Edition
  - C-GEN-IDE
  - unbegrenztes Entwickeln und Testen von Cartridges an Modellen mit bis zu 20 Komponenten
  - Zugriff auf die ArcStyler MDA Source Forge Website[70]
  - Quellcode der Java 2, C# und EJB-Cartridges
  - Kosten: Frei erhältlich

2.3.1.2 Bedienung

Da ArcStyler ein sehr umfangreiches Werkzeug ist, wird hier nur im Ansatz auf die Bedienung eingegangen[71] und ein Standard-Entwicklungszyklus verfolgt. Für weitere Informationen und tiefergehende Behandlung von speziellen Themen empfiehlt sich ein Studium von [IOSW03c] und [IOSW03e].

---

[69] Es handelt sich um single user licenses, vgl. [IOSW03b].
[70] Auf dieser Seite können open-source MDA Cartridges ausgetauscht werden.
[71] Das Beispielprogramm in diesem Kapitel stammt aus dem Arc-Styler-Tutorial für J2EE und .NET [IOSW03g]. Inhaltlich vergl. [IOSW03c], [IOSW03e], und [IOSW03g].

Convergent Business Object Modeler (C-BOM)

In diesem Modul werden, wie bereits in Kapitel 2.2.4 dargelegt, initial die Anforderungen an das System bestimmt. Der Business Object Modeler wird in Zusammenarbeit mit Experten aus den Fachabteilungen verwendet und dient zur Erstellung eines ersten Modells. In Abbildung 17 wird der C-BOM in der Anwendungsfall-Ansicht gezeigt, alternativ kann man die CRC-Diagramm-Ansicht aktivieren.

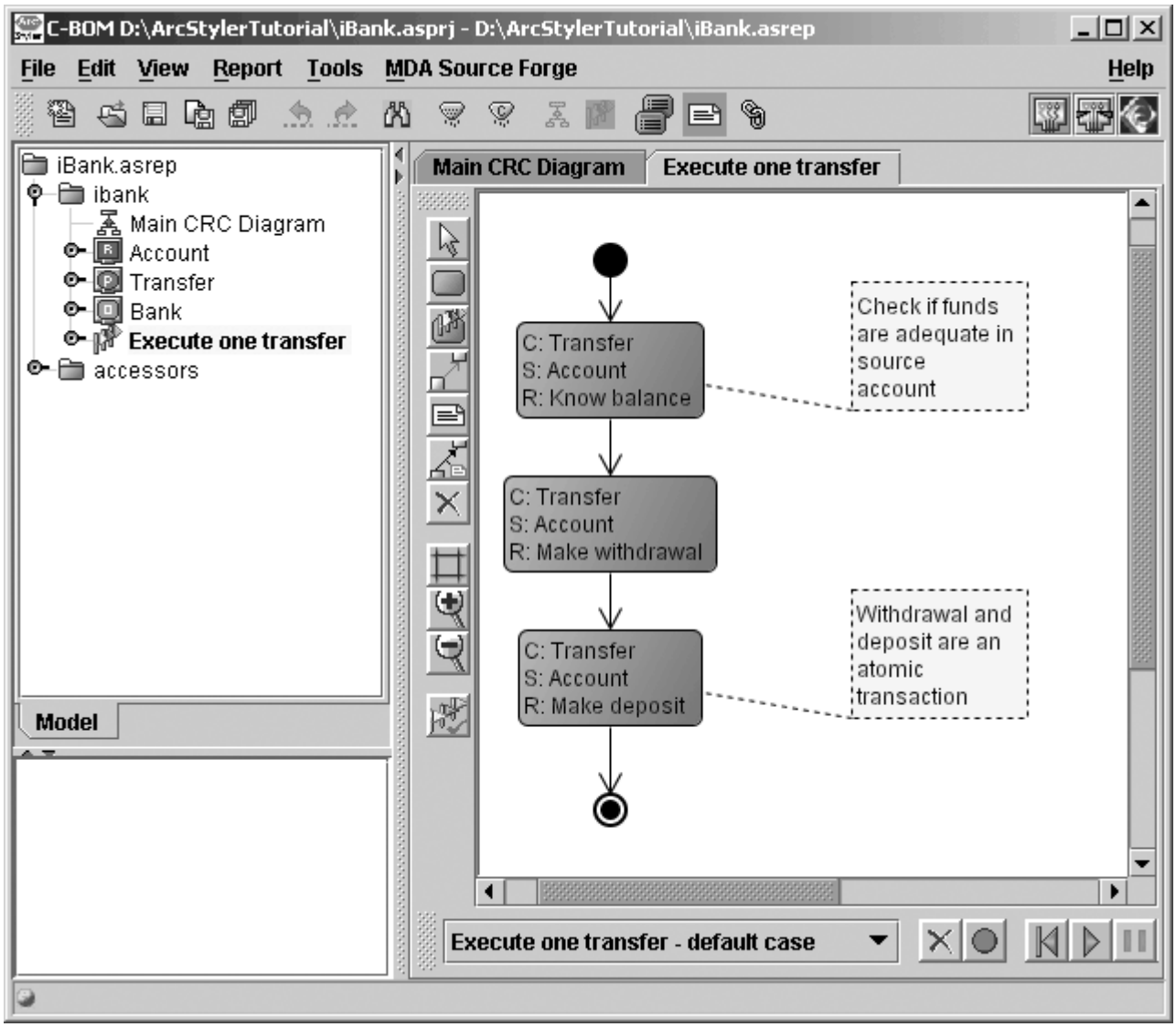


Abbildung 17: Use-Case Walk-Through im C-BOM

Links ist der Browser zu erkennen, welcher in allen Modulen einen Überblick über das System und die einzelnen Geschäftsobjekte bietet. Hier besteht das System aus drei durch OPR-Komponenten (siehe Kapitel 2.2.3.3) repräsentierten Geschäftsobjekten: Account, Transfer und Bank. Diese werden im Browser angezeigt.

Jeder Komponente kann in der CRC-Card-Ansicht eine CRC-Card zugeordnet werden, in der ihre Verantwortlichkeiten modelliert werden. Es werden drei Arten von Verantwortlich-

keiten unterschieden: Wissend, ausführend und besitzend (knowing, doing, owning). Dies wird durch die Symbole vor dem Namen der Verantwortlichkeit angezeigt. Mit diesen Informationen ist es möglich, verschiedene Anwendungsfälle als Walk-Throughs zu erstellen. Im Beispiel wurde ein Anwendungsfall „execute one transfer“ erstellt, der auch links im Browser sichtbar ist.
In der Diagrammansicht werden für jeden Anwendungsfall einzelne Schritte festgelegt. Für jeden Schritt werden Client, Server und Verantwortlichkeit definiert. Es ist möglich, die Szenarien automatisch durchlaufen zu lassen, dazu dienen die Steuerungselemente im unteren Bereich des Fensters. Diese sind in Abbildung 17 nur unvollständig abgebildet, es gibt zusätzlich einen Stop- und einen „Gehe zum Ende“-Button.
Nachdem die Modellierung abgeschlossen ist, ist es möglich das Modell verifizieren zu lassen und es somit auf Fehler im Design wie bspw. zyklische Vererbung oder doppelte Klassennamen hin zu überprüfen. Es ist an diesem Punkt möglich, eine HTML-Dokumentation für unmittelbare Reviews zu generieren. Mit einem Klick auf den mittleren Button in der oberen rechten Ecke der Symbolleiste kann in das folgende Modul, den C-RAS, gewechselt werden.

Convergent Pattern Refinement Assistant (C-RAS)

Das im C-BOM bearbeitete Modell wird automatisch geladen. Links ist wieder der Browser zu sehen, in dem die gleichen Elemente wie im vorherigen Modul angezeigt werden. Im Refinement Assistant wird zusätzlich der Bearbeitungszustand der Geschäftsobjekte angezeigt. Ein bereits verfeinert modelliertes Objekt ist mit einem grünen Häkchen markiert. Ein Objekt, das noch nicht näher detailliert wurde, wird durch ein rotes Ausrufezeichen hervorgehoben (siehe Abbildung 18).
Beim Verfeinern der Modellierung eines Objektes müssen die entsprechenden Verantwortlichkeiten genauer modelliert werden. Je nach Typ der Verantwortlichkeit sind dazu Attribute, Operationen, Assoziationen oder Invarianten genauer zu spezifizieren. Hierbei werden nur die Signaturen spezifiziert, nicht der Inhalt. In Abbildung 18 ist zu sehen, wie der ausführenden Verantwortlichkeit *Make deposit* die Operationssignatur *makeDeposit* zugeordnet wird – mit Parameter und Rückgabetyp. Für Parameter wird in einem spezifischen Dialog eine Richtung (Ein- oder Ausgabeparameter) festgelegt. Weiterhin ist es möglich, besitzende Verantwortlichkeiten in Form von Assoziationen darzustellen, wie z.B. *theAccount* des Geschäftsobjektes Bank. Für Verantwortlichkeiten (z.B. *Know source account*

der Komponente *Transfer*), die sich auf Geschäftsobjekte im Modell beziehen (in diesem Fall *Account*), wird ein Verweis direkt zum Geschäftsobjekt erstellt.

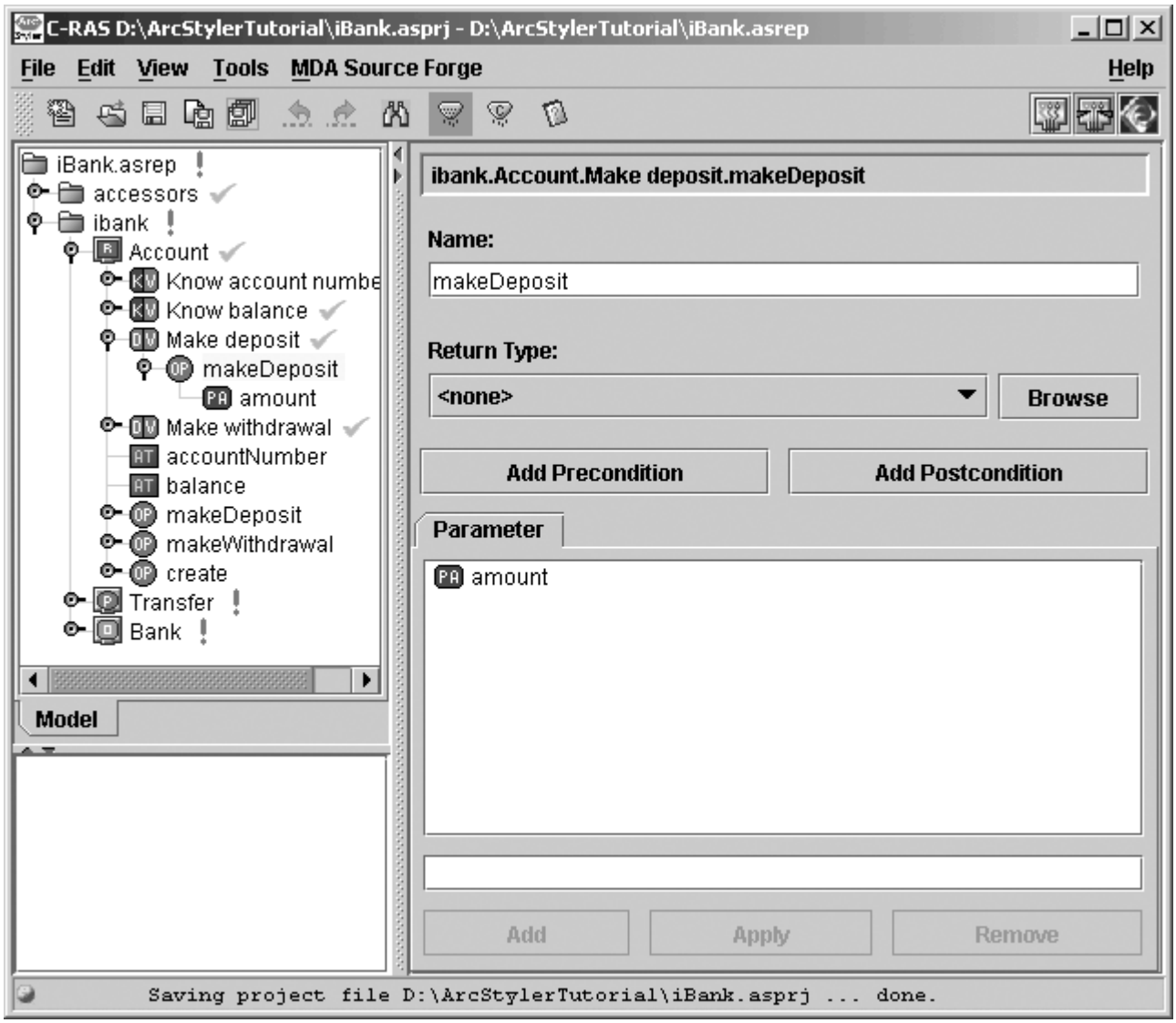


Abbildung 18: Verfeinerung der Geschäftsobjekte in C-RAS

Die Symbole vor den einzelnen Elementen zeigen an, welcher Art diese sind: So bedeutet z.B. „R“ vor *Account*, dass es sich hierbei um eine Ressource handelt, „KV“ vor *Know account number* bezeichnet eine wissende Verantwortlichkeit (knowing), die sichtbar ist (visible). „OP“ repräsentiert eine Operation und „PA“ zeigt an, dass das Element ein Parameter ist.

Sind die Verantwortlichkeiten aller Komponenten korrekt detailliert, wird auch das Package (*iBank*) mit einem Häkchen markiert und es ist möglich das Modell zu verifizieren. Nach einer erfolgreichen Verifikation lässt sich über den Button oben ganz rechts in der Symbolleiste das C-REF-Modul starten.

Convergent UML Refinement Assistant (C-REF)

Dieses Modul bindet Rational Rose ein und verwendet dessen Oberfläche bei der Weiterentwicklung des Modells. Die Oberfläche von Rational Rose ist um einige Schaltflächen und Menüpunkte erweitert, eine Projektverwaltung wird aufgesetzt, jedoch sind die Unterscheide zu Rose vergleichsweise gering.

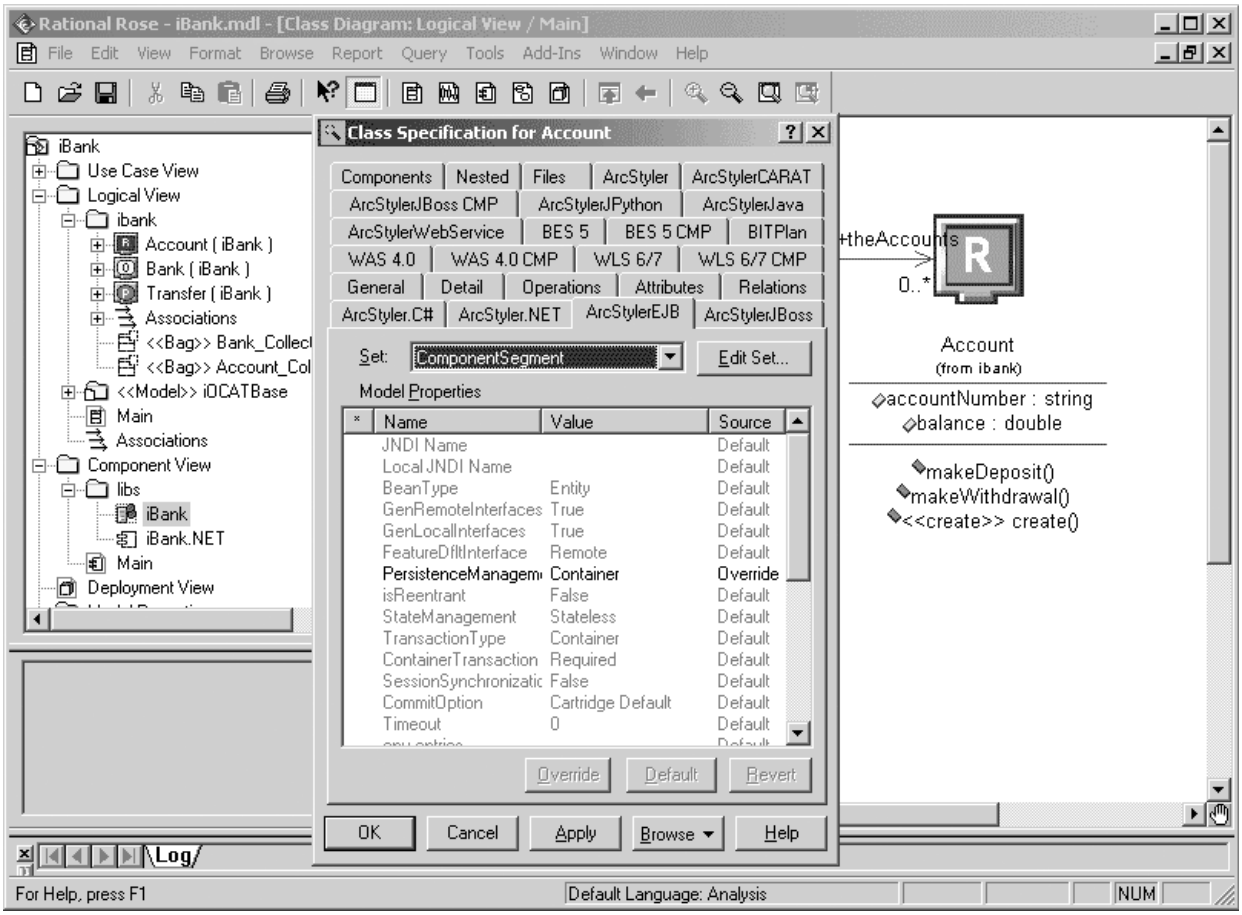


Abbildung 19: EJB-Modellierung im C-REF.

Im C-REF wird das Modell technisch weiter verfeinert. Hauptansatzpunkt ist das Modellieren von technologischen Aspekten wie z.B. EJB's. Auch Aspekte des Deployments können hier modelliert werden. Je nach Technologie müssen plattformbezogene Unterschiede berücksichtigt werden. Diese Einstellungen werden im *Properties*-Menü der Komponenten durch spezifische Karteikarten für verschiedene Sprachen und Plattformen eingetragen. In Abbildung 19 ist zu sehen, wie die Komponente *Account* als EJB modelliert wird. Dies geschieht unter der Karteikarte *ArcStylerEJB*. Soll das Modell auch für .NET verwendbar sein, ist es möglich, auf unter der Karteikarte *ArcStyler.NET* entsprechende Einstellungen für die .NET Umgebung vorzunehmen.

Es ist möglich, im C-REF Klassendiagramme zu erstellen, neue Klassen, Assoziationen und Ähnliches zu modellieren. Dieses bietet sich vor allem an, wenn bereits eine vollständige Anforderungsanalyse des Systems vorliegt.

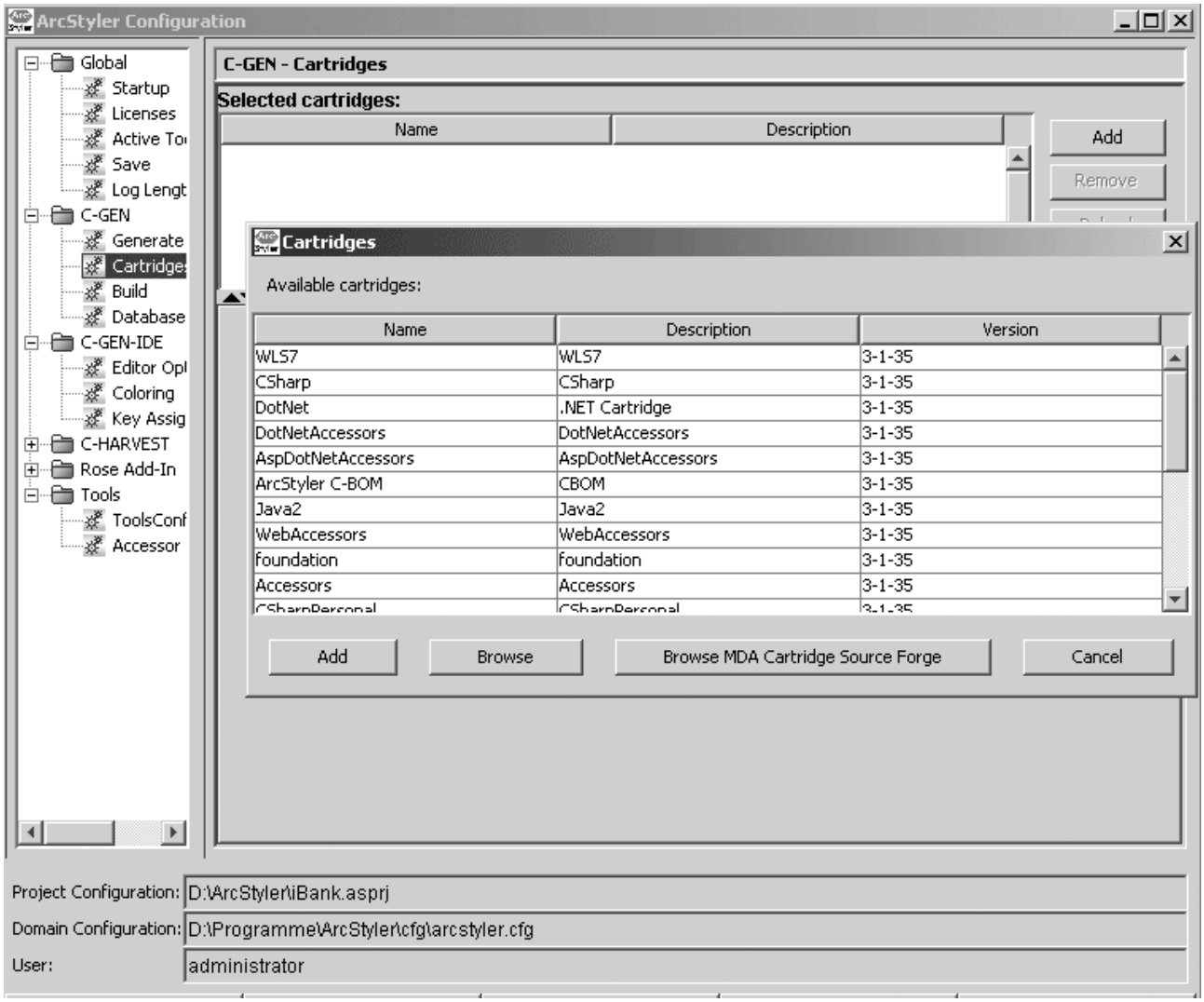


Abbildung 20: Cartridges auswählen im C-GEN

In diesem Fall kann auf die Verwendung der Module C-BOM und C-RAS verzichtet und direkt in Rational Rose mit der Modellierung begonnen werden. Andernfalls sollten alle wesentlichen Elemente bereits existieren, wenn der C-REF aktiviert wird; lediglich übergeordnete Elemente wie die Klassen *Bank_Collection* und *Account_Collection* werden hier noch hinzugefügt.

Wurden alle Aspekte des Modells im C-REF weit genug verfeinert, ist es möglich, aus dem Modell Quellcode und andere Artefakte zu generieren.

Convergent Translative Generator (C-GEN)

Vor dem Generieren muss zunächst der Code-Generator konfiguriert werden.
Über das Rose-Menü *Tools -> ArcStyler -> Configure* lässt sich das ArcStyler-Konfigurationsmenü öffnen. Hier lassen sich Projektname, Output-Pfad, verwendete Cartridges, verwendete Datenbanken und weitere Eigenschaften (z.B. der zu verwendende Applikationsserver) einstellen (siehe Abbildung 20).

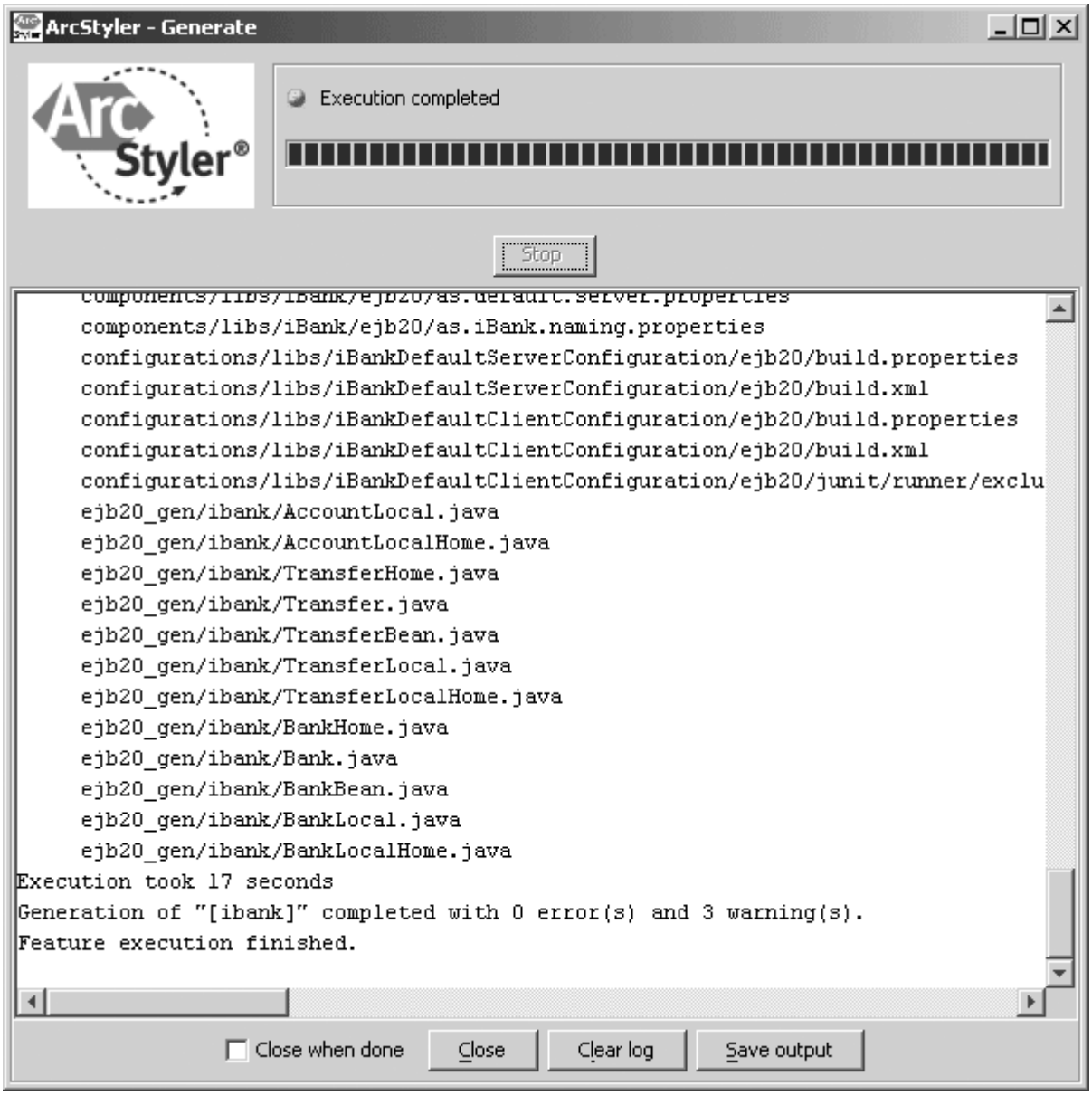


Abbildung 21: Generator-Output in C-GEN

Wie in der Abbildung zu sehen ist, stehen neben Cartridges für Programmiersprachen (CSharp, Java2) auch Cartridges für Webserver und Middleware (WLS7[72], DotNet) sowie

[72] BEA Web Logic Server

ArcStyler-spezifische Cartridges, welche z.B. für das Erstellen neuer Cartridges verwendet werden, zur Verfügung.
Nachdem die notwendigen Einstellungen vorgenommen wurden, ist es möglich über das Kontext-Menü im Browser das Modell zu verifizieren und zu generieren. Während des Generierungsprozesses wird eine Konsole geöffnet, in welcher Generierungsfortschritt und -informationen dargestellt werden (siehe Abbildung 21). Nach Beendigung der Generierung kann man die generierten Artefakte in einer beliebigen IDE öffnen und editieren.
In den meisten Fällen ist ein Ergänzen mit handgeschriebenen Code notwendig, um die volle Funktionalität zu gewährleisten. Der Generator stellt durch seine Implementierung sicher, dass das generierte System jedoch sofort compilierbar ist, auch wenn die Funktionen noch nicht implementiert sind.

Convergent Generator IDE (C-GEN-IDE) und die CARAT Foundation

Neben der bisher beschriebenen Funktionalität stellt ArcStyler auch umfangreiche Erweiterungsmöglichkeiten zur Verfügung. Darunter fällt die Möglichkeit, bestehende Cartridges zu erweitern bzw. neue Cartridges zu erstellen. Hier wird der eigentliche Gedanke von MDA umgesetzt. Durch neue Cartridges können Modelle auf andere Plattformen umgesetzt werden, so ist es möglich, das Wissen in den PIMs für verschiedene Umgebungen zu nutzen.
Das Bearbeiten der Cartridges geschieht in dem eigens für diesen Zweck entwickelten Framework *CARAT*[73] *Foundation*. Dieses Framework ist sehr umfangreich, so dass im Rahmen dieser Arbeit keine detaillierte Schilderung möglich ist. Im Folgenden wird überblickartig auf die wesentlichen Merkmale von CARAT eingegangen[74].
Zum Definieren der Cartridges wird die Sprache JPython[75] verwendet. Die aktuelle Version CARAT 2.0 bietet die komfortable Möglichkeit, Cartridges in UML zu modellieren. Das Framework definiert einen Modellierungsstil in Form von UML-Profilen, der ein Metamodell für die Modellierung der Cartridges vorgibt.
Der Hersteller, die Interactive Objects Software GmbH, bietet eine Reihe von fertigen Cartridges für viele gängigen Programmiersprachen und Plattformen an. In der umfangreichsten Version von ArcStyler sind alle Standard-Cartridges enthalten, die aufeinander aufbauen, wie in Abbildung 22 dargestellt ist. An jedem Knoten des Vererbungsbaumes können neue Cartridges eingefügt werden.

[73] CARAT steht für CARtridge ArchiTecture (siehe [IOSW03g]).
[74] In enger Anlehnung an [IOSWSK].
[75] Ab Version 2.0: „Jython“.

Jede Cartridge verwendet eine Reihe von Klassen, die durch Aufrufe untereinander verknüpft werden, so dass ein Aufrufbaum entsteht. Ergänzt wird diese Struktur durch die Möglichkeit zur Wiederverwendung durch (Mehrfach-)Vererbung sowie einem losen Kopplungsmechanismus, mit dem die strenge Abhängigkeit im Klassenbaum wieder aufgelöst werden kann.

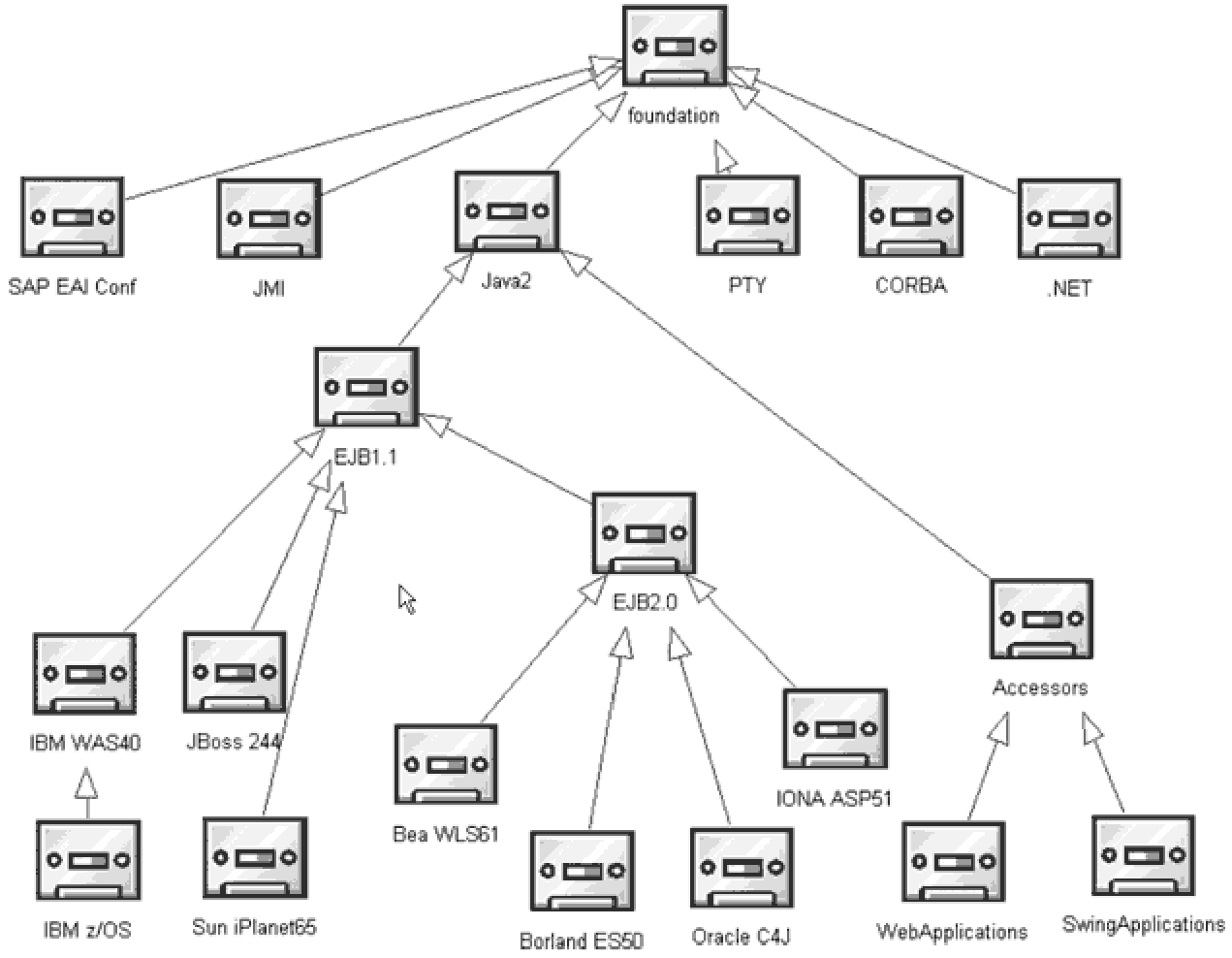


Abbildung 22: Vererbungshierarchie der Standard-Cartridges für ArcStyler[76]

Die Klassen lassen sich in vier Kategorien einteilen[77].

- Artefakte (Artifacts)
- Artefakt-Abschnitte (Artifact sections)
- Artefakt-Gruppen (Artifact sets)
- Blaupausen (Blueprints)

[76] Quelle: [IOSW03g].
[77] Für den folgenden Absatz vgl. [IOSW03d], S. 17f..

Artefakte sind Elemente des Zielmodells, die bei der Generierung erstellt werden. Dies ist bei der Umsetzung in Code eine Datei, bei der Umsetzung in ein UML-Modell bspw. eine Klasse.
Diesen Artefakten untergeordnet sind *Artefakt-Abschnitte*, Elemente innerhalb eines Artefaktes, z.B. ein Abschnitt in einer Datei oder die Attribute einer Klasse. Die Struktur, die durch Artefakte und Artefakt-Abschnitte gebildet wird, spiegelt das Metamodell der Transformation wieder.
Artefakt-Gruppen sind Kombinationen von Artefakten, die unter bestimmten Umständen immer zusammen erstellt werden. So würden bspw. die Artefakte für home interface, remote interface, key class, bean class und deployment descriptor einer EJB in einer Artefakt-Gruppe zusammengefasst.
Blaupausen enthalten das Wissen darüber, wie ein bestimmtes Element des Metamodells in der durch Artefakte und Artefakt-Abschnitte gegebenen Umgebung umgesetzt werden. Im Modell werden diese Elemente durch Stereotypen unterschieden. Klassen, die einem dieser Stereotypen entsprechen, stellen hinterher in ihrer konkreten Form als JPython-Klasse eine Spezialisierung der entsprechenden abstrakten Oberklasse des CARAT Frameworks dar.
Zum Erstellen einer Klasse wird folgendermaßen vorgegangen:
Zunächst wird in der C-GEN-IDE die Klasse als Element im UML Modell erstellt. Hier werden Abhängigkeiten und Vererbungen modelliert. Die fertigen Modelle können dann mit der JPython-Cartridge oder der erweiterten CARAT-Cartridge in JPython-Code übersetzt werden. Nun liegt eine JPython Klasse vor, welche leere Methodenrümpfe besitzt. Diese werden von einem Entwickler aufgefüllt, so dass die Klasse funktionsfähig ist. Um den gewünschten Output zu liefern, verwendet sie vom Entwickler definierte Template-Dateien, die zur Laufzeit geparst und wiederholt befüllt werden. Eine Cartridge bindet mehrere Klassen ein, um so die gewünschte Ausgabe zu erreichen. Das Ändern und Erstellen neuer Cartridges wird im Modul C-GEN-IDE durchgeführt.
Die C-GEN-IDE lässt sich aus Rose über *Tools -> ArcStyler -> Show embedded ArcStyler* und einen Klick auf den entsprechenden Button im sich öffnenden Fenster aktivieren.
In der Generator IDE (siehe Abbildung 23) kann man oben aus dem Pull-Down-Menü eine Cartridge zum Bearbeiten auswählen. Links im Browserbereich kann über die Registerkarten einer von drei Ansichtsmodi gewählt werden: *Model*, *Files* oder *Classes*. Entsprechende Elemente zur aktuellen Cartridge werden dann im Browser angezeigt. Über *Datei -> Laden* ist es möglich, den Quellcode der aktuellen Cartridge zu laden und zu editieren.

Details der Verwendung von JPython beim Arbeiten mit Cartridges werden in [IOSW03d] erläutert.

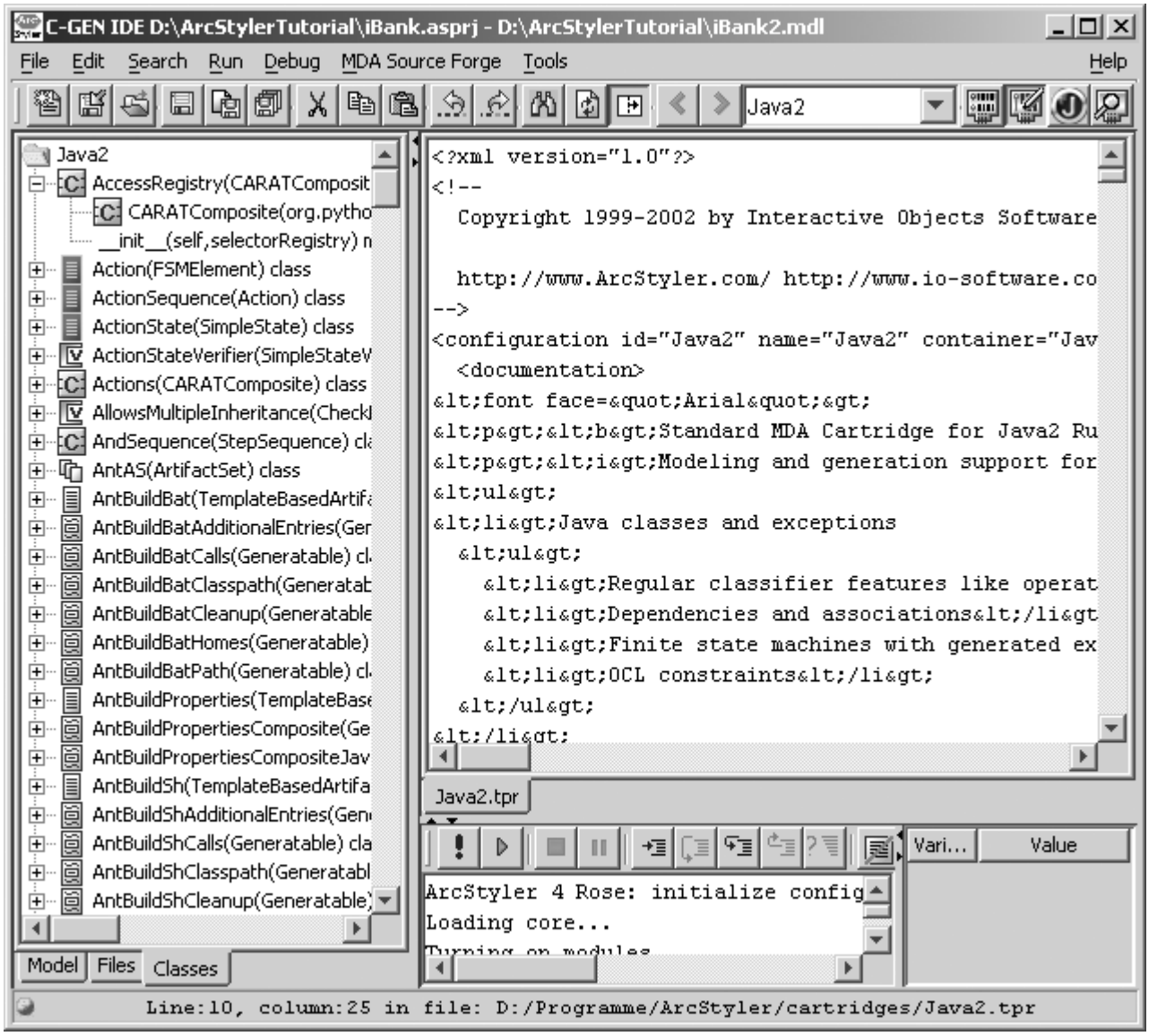


Abbildung 23: C-GEN-IDE mit geöffneter Java2-Cartridge

Convergent Implement, Test & Deploy (C-IX)

Über das Rose-Menü *Tools -> ArcStyler -> Show embedded ArcStyler* und einen Klick auf den entsprechenden Button lässt sich die C-IX-Umgebung starten. Hier wird Deployment und Test der fertigen Anwendungen unterstützt. Ein Teil der mitgelieferten Cartridges generieren neben Quellcode auch Test- und Deployment-Informationen, die im C-IX verwendet werden. Abbildung 24 gibt einen Überblick über die C-IX-Oberfläche: Links im Browser werden alle für Test und Deployment relevanten Elemente des Modells angezeigt, alle anderen Elemente werden ausgeblendet. Die *Testumgebungselemente* (test environment nodes) werden nach der Cartridge benannt.

Im C-IX werden drei Aktionen unterstützt:

- der Build des generierten Systems
- das Starten eines Test Servers
- JUnit-Testläufe.

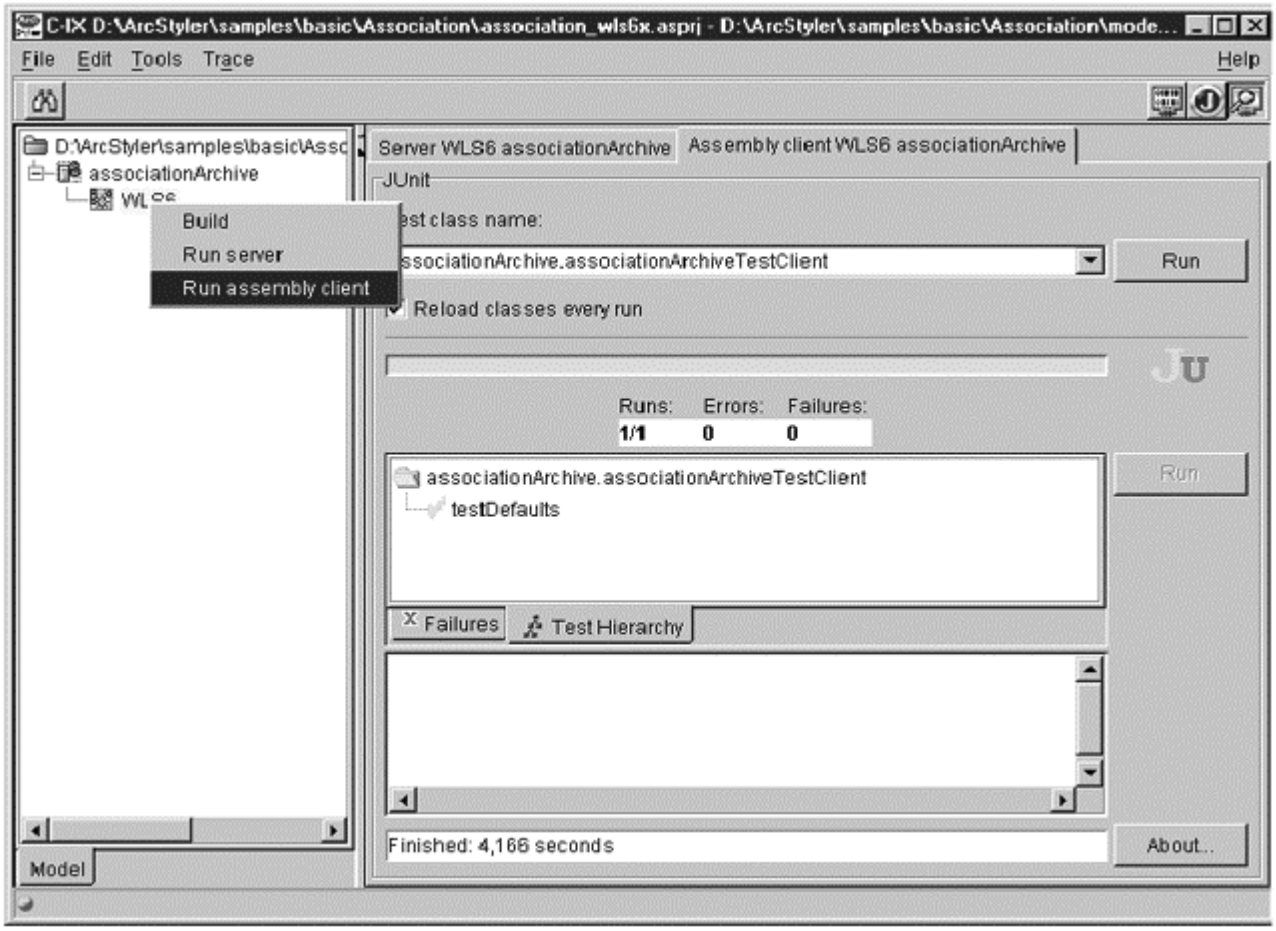


Abbildung 24: JUnit-Tests in C-IX [78]

Diese Aktionen lassen sich durch das Kontextmenü der Testumgebungselemente ausführen. Meldungen zum Build und Statusmeldungen beim Starten des Testservers werden in der jeweiligen Konsole ausgegeben, die bei Bedarf geöffnet wird. C-IX integriert JUnit inklusive der JUnit-GUI (siehe Abbildung 24).

Beim Testen ist zu beachten, dass zu einem Zeitpunkt immer nur ein Testserver und ein JUnit Test ausgeführt werden können.

[78] Quelle: [IOSW03c], S.110.

2.3.2 OptimalJ

2.3.2.1 Überblick

OptimalJ der Firma Compuware bietet genau wie ArcStyler eine sehr umfassende Lösung für die modellgetriebene Entwicklung, beschränkt sich aber auf die Unterstützung von Anwendungen für die J2EE-Plattform.

An Hardwarevoraussetzungen benötigt OptimalJ einen Rechner mit einer CPU-Geschwindigkeit von mindestens 400 MHz, 512 MB Arbeitsspeicher und 250 MB freiem Festplattenspeicher. Während der Installation werden weitere 250 MB benötigt[79]. Ohne diese Hardwaremerkmale (z.B. zu wenig Arbeitsspeicher) wird das Werkzeug teilweise erst gar nicht installiert oder gestartet, aber auch mit einer 1GHz-CPU und 512 MB Arbeitsspeicher dauert das Starten der Software noch länger als eine Minute und liegt dabei weit über den Werten anderer MDA-Entwicklungsumgebungen.

Die Software ist seit 2001 auf dem Markt und momentan in Version 3.0 für Windows NT, 2000 und XP sowie für Linux und Solaris[80] erhältlich. Sie wird zusammen mit Tomcat, JBoss und dem DBMS SOLID ausgeliefert.

Mit dem Ziel, eine objektive Einschätzung der Produktivitätssteigerung bei der Entwicklung mit OptimalJ zu gewinnen, hat Compuware eine Studie bei The Middleware Company[81] in Auftrag gegeben. Im Rahmen dieser Studie entwickelten zwei Expertenteams gleichzeitig die Pet Shop-Anwendung[82]. Eine detaillierte Beschreibung der Vorgaben und des Projektverlaufes ist in [CMPWARk] zu finden. Das modellbasiert entwickelnde Team war drei Tage in OptimalJ geschult worden. Diese Entwickler erzielten einen Vorteil von 35% (an Arbeitsstunden) gegenüber dem Team, welches die Anwendung auf konventionelle Art erstellte. Unter Berücksichtigung der Tatsache, dass dies die erste Anwendung war, welches von den Entwicklern modellbasiert entwickelt wurde, ist bei folgenden Anwendungen eine weitere Arbeitsersparnis von 10-20 % zu erwarten[83].

Compuware konzentriert sich vor allem auf das Generieren von Datenbanken, JSP-Web-Interfaces und EJBs. Geschäftslogik muss von Hand eingefügt werden, was bereits im Modell geschehen kann.

---

[79] Siehe [CMPWARj].
[80] Siehe [BUTL03] S. 6.
[81] TMC ist einer der führenden Anbieter von Schulungen und Beratungen im Bereich J2EE [TMCWWW].
[82] Die Pet Shop-Anwendung(auch bekannt als Pet Store oder Pet Market) ist eine Referenz-Anwendung, die häufig für Beispielimplementierung und bei Vergleichen verwendet wird.
[83] Aus [CMPWARk], S. 13.

OptimalJ ist nicht in einzelne Module gegliedert, sondern orientiert sich direkt an dem MDA-Konzept von PIM, PSM und CM. Analog dazu wird ein *domain model,* ein *application model* und ein *code model* angelegt (siehe Abbildung 25).
Wie in der Abbildung zu erkennen ist, werden sowohl für die Modellierung der einzelnen Modelle als auch für das Mappen des Systems auf ein anderes Modell Muster (Patterns) eingesetzt[84].

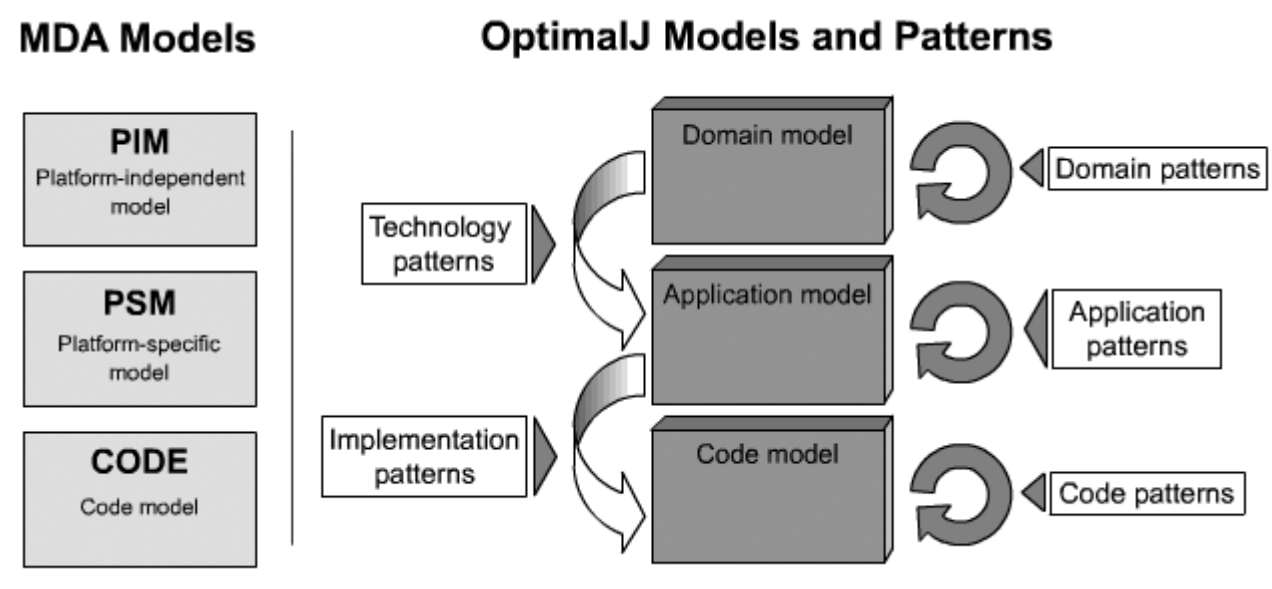


Abbildung 25: Umsetzen der MDA-Modelle in OptimalJ[85]

Dabei wird zwischen Technologie-Mustern (Technology Patterns) und Implementierungs-Mustern (Implementation Patterns) unterschieden. Technologie-Muster zeichnen sich dadurch aus, dass sie ein Modell in ein anderes Modell umsetzen, Implementierungs-Muster dienen zum Erzeugen von Quellcode aus einem Modell[86]. Dabei verwendet Compuware Design Patterns, die von SUN Microsystems als „100% compliant with SUNs J2EE design patterns“[87] zertifiziert wurden. Weiterhin wird eine Unterscheidung zwischen *Funktionalen Mustern* (Functional Patterns) und *Transformations-Mustern* (Transformation Patterns) getroffen. Funktionale Muster dienen zum Erzeugen von neuen Modellen innerhalb eines Abstraktionsgrades, wohingegen Transformations-Muster Modelle in einem neuen Abstraktionsgrad erzeugen. Ein Beispiel für Funktionale Muster sind GoF-Patterns[88]. Die oben

[84] Compuware verwendet für Cartridges die Bezeichnung *Transformationen*. Es wird jedoch weiterhin der Begriff *Cartridges* verwendet, um eine einheitliche Bezeichnung zu gewährleisten.
[85] Diese Grafik wurde aus Ausschnitten der Animation „Concepts of OptimalJ“ erstellt, die beim Starten des Programms gezeigt wird.
[86] In Anlehnung an [CMPWSK].
[87] Aus [CMPWSK].
[88] *Gang of Four-Design Patterns* sind Referenzlösungen für Standard-System- und Softwarestrukturen, siehe [GHJV96] und [GHJV02].

genannte Implementierungs-Muster werden unter Transformations-Muster[89] eingeordnet. Die Analysephase wird von OptimalJ nicht unterstützt, Die Entwicklung beginnt mit der Erstellung des Klassendiagramms, welches hier die Basis für die modellbasierte Entwicklung einer Anwendung ist.
Es besteht die Möglichkeit, Cartridges zu ändern oder zu erweitern, auch Deployment und Test wird unterstützt. Aus der Sichtweise der konvergenten Architektur unterstützt das Werkzeug somit die Funktionalität der Module C-REF, C-GEN, C-GEN-IDE und C-IX des umfassenden Werkzeugpakets.
OptimalJ wird in zwei[90] Editionen ausgeliefert[91]:

- Professional Edition
    - Modellieren und Generieren des Systems
    - Kosten: 5900 €
- Architect Edition
    - Modifizieren von Cartridges, Modellieren und Generieren des Systems
    - Kosten: 11 800 €

2.3.2.2 Bedienung

Die Bedienung von OptimalJ ist ähnlich komplex wie die von ArcStyler, auch wenn beide Werkzeuge sehr unterschiedliche Ansätze verfolgen. Daher wird hier nur auf die Kernelemente der Bedienung anhand eines Tutorials eingegangen[92]. Für detailliertere Informationen zur Bedienung, z.B. Umgang mit verschiedenen Datenbanken empfiehlt sich das Studium von [CMPWARi] und [CMPWARl].
Die Oberfläche von OptimalJ ist in 3 Bereiche gegliedert (siehe Abbildung 26). Links im Fenster ist der *Explorer* positioniert. Die Registerkarten über dem Explorer sind selbsterklärend und werden bei fortgeschrittener Entwicklung verwendet. Standardmodus nach dem Starten von OptimalJ ist der Modus *Editing*.
Im Explorer wird die Hierarchie der Modellelemente in einem Baum angezeigt, wobei mit den Registerkarten unter diesem Bereich ausgewählt werden kann, ob man Domain Model, Application Model oder Code Model bearbeiten möchte. Zusätzlich kann man weitere Modi wie *Meta Model* (Informationen über die Cartridge), *Diagram Thumbnail* (kleine An-

[89] Aus [CMPWSK].
[90] Es gibt genau genommen drei Editionen; die dritte Edition ermöglicht lediglich das Editieren des generierten Codes und kann nicht als MDA-Lösung eingestuft werden.
[91] Die Preise gelten jeweils für eine „Named user"-Lizenz.
[92] Technische Informationen in diesem Kapitel wurden entnommen aus [CMPWARl], das Tutorial basiert auf [CMPWARi].

sicht des aktuellen Modells), *Project <Projektname>* (das aktuelle Projekt sowie generierter Web-Code), *Javadoc* (verfügbare Dokumentation[93]) und *Runtime* (Überblick der Laufzeit-Ressourcen) auswählen. Unter dem Explorer ist das *Properties Window* eingeblendet, welches die Eigenschaften der Modellelemente im Detail anzeigt.

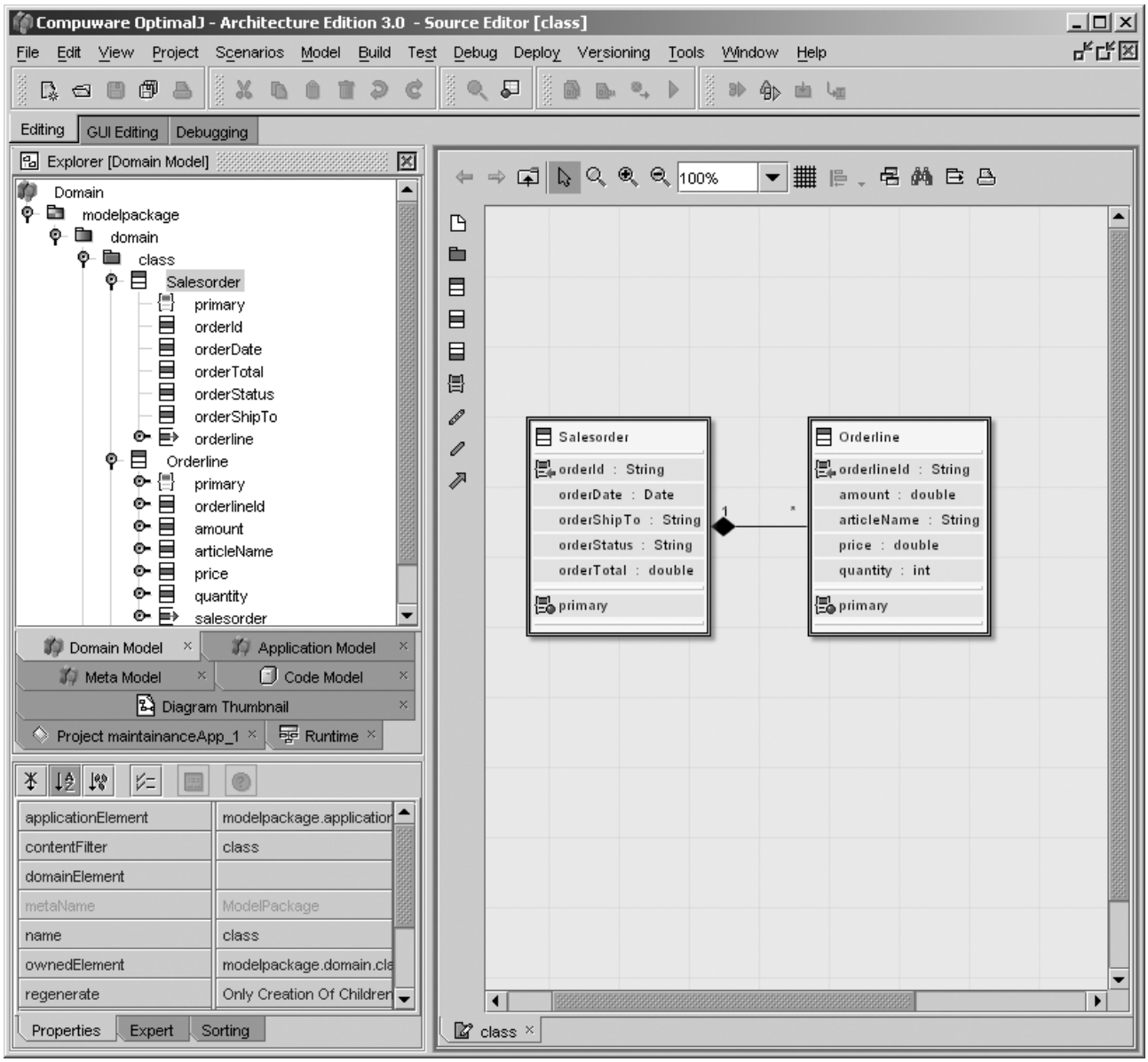


Abbildung 26: Gesamtansicht OptimalJ

In der Mitte des Fensters ist der Arbeitsbereich zu sehen. Hier werden die Modellelemente je nach Modell grafisch oder als Quelltext angezeigt und können bearbeitet werden. Während des Verifizierens, Generierens oder Ausführens wird zusätzlich unter dem Arbeitsbereich eine Konsole[94] eingeblendet, die den aktuellen Status ausgibt. Im Folgenden wird die

[93] Diese Registerkarte ist nur eingeblendet wenn Dokumentation existiert.
[94] In OptimalJ wird die Konsole als *Output Window* bezeichnet.

Entwicklung eines Beispiel-Systems mit OptimalJ beschrieben[95]. Es handelt sich dabei um ein einfaches Bestellverwaltungssystem, das Bestellungen (salesorder) mit beliebig vielen Posten (orderline) verarbeitet. Zu Beginn wird im Projektmanager (*Project -> Project Manager*) ein neues Projekt angelegt. In jedem Projekt müssen Verzeichnisse und Archive, in denen das Modell gespeichert werden soll oder die benötigte Bibliotheken enthalten, *gemounted* (aktiviert) werden. Dies geschieht in der Explorer-Ansicht *Code Model* durch einen Klick auf *Mount -> Local Directory* im Kontext-Menü des Knotens *Code Model* (siehe Abbildung 27).

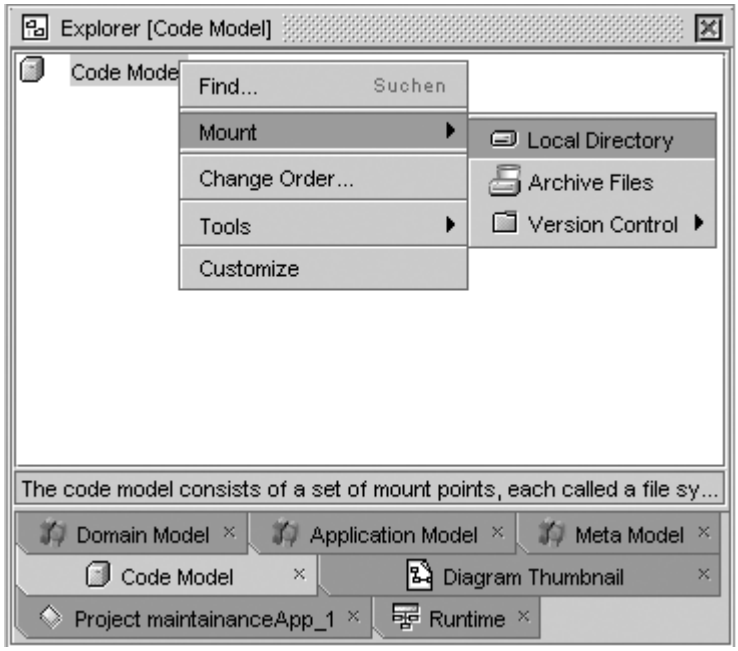


Abbildung 27: Mounten eines Laufwerkes in OptimalJ

Weiterhin muss eine Datenbank für das zu erstellende System angelegt werden. Compuware liefert das Datenbankmanagementsystem SOLID mit. Es ist jedoch auch möglich, externe Datenbankmanagementsysteme zu integrieren[96]. Im Beispiel wird eine SOLID-Datenbank verwendet. Wie die Datenbank im Einzelnen erstellt wird, ist in [CMPWARi] ausführlich erläutert. Nun ist die Umgebung vorbereitet und es kann mit dem Modellieren begonnen werden. Für jede Applikation wird zunächst ein *model package* erstellt. Model packages dienen zur Gliederung der Modellelemente. Es ist möglich, Filter bezüglich des Inhaltes zu setzen[97], so dass Inhalte der gleichen Art zusammengefasst sind. Das umfassende model package enthält das komplette System, welches in mehreren hierarchisch untergeordnete Packages gegliedert ist. Dieses Package wird im domain model mit einem

[95] In Anlehnung an [CMPWARi] S. 1-1 bis S. 1-43.
[96] Unterstützt werden derzeit neben SOLID DB2, MS SQL Server, und Oracle ([CMPWARh]).
[97] Beispiele für Filter sind: class, corba, integration, webservices.

Klick im Kontextmenü des Knotens *domain* erstellt, ein Wizard unterstützt den Erstellungsvorgang. Für das umfassende Package wird kein Filter gewählt. Während der Erstellung mit dem Wizard kann für das System eine Architektur ausgewählt werden, was in OptimalJ als *Initial Structure* bezeichnet wird (siehe Abbildung 28).

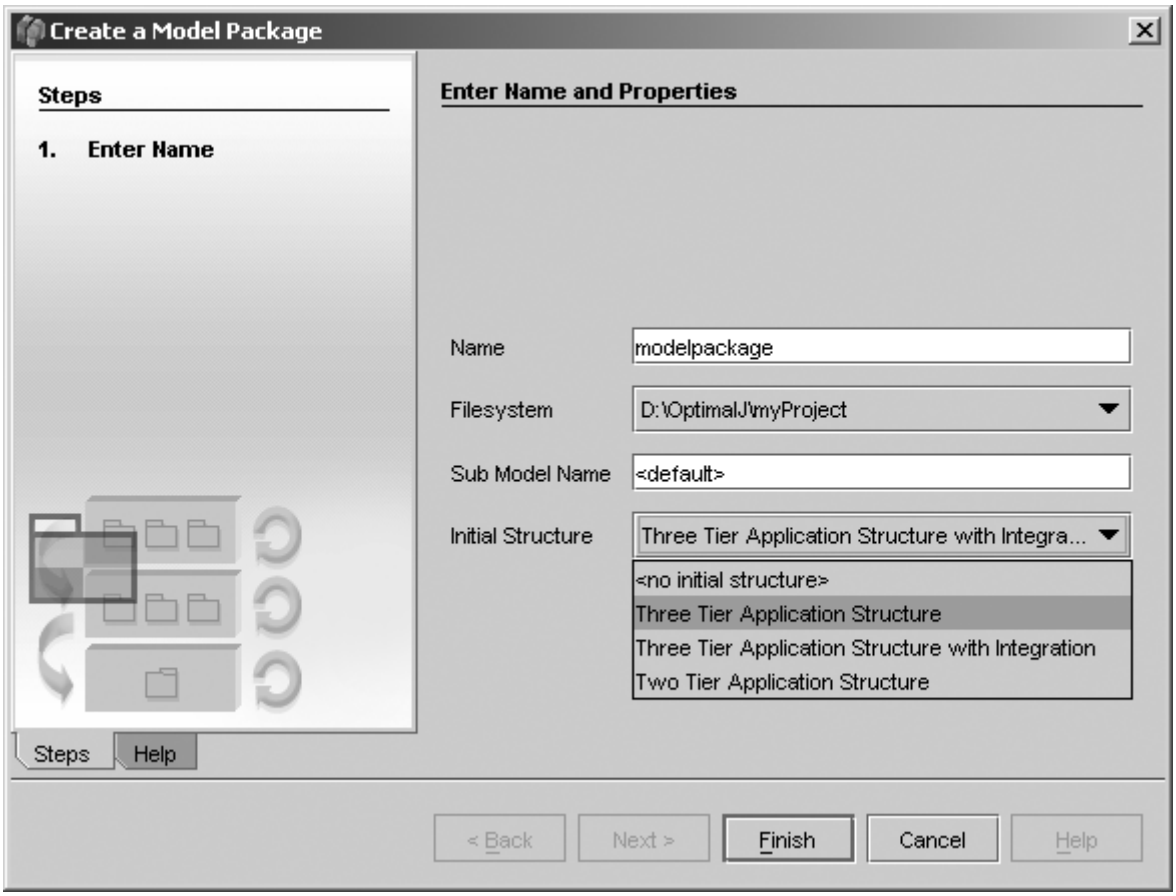


Abbildung 28: Anlegen eines neuen model package in OptimalJ

Je nach gewählter Struktur werden standardmäßig untergeordnete model packages angelegt[98]. In diesem Fall (3-Tier-Application Structure) sind dies:

- ein *domain model* mit den untergeordneten Packages *class* und *service*, wobei class die statischen, service die dynamische Informationen zum Modell aufnimmt[99],
- ein *application model* mit den untergeordneten Packages *dbms*, *ejb* und *web*, die jeweils die Informationen der einzelnen Tiers aufnehmen

Analog zu dieser Struktur werden Elemente im Code Model angelegt, diese repräsentieren zu erstellende Verzeichnisse und Dateien.

Im nächsten Schritt werden im domain model Klassen angelegt. Dies erfolgt über das Kontextmenü des Elements *class* (denn hier werden statische Informationen abgelegt). Durch

[98] Für genaue Beschreibung der einzelnen Strukturen siehe [CMPWARI], S. 44.
[99] Unter class sind vor allem Klassendiagramme abgelegt, service enthält verschiedene Dienste die das fertige System anbieten wird.

einen Klick auf *New Child -> DomainClass* öffnet sich ein Wizard in welchem die Eigenschaften der Klasse zu spezifizieren sind[100] (siehe Abbildung 29). Die angelegten Attribute werden im domain model hierarchisch unter der Klasse angezeigt. Mit einem Doppelklick auf die entsprechende Klasse ist es möglich, das UML-Diagramm im Arbeitsbereich anzuzeigen. Durch Auswählen eines entsprechenden Werkzeuges können Assoziationen per Mausklick gezogen und ihre Eigenschaften festgelegt werden (siehe Abbildung 30).

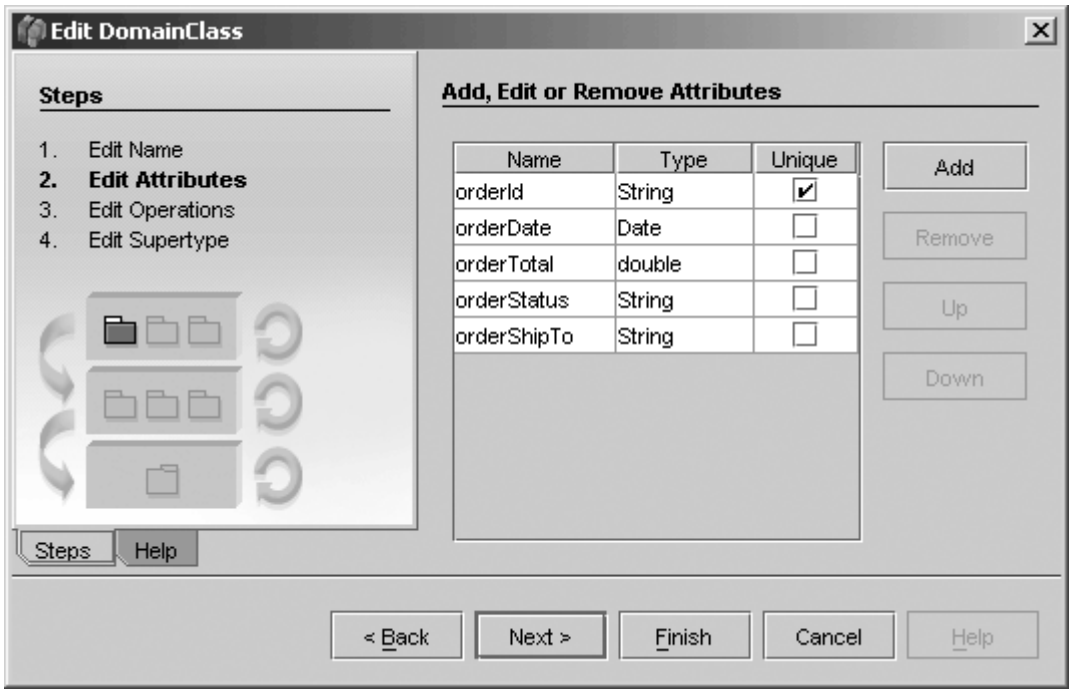


Abbildung 29: Erstellen einer Klasse im domain model von OptimalJ

Nachdem das PIM in Form aller Klassen mit Attributen, Operationen und Assoziationen vorliegt, kann daraus das PSM generiert werden. Durch Klick auf das Menü *Model -> Generate Model -> Generate DBMS from Domain* öffnet sich ein Wizard, der bei der Eingabe einer Quelle und eines Ziels für das Generieren des Datenbankschemas hilft:

- im *domain model* wird der Knoten ausgewählt, der die Informationen enthält (domain)
- im *application model* wird der Knoten ausgewählt, in welchem das generierte Datenbankschema abgelegt werden soll (dbms).

Nachdem alle Einstellungen vorgenommen wurden, wird das Datenbankschema sowie alle nötigen Scripts zum Arbeiten mit der Tabelle (create, drop etc.) generiert. Das generierte Schema wird im Arbeitsbereich grafisch dargestellt, so dass der Benutzer die Ergebnisse sofort überprüfen kann.

[100] Dies beinhaltet Name, Attribute, Operationen (Signaturen) und Oberklasse.

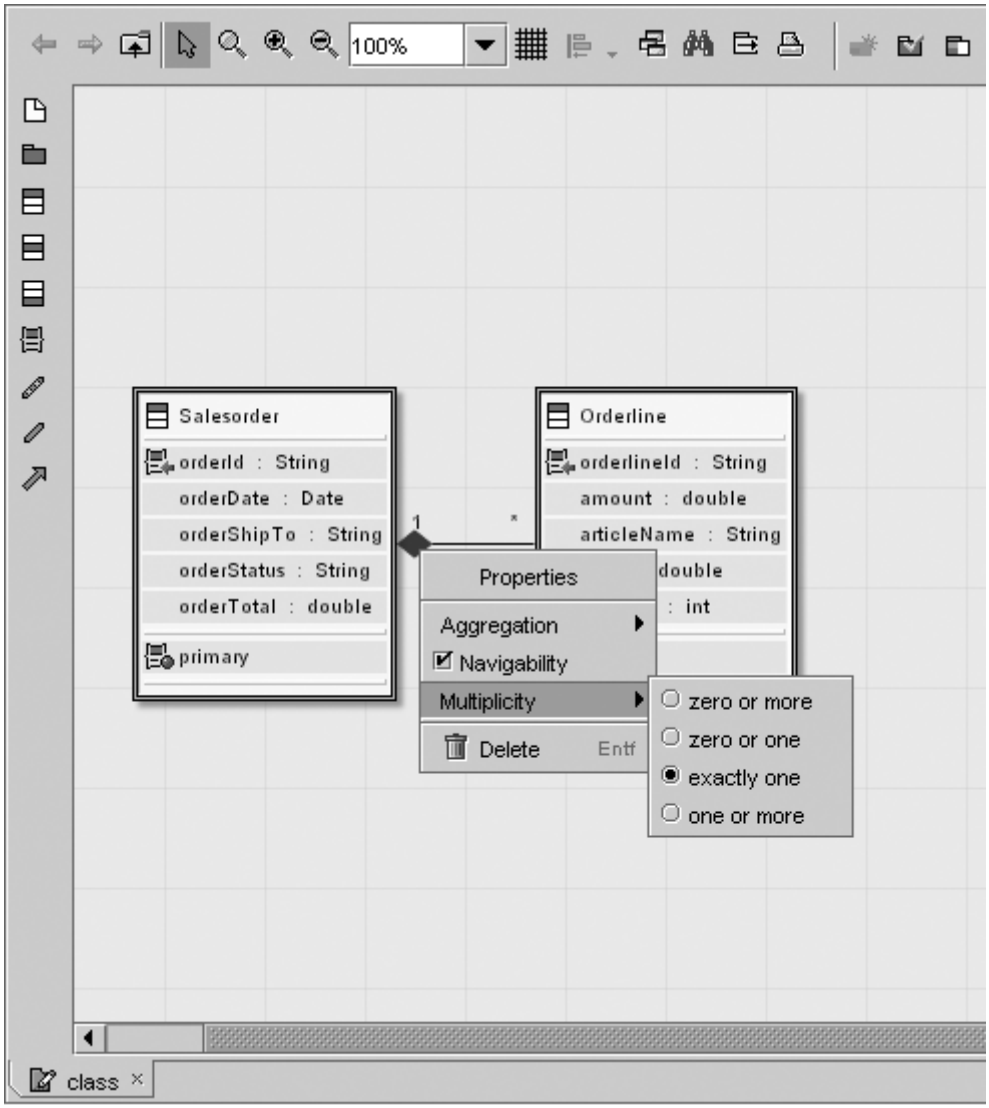


Abbildung 30: Modellieren von Assoziationen in OptimalJ

Ähnlich wird beim Generieren des ejb models verfahren. Dieses wird im application model im Knoten *ejb* abgelegt. Je nach Art der modellierten Assoziationen wird automatisch entschieden, für welche Elemente Beans generiert werden sollen[101].

Auch das Generieren des web models erfolgt analog, die Komponenten werden direkt an die EJBs angebunden, wie in Abbildung 31 gezeigt wird. Links im Explorer ist eine Übersicht der generierten Komponenten zu sehen, rechts im Arbeitsbereich ist die Anbindung des Web-Interfaces an die EJB dargestellt.

In Abbildung 31 ist ebenfalls gut zu erkennen, wie das PSM aufgebaut ist. Unter den drei Oberkategorien *web*, *ejb* und *dbms* sind die jeweiligen Elemente (JSP, EJBs und Datenbankschemata) generiert worden, die die modellierten Elemente *salesorder* und o*rderline*

[101] Da Salesorder eine Aggregation von Orderlines ist, wird für Orderline keine entity bean angelegt.

repräsentieren. Dabei wurde berücksichtigt, dass Salesorder eine Menge von Orderlines ist.

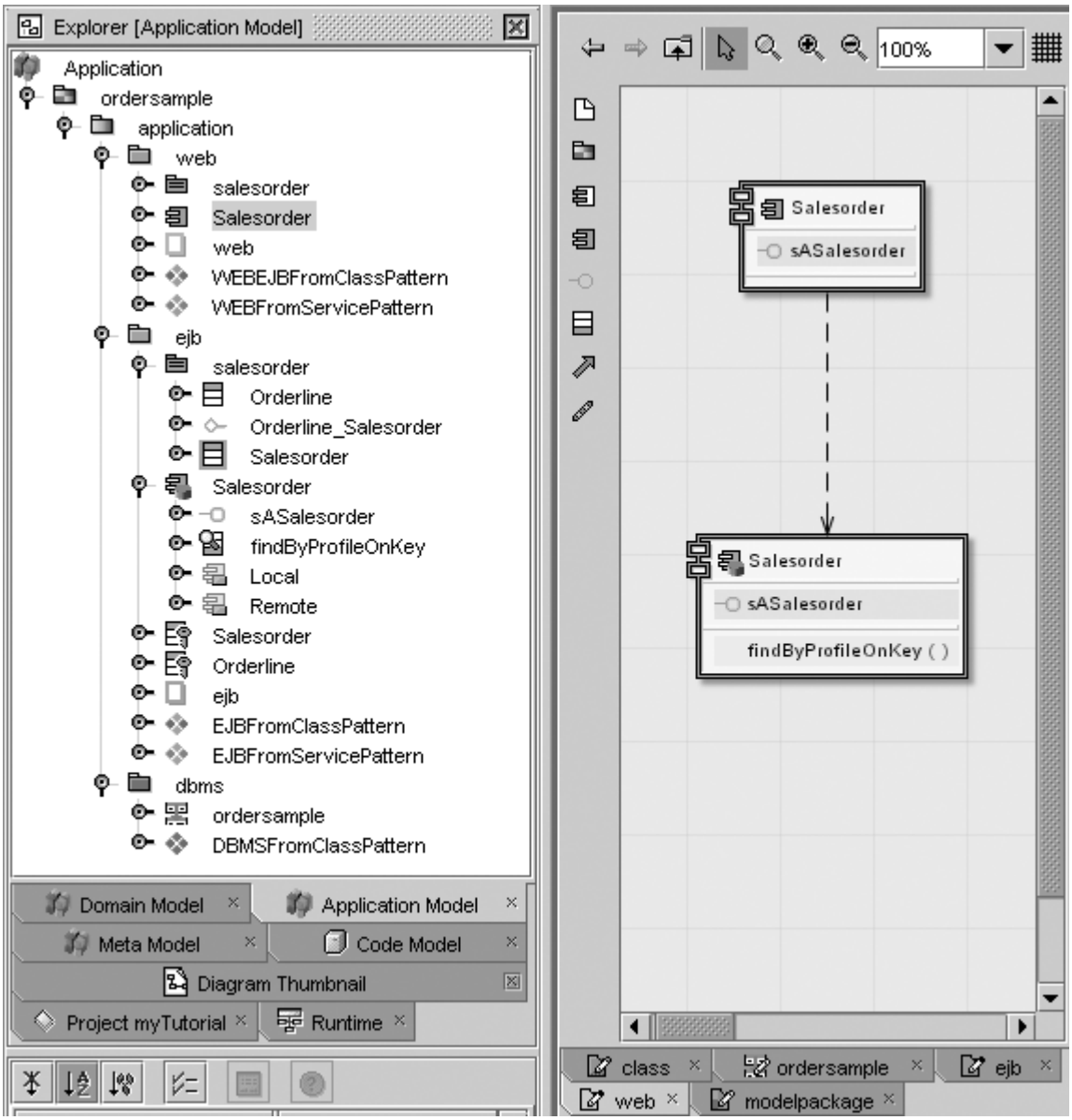


Abbildung 31: Generiertes application model in OptimalJ

Diese Struktur ist charakteristisch für eine 3-Tier-Architektur, bei einer 2-Tier-Architektur fehlt die *ejb*-Komponente. Selbstverständlich ist es auch möglich, eigene Architekturen zu entwerfen, doch erwartet OptimalJ bspw. zum Generieren einer Datenbank immer ein Package mit Filter auf *dbms*.

Nun ist das PSM vollständig generiert und kann in Code umgesetzt werden. Über *Model -> Generate All Code* wird der Generierungsvorgang gestartet. Bevor das Modell in Quell-

code umgesetzt wird, überprüft OptimalJ die Integrität des Modells und erwartet die Eingabe der Verzeichnisse, in denen die generierten Dateien abgelegt werden sollen[102]. Während des Generierungsvorganges werden Statusmeldungen in der Konsole angezeigt. Nachdem der Code fertig generiert ist, wird in die Ansicht *Code Model* gewechselt. Hier kann im Kontextmenü der generierten Komponenten mit Klick auf *Compile All* der generierte Code kompiliert werden. Der nächste Schritt zu einer lauffähigen Anwendung ist das Erstellen der Datenbanktabellen aus dem bereits generierten Datenbankschema. Die DBMS-Metadatei sowie alle Scripts für Operationen (create, initialize, drop) sind im package *dbms* abgelegt, mit deren Hilfe die Generierung der Datenbank möglich ist. Wie die Erstellung im Einzelnen geschieht, ist vom verwendeten DBMS abhängig und wird in [CMPWARi], S. 1-30 erläutert.

Abbildung 32: In OptimalJ generierte Web-Oberfläche

Für das Deployment der Applikation muss der in OptimalJ integrierte EJB Server über *Test-> Start EJB Server* aktiviert werden. Dabei wird das Verzeichnis, welches die Deployment-Deskriptoren enthält, angefordert. Im Beispiel ist dies im code model der Ordner *ejb*. Danach kann im Code Model durch Klick auf *Execute* im Kontextmenü der Komponente MainMenu die Applikation gestartet werden. Die generierte Oberfläche wird au-

[102] Es ist auch möglich, diese Verzeichnisse schon zu Beginn des Projektes zu mounten, dann werden sie automatisch verwendet. Wichtig ist, dass die Verzeichnisse sich vom Projektverzeichnis unterscheiden.

tomatisch im Browser angezeigt und die Applikation kann verwendet werden (siehe Abbildung 32).

Auch Compuware stellt eine Erweiterungsfunktionalität für die Transformationsregeln[103] zur Verfügung, so dass OptimalJ auf individuelle Bedürfnisse anpassbar ist. Allerdings ist die Entwicklung eigener Cartridges in diesem Werkzeug ebenso komplex wie in ArcStyler.

```
[TPL]
/*
 * myPatternTPL.tpl
 *
 * Created on Fri Jul 25 00:17:29 CEST 2003
 */
[/TPL]

[PATTERN patternpackage ]

[TPL]
/**
 *
 * @author Administrator
 * @version
 */
[/TPL]

[TEMPLATE PUBLIC MYPATTERNTPL()]
[/TEMPLATE]
```

Abbildung 33: Cartridge-Gerüst in TPL

OptimalJ bietet die Möglichkeit, eigene Meta-Modelle und Implementierungs-Muster zu erstellen, seit Version 3.0 ist es auch möglich, Technologie-Muster zu erweitern. Zu Letzterem existiert noch keine Dokumentation, deshalb wird auf diese Erweiterungsmöglichkeit nicht näher eingegangen.

Ein austauschbares System von Cartridges ermöglicht es dem Architekten, eine eigene *Software Factory* zu erstellen. Eine Software Factory besteht aus Implementierungs-

[103] Compuware nennt Cartridges „Patterns“ oder „Transformationen“. Um eine bessere Verständlichkeit zu gewährleisten, wird hier weiterhin der Begriff „Cartridges“ verwendet.

Mustern für die verschiedenen technologischen Aspekte der Zielarchitektur. Beispiele hierfür sind Ejb Pattern, Sql Pattern, Web Pattern[104]. Meta-Informationen über diese Patterns werden im SFD (software factory definition file) gespeichert, mit dessen Hilfe sichergestellt wird, dass keine ungültige Menge von Mustern verwendet wird.

```
/*
 * myPatternJava.java
 *
 * Created on Fri Jul 25 00:17:34 CEST 2003
 */

package patternpackage;

import com.compuware.tpl.TPL.*;
import com.compuware.tpl.TPL.LANG.*;
//import com.compuware.alturadev.models.optimalFoundation.Tag;

/**
 *
 * @author Administrator
 * @version
 */
public class myPatternJava extends GeneratorBase {
    public myPatternJava(TplContext c) {
        super(c,"myPatternJava");
    }
    public myPatternJava(TplContext c, java.io.Writer w) {
        super(c,"myPatternJava",w);
    }
    public myPatternJava(TplContext c, java.io.Writer w, java.ut
        super(c,"myPatternJava",w, s, i);
    }

    public void    generate  (java.io.Writer newWriter, Object[]
        print("No source found");
    }
}
```

Abbildung 34: Cartridge-Gerüst in Java

Für die Modifikation bestehender Implementierungsmuster und das Erstellen eigener Muster stellt Compuware die TPL (Template Pattern Language) zur Verfügung. Der TPL-Code wird vom TPL-Compiler in Java-Code übersetzt und als Implementierungs-Muster-Modul in OptimalJ verwendet.

[104] Aus [CMPWARc].

Die Struktur der Implementierungs-Muster wird in *Implementierungs-Muster-Modellen* (implementation pattern models) beschrieben[105].

Die neuen Muster können an so genannten *Joinpoints,* Knoten in der Hierarchie der Pattern, eingefügt werden. Bei dem Erstellen eines eigenen Implementierungs-Musters wird in der Ansicht *Meta Model* des Explorers ähnlich vorgegangen wie beim Erstellen eines neuen Modells in der Ansicht *domain model*: In einem neuen Package wird ein Implementation Pattern angelegt und dessen Eigenschaften konfiguriert. Dabei ist frei wählbar, ob die neue Cartridge in TPL oder direkt in Java erstellt werden soll.

Ist die Cartridge fertig modelliert, kann das Modell mit Klick auf *Generate Code* im Kontextmenü in Quelltext umgesetzt werden. Dabei wird Code in der vorher gewählte Zielsprache erstellt.

Der generierte Code kann geöffnet und modifiziert werden, so dass die gewünschte Funktionalität erzielt wird. Als Einblick in die Charakteristik der Cartridges zeigen Abbildung 33 und Abbildung 34 je eine generierte Cartridge in TPL und Java, die noch nicht mit Logik aufgefüllt wurde.

Die fertige Cartridge wird in *Tools -> Options -> IDE Configuration -> System -> Modules* mit *Add -> Module* dem Pool an Transformationen hinzugefügt.

[105] In Anlehnung an [CMPWARc].

2.3.3 SmartGenerator

2.3.3.1 Überblick

Dieses Werkzeug der Firma BITPlan ist im Vergleich zu den beiden „Komplettlösungen“ ArcStyler und OptimalJ weniger umfangreich. Es bietet jedoch auch interessante Features und gute Erweiterungsmöglichkeiten. Die seit 1999 auf dem Mark erhältliche Software wird als „Generator-Generator[106]“ verkauft, also als Cartridge-Generator: Sie bietet neben dem Generieren von Quellcode einfache und komfortable Entwicklungsmöglichkeiten für eigene Cartridges. Diese werden im Gegensatz zu den meisten Produkten anderer Hersteller in Java bzw. in einer Template-Sprache definiert.

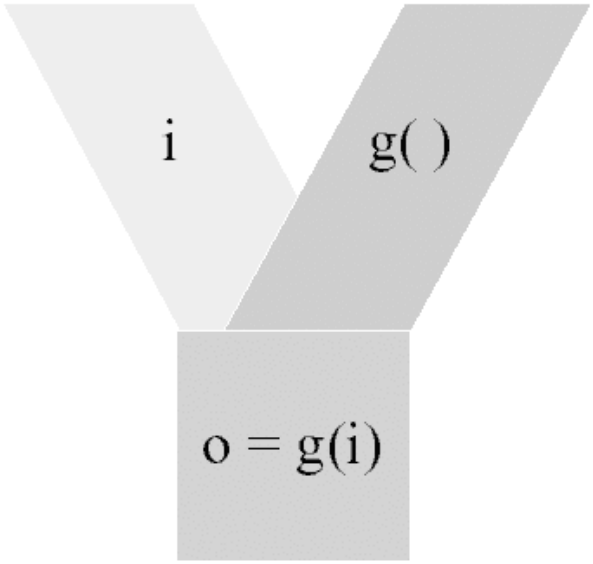


Abbildung 35: Y-Prinzip[107]

BITPlan legt smartGenerator das so genannte Y-Prinzip zugrunde (siehe Abbildung 35)[108]:

- aus strukturierter Information i [Input]
- wird nach einer festgelegten Vorschrift g() [Generator]
- ein genau definiertes Ergebnis o=g(i) erzeugt [Output]

Mit smartGenerator lassen sich unter anderem Architekturen wie CORBA, EJB, Palm Computing Platform oder die Einbindung von Mainframes unterstützen. Zielsprachen der Codegenerierung können bspw. Java, C++, Delphi oder ANSI COBOL[109] sein.

---

[106] *Generator* ist bei BITPlan die Bezeichnung für eine Cartridge.
[107] Quelle:[BITP02a], S. 9.
[108] Entnommen aus [BITP02a], S. 9.

Das MDA-Werkzeug kann laut [BITP02a] auf Erfolge in großen Projekten z.B. bei der Sparkassen Informatik GmbH und der RWE Systems Application GmbH verweisen. Systemvoraussetzungen werden vom Hersteller nicht angegeben, ein Rechner mit 256 MB Arbeitsspeicher und einer 700 MHz Pentium-CPU ist bei weitem ausreichend, auch etwas ältere Systeme dürften keine Probleme bereiten. Die Vollversion lässt sich sowohl auf Windows- als auch auf Linux-Systemen installieren[110]. In seiner Kernfunktionalität deckt smartGenerator die Bereiche C-GEN und C-GEN-IDE ab. Folglich bietet das Werkzeug keine Benutzerführung bei der Erstellung des Modells, keine Verfeinerungsmechanismen und Verifikationsmöglichkeiten im Sinne eines in sich abgeschlossenen und benutzerleitenden Entwicklungsprozesses und unterstützt keinen Architekturstil.

Die Modellierung des Systems erfolgt mit externer Software, welches über eine Schnittstelle an smartGenerator angebunden wird. Zur Zeit werden die UML-Werkzeuge Rational Rose, microTOOL objectiF, ObjectDomain, Poseidon und ArgoUML unterstützt[111], weitere Schnittstellen sind in Vorbereitung.

Die Bearbeitung des Quellcodes (und auch der Cartridges) erfolgt in einem beliebigen Editor, vorgeschlagen wird der Shareware-Editor UltraEdit, für den auch eine Syntax-Highlighting Unterstützung für Cartridges existiert.

smartGenerator setzt beim Austausch mit externer Software auf Standards wie UML 1.3 und XMI 1.1 und wird auch von der erweiterten Funktionalität von UML 2.0 profitieren, da zusammen mit UML 2.0 eine Erweiterung des Funktionsumfanges der Software geplant ist.

In Abbildung 36 ist schematisch dargestellt, wie modellbasierte Softwareentwicklung mit smartGenerator funktioniert. Das in einem externen Werkzeug erstellte Modell wird von einem Generator in Quellcode umgesetzt (untere Zeile, von links nach rechts). Der Generator wird von smartGenerator aus einer Vorlage erstellt, die aus einem kleinen spezifischen Teil und einem großen systematischen Teil besteht (obere Zeile, von rechts nach links). Momentan können mit smartGenerator Strukturen und statische Informationen erstellt werden, jedoch keine Geschäftslogik. Aufgrund der komfortablen Erstellung von Cartridges durch Java-Syntax und Generator-Generator bietet es sich an, die Funktionalität von smartGenerator durch eigene Cartridges zu ergänzen die bspw. Design Pattern umsetzen.

---

[109] Aus [BITPLN].
[110] Aus [BITPSK].
[111] Vgl. [BITP02b].

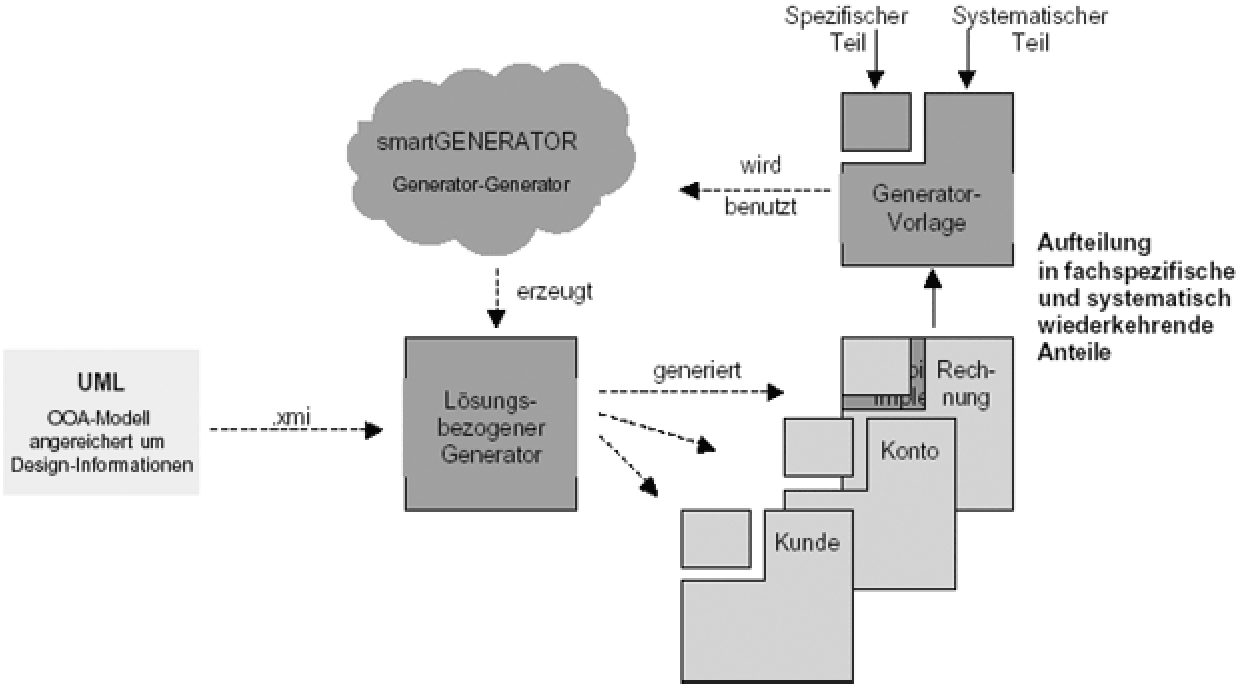


Abbildung 36: Funktionsweise von smartGenerator[112]

In [SCHW03] wird die Umsetzung der Patterns Singleton, Abstract Factory/Factory Method und Visitor ausführlich beschrieben.

Ein Preis für smartGenerator ist nur schwer zu bestimmen. Sollen eigene Cartridges entwickelt werden, so muss in jedem Fall eine vollständige Lizenz für smartGenerator gekauft werden, diese gilt dann für bis zu 50 Mitarbeiter und kostet 25.000 €. Einzelne Lizenzen gibt es nicht zu kaufen, jedoch ist laut [BITPSK] eine Verhandlung sicherlich möglich. In der Lizenz sind jedoch noch keine Cartridges enthalten, diese können für 2.000 € (bestehende Cartridges) und 2.500 € (neu zu entwickelnde Cartridges) bei BITPlan gekauft werden[113].

2.3.3.2 Bedienung

Konfiguration und Import der Modelle

smartGenerator ist mit einer übersichtlichen Oberfläche ohne Untermenüs ausgestattet, diese ist in Abbildung 37 dargestellt.

Über das Scroll-Down Menü *Konfiguration* ist es möglich, verschiedene Konfigurationen von Cartridges, Quellpfaden und Zielpfaden zu verwalten. Die Buttons *Neu, Bearbeiten*

[112] Quelle: [BITP02a].
[113] In Anlehnung an [BITP03a] und [BITP03b].

und *Löschen* dienen zum Anlegen, Ändern (des Namens) und Entfernen, der Button rechts zum Speichern von Konfigurationen. Der Pfad des Modells, aus dem Code generiert werden soll, wird über die Schaltfläche *Modell* angegeben. Über die Schaltfläche *Generator* ist es möglich eine Cartridge zu wählen, mit welcher man das Modell in Quellcode umsetzen möchte. Unter *Optionen* kann man zusätzliche Einstellungen für die Generierung wie z.B. den Pfad einer Logdatei angeben und unter *Zielverzeichnis* wird der Pfad für den erzeugten Quellcode festgelegt.

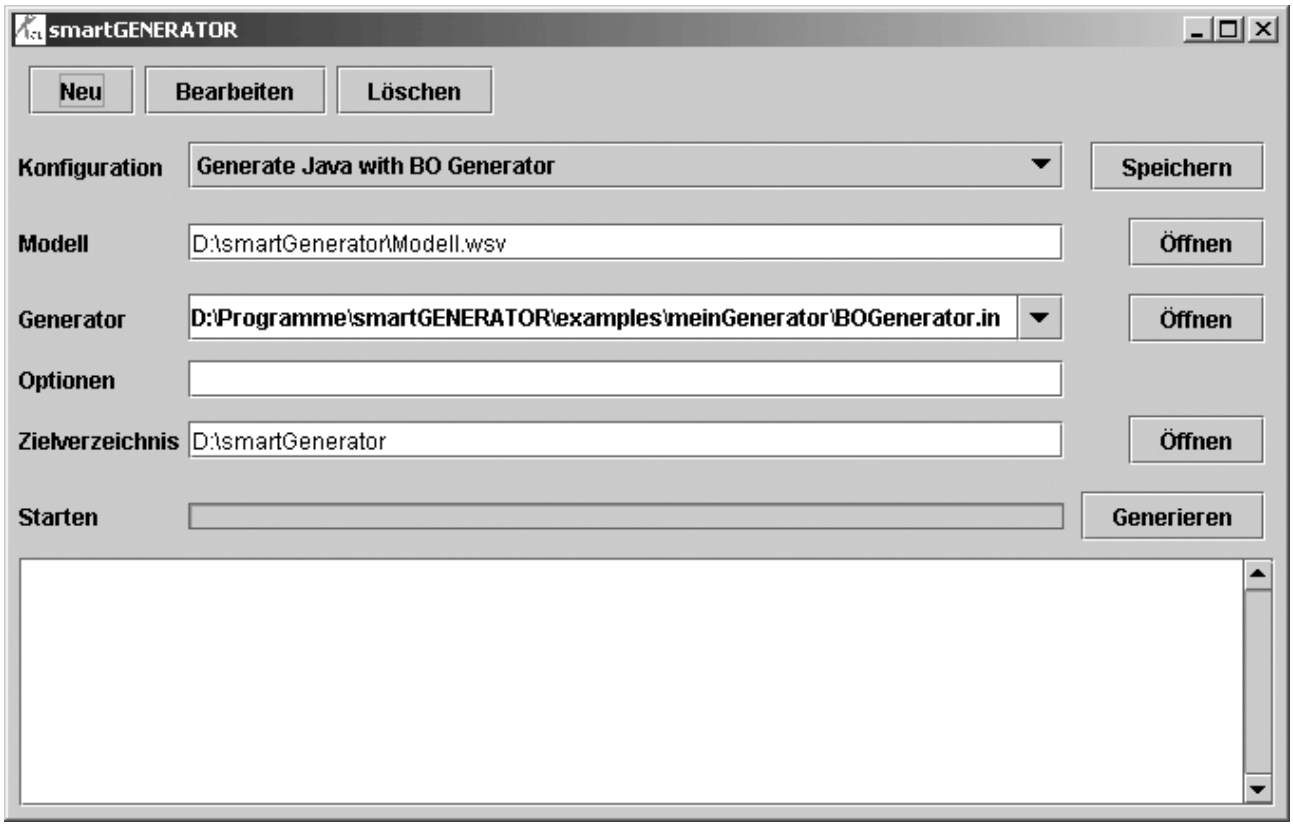


Abbildung 37: Oberfläche smartGenerator

Generierung (C-GEN)

Die UML-Modelle lassen sich in den unterstützten Modellierungswerkzeugen[114] über eine smartGenerator-spezifische Schaltfläche exportieren. Diese Schaltfläche wird bei der Installation von smartGenerator automatisch in die Menüs des bereits installierten UML-Tools eingefügt. Vor der Generierung wird der Pfad des exportierten XMI-Dokumentes angegeben, der Generatorlauf wird über die entsprechende Schalfläche gestartet. In der Konsole werden während der Generierung aktuelle Informationen zum Fortschritt der Erstellung angezeigt. Der generierte Code wird in einer Ordnerstruktur, welche die Packages

[114] Rational Rose, microTOOL objectiF, ObjectDomain, Poseidon oder ArgoUML.

repräsentiert, abgelegt und kann in beliebigen Entwicklungsumgebungen oder Editoren bearbeitet werden.
Die generierten Anwendungen sind, zwar noch ohne Funktionalität, prinzipiell compilierbar und lauffähig.

Erstellen und Modifizieren von Cartridges (C-GEN-IDE)

Auch BITPlan verwendet für seine Cartridges den template-basierten Ansatz. Die erstellten Templates werden im Format *.in gespeichert und können in dieser Form als Cartridge verwendet werden, sie werden von smartGenerator automatisch in Java-Dateien übersetzt. Durch den zweistufigen Ansatz („Generator-Generator“) muss der Benutzer lediglich die Logik der Cartridge programmieren, die Software ergänzt beim Übersetzen in die Java-Datei den Rahmen. So kann der Entwickler Standardaufgaben wie z.B. das Öffnen einer Datei oder Ausgabebefehle generieren lassen.
Das Bearbeiten der Transformationsregeln erfolgt nach Empfehlung von BITPlan im Shareware-Editor UltraEdit. Dort können die in Java geschriebenen Templates modifiziert bzw. eigene Cartridges erstellt werden.
Bei der Erstellung von Templates gelten folgende aus [SCHW03] entnommene Regeln:

- *Zeilen, die mit `#META, #TYPE, #for, #if` beginnen, sind Anweisungen, die von smartGENERATOR selbst ausgeführt bzw. ausgewertet werden.*
- *Insbesondere sind Zeilen mit `#.` Kommentare.*
- *Ausdrücke in spitzen Klammern `<...>` sind Variablen, die von smartGENERATOR ersetzt werden, z.B. durch einen Klassennamen.*
- *Zeilen, die mit einem Punkt beginnen, werden genauso in den Java-Generator übernommen, nur ohne den Punkt [..].*

  Beispiel:

  `.System.out.println(xy);` in der Generatorvorlage wird zu

  `System.out.println(xy);` im Java-Generator.
- *Alle anderen Zeilen werden direkt ins Generat (d.h. in die generierten Java-Dateien) übernommen[..].*
- *Im Java-Generator beginnen diese Zeilen mit `currentOutput().println...`*

BITPlan liefert auf Verhandlungsbasis einige Standard-Cartridges zusammen mit dem Werkzeug aus, namentlich Java-, COBOL-, Delphi-, C++-, ObjectiveC- und C#-Cartridges sowie einen Testfall-Generator für JUnit-Tests[115].

2.3.4 Andere

Es werden auf dem Markt eine Reihe anderer Werkzeuge angeboten. Hier sollen beispielhaft einige weitere Namen genannt werden. Eine umfangreiche, mit ArcStyler und OptimalJ vergleichbare Lösung ist „Model-In-Action“ der Firma Sodifrance. Unter den kleineren Tools ist bspw. IQGen der Firma Innoq zu nennen, welches durch die Verwendung von JSP[116] eine ähnlich komfortable Erstellung von Cartridges ermöglicht wie smartGenerator, jedoch nicht 100% MDA konform ist. Während die vorliegende Arbeit schon in Bearbeitung war, brachte IBM ihren Rational Rapid Developer heraus, ein recht umfangreiches Werkzeug, das auch den kompletten Lebenszyklus der Software unterstützt.
Neben den kommerziellen Werkzeugen existiert eine open-source-Initiative, die das Werkzeug AndroMDA[117] (gesprochen: Andromeda) entwickelt, welches in seiner Funktionalität auch zu den kleineren Werkzeugen gezählt wird, der Funktionsumfang ist mit dem von smartGenerator vergleichbar.
In diesem Kapitel sowie in Kapitel 3.3-3.5 und in Kapitel 4 wird deutlich, wie unterschiedlich Werkzeuge für die gleiche Technologie ausfallen können, und wie verschieden Stärken und Schwächen verteilt sind. Um so mehr Sorgfalt ist bei der Auswahl des Werkzeuges für die eigene Entwicklung gefragt.

[115] Aus [BITPSK].
[116] Vgl. [INNO02b].
[117] Mehr Informationen dazu unter www.andromda.org.

2.4 Basistechnologien der Management-Architektur

In Kapitel 3 wird das in [ZACH02] vorgestellte Managementsystem in leicht abgewandelter Form mit MDA umgesetzt. Dieses System verwendet neben der Programmiersprache Java und den darin fest verankerten Konzepten die Technologien JMX und J/XFS, die im Folgenden kurz vorgestellt werden.

2.4.1 Java Management Extensions

Java Management Extensions (JMX) ist ein Standard, der ein Framework für das Management verteilter Applikationen im Bereich J2EE definiert. [THOM03] fasst zusammen:

> *"JMX makes it possible to manage and monitor applications using a choice of management systems and consoles […]. It also simplifies the task of making applications manageable, i.e. management instrumentation, and enables improved management of deployed applications in the enterprise."*

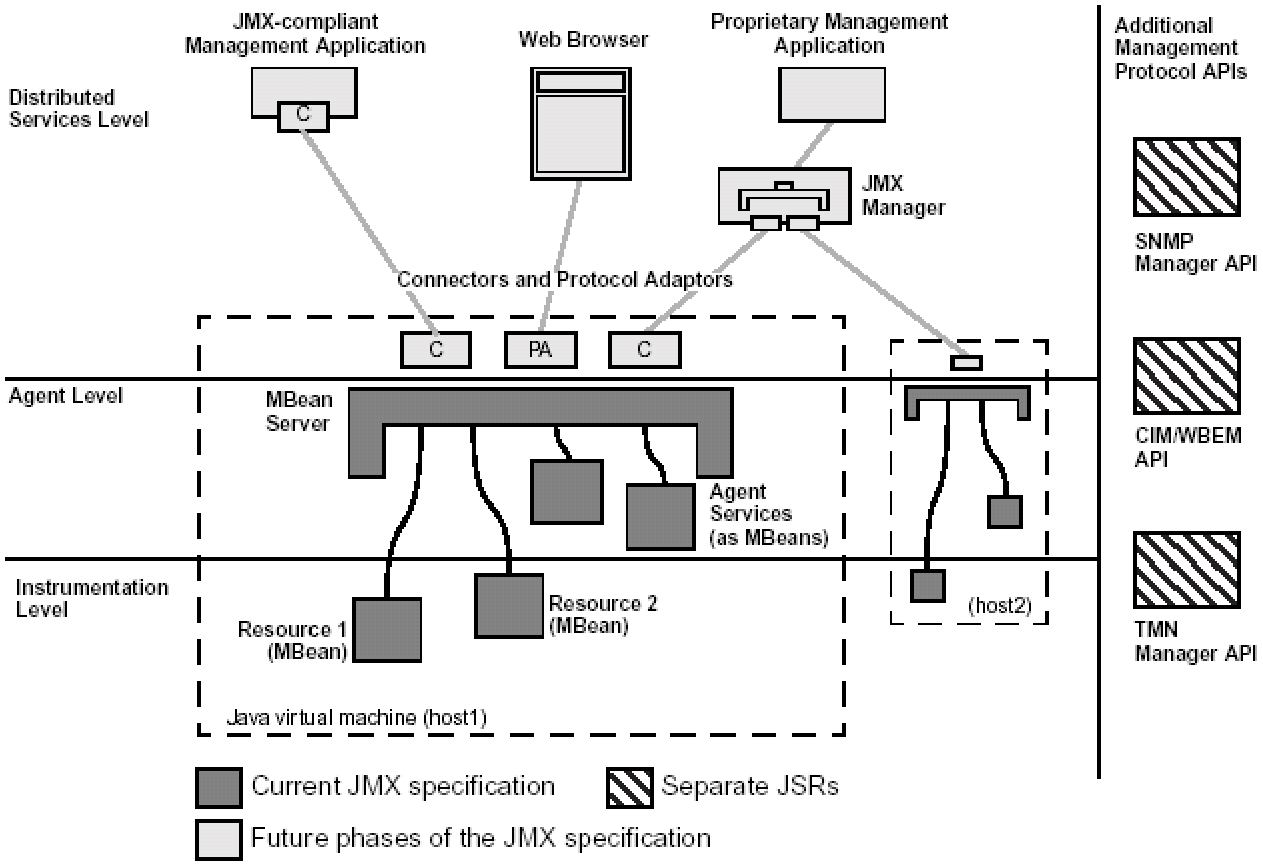


Abbildung 38: Layer des JMX-Standards[118]

[118] Quelle: [SUN 00], S. 21.

Das JMX-Framework basiert auf einem Manager- und Agent-Model, welches in drei Schichten gegliedert ist: *Instrumentation Level, Agent Level* und *Distributed Services Level*[119] (siehe Abbildung 38).

- Instrumentation Level

  In dieser Schicht wird die Funktionalität der Anwendung in Form einer oder mehrerer *Managed Beans* (MBeans) offengelegt. Eine MBean ist ein beliebiges Java-Objekt, welches so strukturiert ist, dass es der JMX Spezifikation entspricht. [ZACH02] fasst die Anforderungen an das Objekt zusammen:

  - *Es muss eine konkrete und öffentliche Klasse sein.*
  - *Es muss einen öffentlichen (public) Konstruktor besitzen.*
  - *Es muss der Java Beans Spezifikation genügen.*
  - *Es muss ein korrespondierendes MBean Interface bzw. das FBean Interface implementieren.*
  - *Es kann das Notification Broadcaster Interface implementieren, wenn es Ereignisse an andere Komponenten melden soll.*

  Diese Eigenschaften stellen sicher, dass die Ressource bzw. Funktionalität durch das Agent-Level verwaltet werden kann, wobei eine MBean keine Informationen über den verwaltenden Agenten besitzt. Es werden vier Arten von MBeans unterschieden: *standard, dynamic, open* und *model*. Eine *standard MBean* besitzt ein statisches Interface, wohingegen eine *dynamic MBean* ihr Interface während der Laufzeit mit Hilfe von Metadaten an den Agenten übergibt. Eine *open MBean* gleicht einer dynamic MBean, bietet jedoch besseres Laufzeitverhalten durch die Verwendung von vordefinierten Java-Datentypen, was aber gleichzeitig den Funktionsumfang leicht einschränkt. Eine *model MBean* ist eine von der dynamischen MBean abgeleitete generische Bean, die als Template verwendbar ist, um dem Aufwand zu entgehen, eine eigene Bean-Klasse zu definieren.

- Agent Level

  In dieser Schicht sind die Agenten angesiedelt, die die im vorigen Abschnitt beschriebenen MBeans verwalten und (meist) entfernten Management-Applikationen zur Verfügung stellen. Ein JMX Agent besteht aus einem MBean-Server und einer Reihe von Services, die bei dem Umgang mit den MBeans benötigt werden. Er stellt der entfernten Anwendung die Dienste aller bei ihm registrierten MBeans zur Verfügung, ohne jedoch selbst Informationen über diese Anwendung zu besitzen. Der

[119] Aus [SUN 00], S. 17.

Agent benötigt einen Adapter oder Konnektor, welcher für die Kommunikation mit dem restlichen System verantwortlich ist (siehe *Distributed Services Level*).

- Distributed Services Level
  Diese Schicht definiert Schnittstellen und Komponenten, die von der Managementanwendung (oder einem evtl. vorhandenen Manager) verwendet werden, um mit dem Agenten zu kommunizieren. Dies geschieht in Form von Adaptern und Konnektoren.

Wie bereits erwähnt, stellt JMX eine Reihe von Standard-Diensten zur Verfügung, welche im Umgang mit den MBeans benötigt werden. Folgende Dienste gibt es[120]:

- Dynamic class loading: Der M-let-Service (management applet service) dient dem dynamischen Laden und Instanziieren von MBeans über ein Netzwerk.
- Monitoring: Durch den Monitoring-Service können Attribute der MBeans überwacht und bei einer Änderung andere Objekte informiert werden.
- Timer: Der Timer sendet einmalige oder wiederkehrende Notifications, um so zeitlich festgelegte Aktionen bei MBeans zu steuern.
- Relation Service: Dieser Service definiert, pflegt und löst Beziehungen zwischen den MBeans. Er überwacht die Beziehungen in Anlehnung an vordefinierte Relationen und benachrichtigt bei Bedarf registrierte Ereignissenken[121].

In ihren grundlegenden Eigenschaften lassen sich MBeans mit EJBs vergleichen, jedoch sind sie einfacher strukturiert, besitzen keine Eigenschaften wie Transaktionen oder Sicherheit, was ihre Verwendung weniger komplex macht. Für die in Kapitel 3 vorgestellte Management-Anwendung ist JMX mit Managed Beans ideal: Die benötigten Grundeigenschaften und -Dienste stehen zur Verfügung, die Beans sind schlank und leicht einzusetzen.

[120] Aus [SUN 00], S. 28.
[121] In Anlehnung an [ZACH02], S. 110f..

2.4.2 J/eXtensions for Financial Services for the Java™ Platform

Der 1998 definierte[122] Standard J/eXtensions for Financial Services (J/XFS) for the Java™ Platform bietet plattformunabhängige Java-Schnittstellen für die in der Finanzwirtschaft verwendeten Bankperipheriegerätegeräte. Zu diesem Zweck kooperieren die Unternehmen DeLaRue, IBM, NCR, Wincor Nixdorf und Sun Microsystems. Die Version J/XFS 2.1, auf der das in [ZACH02] beschriebene Managementsystem basiert, wurde 2001 verabschiedet[123]. Aktuell ist die 2003 im CEN[124] Workshop Agreement veröffentlichte Version 2.1.1.

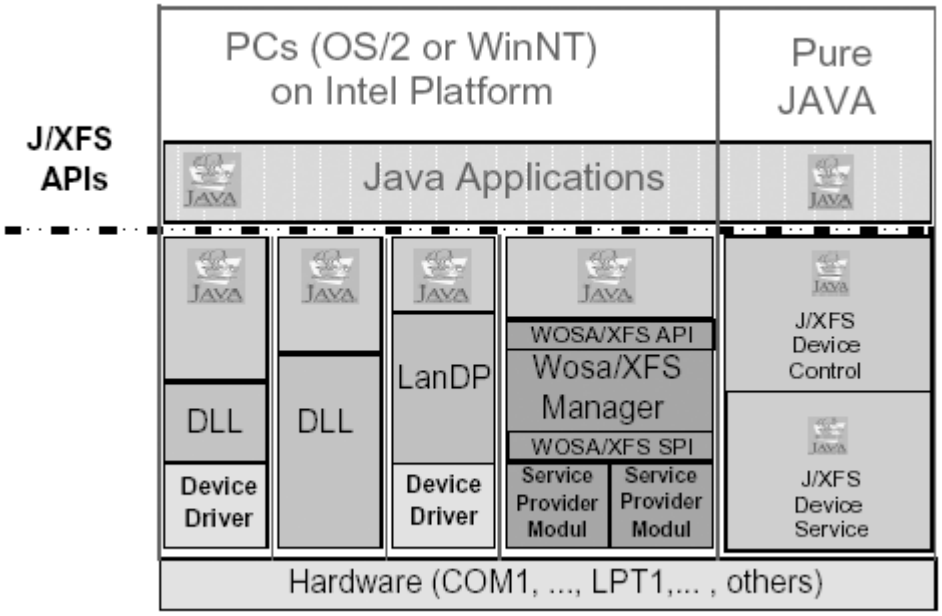


Abbildung 39: Kompatibilität mit älteren Technologien durch Wrapping[125]

Das Ziel bei der Definition des Standards war, eine Unabhängigkeit von Anwendungen zu der herstellerspezifischen Hardware zu schaffen. Zu diesem Zweck wurde eine mehrschichtige Architektur definiert, die Standardschnittstellen für folgende Gerätegruppen bietet[126]:

- Drucker (Kontoauszug, Quittung, Sparbuch etc.)
- Scanner
- Geldautomat/Geldannahmeautomat/Terminal

---

[122] Vgl. [ZACH02], S. 117.
[123] Aus [JXFSWW].
[124] CEN steht für „Comité Européen de Normalisation“, auf deutsch: „Europäisches Komitee für Normung"
[125] Quelle: [JXFS03a].
[126] Vgl. [JXFSDOC], S. 17.

- Tastenfelder
- Magnet- und Chipkartenlesegeräte
- Textein- und -ausgabegeräte
- Alarmanlagen
- Depoteinheit
- Scheck-Lesegeräte und -Scanner
- Sensoren und externe Kontakte
- Kameras

Um eine Kompatibilität mit älteren Technologien wie WOSA/XFS[127] zu erreichen, wurde neben der reinen Java-Lösung (siehe Abbildung 39 rechts) eine wrapping-Lösung (siehe Abbildung 39 links) von J/XFS erstellt. In ihr werden die unteren Schichten der Architektur durch die Kapselung der älteren Technologien simuliert.

Die Architektur besteht aus folgenden Schichten:

- J/XFS Device Manager
- J/XFS Device Control
- J/XFS Device Communication (optional)
- J/XFS Device Services

Abbildung 40 gibt einen Überblick über die Architektur. Die Anwendung auf dem Server greift per TCP/IP sowie HTTP, RMI oder IIOP über feste Schnittstellen auf den J/XFS-Kernel zu. In diesem werden die ersten drei Schichten von der Implementierung des J/XFS-Standards zur Verfügung gestellt. Die J/XFS-konformen Device Services werden vom Hersteller des Bankperipheriegerätes implementiert. Zentrales Element der Architektur ist der *J/XFS-Device Manager*. Er organisiert und koordiniert die Lokalisierung und den Zugriff auf die Geräte. Innerhalb einer Java VM existiert immer genau ein Device Manager. Seine Aufgaben sind:

- Listen der Geräte, Dienste und Kommunikation führen
- Controls, Kommunikation und Dienste auf eine transparente Art verbinden
- Zentral konfigurierte Daten abfragen
- Controls und Dienste davon abhalten, nicht standardisierte Java-Elemente zu verwenden
- Programmierung von Controls und Diensten vereinfachen

[127] Windows Open Services Architecture Extensions for Financial Services.

Der Device Manager führt eine Liste darüber, für welche Geräte bereits ein Service-Objekt existiert. Existiert noch kein Service-Objekt, so wird eines erzeugt, andernfalls wird das bereits existierende Objekt zurückgegeben. Der Device Manager kapselt alle anderen J/XFS-Komponenten.

Die J/XFS *Device Control* unterstützt die Anwendung bei der Verwendung der Geräteschnittstelle. Diese Schnittstelle spezifiziert für jede generische Gruppe von Geräten eine Menge an Eigenschaften, Methoden und Ereignissen. Diese Elemente werden vom J/XFS-Standard dergestalt umgesetzt, dass sie den äquivalenten Entitäten des Java-Komponentenmodells entsprechen, konkret sind dies Java Beans. Die Device Control trennt die Anwendung von der Device Service Software, welche eher an das Gerät als an die Anwendung gebunden ist und unmittelbar auf der Hardware arbeitet.

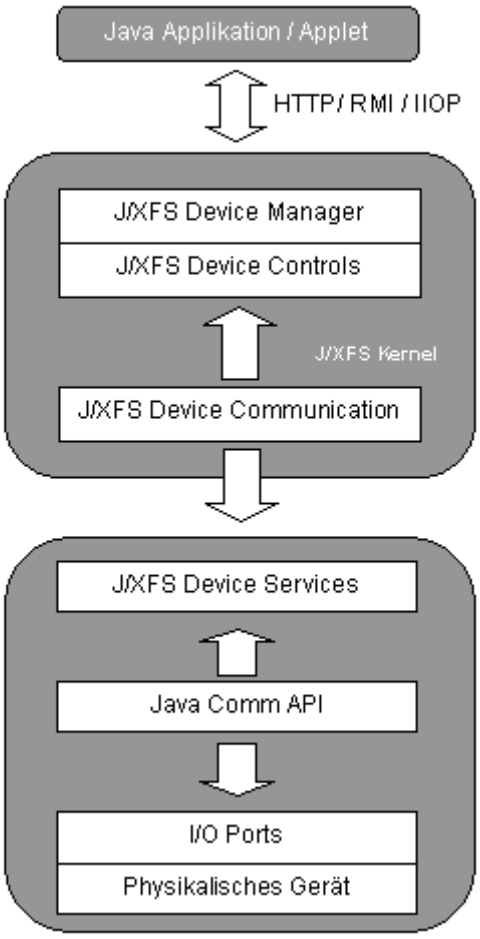


Abbildung 40: Die J/XFS Architektur[128]

Die *Device Communication* ist eine optionale, transparente Schicht zwischen Device Control und Device Service. Diese Schicht dient dem Zweck, die doppelte Nutzung von Geräten zu koordinieren sowie Netzwerkkommunikation zu kapseln Sie stellt der nächsthöheren Schicht eine

eine Device Service-ähnliche API und der nächstniedrigeren Schicht eine Control-ähnliche API zur Verfügung. Neben diesen Schichten und den in ihnen implementierten Funktionalitäten stellt J/XFS einen Logger und ein Repository zur Verfügung[129].

In [JXFS03b] werden sechs Vorteile hervorgehoben, die J/XFS dem Entwickler bietet:

- Vorteile der Sprache Java

  Durch den Einsatz von Java können Mechanismen wie Garbage Collection und Multithreading verwendet werden. Java-Anwendungen sind erfahrungsgemäß stabi-

[128] In enger Anlehnung an [ZACH02], S. 123.
[129] Vgl. [ZACH02], S. 125.

ler als in anderen Sprachen erstellte Applikationen, was ein wichtiges Kriterium für Unternehmen der Finanzwirtschaft ist.

- Geringere Konfigurationskosten für Clients
  Java-Anwendungen benötigen für ihre Ausführung lediglich eine Java Virtual Machine (JVM) auf dem System. So kann die Hardwarekonfiguration der Clients soweit verringert werden, das auf ihnen eine JVM ausgeführt werden kann (thin client).
- Geringere Administrationskosten für Clients
  Da Java den Einsatz von thin clients ermöglicht, kann beim Einsatz von Applets oder ladbaren Applikationen der gesamte Programmcode auf dem Server abgelegt werden. Neue Software muss lediglich einmal auf dem Server installiert werden und wird automatisch beim nächsten Booten auf den Client übertragen. Die dadurch eingesparten Aufwände machen sich vor allem bei Systemen mit vielen Clients bemerkbar.
- Geräteunabhängigkeit
  Durch die Abstraktion der herstellerspezifischen Geräte auf einfache Stereotypen wie Kartenleser, Geldautomat etc. sind J/XFS-konforme Applikationen unabhängig von der darunterliegenden Hardware. Folglich kann diese Hardware beliebig aufgerüstet oder ausgetauscht werden ohne die Funktionalität der Software zu beeinflussen.
- Ortsunabhängigkeit
  Die Kommunikationsschicht des J/XFS-Standards stellt den Zugriff auf Geräte sicher, egal ob sie lokal oder entfernt angeschlossen sind. Dies erlaubt eine beliebige physikalische Verteilung der Geräte und den Zugriff von mehreren Geräten auf eine zentrale Ressource wie z.B. einen Dokumentenscanner. Diese Dienste werden automatisch von der J/XFS Implementierung bereitgestellt, so dass der Programmierer keine Verwaltung mehr implementieren muss.
- Plattformunabhängigkeit
  Die JVM als Plattform für J/XFS-Dienste kann in einem Browser, einem Betriebssystem oder einem Chip eingebettet sein. So ist die Basis der J/XFS-Technologie unabhängig von jeglicher Hardware und kann auf beliebigen Systemen (Hard- und Software) laufen.

3 Umsetzung einer Managementarchitektur mit MDA

3.1 Allgemeines

Zur Evaluierung der Möglichkeiten von MDA in der Praxis soll prototypisch ein bestehendes Management-System modellbasiert umgesetzt werden. Zu diesem Zweck wurde das in [ZACH02] beschriebene und umgesetzte Management-System für Bankperipheriegeräte ausgewählt.

3.2 Abgrenzung und Modellierung des Systems

Im Folgenden wird nicht das vollständige System betrachtet, sondern nur die unteren Schichten. Die Anbindung über SOAP wird in dieser Arbeit nicht behandelt, nur der in Abbildung 41 hell hervorgehobene Bereich des Systems wird umgesetzt. Die Umsetzung konzentriert sich vor allem auf die Technologie JMX (siehe Kapitel 2.4.1) und die Möglichkeiten die MDA in diesem Zusammenhang bietet. Zur Anbindung der Bankperipheriegeräte wird J/XFS (siehe Kapitel 2.4.2) verwendet. Dies ist allerdings im Zusammenhang mit MDA nicht von besonderer Bedeutung, da sich daraus keine besonderen Merkmale in der Struktur des Managementsystems ergeben.
Diese Einschränkung des Systems erfordert eine Anpassung der Benutzer-Schnittstellen. Die Oberfläche für Endbenutzer kann nicht mehr umgesetzt werden. Daher wird prototypisch eine neue GUI für das Überwachungspersonal implementiert.
Für ausführliche Informationen über das Management-System empfiehlt sich die Lektüre von [ZACH02]. Im Folgenden wird ein kurzer Überblick über die Use Cases und das Klassendiagramm gegeben.

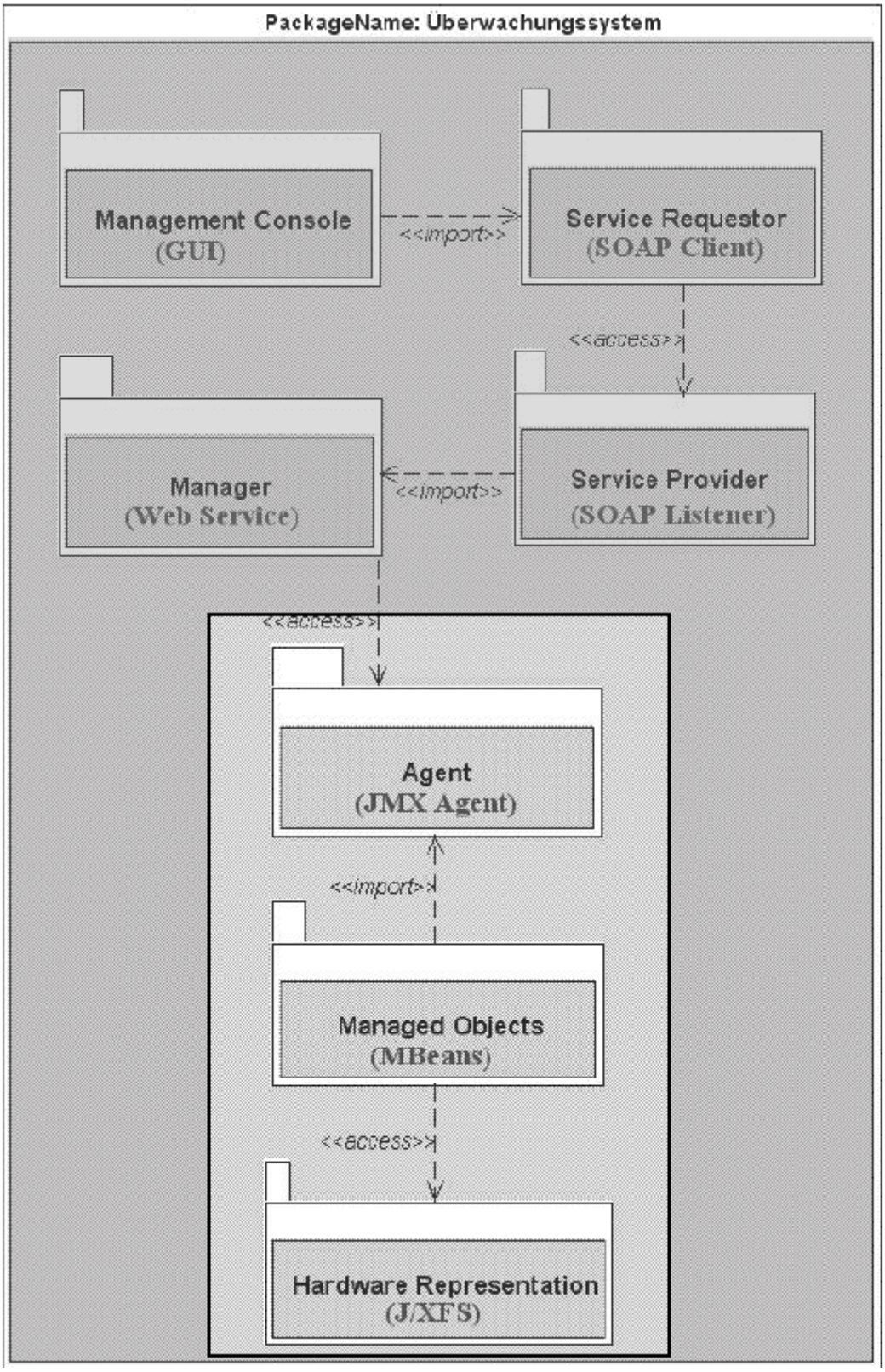


Abbildung 41: Umsetzung des Managementsystems[130]

[130] Quelle: [ZACH02], S. 173.

3.2.1 Use Case Diagramm

Bei der prototypischen Umsetzung des Managementsystems werden die drei Use Cases betrachtet, die in [ZACH02] beschrieben werden. Diese sind in Abbildung 42 im Use Case Diagramm abgebildet. Die genauen Spezifikationen der einzelnen Use Cases sind für die vorliegende Arbeit nicht relevant und können bei Interesse in [ZACH02], S. 141ff. nachgelesen werden. Im Folgenden werden die Use Cases nur kurz vorgestellt.

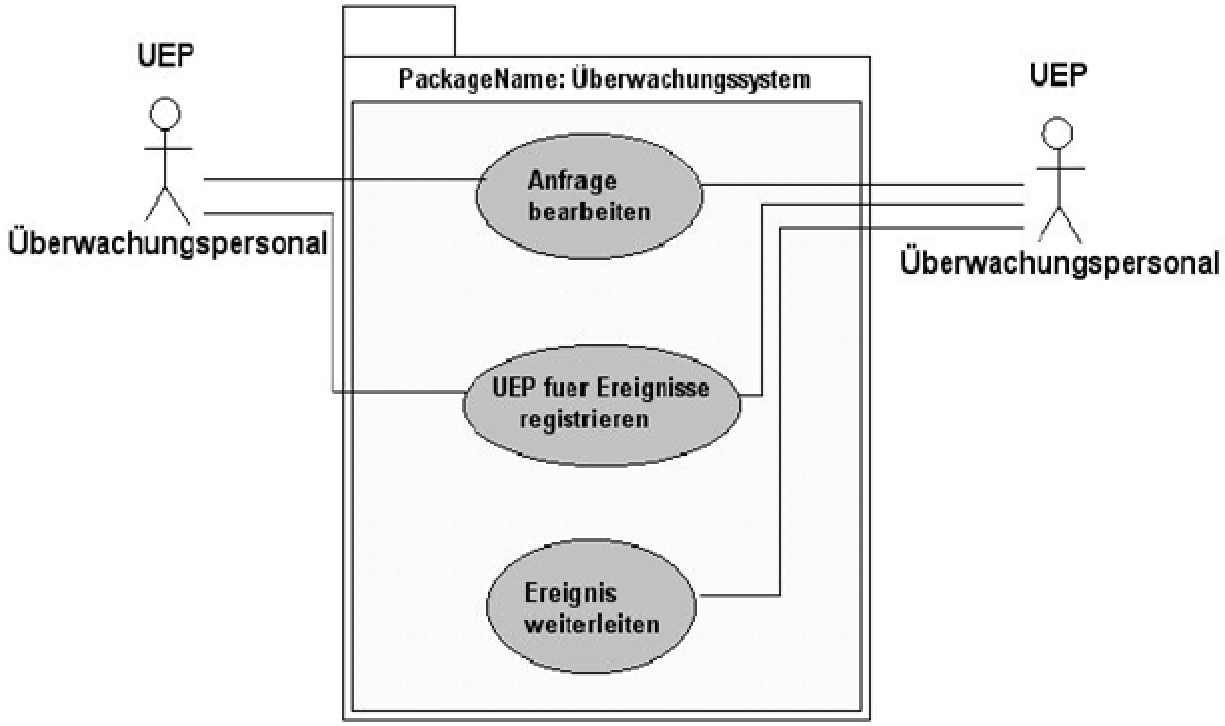


Abbildung 42: Use Case Diagramm

Use Case Nr. 1: „Anfrage bearbeiten“

In diesem Use Case stellt das Überwachungspersonal eine Anfrage an das System. Die für diese Arbeit realisierte Version des Systems bietet die Möglichkeit, ein Gerät auszuwählen.

Use Case Nr. 2: „Für Ereignisse registrieren“

Das Überwachungspersonal kann sich für Ereignisse, die bei den Geräten auftreten, registrieren.

Use Case Nr. 3: „Ereignis weiterleiten“

Aufgetretene Ereignisse (die in der für diese Arbeit realisierten Version des Systems in Ermangelung eines Benutzers zufällig generiert werden) werden automatisch an das Überwachungspersonal weitergeleitet.

3.2.2 Klassendiagramm

Für die Darstellung der Klassenstruktur des Systems wurde ein Klassendiagramm in Rational Rose erstellt[131] (siehe Abbildung 43). Das System besteht aus drei Schichten: *GUI*, *JMX* und *J/XFS*.

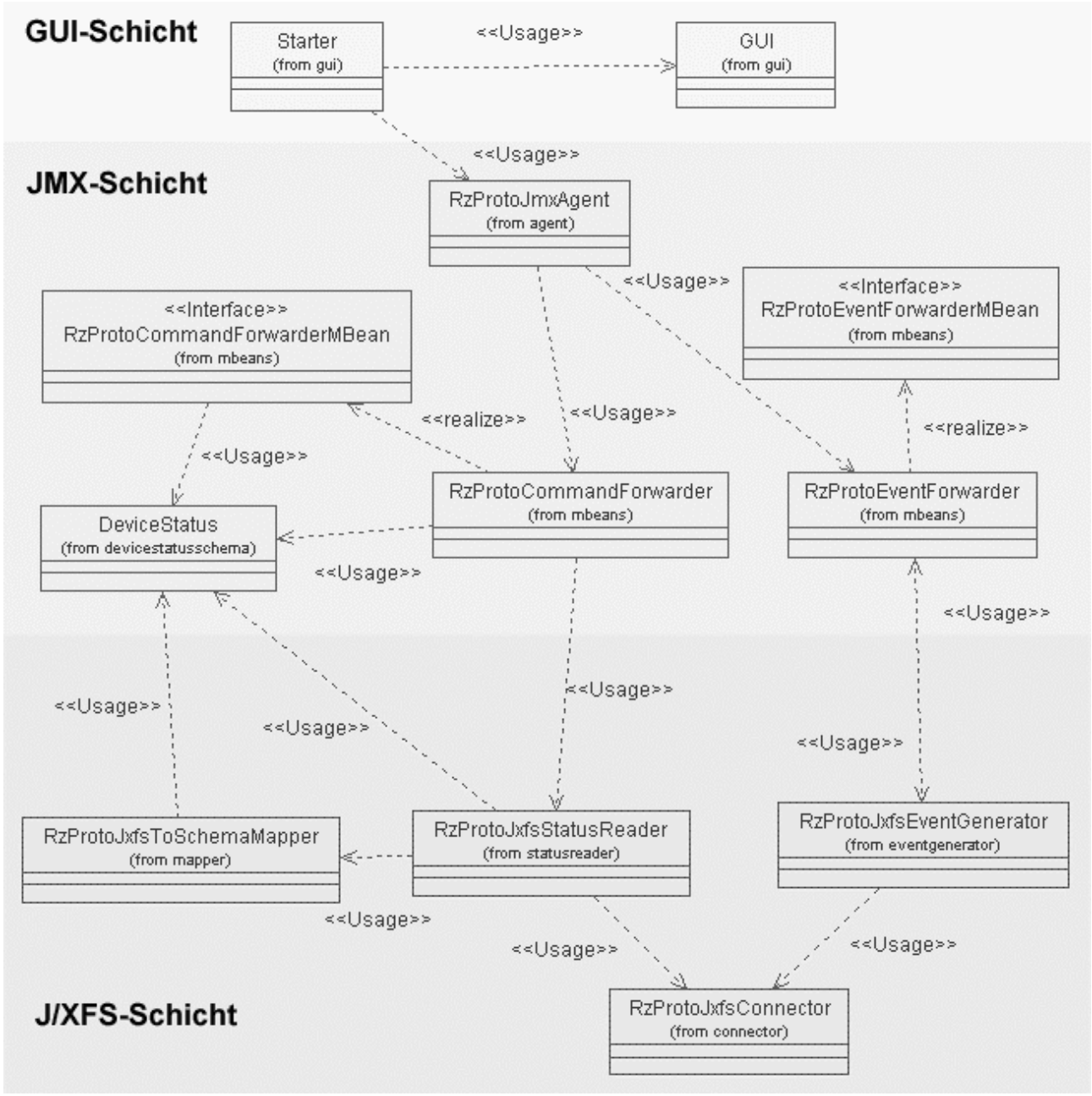


Abbildung 43: Klassendiagramm

[131] Zu diesem Zweck wurde Harvester, das Reverse-Engineering-Modul von ArcStyler verwendet. Es wäre aber auch möglich, das Modell von Hand oder mit einem beliebigen anderen Reverse-Engineering-Werkzeug zu erstellen. Die genaue Beschreibung eine Importvorganges mit Harvester ist in Kapitel 3.3.1 zu finden.

Die direkte Anbindung an die Bankperipheriegeräte wird in der als Singleton umgesetzten Klasse RzProtoJxfsConnector realisiert. Diese wird von den Klassen RzProtoJxfsEvent-Generator und RzProtoJxfsStatusReader verwendet. Der RzProtoJxfsEventGenerator ist für das Verwalten von Events verantwortlich, der RzProtoJxfsStatusReader ruft Informationen von den Geräten ab. Die Klasse RzProtoJxfsSchemaMapper setzt den Status eines Gerätes (dieser wird von J/XFS-Diensten geliefert) um, so dass er in der JMX Schicht verwendet werden kann. Das Format des Status ist in der Klasse DeviceStatus implementiert. In der JMX-Schicht werden diese Dienste durch die Managed Beans RzProtoCommandForwarder und RzProtoEventForwarder repräsentiert, welche in einem MBean-Server verwaltet werden. Kontrolliert und angesprochen wird der MBean-Server von der Klasse RzProtoJMXAgent. Für das Starten dieses Agenten und der GUI ist die Klasse Starter verantwortlich. Sie initialisiert das gesamte System.
Das vorgestellte Modell ist bereits recht spezifisch und entweder als sehr detailliertes PIM oder als grobes PSM einzuordnen.

3.3 Umsetzung mit ArcStyler

Aufgrund des bereits bestehenden, vorgegebenen Systems kann auf die Analyse und Modellierung der Geschäftsvorfälle mit den Modulen C-BOM und C-RAS verzichtet werden. Diese finden (wie in Kapitel 2.2.4 beschrieben) ihre Anwendung beim Analysieren der Systemanforderungen in Zusammenarbeit mit den Experten aus den Fachabteilungen, was in diesem Fall nicht mehr notwendig ist.
Es gibt zwei Ansätze, um das bereits bestehende System umzusetzen. Zum einen kann das Klassendiagramm im C-REF-Modul neu modelliert werden, so dass das ganze System neu eingegeben wird. Zum anderen bietet sich als arbeitssparende Alternative die Verwendung des ArcStyler-Harvesters an. Dieses Reverse-Engineering-Werkzeug ermöglicht den Import eines bestehenden Java-Systems, indem es den in einem Archiv oder einer Verzeichnisstruktur vorliegenden Quellcode in ein Klassendiagramm umwandelt. Hier wird aus Effektivitätsgründen und um die Funktionalität des Werkzeuges zu testen die Alternative gewählt.

3.3.1 Import des Systems unter Verwendung von Harvester

Das Reverse-Engineering-Werkzeug Harvester (von englisch „to harvest“[132]) bietet die komfortable Möglichkeit, das Klassendiagramm des bestehenden Systems zu importieren. Allerdings beschränkt sich diese Funktionalität auf Java-Anwendungen[133]. Das Importieren von Anwendungen in anderen Programmiersprachen ist nicht möglich. Zu diesem Zweck könnten Reverse-Engineering-Werkzeuge anderer Hersteller eingesetzt werden.
Beim Importieren wird folgendermaßen vorgegangen: Über das Kontextmenü in der Explorerleiste links wird Harvester mit Klick auf *ArcStyler -> MDA Harvesting* gestartet. Es stehen die drei Optionen *Class by Class*, *All Classes At Once* und *PA Merge* zur Auswahl. Auf letztere Option wird später eingegangen. Wählt man die zweite Option, werden alle Klassen aus dem Quellcode in einem Arbeitsgang in das Modell importiert. Option eins ermöglicht, vor dem Import jeder einzelnen Klasse zwischen Harvester und Modell in Rose umzuschalten und die bisherigen Ergebnisse zu betrachten bzw. Details des Imports zu steuern[134]. Da noch kein Modell existiert, soll der Code ohne Rücksicht auf Differenzen zum Modell in Rose importiert werden, es wird die Option *All Classes at Once* gewählt.

[132] Zu deutsch: „ernten“ oder „abernten“.
[133] Siehe [IOSW03e], S. 309.
[134] Vgl. [IOSW03e], S. 311.

Harvester arbeitet in diesem Modus laut [IOSW03e], S. 311 doppelt so schnell wie im Modus *Class by Class*. Im folgenden Dialog wird ein Abstraktionslevel für den Import ausgewählt. (siehe Abbildung 44).

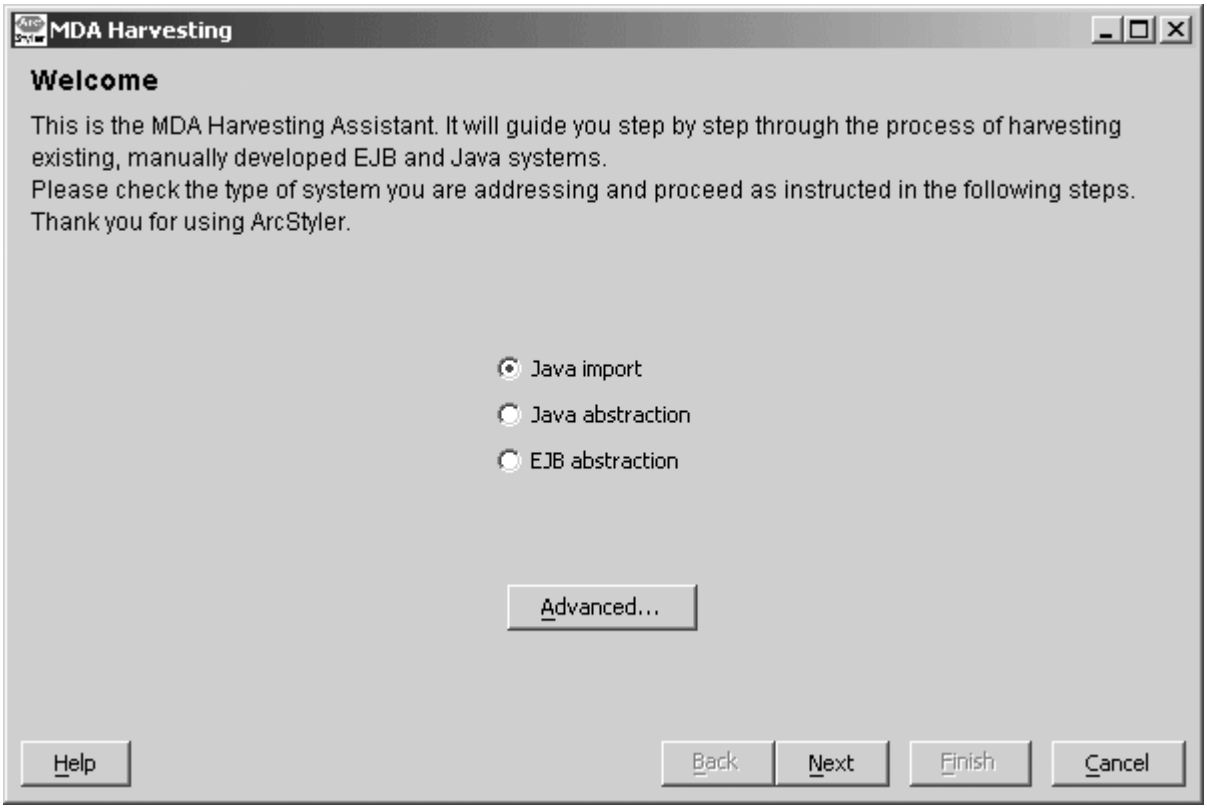


Abbildung 44: Wählen des Abstraktionslevels in Harvester

Java import bietet einen 1:1 Import der Java-Klassen, wohingegen *Java Abstraction* Variablen und deren Getter/Setter mit einer gewissen Logik behandelt[135]. Das dritte Abstraktionslevel ist *EJB abstraction*. Dieser Modus ermöglicht, EJB-spezifische Eigenschaften in das Modell zu übernehmen. Unter dem Menüpunkt *Advanced* können zusätzliche Einstellungen bzgl. der Umsetzung von bestimmten Codeelementen vorgenommen werden, für weitere Informationen wird auf [IOSW03e], S. 312 verwiesen. In der Beispielanwendung wird der Modus *Java import* gewählt, damit das System möglichst unverändert in das Modell übernommen wird.

Im nächsten Dialog (siehe Abbildung 45) werden auf der rechten Seite neben den zu importierenden Elementen alle Klassen und Bibliotheken aus der Verzeichnisstruktur gewählt, welche im Klassenpfad der Anwendung stehen. Mit den horizontalen Pfeilen werden die Komponenten, die importiert werden sollen (Bibliotheken werden nie importiert, nur der Quellcode der Anwendung) in das linke Fenster bewegt. Mittels der vertikalen Pfeil-Buttons ist es möglich, Elemente nach oben oder nach unten zu bewegen und so die Rei-

[135] Für genauere Informationen siehe [IOSW03e], S. 24ff..

henfolge der Verwendung bzw. des Imports zu bestimmen. Im Feld *Physical Component* kann optional ein Name spezifiziert werden, welcher das importierte Element im C-REF „physisch“ repräsentiert[136].

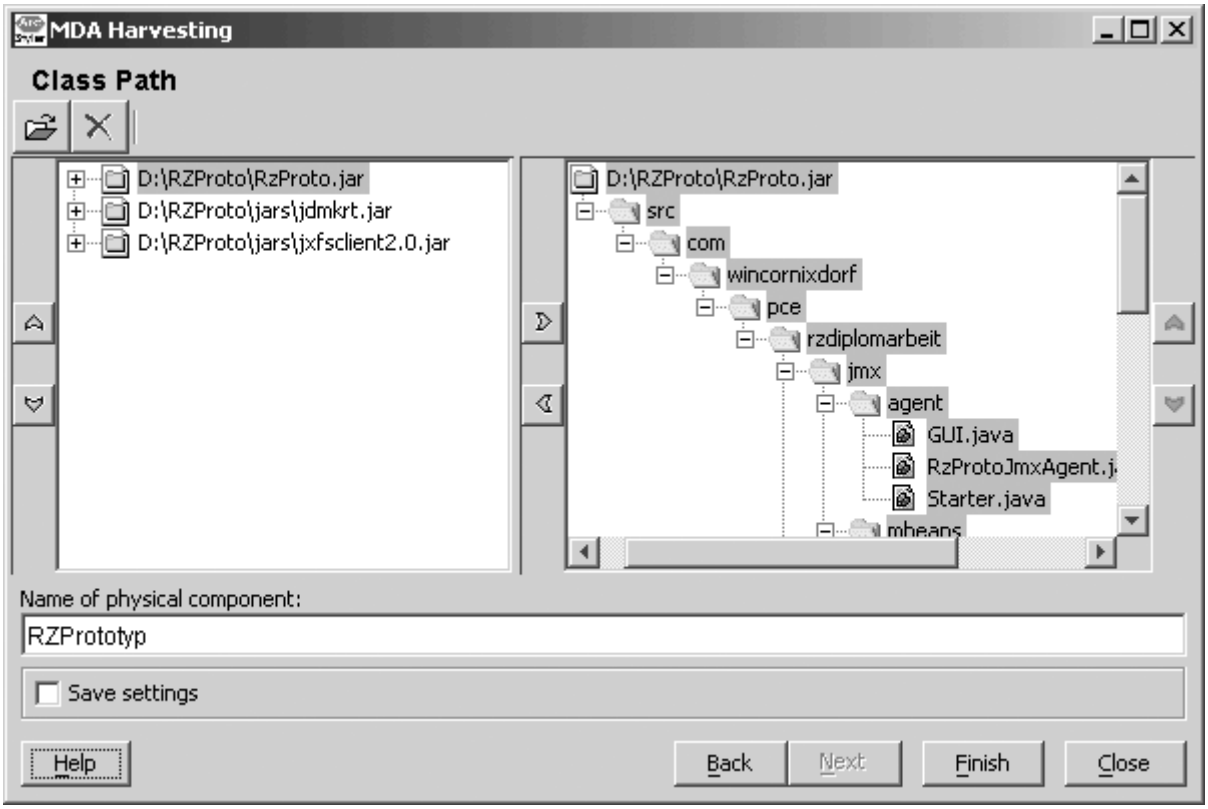


Abbildung 45: Importieren einer bestehenden Klassenstruktur

Ein Klick auf *Finish* startet den Import, welcher im Falle des Beispieles ca. eine halbe Minute dauert. Während des Importierens werden Aktionen und der aktuelle Status in einer Konsole ausgegeben.

Ist der Import beendet, so ist das komplette System in Rose als Klassendiagramm modelliert wobei verwendete Bibliotheksklassen als leere Klassen dargestellt werden. Abbildung 46 zeigt die Baumstruktur der importierten Klassen. Diese sind in entsprechenden Packages hierarchisch gegliedert, auch die verwendeten Bibliotheksklassen sind in diese Hierarchie eingefügt.

[136] Vgl. [IOSW03e], S. 315.

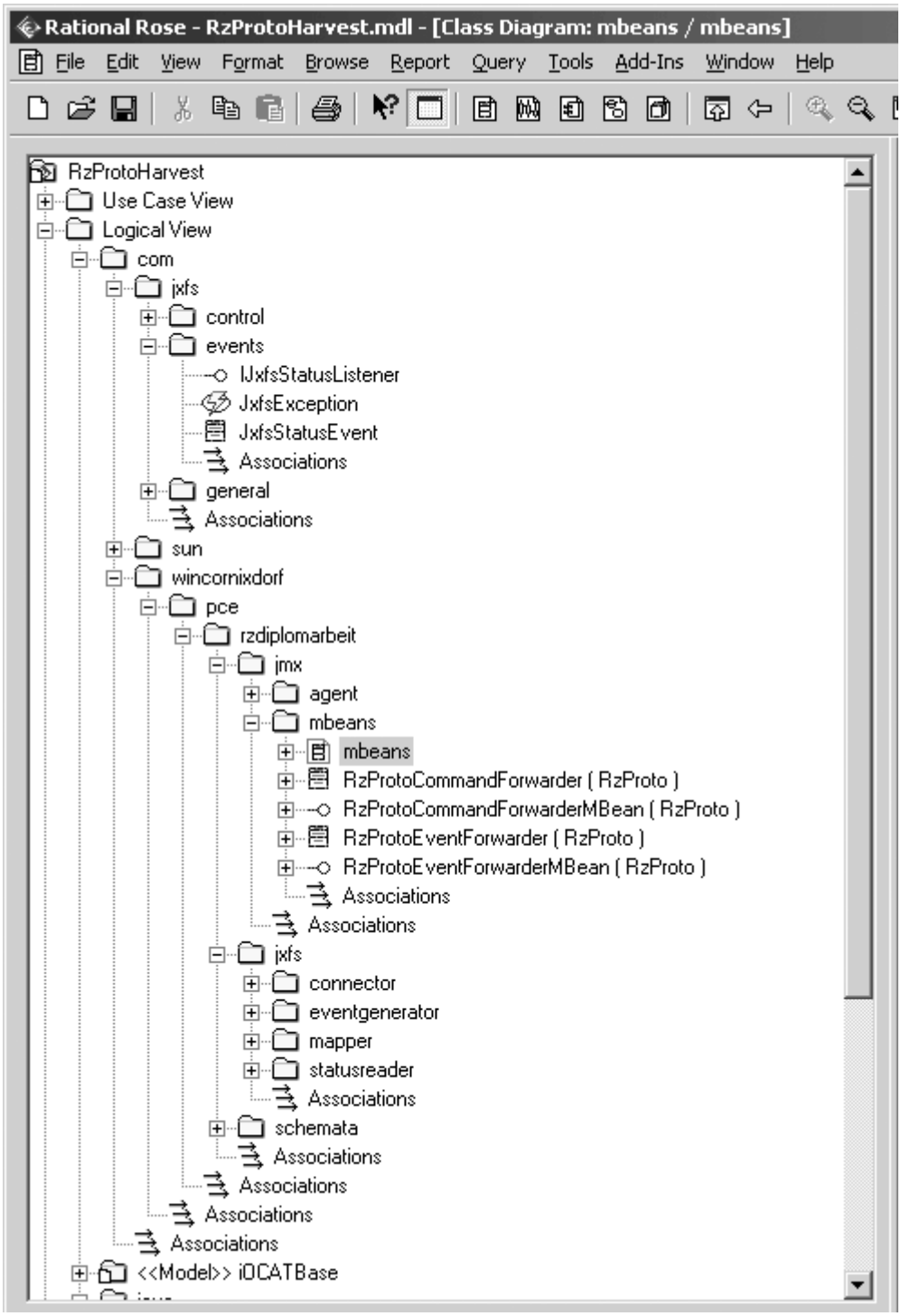


Abbildung 46: Klassenstruktur der importierten Elemente

Für jedes Package wird automatisch ein Klassendiagramm erstellt. In Abbildung 47 ist beispielhaft die Umsetzung der Managed Bean RzProtoEventForwarder dargestellt. Hier ist zu erkennen, dass auch Relationen umgesetzt werden. Kommentare aus dem Quellcode werden in das Modell übernommen und in die Spezifikationen der jeweiligen Elemente eingefügt. Das Modell kann nun im C-REF weiter verfeinert werden. Es ist beispielsweise möglich, Eigenschaften von anderen Sprachen und Plattformen zu modellieren.

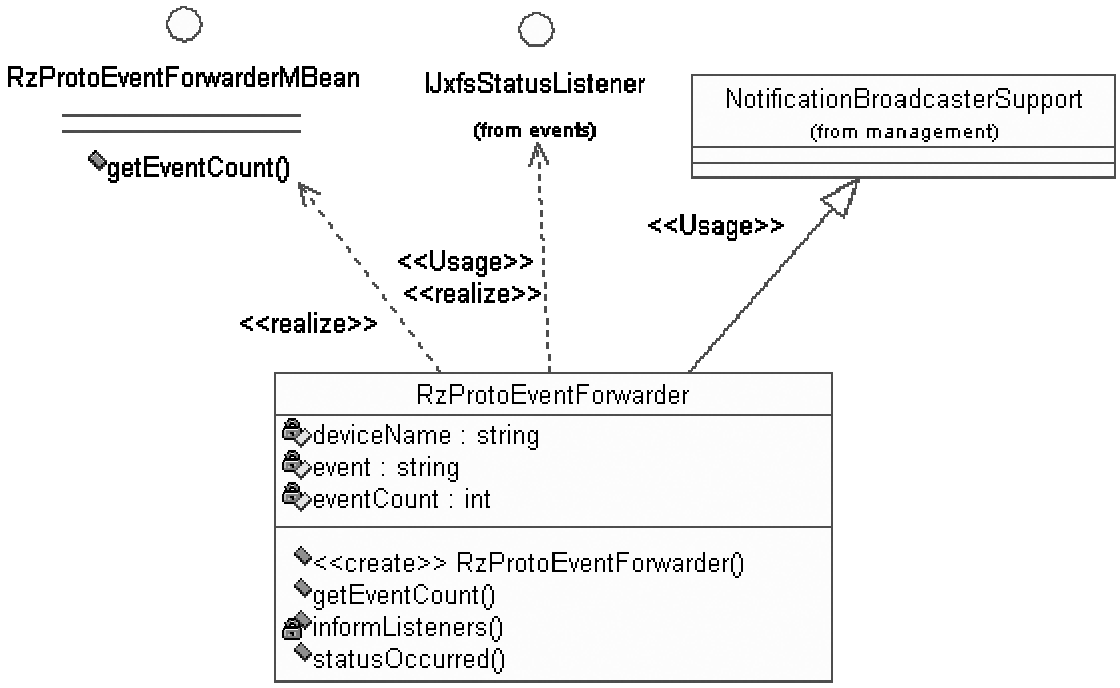


Abbildung 47: Umsetzung der EventForwarderMBean

Die Fähigkeiten von Harvester beschränken sich auf das Extrahieren von Klassen- und Methodenstrukturen, Fachlogik kann nicht bzw. im Rahmen eines EJB Imports nur in sehr begrenztem Maße übertragen werden. Wenn man das Modell in Quelltext umsetzen lässt, werden folglich nur die Strukturen generiert, die Rümpfe sind leer. Damit der Quelltext dennoch mit verwendet werden kann, besteht die Möglichkeit einen PA Merge (Protected Area[137] Merge) durchzuführen. Dazu wählt man im Kontextmenü *ArcStyler -> MDA Harvesting* und im folgenden Dialog *Protected Area Merge*. Nach Angabe des Quellen- und Zielverzeichnisses werden die Methodenrümpfe automatisch in die generierten Strukturen übertragen. Dieses Vorgehen ist im Rahmen der modellgetriebenen Entwicklung jedoch nicht zu empfehlen, da hier wieder auf Codeebene gearbeitet wird. Günstiger ist es, mittels geeigneten Diagrammen bzw. Einstellungen das Modell zu erweitern, so dass Teile der Geschäftslogik generiert werden können.

3.3.2 Modellierung und Verfeinerung im C-REF

Die Eigenschaften des importierten Modells werden von Harvester relativ wenig verändert. Nach dem Import ist es möglich, im Dialog *Eigenschaften* jeder Komponente detaillierte Einstellungen vorzunehmen.

[137] Mit Protected Area werden Abschnitte im Quellcode bezeichnet, welche vom Benutzer implementiert und vom Codegenerator nicht verändert werden.

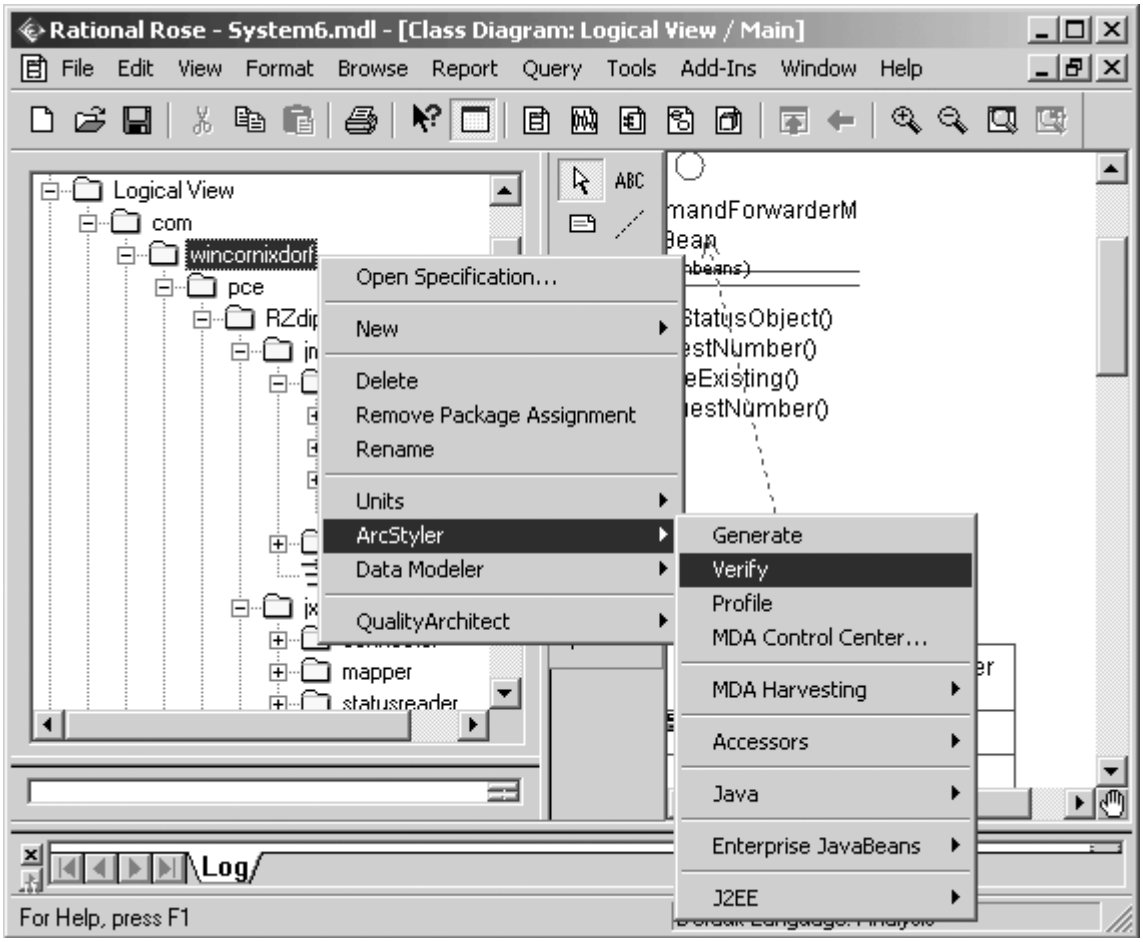


Abbildung 48: Verifikation und Generierung

Von programmiersprachenspezifischen Details (*ArcStylerC#, ArcStylerJava*) über Einstellungen für potentielle Applikationsserver (z.B. *ArcStylerJBoss*) bis hin zu Dokumentation kann alles eingegeben werden. Es ist möglich, zusätzliche Exceptions zu modellieren, die Zugehörigkeit von Klassen zu Packages zu definieren, ohne diese in Rose anzulegen und EJB-spezifische Einstellungen vorzunehmen. Da es sich hier um ein System mit recht einfacher Struktur handelt, sind allerdings keine weiteren Einstellungen oder Verfeinerungen notwendig.

Neben dem Klassendiagramm existieren für das Beispiel keine weiteren sinnvollen Modellierungsmöglichkeiten, da sich mit Aktivitätsdiagrammen und endlichen Automaten das Verhalten des Systems nicht näher beschreiben lässt.

3.3.3 Verifikation und Generierung

Nach abgeschlossener Modellierung besteht die Möglichkeit, das Modell oder Teile des Modells zu verifizieren und Quellcode generieren zu lassen. Dazu muss zunächst der Generator konfiguriert werden. Da ein Vergleich zu dem bestehenden Java-System stattfinden soll, wird die Java2-Cartridge gewählt. Sobald eine Cartridge gewählt ist, kann über

das Kontext-Menü die Verifikation des Modells und die Codegenerierung gestartet werden (siehe Abbildung 48).

3.3.4 Der generierte Code

ArcStyler generiert nicht nur die Klassen des Systems selbst, sondern auch Klassen für die Imports von Java-Standard-Klassen und JUnit-Testtreibern. Mit Letzterem ist es möglich, im C-IX einen Build des Systems durchzuführen und einen Testlauf zu starten, der prüft, ob alle Komponenten problemlos zusammenarbeiten (siehe Abbildung 49).

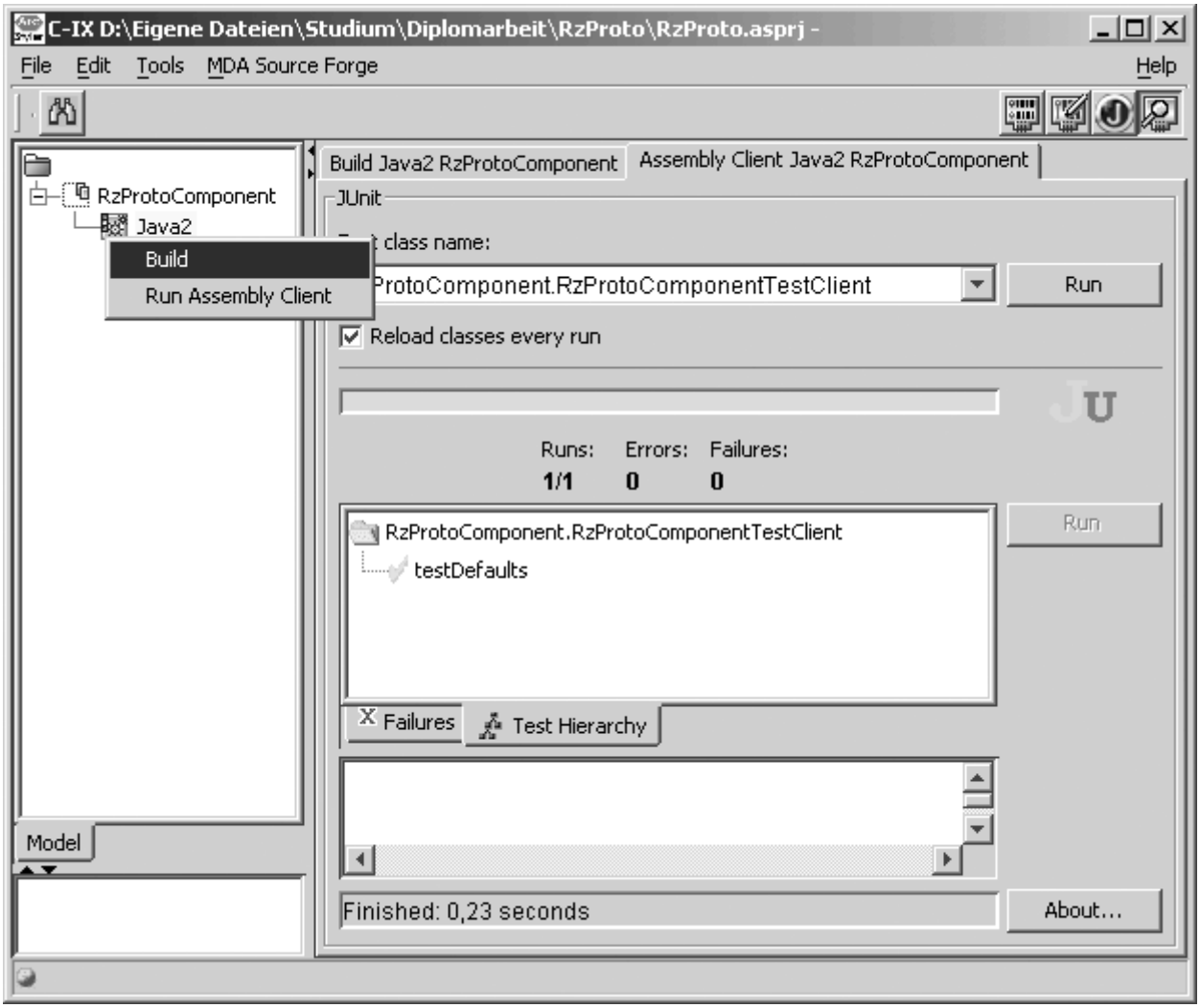


Abbildung 49: Test des Systems mit JUnit-Testtreibern

Im Hinblick auf den Quellcode werden alle statischen Strukturen generiert, also Klassen- und Methodenrahmen, Attribute inklusive ggf. vorhandener Initialisierungen, Imports und Vererbungen.

Der generierte Code ist fehlerfrei. Für einen Vergleich von handgeschriebenem und generiertem Code werden im Folgenden zwei ausgewählte Codeabschnitte näher betrachtet.

Zunächst wird die Implementierung der Klasse RzProtoEventForwarderMBean untersucht. Bei dieser Klasse handelt es sich um ein Interface mit einer Methode.
Im folgenden Listing ist die manuelle Umsetzung zu sehen:

```
package com.wincornixdorf.pce.rzdiplomarbeit.jmx.mbeans;


/**
 * defines the functionality of the RzProtoEventForwarder. <br>
 * @author Roger Zacharias
 */
public interface RzProtoEventForwarderMBean {


    /**
     * returns the number of occurred events.
     * @return  number of occurred events
     */
    public int getEventCount();
}
```

Abbildung 50: RzProtoEventForwarderMBean manuell programmiert

Unten ist zum Vergleich der generierte Code abgedruckt. Was sofort auffällt, ist der unterschiedliche Umfang der beiden Abschnitte, umfasst die handgeschriebene Version ca. 12 Zeilen Code und Kommentare, so ist die generierte Version ungefähr viermal so umfangreich, was die Lesbarkeit erschwert. Allerdings ist die generierte Klasse fehlerfrei und sofort einsatzbereit, ohne dass der Entwickler manuell nachbessern muss: ein großer Zeitgewinn. Hier wird deutlich, dass sich für die Erstellung von Interfaces ohne Geschäftslogik und ähnliche Komponenten das Generieren durchaus lohnt.

```
/**
 * Generated by ArcStyler.
 *
 * ArcStyler is copyrighted 1999-2002 by Interactive Objects
 * Software GmbH. All rights reserved.
 * http://www.ArcStyler.com/ http://www.io-software.com/
 */


/**
 *
@(#)com.wincornixdorf.pce.rzdiplomarbeit.jmx.mbeans.RzProtoEventForwarderMBea
n.java
 */
```

```
package com.wincornixdorf.pce.rzdiplomarbeit.jmx.mbeans;

/* START OF PROTECTED AREA  <<imports>> */
// insert custom code here
/* END OF PROTECTED AREA 17b460850000001c */

/**
 *
 * defines the functionality of the RzProtoEventForwarder. <br>
 * <!-- START OF PROTECTED AREA  <<Docu-RzProtoEventForwarderMBean-:-3>> -->
 * <!-- END OF PROTECTED AREA 0005090c00000004-:-4 -->
 *
 */
public interface RzProtoEventForwarderMBean
/* START OF PROTECTED AREA  <<inheritance>> */
/* END OF PROTECTED AREA 0000000a00000001 */
{
  /* START OF PROTECTED AREA  <<top>> */
  // insert custom code here
  /* END OF PROTECTED AREA f9f09dc500000020 */

  /**
   *
   * returns the number of occurred events.
   * @return number of occurred events
   * <!-- START OF PROTECTED AREA  <<Docu-getEventCount-:-3>> -->
   * <!-- END OF PROTECTED AREA 12e26f8c00000006-:-4 -->
   *
   * @return int
   *
   */
   int getEventCount();

  /* START OF PROTECTED AREA  <<bottom>> */
  // insert custom code here
  /* END OF PROTECTED AREA f9f09dc500000020 */
}
```

Abbildung 51: RzProtoEventForwarderMBean mit ArcStyler generiert

Einen anderen Blickwinkel des Systems bietet das folgende Beispiel. Es wird die Implementierung des Konstruktors der Klasse RzProtoJmxAgent betrachtet. An dieser Stelle

werden die durch MBeans repräsentierten Dienste für den Zugriff auf die J/XFS Funktionalität bei dem MBean-Server registriert, es handelt sich also um einen Codeabschnitt mit sehr viel Geschäftslogik.

Hier zunächst die manuelle Implementierung:

```
 /**
  *The class constructor. <br>
  * creates the MBean Server.
  */
public RzProtoJmxAgent() {
      System.out.println("<RzProtoJmxAgent> -> Creating MBean server ...");
     this.server = MBeanServerFactory.createMBeanServer();
     // register the Command Forwarder MBean
     System.out.println("\n<RzProtoJmxAgent> -> Creating <RzProtoCommand
                          Forwarder> ...");
     RzProtoCommandForwarder jxfsMBean = new RzProtoCommandForwarder();
      this.addMBean(jxfsMBean,"DefaultDomain:name=RzProtoCommandForwarder,
                    id=1,desc=command forwarder");
    // register Event Forwarder MBean
    System.out.println("\n<RzProtoJmxAgent> -> Creating <RzProtoEvent
                         Forwarder> ...");
    RzProtoEventForwarder eventMBean = new RzProtoEventForwarder();
    this.addMBean(eventMBean, "DefaultDomain:name=RzProtoEventForwarder,
    id=2,desc=event forwarder");
}
```

Abbildung 52: Konstruktor des RzProtoJmxAgent manuell programmiert

Unten ist nun als Vergleich der generierte Code zu sehen.

```
/**
 *
 * The class constructor. <br>
 * creates the MBean Server.
 * <!-- START OF PROTECTED AREA  <<Docu-Constructor-RzProtoJmxAgent-:-3>> -->
 * <!-- END OF PROTECTED AREA 12e26f8c00000006-:-4 -->
 *
 */
  public RzProtoJmxAgent()
  {
    /* START OF PROTECTED AREA  <<RzProtoJmxAgent>> */
    // insert custom code here
    /**@todo: Implement logic for RzProtoJmxAgent() */
```

```
    /* END OF PROTECTED AREA 2a2f5c1e0000005b */
  }
```

Abbildung 53: Konstruktor des RzProtoJmxAgent mit ArcStyler generiert

Hier ist der Umfang des generierten Code deutlich geringer als der des handgeschriebenen Codes. Trotz der Signaturen für Protected Areas, wo dem Benutzer das manuelle Einfügen von Geschäftslogik ermöglicht wird, ist hier sofort zu erkennen, dass nichts außer der Signatur des Konstruktors generiert wurde. Alle Aktionen, die im Konstruktor durchgeführt werden, müssen von Hand implementiert werden. Folglich sind für die Entwicklung einer Komponente mit viel Geschäftslogik folgende Arbeitsschritte erforderlich:

- Modell entwerfen
- Code generieren
- Im generierten Code die entsprechende Stelle (PA) suchen
- Manuell Geschäftslogik implementieren

Als Konsequenz bleibt hier festzustellen, dass bei Klassen, die viel Geschäftslogik enthalten, der Generator dem Entwickler nur einen Bruchteil der Arbeit abnimmt. Hinzu kommt, dass es nicht leicht fällt, sich in dem recht unübersichtlichen Code aus Protected Areas und generierten Fragmenten zurechtzufinden.

3.4 Umsetzung mit OptimalJ

3.4.1 Importieren des Klassendiagramms

Als erster Schritt bei der Entwicklung einer neuen Applikation wird, wie in Kapitel 2.3.2.2 erläutert, ein neues Projekt angelegt. Weiterhin werden die Verzeichnisse in denen Projekt und Code abgelegt werden sollen, aktiviert und die Bibliotheken für JMX und JXFS geladen (siehe Abbildung 54).

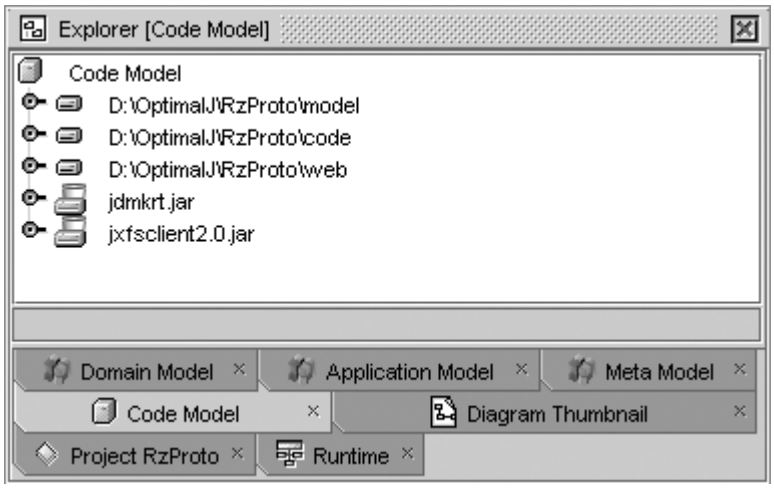


Abbildung 54: Mounten der Bibliotheken und Zielverzeichnisse in OptimalJ

Es wird ein Package ohne vorgegebene Struktur erstellt, da das System keine klassische 2-Tier oder 3-Tier Architektur verwendet.

OptimalJ erwartet als Basis für die Entwicklung einer Applikation ein Klassendiagramm. Dieses Diagramm kann entweder in OptimalJ modelliert oder importiert werden und muss im *domain model* in einem Package abgelegt werden, welches einen Filter auf Klassen gesetzt hat (siehe Kapitel 2.3.2.2).

Das Import-Werkzeug unterstützt XMI 1.2 und UML 1.3, es können laut [CMPWARi], S. 67 aus Rational Rose 2002 und Objecteering 5.2.1 exportierte Modelle gelesen werden. Das in Kapitel 3.2.2 beschriebene, in Rational Rose vorliegende Klassendiagramm wird exportiert und dann über einen Klick auf *Model-> Import Model -> Import Class from UML* mit Hilfe des sich öffnenden Wizards in OptimalJ importiert. Auffällig beim Importieren ist, dass Klassen mit dem Stereotyp Interface (die MBean-Interfaces) nicht importiert werden. OptimalJ unterstützt generell keine Stereotypen[138], Elemente dürfen nur einfache Klassen

[138] Siehe [CMPWSK].

ohne ergänzenden Stereotyp sein. Weiterhin gehen Abhängigkeiten verloren, nur Assoziationen werden von OptimalJ übernommen.

3.4.2 Generieren des PSMs

Nachdem das Klassendiagramm in OptimalJ importiert und nachgebessert wurde, soll im nächsten Schritt ein plattformspezifisches Modell generiert werden. Dazu bietet OptimalJ unter dem Menüpunkt *Model -> Generate Model* verschiedene Menüpunkte an, mit welchen u.a. die verschiedenen Tiers einer J2EE Anwendung generiert werden können. Eine reine Java-Umsetzung ist nicht möglich (siehe Abbildung 55).

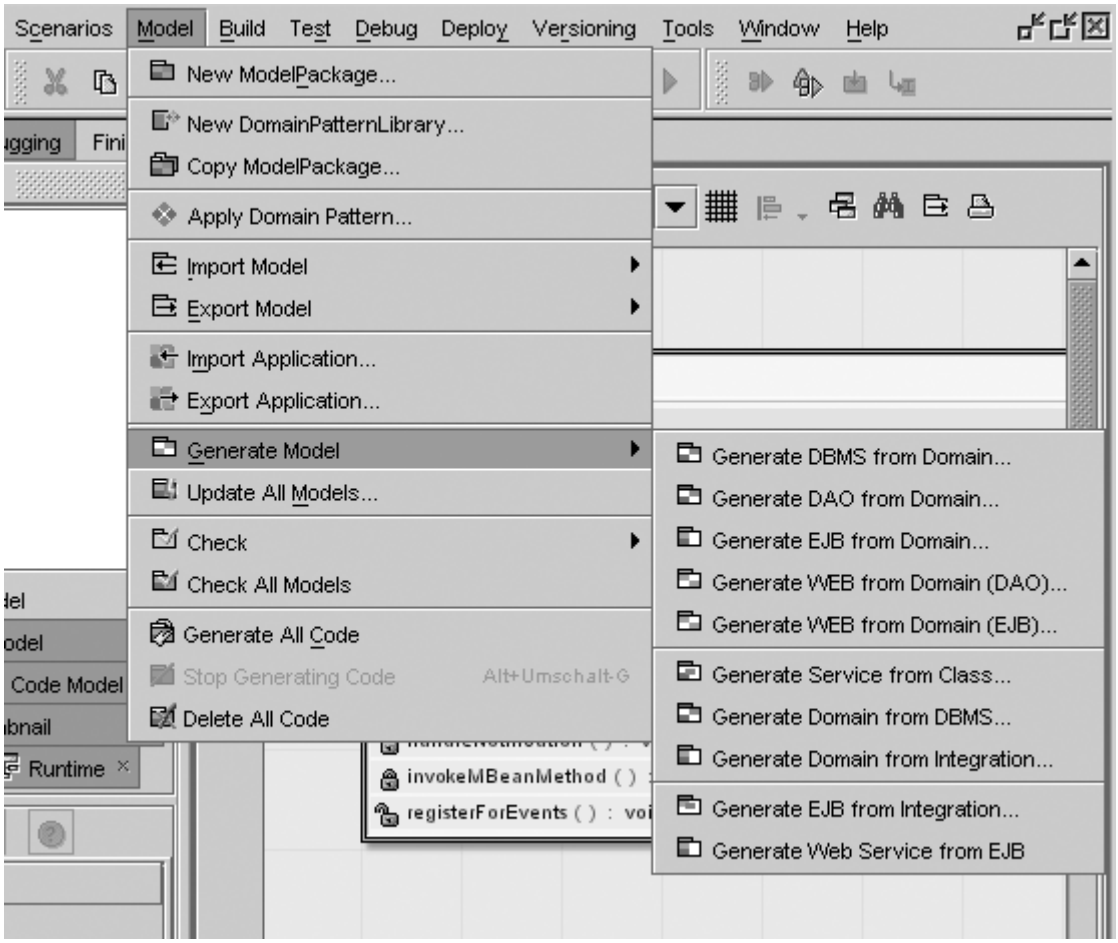


Abbildung 55: Erstellen eines PSM in OptimalJ

An dieser Stelle wird klar, dass eine Umsetzung des Beispielsystems mit OptimalJ so nicht möglich ist, da nur reine J2EE-Architekturen entwickelt werden können. Das vorliegende Beispielsystem verwendet weder EJBs, noch wird eine Datenbank oder eine Web-Oberfläche mit JSP benötigt.
Als Konsequenz wird die prototypische Umsetzung an dieser Stelle abgebrochen.

3.5 Umsetzung mit smartGenerator

3.5.1 Import des Klassendiagramms und Generierung

Da smartGenerator zur Zeit nur Klassendiagramme umsetzen kann, wurde lediglich das in Kapitel 3.2.2 vorgestellte Klassendiagramm verwendet. In Rational Rose modelliert, wird es im Menü *Tools* über die von smartGenerator hinzugefügte Schaltfläche *BITPlan -> XMI Export* exportiert.

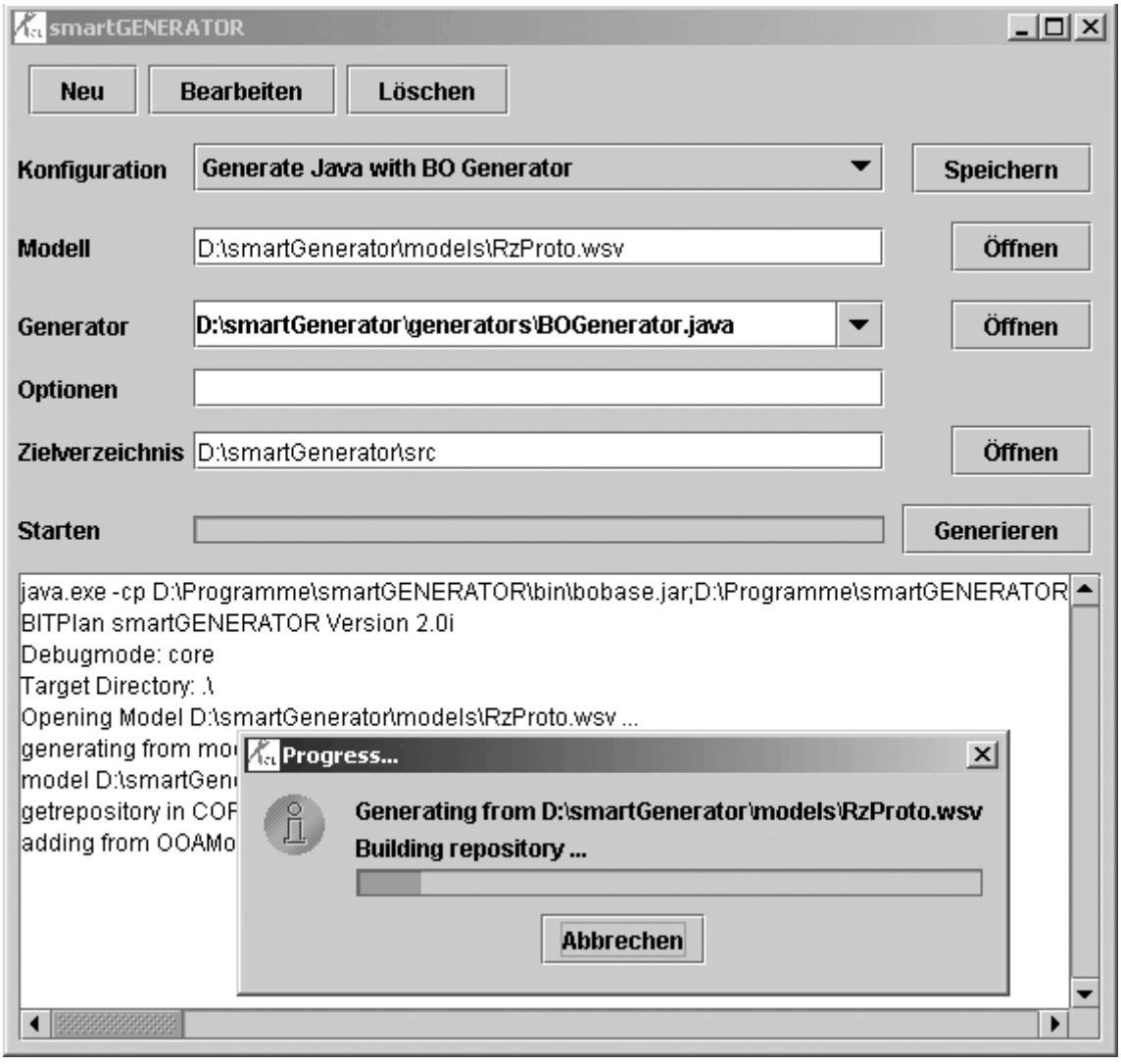


Abbildung 56: Generierungsvorgang in smartGenerator

Beim Exportieren wird das Modell in den Formaten *.xmi und *.wsv[139] gespeichert. Eines dieser Modelle dient smartGenerator als Input zum Generieren. Zur Zeit ist es nur möglich, die *.wsv-Datei zu verwenden, da smartGenerator nur XMI Version 1.1 unterstützt, die

[139] Das Format *.wsv wurde von BITPlan zu Optimierungszwecken entwickelt. Es benötigt weniger Platz und ist für smartGenerator schneller zu verarbeiten als *.xmi.

Diagramme jedoch noch in Version 1.0 exportiert werden[140]. Alternativ können auch XMI-Exporte der unterstützten UML-Tools verwendet werden.
Als Cartridge (oder Generator, wie BITPlan die Transformationen bezeichnet) wird der BOGenerator[141] von BITPlan verwendet. Dies ist eine einfache Cartridge, die reines Java implementiert. Durch einen Klick auf die Schaltfläche *Generieren* wird der Übersetzungsvorgang gestartet.
Während des weniger als 5 Sekunden dauernden Generierungsvorganges werden Statusmeldungen auf der Konsole ausgegeben (siehe Abbildung 56).

3.5.2 Der generierte Code

SmartGenerator erstellt im Wesentlichen die Systemstatik: Klassen- und Methodenrahmen, Attribute mit Initialisierungen (sofern modelliert) und import-Statements. Es werden Getter und Setter[142] für alle Attribute generiert, wobei zusätzlich ein Flag mit dem Namen _set<Variablenname> angelegt wird, welches überprüft, ob die jeweilige Variable initialisiert wurde. Über die Methode _isset<Variablenname>, die dieses Flag als return-Wert liefert, ist es möglich den Status einer Variable zu überprüfen. Zu beachten ist, dass je nach Anwendungsbereich ein oder mehrere von BITPlan mitgelieferte Archive zum Ausführen der Anwendung notwendig sind. Im dem hier behandelten Beispiel war das Einbinden des Archivs *common.jar* notwendig, welches den Umgang mit Exceptions abdeckt.
Beim Betrachten des generierten Codes sind vor allem folgende Punkte auffällig:
Die Package-Deklaration ist fehlerhaft, was zu Fehlern bei den folgenden Codezeilen führt, da die zwei Punkte am Ende der Zeile vom Compiler nicht interpretiert werden können.

```
package com.wincornixdorf.pce.rzdiplomarbeit.jmx.mbeans..;
```

Es gibt Probleme beim Import von Packages wie *java.lang.string*, da diese nur als *String* angegeben und folglich nicht gefunden werden.
Alle öffentlichen Methoden werden mit dem Schlüsselwort *synchronized* transaktionsicher implementiert. Dies wird auch bei den Konstruktoren versucht, was nicht erlaubt ist:

```
public synchronized RzProtoCommandForwarder() throws Exception
```

Insgesamt wird dem Entwickler viel Tipparbeit abgenommen, aber der Code ist durch das großzügige, aber nicht sinnvolle Getter- und Setter-Konzept sowie die generelle Anwendung des Schlüsselwortes *synchronized* recht unübersichtlich und nicht immer korrekt.

---

[140] Aus [BITPSK].
[141] Business Object Generator.
[142] Diese werden allerdings als *private* gekennzeichnet.

In den Klassen muss die Geschäftslogik in den Protected Areas aufgefüllt werden. Die MBean-Interfaces werden verwendungsfähig generiert, da hier nur die Signaturen der Methoden benötigt werden. Der Zusammenhang von MBean-Klasse und MBean-Interface über das Schlüsselwort *implements* wird leider nicht umgesetzt.
Vorsicht ist geboten beim Umgang mit Strings. Soll im generierten Code

```
"c:\\jxfsconnector.properties"
```

stehen, so muss im Modell der String in das entsprechende Feld folgendermaßen eingetragen werden:

```
c:\\\\jxfsconnector.properties
```

Handgeschriebener und generierter Code werden nun in Beispielen gegenübergestellt. Dazu werden die gleichen Codeabschnitte betrachtet wie bei der Umsetzung mit ArcStyler in Kapitel 3.3.3.
Zunächst die von Hand geschriebene Implementierung des Interfaces RzProtoEventForwarderMBean:

```
package com.wincornixdorf.pce.rzdiplomarbeit.jmx.mbeans;


/**
 * defines the functionality of the RzProtoEventForwarder. <br>
 * @author Roger Zacharias
 */
public interface RzProtoEventForwarderMBean {


    /**
     * returns the number of occurred events.
     * @return  number of occurred events
     */
    public int getEventCount();
}
```

Abbildung 57: RzProtoEventForwarderMBean manuell programmiert

Im Vergleich dazu die generierte Version:

```
/*   Copyright (C) 1999-2002 BITPlan GmbH
     Meerbuscher Str. 58-60
     40670 Meerbusch-Osterath
     Tel. +49-(0)2159/5236-0
     Fax. +49-(0)2159/5236-100
     generated: 07.07.2003 um 16:44
     $Header$
     $Id$
```

```
// >>>{RCS-Log secti-
on}{com\wincornixdorf\pce\rzdiplomarbeit\jmx\mbeans\.\RzProtoEventForwarderMB
ean.java}{com\wincornixdorf\pce\rzdiplomarbeit\jmx\mbeans\.\RzProtoEventForwa
rderMBean.java}
$Log$
// <<<{RCS-Log secti-
on}{com\wincornixdorf\pce\rzdiplomarbeit\jmx\mbeans\.\RzProtoEventForwarderMB
ean.java}{com\wincornixdorf\pce\rzdiplomarbeit\jmx\mbeans\.\RzProtoEventForwa
rderMBean.java}
*/
/**
 *
 */
package com.wincornixdorf.pce.rzdiplomarbeit.jmx.mbeans..;


import com.wincornixdorf.pce.rzdiplomarbeit.jmx.mbeans.*;
import com.bitplan.common.ExceptionStack;
// needed imports for RzProtoEventForwarderMBean
// >>>{User defined import sec-
tion}{::com::wincornixdorf::pce::rzdiplomarbeit::jmx::mbeans::RzProtoEventFor
warderMBean}{S3F058CC700CD}
// no implementation yet !!!
// <<<{User defined import sec-
tion}{::com::wincornixdorf::pce::rzdiplomarbeit::jmx::mbeans::RzProtoEventFor
warderMBean}{S3F058CC700CD}
/**
 *  defines the functionality of the RzProtoEventForwarder. <br>
 *  @version $Id$
 *  @author  $Author$
 *  <DL><DT><B><DD>Template-Version:</B> Id: implgen.in,v 2.5 2003/01/16
 * 09:48:48 wf Exp </DD></DL>
 */
public  class RzProtoEventForwarderMBean   {
/**
 * ID for GNU Revision Control System - will show in the class file and
 * can be looked for
 * using the ident command of RCS
 */
    private final static String RCSID="$Id$";
// >>>{User defined global sec-
tion}{::com::wincornixdorf::pce::rzdiplomarbeit::jmx::mbeans::RzProtoEventFor
warderMBean}{S3F058CC700CD}
// no implementation yet !!!
// <<<{User defined global sec-
tion}{::com::wincornixdorf::pce::rzdiplomarbeit::jmx::mbeans::RzProtoEventFor
warderMBean}{S3F058CC700CD}
```

```
// >>>{global reverse engineer-
ing}{::com::wincornixdorf::pce::rzdiplomarbeit::jmx::mbeans::RzProtoEventForw
arderMBean}{S3F058CC700CD}
// no implementation yet !!!
// <<<{global reverse engineer-
ing}{::com::wincornixdorf::pce::rzdiplomarbeit::jmx::mbeans::RzProtoEventForw
arderMBean}{S3F058CC700CD}
// Attributes
// Operations
/**
 *  returns the number of occurred events.
 * @throws Exception if getEventCount fails
 */
    public synchronized int getEventCount() throws Exception {
          String ExcHint="?";
          try {
// >>>{Op }{getEventCount}{S3F058CC90292}
// no implementation yet !!!
// <<<{Op }{getEventCount}{S3F058CC90292}
          }     catch (Throwable _t) {
                throw new ExceptionStack("int getEventCount() failed:
                 "+ExcHint,_t,this);
    }
    } // getEventCount
// Relations
} // RzProtoEventForwarderMBean
```

Abbildung 58: RzProtoEventForwarderMBean mit smartGenerator generiert

Wie leicht zu erkennen ist, ist der generierte Code sehr ausführlich kommentiert und durch Transaktionssicherheit und Ausnahmebehandlung auf einem hohen Sicherheitsniveau umgesetzt. Die mit `// >>>` beginnenden Zeilen begrenzen die Protected Areas. Durch automatisierte Implementierung vieler Protected Areas wird die Datei auf ein Vielfaches der ursprünglichen Größe vergrößert und ist unübersichtlicher.

Durch Betrachten der wesentlichen Codezeilen wird jedoch deutlich, dass bis auf zwei unnötige Imports (die ihren Ursprung nicht im Modell haben) das Gleiche an „Funktionalität“[143] wie im Original umgesetzt wurde.

Als zweites Beispiel wird der Konstruktor der Klasse RzProtoJmxAgent betrachtet:

Hier das handgeschriebene Original:

```
/**
```

[143] Ein Interface hat keine Funktionalität im eigentlichen Sinne.

```
  * The class constructor. <br>
  * creates the MBean Server.
  */
public RzProtoJmxAgent() {
     System.out.println("<RzProtoJmxAgent> -> Creating MBean server...");
     this.server = MBeanServerFactory.createMBeanServer();

     // register the Command Forwarder MBean
    System.out.println("\n<RzProtoJmxAgent> -> Creating <RzProtoCommand
                        Forwarder> ...");
     RzProtoCommandForwarder jxfsMBean = new RzProtoCommandForwarder();

     this.addMBean(jxfsMBean,"DefaultDomain:name=RzProtoCommandForwarder,id=1
                   ,desc=command forwarder");

     // register Event Forwarder MBean
     System.out.println("\n<RzProtoJmxAgent> -> Creating <RzProtoEvent
                         Forwarder> ...");
     RzProtoEventForwarder eventMBean = new RzProtoEventForwarder();
     this.addMBean(eventMBean, "DefaultDomain:name
      =RzProtoEventForwarder,id=2,desc=event forwarder");
}
```

Abbildung 59: Konstruktor des RzProtoJmxAgent manuell programmiert

Zum Vergleich der generierte Code:

```
**
 *  The class constructor. <br>
 * creates the MBean Server.
 * @throws Exception if RzProtoJmxAgent fails
 */
public synchronized  RzProtoJmxAgent() throws Exception {
// >>>{constructor calls}{RzProtoJmxAgent}{S3F058CBE0079}
// no implementation yet !!!
// <<<{constructor calls}{RzProtoJmxAgent}{S3F058CBE0079}
          String ExcHint="?";
          try {
// >>>{Op }{RzProtoJmxAgent}{S3F058CBE0079}
// no implementation yet !!!
// <<<{Op }{RzProtoJmxAgent}{S3F058CBE0079}
          }     catch (Throwable _t) {
                throw new ExceptionStack(" RzProtoJmxAgent() failed: "
                                         +ExcHint,_t,this);
```

```
    }
} // RzProtoJmxAgent
```

Abbildung 60: Konstruktor des RzProtoJmxAgent mit smartGenerator generiert

Hier werden die Schwächen des Generators deutlich. Die Systemstatik wird generiert, es wird automatisch Exception-Handling und Transaktionssicherheit implementiert, obwohl die Verwendung des Schlüsselwortes *synchronized* beim Konstruktor nicht erlaubt ist und zu einer Fehlermeldung führt. Das Erstellen des MBean Servers und das Registrieren der MBeans konnte nicht modelliert werden und wird auch nicht umgesetzt, hier ist Implementierung von Hand notwendig.

Insgesamt ist eine Überarbeitung des Codes durch einen Programmierer zwingend. Selbst Klassen, in denen kein Auffüllen von Geschäftslogik in Protected Areas vorgenommen werden muss, enthalten kleine Fehler, die von Hand beseitigt werden müssen. Diese kommen durch Fehler in der Cartridge zustande, die für einen professionellen Einsatz auf jeden Fall überarbeitet und erweitert werden müsste.

Je mehr Geschäftslogik implementiert werden muss, desto mehr Arbeit hat der Programmierer, der sich durch den generierten Code arbeiten, die entsprechende Protected Area suchen und dort die Logik implementieren muss.

4 Bewertung

4.1 MDA im Vergleich zur manuellen Programmierung

Es gibt in der Softwarebranche sehr viele Arten von Anwendungen. Wie in den vorherigen Kapiteln gezeigt wurde, ist nicht jede Art für die modellbasierte Entwicklung geeignet. MDA-Werkzeuge können vor allem Datenbanken, Transaktionen und Logik, die in Verwaltungssystemen etc. benötigt wird[144], recht gut umsetzen. Im Idealfall (z.B. J2EE, sehr wenig Geschäftslogik) ist es möglich, in einzelnen Klassen 80% bis 100% des Quellcodes zu generieren[145]. Sobald ein System jedoch individueller wird und die interne Logik schwer zu modellieren ist, da es sich weniger um transaktionsähnliche Abläufe handelt, stoßen aktuelle MDA-Werkzeuge schnell an ihre Grenzen. Dies liegt zum Teil daran, dass heute noch nicht hinreichend mächtige Modellierungssprachen existieren, um bestimmte Sachverhalte plattformunabhängig darzustellen. Zum Teil kann Geschäftslogik nicht (effektiv) modelliert werden, sondern muss von einem Programmierer direkt umgesetzt werden.
Im Allgemeinen ist zur erfolgreichen Anwendung von MDA ein Umdenken in den Entwicklungsabteilungen notwendig. Es muss von der code-zentrierten Entwicklung zur modellbasierten Entwicklung übergegangen werden, in deren Rahmen die Strukturen vor dem Programmieren erst sorgfältig modelliert werden. Änderungen sollten im Modell und nicht im fertigen Code durchgeführt werden. Dies ermöglicht eine kontrolliertere Entwicklung, höhere Wiederverwendung und sorgt dafür, dass das Wissen um Struktur und Inhalt der Programme nicht personengebunden, sondern modellgebunden ist.
Zusammenfassend lässt sich sagen, dass der Ansatz der modellbasierten Entwicklung bei richtigen Vorraussetzungen zahlreiche Vorteile bietet, die die Nachteile bei Weitem übersteigen. Wie sinnvoll der Einsatz von MDA ist, hängt jedoch vom Einsatzgebiet und den Entwicklern ab. Das Gleiche gilt für den Einsatz der konvergenten Architektur - ob mit oder ohne Unterstützung eines MDA-Werkzeugs: Lohnenswert ist der Einsatz auf jeden Fall, sofern das Projekt groß genug ist und die Entwickler entsprechend diszipliniert sind.

[144] Zum Beispiel für Bankkontoführung oder Systeme zur Verwaltung von Büchern etc..
[145] Aus [IOSWDB].

4.2 ArcStyler

ArcStyler als eines der größten MDA-Werkzeuge bietet Unterstützung für den gesamten modellgetriebenen Entwicklungszyklus von Analyse bis Deployment und Test und erweist sich dabei als extrem vielseitig. Von der ersten Anforderungsanalyse bis hin zum endgültigen Modell wird der Benutzer durch verschiedene Module geleitet. Durch Hilfestellungen und ständige Verifikation wird sichergestellt, dass das Modell dem Architekturstil der konvergenten Architektur (siehe Kapitel 2.2) gerecht wird. Dadurch entsteht eine qualitativ hochwertige Vorlage, deren Niveau in den Quellcode übernommen werden kann. Durch die Verwendung von Rational Rose in einem Modul wird einerseits eine Bindung zu dieser Software eingegangen, was unter Umständen zu einem erhöhten Einarbeitungsaufwand führen kann. Andererseits hat man ein sehr komfortables und weit verbreitetes Modellierungstool an der Hand. Etwas problematisch ist die Mischung Rose/ArcStyler, da z.B. die ArcStyler Projektverwaltung deutlich aufgesetzt und nicht Bestandteil von Rose ist. Dadurch ergeben sich teilweise Schwierigkeiten beim Verwalten von Projekten oder bei dem eigenständigen Einsatz von Rose.

Für den Einstieg in ArcStyler empfiehlt sich die Lektüre des Buches „Convergent Architecture“[146] von Richard Hubert[147]. Es ist deutlich zu erkennen, dass Buch und Software aus dem gleichen Haus stammen und dass das Konzept des Buches auch hinter der Software steht. Durch die Kenntnis der Hintergründe ist es möglich, sich in den verschiedenen Modulen von ArcStyler zurechtzufinden, deren Zusammenwirken sonst auf den ersten Blick nicht zu erfassen ist. Durch eine sehr ausführliche Benutzerdokumentation sind zahlreiche Hilfestellungen gegeben. Das Werkzeug überzeugt durch seine weitestgehend intuitive Bedienung, nur die Verteilung der Funktionen auf ein Kontextmenü und ein Menü in Rose ist etwas gewöhnungsbedürftig.

Der generierte Code hat eine sehr hohe Qualität und ist im Anwendungsbeispiel fehlerfrei. Wie viel prozentual an Programmcode generiert werden kann, hängt sehr stark von dem Einsatzgebiet ab. Bei der Umsetzung von Transaktionen, Web-Interfaces und einfachen Geschäftsabläufen[148] bietet ArcStyler eine sehr umfassende Unterstützung in Form von endlichen Automaten, Modellierung mit OCL und Unterstützung bei der Modellierung von Zugriffskomponenten und Repräsentanten[149]. Bei individuelleren Anwendungen wie dem

[146] Quelle: [HUBE02].
[147] Herr Hubert ist CEO bei der Interactive Objects Software GmbH.
[148] Wie z.B. die Umsetzung eines Bankkontos.
[149] Siehe auch Kapitel 2.2.3.3.

in Kapitel 3 vorgestellten Beispiel kann kaum mehr als die Systemstatik generiert werden. Zusätzliche Geschäftslogik muss vom Entwickler von Hand in den Code eingefügt werden. Eine besondere Stärke von ArcStyler sind die zahlreichen existierenden Cartridges für dieses Werkzeug. Java, C#, J2EE, .NET sowie verschiedene Middleware-Lösungen werden unterstützt. ArcStyler sorgt dafür, dass ein System kontrolliert und konform zu einem Architekturstil entwickelt wird, was vor allem bei Projekten mit mehreren Teams die Integration der einzelnen Softwareelemente fördert. Auch werden dem Entwickler viele Standard-Aufgaben abgenommen. Durch ein eigenes Framework für die Erweiterung bestehender und die Entwicklung neuer Cartridges ist es möglich, ArcStyler an individuelle Bedürfnisse anzupassen.
Da das Werkzeug recht umfangreich ist, muss auf jeden Fall eine angemessene Einarbeitung in ArcStyler und in die CARAT Foundation stattfinden. Im Überblick lässt sich sagen, dass ArcStyler ein umfangreiches Werkzeug ist, welches trotzdem recht gut zu bedienen ist. Ihre volle Leistungsfähigkeit kann die Software jedoch nur in bestimmten Systemen umsetzen. Im Hinblick auf Einarbeitungszeit und Lizenzkosten muss hier bedacht werden, ob im Falle Wincor Nixdorf der potentielle Einsatzbereich die Möglichkeiten von ArcStyler optimal nutzt, andernfalls wird viel Kapazität verschwendet. Betrachtet man das prototypisch umgesetzte System, so wird deutlich, dass ein Einsatz von ArcStyler für Wincor Nixdorf nur in Verbindung mit einer Cartridge für JMX sinnvoll ist. Durch die Verwendung einer solchen Cartridge wäre es möglich, Implementierungsdetails wie die MBean-Inferfaces automatisiert zu erstellen. Dennoch wäre auch hier noch ein großer Anteil an Geschäftslogik von Hand zu implementieren, da ArcStyler nicht in der Lage ist, die in diesem System verwendete Geschäftslogik zu generieren. Das Entwicklungsteam müsste sowohl in der modell-zentrierten Entwicklung als auch in ArcStyler gut geschult werden, um das Potential von ArcStyler voll auszuschöpfen.

4.3 OptimalJ

OptimalJ ist das komfortabelste der drei vorgestellten Programme. Mit sehr wenig Input wird ein umfangreicher Output generiert, was unter Anderem an der Beschränkung auf J2EE liegt. Durch die Spezialisierung auf eine Plattform müssen wenig Einstellungen und Verfeinerungen an dem Modell vorgenommen werden: In der Software ist voreingestellt, wie ein Klassendiagramm in J2EE umgesetzt wird. Als umfangreiche Lösung umfasst das Werkzeug die modellgetriebene Entwicklung von der (internen) Modellierung des Klassendiagramms bis hin zum Deployment, die Analyse wird nicht unterstützt.

OptimalJ ist als einziges der vorgestellten Programme vollkommen unabhängig von externen Modellierungswerkzeugen, kann jedoch auch Modelle importieren. Die vorgegebenen Standardarchitekturen und eine strenge Bindung an die J2EE-Plattform stellen eine hohe Qualität des Modells sicher. Durch die enge Benutzerführung werden viele Fehler vermieden. Der Benutzer wird während der Entwicklung durch zahlreiche Wizards unterstützt und angeleitet. Es ist könnte allerdings sein, dass man sich mit der Zeit eine weniger eng geführte Bedienung wünscht, da die Arbeit mit Assistenten auf Dauer zeitintensiver ist als die manuelle Bedienung. An manchen Stellen ist es möglich, die Wizards nicht zu verwenden, man kann sie jedoch nicht deaktivieren. Die im Vergleich mit anderen Werkzeugen hohen Systemvoraussetzungen führen bei etwas älteren Rechnern zu mühsamer Bedienung, und auch nicht alle Schaltflächen sind dort angeordnet wo man sie vermutet.
Für die Umsetzung des Beispielsystems war OptimalJ nicht geeignet, da nur reine J2EE-Systeme erstellt werden. Weiterhin ist zu erwähnen, dass das Programm in der Version 2.2 fehlerhaft installiert wurde und nicht automatisch deinstalliert werden konnte. Version 3.0 lief selbst nach mehreren Kontakten mit dem Support nicht fehlerfrei und es war nicht möglich, das Tutorial vollständig durchzuarbeiten.
Im J2EE Bereich ist es möglich, signifikante Einsparungen zu machen, da das Werkzeug große Teile der Anwendungen generiert, wie in Kapitel 2.3.2.2 gezeigt wird. Hier stellt sich jedoch die Frage, inwieweit die Vision von MDA durch eine auf eine Plattform begrenzte Lösung umgesetzt wird. Es besteht zwar die Möglichkeit, eigene Cartridges sowohl für die Generierung von PSMs als auch für die Erstellung von Code zu entwerfen, womit eine Entwicklung für andere Plattformen möglich wäre. Dies ist allerdings ein recht komplexer Vorgang, da sowohl für die Generierung der PSMs als auch für die Erstellung des Codes eigene Regeln entwickelt werden müssen. Der Hersteller bietet keine Erweiterungen an und auch die Dokumentation für Teile der Erweiterungsfunktionalität war zum Abgabezeitpunkt dieser Arbeit noch nicht veröffentlicht.
Als Fazit bleibt, dass das komfortable OptimalJ für das Einsatzgebiet „Managementsysteme mit JMX und J/XFS“ nicht in Frage kommt. Dieses Werkzeug ist nur für die Entwicklung reiner J2EE Anwendungen geeignet. Eine Entwicklung eigener Cartridges wäre extrem aufwändig und würde den Nutzen nicht rechtfertigen, da sich die Entwickler komplett in OptimalJ und die Erweiterungsfunktionalität einarbeiten müssten und vollkommen neue Umsetzungsregeln benötigt werden.

4.4 smartGenerator

Ein Vergleich mit den sehr umfassenden Werkzeugen ArcStyler und OptimalJ ist schwer zu ziehen, da smartGenerator der Firma BITPlan lediglich die Generierung und das Erstellen von Cartridges unterstützt. Die Modellierung wird vom Entwickler ohne Hilfestellungen oder Anleitungen in einem externen Tool durchgeführt. Der Vorteil liegt dabei klar auf der Hand: Da smartGenerator viele Standard-Modellierungswerkzeuge unterstützt, ist es sehr wahrscheinlich, dass der Entwickler mit einem dieser Werkzeuge bereits Erfahrung hat; so werden Einarbeitungszeit und ggf. auch Lizenzkosten gespart. Nachteilig ist die fehlende Kontrolle des Architekturstils und der Integrität des Modells. Fehler werden erst sehr spät bemerkt. Eine gewisse Hilfestellung bietet die „check model“ Funktionalität, die von BITPlan in das UML-Werkzeug integriert wird. Hier werden in einer Konsole Fehlermeldungen und Warnungen ausgegeben, auf deren Basis das Modell überarbeitet werden kann. Die Fehlerprüfung ist allerdings nur bedingt hilfreich, da sie zum Teil Fehler meldet, die später bei der Generierung kein Problem darstellen. Beispielsweise wird bei Methoden ohne Rückgabewert eine Fehler gemeldet, obwohl beim Generieren der Rückgabetyp ordnungsgemäß als *void* deklariert wird.

Die Bedienung von smartGenerator ist einfach und schnell erlernbar, da nur Modell, Cartridge und Zielverzeichnis gewählt werden müssen. Weiterhin ist eine Verwendung über die Kommandozeile möglich. Durch die einfach gehaltene Oberfläche ist smartGenerator übersichtlich und intuitiv zu bedienen, es ist keine nennenswerte Einarbeitung erforderlich. Beim sehr zügig ablaufenden Generieren erstellt smartGenerator reine Systemstatik. Mit UML 2.0 ist die Umsetzung von Geschäftslogik geplant, doch momentan beschränkt sich die Funktionalität auf das Generieren von Geschäftsobjekten und deren Beziehungen. Dies schließt Deklarationen von Attributen und Methoden, Ausnahmebehandlung, Transaktionssicherheit und Importe ein. Der Code ist teilweise fehlerhaft und muss von Hand nachgebessert werden, die Geschäftslogik muss zusätzlich in vom Generator vorgesehenen Protected Areas aufgefüllt werden. Es werden in mancher Hinsicht Elemente generiert, die nicht benötigt werden[150] oder die nicht sinnvoll eingesetzt werden können[151].

smartGenerator ist genau das richtige Werkzeug, um dem Programmierer Tipparbeit zu ersparen. Ohne viel Aufwand kann das Werkzeug schnell und unkompliziert für beliebige Problemstellungen eingesetzt werden. Dafür ist allerdings eine fehlerfreie Cartridge unerlässlich. Für Technologien wie JMX ist es möglich, ohne allzu großen Arbeitsaufwand

[150] Alle Methoden werden als „synchronized“ deklariert, was nicht immer notwendig ist.
[151] Beispielsweise wird für jedes private Attribut ein Getter und ein Setter generiert.

Cartridges zu erstellen, die entsprechende (statische) Merkmale wie MBean-Interfaces umsetzen. Leider existiert für dieses Werkzeug zur Zeit noch keine ausführliche Benutzerdokumentation, was viele Fragen zur Verwendung offen lässt. Diese werden zwar vom sehr guten Support beantwortet, trotzdem bleibt der erhöhte Arbeitsaufwand. Weiterhin wären leichter verständliche Fehlermeldungen sehr nützlich, da die Fehlersuche bei Generierungsproblemen sonst sehr mühsam ist.
Zusammenfassend lässt sich sagen, dass smartGenerator ein leicht verständliches und gut zu bedienendes Werkzeug für jedes Einsatzgebiet ist, welches sich mit vergleichsweise geringem Aufwand auf individuelle Problemstellungen anpassen lässt. Der Funktionsumfang ist allerdings recht klein und teilweise lässt die Benutzerführung zu wünschen übrig. Weiterhin macht der nicht unbeträchtliche Preis von 25.000 € einen Kauf für kleinere Unternehmen recht unattraktiv. Für Wincor Nixdorf wäre ein Einsatz von smartGenerator in Verbindung mit der Entwicklung einer JMX-Cartridge denkbar, allerdings müsste geklärt werden, wie viel Zeit und Aufwand durch MDA gegenüber einem Codegenerator gespart werden könnte. Dazu ist vor allem eine Bestimmung der denkbaren Zielarchitekturen sowie der Aufwand zum Entwickeln einer genau angepassten Cartridge notwendig.

4.5 Übersicht

Die Ergebnisse aus den vorhergehenden Kapiteln werden hier in einer Übersicht zusammengefasst. Ziel ist es, einen direkten Vergleich zu bieten, wie die Werkzeuge im Hinblick auf verschiedene Kriterien einzustufen sind. Dabei werden die Bewertungen qualitativ oder quantitativ in fünf Abstufungen vorgenommen:

++ sehr gut, sehr umfangreich
+ gut, umfangreich
o ausreichend, angemessen
- mangelhaft, geringer Umfang
-- unbefriedigend, zu wenig

Es ist zu beachten, dass bei Kriterien wie „Systemvoraussetzungen“ geringe Anforderungen als positiv dargestellt werden.
In der Kategorie „Preis“ wird der tatsächliche Preis in Euro für eine Einzellizenz angegeben, damit es dem Leser möglich ist, einen Überblick über das Preis/Leistungsverhältnis zu gewinnen.

Die Bewertung erfolgt schwerpunktmäßig im Hinblick auf die in Kapitel formulierte Fragestellung, ob und in welchem Maße sich der Einsatz der vorgestellten Werkzeuge für Wincor Nixdorf lohnt.

Verwendungsspezifische Kriterien

	ArcStyler	OptimalJ	smartGenerator
Bedienung	+	+	++
Einarbeitung	o	o	++
Benutzerführung	+	++	-
Funktionsumfang[152]	++	+	o
Unterstützte Plattformen	++	-	+
Performance	+	o	++
Umfang Generierung[153]	o/+	o/++	o
Qualität Code	++	n. a.	-
Erweiterbarkeit	++	o	+

Sonstige Kriterien

	ArcStyler	OptimalJ	smartGenerator
Systemvoraussetzungen	+	o	++
Lizenzkosten[154]	9.800 €	11.500€	25.000€
Dokumentation	++	++	-
Support	+	o	++

[152] Funktionsumfang im Bezug auf den Software-Entwicklungszyklus.
[153] Der Umfang der Generierung ist stark abhängig vom Einsatzgebiet.
[154] Hier werden die Kosten für die umfangreichste Version der Software mit allen Funktionalitäten angegeben. Bei ArcStyler und OptimalJ handelt es sich um eine Named-User Lizenz, die Lizenz von smartGenerator ist für bis zu 50 Mitarbeiter gültig.

5 Zusammenfassung/Fazit

> *„We are aware of the fact that the grand vision of MDA [..] is not yet reality. However, some parts of MDA can be used today, while others are under development.“*[155]

Dieses Zitat beschreibt den momentanen Stand der Model Driven Architecture sehr treffend. In der Theorie ist die Vision einer kontrollierten, modell-zentrierten Entwicklung der perfekte Ausweg aus dem Dilemma schnell entwickelter Software von minderwertiger Qualität, der Bindung von Programmkenntnissen an Personen und zu hoher Wartungskosten. Wie aus den Ergebnissen der vorliegenden Arbeit zu ersehen ist, besteht zur Zeit jedoch noch eine beträchtliche Diskrepanz zwischen der Theorie der Model Driven Architecture und der Leistungsfähigkeit der Softwarelösungen auf diesem Gebiet. Auch qualitativ hochwertige Werkzeuge sind meist nur von begrenztem Nutzen, vor allem für individuelle Lösungen mit wenigen Standardabläufen. Eine Schlüsselkomponente zur verbreiteten und erfolgreichen Anwendung von MDA ist die Unterstützung der Werkzeuge bei der Erstellung eigener Cartridges. Diese ist jedoch häufig sehr komplex und würde einen sehr großen Einarbeitungs- und Entwicklungsaufwand fordern, bevor überhaupt an den Einsatz von MDA zur Entwicklung gedacht werden kann. Beim Einsatz einer MDA-Entwicklungsumgebung muss folglich davon ausgegangen werden, dass eine nicht unerhebliche Investition in Lizenzkosten, Schulungen und eventuell in die Entwicklung einer passenden Cartridge fließen. Ziel der Entwicklung mit MDA kann nicht ein schneller ROI sein, hier muss langfristig geplant werden. Grund dafür sind die Stärken der MDA, die sich z.B. in Wiederverwendung und nicht personengebundenem Wissen erst später auszahlen.
Ebenfalls sollte der Einstellungswechsel berücksichtigt werden, der im Denken der Entwickler stattfinden muss. Nur ein sehr diszipliniertes Team, das modell-zentriert entwickelt und von dem code-zentrierten Ansatz Abstand nehmen kann, ist überhaupt in der Lage, erfolgreich mit MDA zu arbeiten.
Für die Wincor Nixdorf GmbH & Co. KG lohnt sich der Einsatz von MDA auf dem untersuchten Gebiet der Managementsysteme nur in sehr beschränktem Umfang. Die Nutzung würde in diesem Fall zwar eine bedeutende Erleichterung gegenüber der manuellen Programmierung bedeuten, jedoch nur minimal über generative Programmierung mit gängigen UML-Codegeneratoren hinausgehen. Gerade die Umsetzung von Anwendungen mit

[155] [KWBW03], S. XIII.

JMX ist recht aufwändig, da diese Technologie noch von keinem Werkzeug unterstützt wird. Ein unerlässlicher Schritt wäre also, eine eigene Cartridge für JMX zu entwickeln.
Ein Ansatz, der die (spätere) Verwendung einer MDA-Entwicklungsumgebung begünstigen würde, wäre die Verwendung der konvergenten Architektur auch ohne das Werkzeugpaket und die Technologieprojektionen. Durch den geregelten Entwicklungsprozess entstehen viele (z. T. qualitative) Vorteile. Zusätzlich wird die Disziplin der Entwickler gefördert und eine spätere Einführung von MDA erleichtert.
Abschließend lässt sich sagen, dass Model Driven Architecture ein guter Ansatz ist, der mehr Qualität und schnellere Entwicklung in der Softwarebranche bringen wird. Allerdings ist die neue Technologie noch sehr in den Anfängen und zum aktuellen Zeitpunkt nur in beschränktem Maße anwendbar, was sich aber in den nächsten Jahren signifikant verbessern wird.

6 Anhang

Anhang A: Literaturverzeichnis

Bücher

[CKEU02] Czarnecki, Krysztof / Eisenecker, Ulrich W. (2000):
Generative Programming,
1. Auflage 2000, Addison Wesley.

[FRAN03] Frankel, David S. (2003):
Model Driven Architecture,
1. Auflage 2003, Wiley Publishing Inc..

[GHJV96] Gamma, Erich et. al. (1996):
Entwurfsmuster:
Elemente wiederverwendbarer objektorientierter Software,
Aus dem Amerikanischen übersetzt von D. Riehlke
4. korrigierter Nachdruck, Addison-Wesley Verlag.

[HUBE02] Hubert, Richard (2002):
Convergent Architecture,
1. Auflage 2002, Wiley Publishing, Inc..

[HMKG99] Hitz, M. / Kappel, G. (1999):
UML@Work,
1. Auflage 1999, Heidelberg, dpunkt Verlag.

[KWBW03] Kleppke, A. / Warmer, J./ Bast, W. (2003):
MDA Explained,
1. Auflage 2003, Addison-Wesley.

Fachzeitschriften, akademische Arbeiten u. a.

[FAHL03a] Fahl, Wolfgang (2003):
MDA-Template-basierende Codegeneratoren,
Handout zum Vortrag,
09.05.2003, Köln.

[CBDI03a] CBDI Forum Limited (2003):
CBDI Report: Service Oriented Architecture and OptimalJ,
Handout bei der OptimalJ-Roadshow am 10.07.2003 in Frankfurt (Main).

[OBSP03a] OBJEKTSpektrum (Hrsg.)(2003):
Produktneuheiten,
In: OBJEKTSpektrum 02/03, S. 12.

[OBSP03b] Buchholdt, Christian (2003):
Ökonomische Entscheidungskriterien für den Einsatz der MDA,
In: OBJEKTSpektrum 02/03, S. 20ff..

[OBSP03c] Herzig, Andreas (2003):
Wie Entwurfsmuster und die MDA Java-Entwicklungen beschleunigen,
In: OBJEKTSpektrum 02/03, S. 26f..

[OBSP03d] Dietze, A. / Ghadir, P. / Tilkov, S. (2003):
Trennung von fachlicher und technischer Komponentenarchitektur im Sinne der MDA,
In: OBJEKTSpektrum 02/03, S. 63ff..

[TMCP03] The Middleware Company (Hrsg.) (2003):
Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach - A Productivity Analysis,
Erscheinungsdatum: Juni 2003.

[ZACH02] Zacharias, Roger (2002):
Management- und Service-Architekturen: Konzeption und Realisierung eines Überwachungssystems für Bankperipheriegeräte,
Februar 2002,
Diplomarbeit ,
FH Giessen-Friedberg.

Internet-Seiten und elektronische Dokumente

[ALEX03] Alexander, Sascha (2003):
UML 2.0 wird zum Politikum,
http://www.computerwoche.de/index.cfm?pageid=255&artid=44876&type=detail,
Erscheinungsdatum: 2003,
Zugriffsdatum: 11.08.2003.

[AMBR03a] Ambrosio, Johanna (2003):
MDA: What is the standard?,
http://www.adtmag.com/article.asp?id=7851,
Erscheinungsdatum: 01.07.2003,
Zugriffsdatum: 04.08.2003.

[AMBR03b] Ambrosio, Johanna (2003):
Tools for the code generation,
http://www.adtmag.com/article.asp?id=7850,
Erscheinungsdatum: 01.07.2003,
Zugriffsdatum: 04.08.2003.

[ANDROMa] Open-Source Projekt AndroMDA (Hrsg.):
http://www.andromda.org/,
Zugriffsdatum: 20.03.2003.

[BIENAD] Bien, Adam:
Model Driven Architecture,
http://www.compuware.de/events/praesentationen/mda_rs/Adam_Bien.pdf,
Zugriffsdatum: 03.03.2003.

[BINSTO] Binstock, Andrew:
MDA gives us reason to believe in methodology,
http://www.devx.com/SummitDays/Article/7803,
Zugriffsdatum: 26.05.2003.

[BITP02a] BITPlan GmbH (Hrsg.) (2003):
Wie erstellen Sie mehr Software schneller mit mehr Qualität?,
http://www.bitplan.de/download/BITPlan-smartGenerator.pdf, Erscheinungsdatum: 04.12.2002,
Zugriffsdatum: 07.04.2003.

[BITP02b] BITPlan GmbH (Hrsg.) (2002):
BITPlan smartGenerator 2.0 ab sofort verfügbar,
http://www.bitplan.de/xmlweb/index.php?topic=products/smartgenerator,
Erscheinungsdatum: 2002,
Zugriffsdatum: 07.04.2003.

[BITP03a] BITPlan GmbH (Hrsg.):
Einsatzszenarien mit Preisen für smartGenerator (unverbindlich),
Word-Datei per eMail,
Zugriffsdatum: 05.09.2003

[BITP03b] BITPlan GmbH (Hrsg.):
Preisliste smartGenerator(unverbindlich),
PDF-Datei per eMail,

Zugriffsdatum: 05.09.2003

[BITPLN] BITPlan GmbH (Hrsg.):
Infoblatt smartGenerator,
http://www.bitplan.de/download/Infoblatt_smartGenerator_de.pdf,
Zugriffsdatum: 07.04.2003.

[BOWK01] Bowker, Todd (2001):
Superior app management with JMX,
http://www.javaworld.com/javaworld/jw-06-2001/jw0608-jmx_p.html,
Erscheinungsdatum: 06/2001,
Zugriffsdatum: 07.04.2003.

[BROC02] Bibliographisches Institut & F.A. Brockhaus AG (Hrsg.) (2002):
Der Brockhaus in Text und Bild Edition 2002,
(elektronisches Nachschlagewerk).

[BUTL02] Charlesworth, Ian (2002):
Butler Review of ArcStyler,
http://www.io-software.com/as_support/brochures/ArcStyler_TECH_RPS_1098.pdf,
Erscheinungsdatum: Juni 2002,
Zugriffsdatum: 30.06.2003.

[BUTL03] Blowers, Mark (2003):
Butler Group Review of OptimalJ,
http://www.compuware.com/dl/butler_review.pdf,
Erscheinungsdatum: März 2003,
Zugriffsdatum: 12.06.2003.

[CBDI03b] CBDI Forum Limited (2003):
CBDI reviewed OptimalJ in June 2002,
http://www.compuware.com/dl/cbdi_review.pdf,
Erscheinungsdatum: Februar 2003,
Zugriffsdatum: 12.06.2003.

[CHAN01] Chang, Daniel T. (2001):
CWM: Model Driven Architecture,
Erscheinungsdatum: 2001,
http://www.cwmforum.org/cwmMDA_foilset.pdf,
Zugriffsdatum: 12.08.2003.

[CMPWARa] Compuware Corporation (Hrsg.):
Creating Implementation Patterns,
http://javacentral.compuware.com/members/optimalj/documentation/v2.2.01/1928-23-15-11-43.html,
Zugriffsdatum: 23.07.2003.

[CMPWARb] Compuware Corporation (Hrsg.):
Deploying an implementation pattern,
http://javacentral.compuware.com/members/optimalj/documentation/v2.2.01/1928-22-18-16-6gg.html,
Zugriffsdatum: 23.07.2003.

[CMPWARc] Compuware Corporation (Hrsg.):
Implementation pattern models,
http://javacentral.compuware.com/members/optimalj/documentation/v2.2.01/1928-1-15-43-52aa.html,
Zugriffsdatum: 23.07.2003.

[CMPWARd] Compuware Corporation (Hrsg.):
Joinpoints,
http://javacentral.compuware.com/members/optimalj/documentation/v2.2.01/jpIntro.html,
Zugriffsdatum: 23.07.2003.

[CMPWARe] Compuware GmbH (Hrsg.):
OptimalJ: Do more with less,
http://www.compuware.de/events/praesentationen/mda_rs/Compuware.pdf,
Zugriffsdatum: 03.03.2003.

[CMPWARf] Compuware Corporation (Hrsg.):
OptimalJ: How model-driven development enhances productivity,
OptimalJ White Paper,
Zugriffsdatum: 24.06.2003.

[CMPWARg] Compuware Corporation (Hrsg.):
Software Factory,
http://javacentral.compuware.com/members/optimalj/documentation/v2.201/1928-28-11-30-56.html,
Zugriffsdatum: 23.07.2003.

[CMPWARh] Compuware Corporation (Hrsg.):
OptimalJ – Supported Environments,
http://javacentral.compuware.com/optimalj/getting_started/pam.htm#ide,
Zugriffsdatum: 23.06.2003.

[CMPWARi] Compuware Corporation (Hrsg.):
OptimalJ Tutorials,
Bestandteil der Installation von OptimalJ.

[CMPWARj] Compuware Corporation (Hrsg.):
System Requirements for OptimalJ,
http://javacentral.compuware.com/optimalj/getting_started
/systemreq.htm,
Zugriffsdatum: 23.06.2003.

[CMPWARk] Compuware Corporation (Hrsg.):
Productivity Analysis: Model-Driven, Pattern-Based Development with OptimalJ,
http://www.compuware.de/events/praesentationen/mda_rs_2003/tmc.zip,
Zugriffsdatum: 15.07.2003.

[CMPWARl] Compuware Corporation (Hrsg.):
Using OptimalJ,
Bestandteil der Installation von OptimalJ.

[COMP02a] Versteegen, Gerhard (2002):
Wege aus der Plattformabhängigkeit,
COMPUTERWOCHE vom 01.02.02,
http://www.io-software.com/news/publications/german/
CW_%2005_02%20.pdf,
Zugriffsdatum: 24.03.2003.

[COMP02b] Computerwoche (Hrsg.) (2002):
Programmier-Tools verbergen ihre Technik,
COMPUTERWOCHE vom 09.08.02,
http://www.io-software.com/news/publications/german/
COMPUTERWOCHE_9Aug02_Butler_Report%20.pdf,
Zugriffsdatum: 24.03.2003.

[CxO 01] Versteegen, Gerhard (2001):
Mit ArcStyler Rational Rose perfektionieren,
CxO Ausgabe 11/01,
http://www.io-software.com/news/publications/german/
CxO11_Mit_ArcStyler_Rose_perfektionieren.pdf,
Zugriffsdatum: 24.03.2003.

[GART03a] Blechar, M. / Light, M. (2003):
ARAD brings architectural compliance and developer productivity,
Gartner Research Note vom 17.01.2003.

[GART03b] Blechar, M. (2003)
Architected RAD Tools are delivering major benefits,
Gartner Research Note vom 29.01.2003.

[GHJV02] Gamma, E./ Helm, R./Johnson, R./ Vlissides, J. (2002):
The Gang Of Four,
http://hillside.net/patterns/DPBook/GOF.html,
Erscheinungsdatum: 2002,
Zugriffsdatum: 01.09.2003

[HARMON] Harmon, Paul:
MDA: An Idea Whose Time Has Come,
http://www.omg.org/mda/mda_files/MDA_Seminar_Harmon.pdf,
Zugriffsdatum: 04.08.2003.

[HEIS03] Heise-Newsticker (Hrsg):
OMG segnet Unified Modeling Language in Version 2.0 ab,
http://ww.heise.de/newsticker/data/hos-13.06.03-000/,
Erscheinungsdatum: 13.06.2003,
Zugriffsdatum: 06.08.2003.

[HERR03a] Herrington, Jack D.(2003):
Code Generation: Decision Tree,
http://www.codegeneration.net/files/JavaOne_OnePageGuide_v1.pdf,
Erscheinungsdatum: 2003,
Zugriffsdatum: 24.06.2003.

[HERR03b] Herrington, Jack D. (2003):
Richard Hubert MDA Interview,
http://www.codegeneration.net/tiki-read_article.php?articleId=12,
Erscheinungsdatum: 08.05.2003,
Zugriffsdatum: 16.05.2003.

[HOW03] Howard, Phil (2003):
Compuware releases latest version of OptimalJ,
http://www.it-director.com/article.php?articleid=3661,
Erscheinungsdatum: 27.03.2003,
Zugriffsdatum: 10.06.2003.

[HUBE01] Hubert, Richard (2001):
ArcStyler: The Architectural IDE for MDA,
www.omg.org/mda/mda_files/iO_OMG_ArcStyler_20Oct01.pdf,
Erscheinungsdatum: 2001,
Zugriffsdatum: 10.06.2003.

[IBM03] IBM Corp. (2003):
Unified Modeling Language Version 2.0,
http://www.rational.com/uml/resources/uml2/index.jsp,
Erscheinungsdatum: 2003,
Zugriffdatum: 05.08.2003.

[ILOG03] I-Logix Inc. (2003):
UML 2.0,
http://www.ilogix.com/products/rhapsody/UML_20_FAQ.cfm,
Erscheinungsdatum: 2003,
Zugriffsdatum: 12.08.2003.

[INNO02a] Innoq GmbH (2002):
IQGen Users Guide,
http://www.innoq.com/iqgen/userguide.html,
Erscheinungsdatum: 2002,
Zugriffsdatum: 02.06.2003.

[INNO02b] Innoq Gmbh (2002):
IQGen Whitepaper,
http://www.innoq.com/iqgen/whitepaper.html,
Erscheinungsdatum: 2002,
Zugriffsdatum: 26.05.2003.

[IOSW99] Interactive Objects Software GmbH News Archive (Hrsg.) (1999):
Architectural IDE for EJB,
http://www.io-software.com/news/pr_arch_11_1999_d.jsp,
Erscheinungsdatum: November 1999,
Zugriffsdatum: 30.06.2003.

[IOSW02a] Interactive Objects Software GmbH (Hrsg.) (2002):
ArcStyler Modeling Style Guide,
http://www.io-software.com/as_support/docu
/BTA_Modeling_Style_Guide.pdf,
Erscheinungsdatum: 27.09.2002,
Zugriffsdatum: 24.06.2003.

[IOSW02b] Interactive Objects Software GmbH (Hrsg.) (2002):
The Architectural IDE for MDA,
http://www.io-software.com/as_support/factsheets/
iO_ArcStyler_Overview.pdf,
Erscheinungsdatum: 2002,
Zugriffsdatum: 30.06.2003.

[IOSW03a] Interactive Objects Software GmbH (Hrsg.) (2003):
ArcStyler Release Notes for ArcStyler Version 3.1,
http://www.io-software.com/as_support/docu/Release_Notes.pdf,
Erscheinungsdatum: 29.03.2003,
Zugriffsdatum: 30.02.2003.

[IOSW03b] Interactive Objects Software GmbH (Hrsg.) (2003):
ArcStyler License Packages and Pricing,
http://www.io-software.com/sales/data/ArcStyler
_pricing_3.1_new_31Mar03.pdf,
Erscheinungsdatum: 10.04.2003,
Zugriffsdatum: 30.06.2003.

[IOSW03c] Interactive Objects Software GmbH (Hrsg.) (2003):
ArcStyler Tools Guide,
Bestandteil der Installation von ArcStyler,
Erscheinungsdatum: 17.04.2003.

[IOSW03d] Interactive Objects Software GmbH (Hrsg.) (2003):
ArcStyler Extensibility Guide,
http://www.io-software.com/as_support/docu/extensibility_guide.pdf,
Erscheinungsdatum: 22.04.2003,
Zugriffsdatum: 24.06.2003.

[IOSW03e] Interactive Objects Software GmbH (Hrsg.) (2003):
ArcStyler Users Guide,
Bestandteil der Evaluierungsversion von ArcStyler,
Erscheinungsdatum: 22.04.2003.

[IOSW03f] Interactive Objects Software GmbH (Hrsg.) (2003):
ArcStyler Installation, Configuration and Adminstration Guide,
Bestandteil der Evaluierungsversion von ArcStyler,
Erscheinungsdatum: 28.04.2003.

[IOSW03g] Interactive Objects Software GmbH (Hrsg.) (2003):
ArcStyler Tutorial für J2EE und .NET,
Bestandteil der Installation von ArcStyler,
Erscheinungsdatum: 28.04.2003.

[IOSW03g] Interactive Objects Software GmbH (Hrsg.) (2003):
The Architectural IDE for MDA:
ArcStyler CARAT (CARtrirge ArchiTecture) Introduction and Overview,
http://www.io-software.com/tour/extensibility_carat20/extensibility_carat20_viewlet_swf.html,
Zugriffsdatum: 25.06.2000.

[IOSW03h] Interactive Objects Software GmbH (Hrsg.) (2003):
The Architectural IDE for MDA:
Full cycle development in the context of the Unified Process,
http://www.io-software.com ->ArcStyler Product Tour -> Full cycle development in the context of the Unified Process,
Erscheinungsdatum: 2003,
Zugriffsdatum: 30.07.2003.

[IOSWAR] Interactive Objects Software GmbH (Hrsg.):
Success Story: Austrian Railways (ÖBB),
http://www.io-software.com/customers/Success_Stories/SuccessStory_OeBB.pdf,
Zugriffsdatum: 30.06.2003.

[IOSWDB] Interactive Objects Software GmbH (Hrsg.):
Success Story: Deutsche Bank Bauspar AG,
http://www.io-software.com/customers/Success_Stories/SuccessStory_DBBS.pdf,
Zugriffsdatum: 30.06.2003.

[IOSWFQ] Interactive Objects Software GmbH (Hrsg.):
ArcStyler FAQ,
http://www.io-software.com/support/support_login.jsp?user=support&password=R2D2&page=support_faq.jsp&form=posted,
Zugriffsdatum: 04.07.2003.

[IOSWWW] Website der Firma Interactive Objects Software
http://www.io-software.com.

[IX 02] Merkle, Bernhard (2002):
Entscheidungsfreiheit,
iX Ausgabe 02/2002,
http://www.io-software.com/news/publications/german/ix_0202_Entscheidungsfreiheit.pdf,
Zugriffsdatum: 24.03.2003.

[JFST02] Java Forum Stuttgart (Hrsg.) (2002):
Abstracts und Vortragsfolien,
http://www.jfs2002.de/abstracts.html,
Zugriffsdatum: 23.07.2002.

[JXFS03a] JXFS Forum:
J/eXtensions for Financial Services (J/XFS) for the Java Platform:
Conceptual Overview,
http://www.jxfs.com/documents/jxfs_overview.zip,
Erscheinungsdatum: 28.03.2000,
Zugriffsdatum: 20.08.2003.

[JXFS03b] JXFS Forum:
J/eXtensions for Financial Services (J/XFS) for the Java Platform:
Frequently Asked Questions,
http://www.jxfs.com/documents/jxfs_overview.zip,
Erscheinungsdatum: 28.03.2000,
Zugriffsdatum: 20.08.2003.

[JXFSWW] JXFS Forum:
http://www.jxfs.com/,
Zugriffsdatum: 20.08.2003.

[KCLT02] Kennedy Carter Limited (Hrsg.) (2002):
Configurable Code Generation in MDA with iCCG,
http://www.kc.com/cgi-bin/download.cgi?action=ctn/CTN_27v3_0.pdf,
Erscheinungsdatum: 2002,
Zugriffsdatum: 26.05.2003.

[KCLTDE] Kennedy Carter Limited (Hrsg):
Expert Software Design Every Time,
http://www.kc.com/products/iccg/index.html,
Zugriffsdatum: 26.05.2003.

[OBSP02] Koch, Thomas (2002):
An Introduction to Model Driven Architecture,
In: OBJEKTSpektrum 03/2002,
http://www.io-software.com/news/publications/english/MDA_OS0103_e.pdf,
Zugriffsdatum: 24.03.2003.

[OBSP03a] Maurer, T. / Esslinger, A. / Sydow, T. (2003):
Vom Prozessmodell zur lauffähigen Anwendung,
In: OBJEKTspektrum 01/2003,
http://www.io-software.com/news/publications/german/2003/OS_01_03.pdf,
Zugriffsdatum: 24.03.2003.

[OBSP03e] Purgahn, J. / Renner, G. / Reckziegel, J. / Krusch, J. (2003):
Modellgetriebene Architektur in einem J2EE- und COBOL-Großrechner-Umfeld,
http://www.maas.de/de/projects/casestudies/dbb-02/01.html,
Erscheinungsdatum: 2003,
Zugriffsdatum: 24.03.2003.

[OEWE03] Oestereich, B. / Weilkins, T. (2003):
UML 2.0: Alles wird gut?,
Erscheinungsdatum: Januar 2003,
http://www.sigs.de/publications/os/2003/01/oestereich_OS_01_03.pdf,
Zugriffsdatum: 11.08.2003.

[OMG 00] Object Management Group Inc. (Hrsg.) (2000):
CWM™ Audio Briefing: The Key To Integrating Business Intelligence,
http://www.omg.org/news/releases/pr2000/cwm/whitepaper.htm,
Erscheinungsdatum: 2000,
Zugriffsdatum: 11.08.2003.

[OMG 03a] Object Management Group Inc. (Hrsg.) (2003):
Executive Overview Model Driven Architecture,
http://www.omg.org/mda/executive_overview.htm,
Erscheinungsdatum: 2003,
Zugriffsdatum: 26.05.2003.

[OMG 03b] Object Management Group Inc. (Hrsg.) (2003):
MDA FAQ,
http://www.omg.org/mda/faq_mda.htm,
Erscheinungsdatum: 2003,
Zugriffsdatum: 26.05.2003.

[OMG 03c] Object Management Group Inc. (Hrsg.) (2003):
MDA Guide Version 1.0,
http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf,
Erscheinungsdatum: 2003,
Zugriffsdatum: 04.08.2003.

[OMG 03d] Object Management Group Inc. (Hrsg.) (2003):
OMG MDA Member and Analyst Quotes,
http://www.omg.org/mda/mda_files/Member_and_Analyst_Quotes.pdf,
Erscheinungsdatum: 2003,
Zugriffsdatum: 26.05.2003.

[OMG 03e] Object Management Group Inc. (Hrsg.) (2003):
Success Story: Deutsche Bank Bauspar AG,
http://www.omg.org/mda/mda_files/SuccessStory_DBBS.pdf,
Erscheinungsdatum: 2003,
Zugriffsdatum: 26.05.2003.

[OMG 03f] Object Management Group Inc. (Hrsg.) (2003):
Success Story: Gothaer Versicherungen, IDG Informationsverarbeitung und Dienstleistungen GmbH, b+m,
http://www.omg.org/mda/mda_files/b+marchitecturesuccessstory.pdf,
Erscheinungsdatum: 2003,
Zugriffsdatum: 26.05.2003.

[OMG 03g] Object Management Group Inc. (Hrsg.) (2003):
UML 2.0 Standard Officially Adopted at OMG Technical Meeting in Paris,
http://www.omg.org/news/releases/pr2003/6-12-032.htm,
Erscheinungsdatum: 2003,
Zugriffsdatum: 05.08.2003.

[OMGMDA] Internetauftritt der Object Management Group Inc. (Hrsg.):
Überblick über Model Driven Architecture,
http://www.omg.org/mda,
Zugriffsdatum: 04.07.2003.

[OMGCWM] Object Management Group Inc. (Hrsg.):
Überblick über MOF und CWM,
http://www.omg.org/cwm,
Zugriffsdatum: 11.08.2003.

[OMGSPC] Internetauftritt der Object Management Group Inc. (Hrsg.):
Liste aller Spezifikationen,
http://www.omg.org/technology/documents/modeling_spec_catalog.htm,
Zugriffsdatum: 11.08.2003.

[POOL01] Poole, John D. (2001):
Model-Driven Architecture:
Vision, Standards And Emerging Technologies,
http://www.cwmforum.org/Model-Driven%20Architecture.pdf,
Erscheinungsdatum: April 2001,
Zugriffsdatum: 24.03.2003.

[SCHE02] Scheb, Alexander (2002):
Entwicklungshilfe:
COMPUWARE OptimalJ Professional Edition 2.0 im Überblick,
http://www.scheb.de/OptimalJ.htm,
Erscheinungsdatum: 2002,
Zugriffsdatum: 20.05.2003.

[SCHW03] Schwiontek, Melanie (2003):
Design Patterns realisieren mit smartGenerator,
Erscheinungsdatum: 2003.

[SESI03] Serverside.com (Hrsg.)(2003):
The challenge: dbBausparen online,
http://www.theserverside.com/resources/article.jsp?l=BausparenOnline,
Erscheinungsdatum: Mai 2003,
Zugriffsdatum: 26.05.2003

[SJSD02] John Siegel (2002):
Making the Case: OMG's Model Driven Architecture,
http://sdtimes.com/news/064/special1.htm,
Erscheinungsdatum: 15.10.2002,
Zugriffsdatum: 05.08.2003.

[SMITHJ] Smith, John Allen:
Case Study: A Model Driven Architecture for Integrating Enterprise Wide Federal Web Applications,
http://www.omg.org/news/meetings/workshops/presentations/uml2001_presentations/08-1_Smith-OMG_UML_Workshop_Presentation_011205.pdf,
Zugriffsdatum: 24.03.2003.

[STGBOM] Starke, G. / Bohlen, M.:
Was ist MDA?,
http://www.openmda.de/aboutmda.htm,
Zugriffsdatum: 20.03.2003.

[SUN 00] Sun Microsystems Inc.(Hrsg.) (2000):
Java Management Extensions
Instrumentation and Agent Specification v1.0,
http://www.sun.com/jmx/jmx_instr_agent.pdf,
Erscheinungsdatum: 06/2000,
Zugriffsdatum: 22.05.2003.

[THOM03] Thomas, Tony G. (2003):
Using JMX to Manage Web Applications,
http://www.theserverside.com/resources/articles/JMXWebApps/article.html,
Erscheinungsdatum: März 2003,
Zugriffsdatum: 15.05.2003..

[TMCWWW] The Middleware Company Inc.:
Internetauftritt,
http://www.middleware-company.com,
Zugriffsdatum: 18.08.2003.

[TREN02] GEBIT GmbH (Hrsg.) (2002):
Trend Whitepapers,
http://www.gebit.de/download/trend_whitepaper_3032_small.pdf,
Erscheinungsdatum: 2002,
Zugriffsdatum: 26.05.2003.

[WATE03] Waters, John K. (2003):
MDA: IBM Rational unveils RAD Tool,
http://www.adtmag.com/article.asp?id=7852,
Erscheinungsdatum: 01.07.2003,
Zugriffsdatum: 04.08.2003.

[ZDNE02] Judge, Peter (2002):
ArcStyler turns business models into code,
http://news.zdnet.co.uk/story/0,,t269-s2126632,00.html,
Erscheinungsdatum: 27.11.2002,
Zugriffsdatum: 24.03.2003.

Support und Kommunikation mit Firmen

[CMPWSK] Schriftliche, telefonische und persönliche Kommunikation mit Mitarbeitern der Firma Compuware Corporation im Zeitraum vom 23.06.2003 - 22.07.2003.

[IOSWSK] Schriftliche, telefonische und persönliche Kommunikation mit Mitarbeitern der Firma Interactive Objects Software GmbH im Zeitraum vom 16.05.2003 - 27. 08.2003.

[BITPSK] Schriftliche, telefonische und persönliche Kommunikation mit Mitarbeitern der Firma BITPlan GmbH im Zeitraum vom 20.05.2003 - 05.09.2003.

Anhang B: CD mit Quellcode

Auf der beiliegenden CD ist sowohl der Quellcode des Managementsystems in der für diese Arbeit angepassten Version, als auch der von den MDA-Werkzeugen generierte Code zu finden.
Weiterhin wurden die benötigten Bibliotheken für JMX und J/XFS hinzugefügt, um ein Ausführen des Systems zu ermöglichen.

Pfad	**Inhalt**
/ArcStyler	Generierter Quellcode der MDA-Entwicklungsumgebung ArcStyler
/jars	Java Archive für JMX und J/XFS
/RzProto	Quellcode des Managementsystems in der eingeschränkten Version
/smartGenerator	Generierter Quellcode der MDA-Entwicklungsumgebung smartGenerator

Eidesstattliche Versicherung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Wissenbach, den 08.09.2003

_____

Stephanie Weg