Wiederverwendbare Generatorkomponenten in heterogenen Projektumfeldern

Norbert Klein, Robert Nikonowicz

Capgemini sd&m Research
Capgemini sd&m AG
Mühlheimer Str. 9a
53840 Troisdorf
{norbert.klein, robert.nikonowicz}@capgemini-sdm.com

Abstract: Capgemini sd&m setzt seit über 15 Jahren modellgetriebene Generatoren in seinen Projekten ein. Bisher wurden immer wieder neue Infrastrukturen aufgesetzt, Modellierungssprachen entwickelt und Generatoren gebaut. Jetzt entwickeln wir einen einheitlichen Modellierungs- und Generierungs-Ansatz. Dabei setzen wir auf die MDD-Werkzeuge im Umfeld der Eclipse-Plattform. Wir integrieren Werkzeuge wie Eclipse, openArchitectureWare, Maven2 und Enterprise Architect mit eigenen fertigen Generator-Komponenten (Cartridges) zu einem flexiblen und erweiterbaren MDD-Baukasten: dem Capgemini sd&m MDD-Toolkit. Die im MDD-Toolkit enthaltenen Cartridges können zu individuellen projektspezifischen Generatoren zusammengesetzt werden. Ein Projekt wählt die Cartridges aus, die die gewünschte Ziel-Plattform und Architektur unterstützen. Die Cartridges und der von ihnen generierte Code lassen sich weitgehend frei konfigurieren. In diesem Artikel beschreiben wir die Konzepte durch die wir Cartridges kombinierbar und konfigurierbar machen, wie Cartridges im MDD-Toolkit umgesetzt wurden und welche Erfahrungen erste Projekte mit dem MDD-Toolkit gemacht haben.

1 Einleitung

Model-Driven Development (MDD) ist ein Ansatz, der stetig wachsenden Komplexität heutiger Software-Systeme zu begegnen. MDD rückt Modelle in das Zentrum des Entwicklungsprozesses und hat zum Ziel, große Teile einer Anwendung zu generieren. Der Einsatz von Modellen und Generatoren ist nicht neu und beweist seinen Wert bei der Erstellung von Anwendungen bei Capgemini sd&m seit über 15 Jahren [Den93, Sch93].

Ein einheitlicher Ansatz zum Einsatz von MDD existierte bisher aber nicht. Modellierungssprachen und Generatoren wurden projekt- und kundenspezifisch immer wieder neu erstellt. Projektmitarbeiter benutzten immer gerade die Technologien, in denen sie sich auskannten. Da die Projekte von Capgemini sd&m sich sowohl in Fachlichkeit als auch in Technologie stark unterscheiden, war es nicht möglich, Generatoren oder Teile von ihnen mit vertretbarem Aufwand wiederzuverwenden.

Das Ziel von Capgemini sd&m Research ist es, dies zu ändern. Wir entwickeln eine ein-

heitliche Methode zum Einsatz von MDD, und schaffen einheitliche Werkzeuge zur Umsetzung von MDD. Die bei Capgemini sd&m am häufigsten eingesetzten Modellierungswerkzeuge Enterprise Architect und MID Innovator müssen dazu ebenso nutzbar sein, wie eventuell vom Kunden präferierte weitere Modellierungswerkzeuge.

Eine Sammlung von Bausteinen (Cartridges) soll es ermöglichen, Generatoren wie in einem Baukasten aus fertigen Teilen zusammenzustecken. Projekte sollen neue Cartridges selbst erstellen und sie in das MDD-Toolkit zurückfließen lassen können, damit diese zukünftigen Projekten zur Verfügung stehen. Die Cartridges müssen umfassend konfigurierbar sein, damit sie in der heterogenen Projektwelt von Capgemini sd&m bestehen können. Ein Software-Architekt soll den generierten Code seinen Vorstellungen und den Anforderungen des Projektes entsprechend anpassen können.

Existierende MDD-Werkzeuge versprechen einige der geforderten Punkte abzudecken. Die beiden Open-Source Toolkits openArchitectureWare (oAW siehe [OAW]) und AndroMDA [And] können beispielsweise sowohl mit Enterprise Architect als auch mit Innovator zusammenarbeiten. Im Umfeld der beiden MDD-Projekte existieren auch einige Cartridges für Java-Frameworks, wie Hibernate, Spring oder EJB. Die Sammlung der Cartridges für oAW ist aber eher spärlich und diese sind nicht aufeinander abgestimmt und nur schwer zu kombinieren. AndroMDA bietet eine umfangreiche Sammlung von Cartridges, die aber auch nicht immer einfach kombinierbar sind. Die Konfiguration der Cartridges und die Anpassung des generierten Codes ist bei beiden Toolkits nur sehr eingeschränkt möglich. Auch bei AndroMDA scheint man erkannt zu haben, dass Wiederverwendung und Anpassung von Cartridges ein wichtiges Thema sind und will dies in Version 4.0 mit einer grundlegend neuen Architektur verbessern [Boh06].

Die Lösung von Capgemini sd&m ist das MDD-Toolkit. Es bietet Methoden, Werkzeuge und Bausteine zum Einsatz von MDD.

Als gemeinsame Basis haben wir dazu die MDD-Werkzeuge der Eclipse-Plattform gewählt, da Eclipse die primäre Entwicklungsumgebung bei Capgemini sd&m ist. Wir stellen eine MDD-Eclipse-Distribution mit openArchitectureWare und allen benötigten Plugins zur Verfügung. Adapter zur Nutzung von Enterprise Architect und MID Innovator sind ebenfalls in der Distribution enthalten.

Einen Mehrwert bietet das MDD-Toolkit vor allem durch die enthaltenen Cartridges. Es bietet eine Auswahl an Cartridges, die die zur Generierung am besten geeigneten technischen Domänen [HSS05] abdecken.

Die Cartridges basieren auf einem Komponenten-Modell eines modularen Generators, das feste Schnittstellen für Cartridges definiert und dadurch eine Komposition der Cartridges ermöglicht.

Zur umfangreichen Konfiguration der Cartridges greifen wir neben Property-Dateien auf den Ansatz von [VG07] zurück, Generatoren über Aspektorientierung (AO) anzupassen. Die Kombination von Property-Dateien und AO, erfüllt alle Anforderungen an die Anpassungsfähigkeit einer Cartridge.

Neben den einheitlichen Schnittstellen vereinfacht auch die Nutzung eines gemeinsamen Architektur-Metamodells das Zusammenspiel von Cartridges. Basierend auf Quasar, der

Quality Software Architecture von Capgemini sd&m (vgl. [Sie04, Sie03]) haben wir das Metamodell einer Anwendung als Grundlage für alle Cartridges erstellt.

Wir beschreiben hier, wie wir diese Grundgedanken im Capgemini sd&m MDD-Toolkit umsetzen. Abschnitt 2 zeigt die Architektur des MDD-Toolkit und seiner Bausteine. In Abschnitt 3 stellen wir die konkrete Umsetzung der Konzepte vor.

2 Die Architektur des MDD-Toolkit

Das MDD-Toolkit bietet alle Komponenten einer MDD-Umgebung als fertige Bausteine an. Abbildung 1 zeigt die Referenz-Architektur einer solchen MDD-Umgebung.

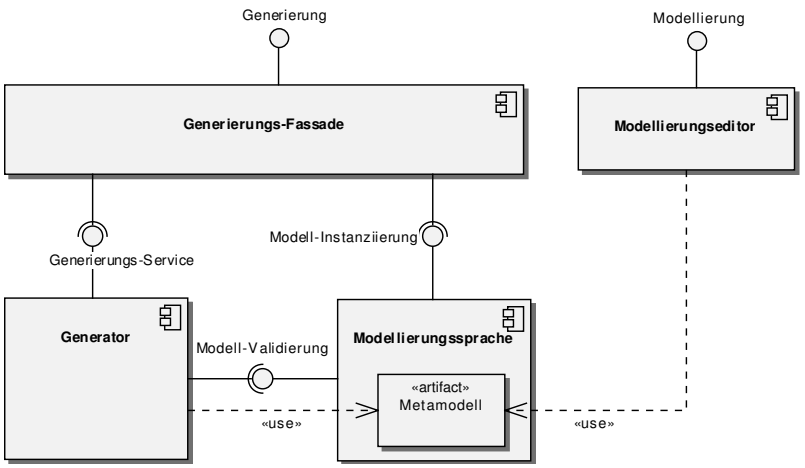

Abbildung 1: MDD-Referenzarchitektur

2.1 Generierungs-Fassade

Die Generierungs-Fassade ist der Berührungspunkt des Anwenders zur MDD-Umgebung. Sie abstrahiert Details von Modellierungssprache und Generator. Der Benutzer hat durch sie eine einfache Schnittstelle, über die er die Generierung anstoßen kann.

2.2 Modellierungssprache und Modellierungseditor

Eine Modellierungssprache definiert im MDD-Toolkit das Metamodell für den Generator und die Modellierungseditoren. Sie bietet auch Schnittstellen zur Instanziierung von auf dem Metamodell basierenden Modellen und zu deren Validierung. Das Metamodell formalisiert die modellierbaren Aspekte einer Anwendung. Das MDD-Toolkit enthält ein allen Cartridges gemeinsames in EMF-ecore realisiertes Metamodell der Quasar-Referenz-Architektur. Es stellt die zentralen Konzepte der Quasar-Referenz-Architektur wie Komponenten, Schnittstellen, Entitäten, Datentypen und Services als Modellierungs-Elemente bereit.

Der Modellierungseditor dient der Erstellung und Pflege der Modelle. Das MDD-Toolkit unterstützt zunächst die Modellierungswerkzeuge Enterprise Architect und MID Innovator. Die Modellierungseditoren benötigen eine externe Notation des Metamodells als Definition der konkreten Modell-Syntax. Dies geschieht im Fall von Enterprise Architect und Innovator durch UML-Profile. MDD-Toolkit liefert UML-Profile für die Bereiche Daten, Komponenten, Schnittstellen, Dienste und Remoting, sowie Modell-zu-Modell-Transformationen um Modelle aus dem Format des Modellierungswerkzeugs in Quasar-Metamodell-konforme Modelle zu überführen. Projekte können auch eigene UML-Profile definieren, die dann über selbstgeschriebene Modell-zu-Modell-Transformationen überführt werden müssen.

2.3 Generator

Der Generator erzeugt aus dem Modell die Ziel-Artefakte. Mit Hilfe des MDD-Toolkit werden Generatoren aus fertigen Cartridges zusammengebaut. Diese Cartridges enthalten jeweils für Teilaspekte der Anwendung die eigentliche Generierungsfunktionalität und werden vom Generator gesteuert.

Um solche Cartridges als Mittel zur Modularisierung und Wiederverwendung zu erstellen, werden sie im MDD-Toolkit als Komponenten mit festen Schnittstellen definiert (siehe Abbildung 2).

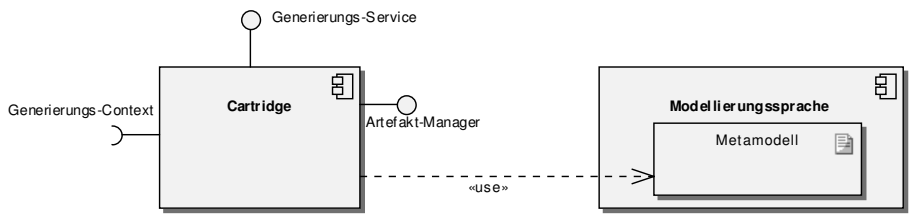

Abbildung 2: Cartridge als Komponente

Die Generierungs-Funktionalität wird über die Schnittstelle Generierungs-Service angeboten. Der Schnittstellen-Operation wird dabei das Modell übergeben. Die Schnittstelle Artefakt-Manager liefert Informationen über die generierten Artefakte, beispielsweise Referenzen auf Java-Klassen. Die Operationen der Generierungs-Context Schnittstelle definieren die Anforderungen der Cartridge an ihre Umgebung. Die Cartridge ist erst betriebsbereit, nachdem diese Schnittstelle versorgt ist. Es ist die Aufgabe des Cartridge-Nutzers, eine geeignete Implementierung bereitzustellen.

Die Schnittstellen Artefakt-Manager und Generierungs-Context stellen das wesentliche Mittel zur Komposition von Cartridges dar. Über Artefakt-Manager liefert eine Cartridge die Informationen, die eine andere Cartridge über die Schnittstelle Generierungs-Context anfordert.

Durch diese Architektur wird die Modularisierung und Wiederverwendung von Generatoren und Cartridges erreicht. Cartridges sind voneinander unabhängig und in unterschiedlichen Generatoren einsetzbar. Der Generator übernimmt die Kopplung der Cartridges (siehe Abbildung 3). Da er selbst als Cartridge gebaut wird, kann auch er in Kompositionen eingesetzt werden.

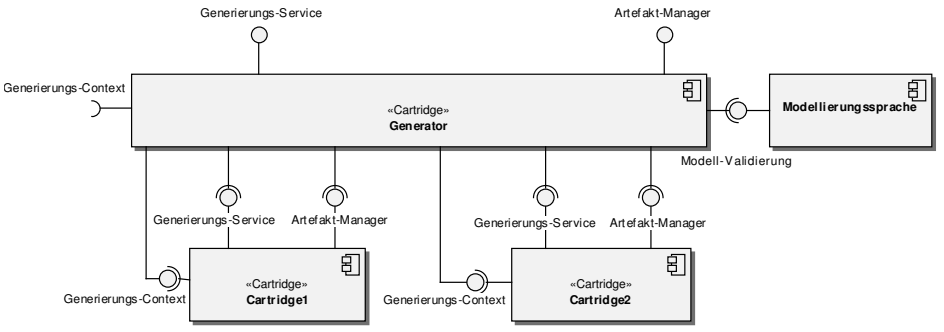

Abbildung 3: Der Generator als Komposition von Cartridges

3 Umsetzung von Cartridges mit openArchitectureWare

3.1 Komposition über Schnittstellen

Eine Cartridge muss die drei Schnittstellen Generierungs-Service, Artefakt-Manager und Generierungs-Context definieren. Leider bietet openArchitectureWare kein Sprachmittel zur Definition von Schnittstellen. Im MDD-Toolkit greifen wir daher auf die oAW-Mittel Workflow und Xtend zurück, um Schnittstellen zu realisieren. Durch ihre Package-Zugehörigkeit werden Workflows und Xtend-Funktionen als Teil der Schnittstelle gekennzeichnet. Abbildung 4 zeigt den Aufbau einer Cartridge mit oAW-Mitteln.

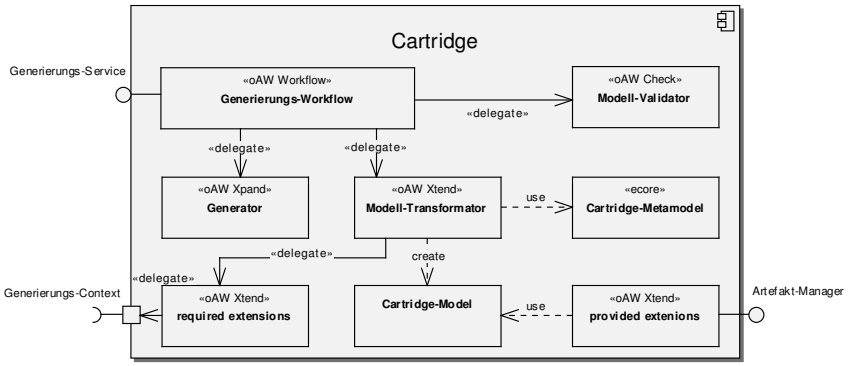

Abbildung 4: Cartridge - Innensicht

Die Schnittstelle Generierungs-Service wird als oAW-Workflow umgesetzt. Der Workflow validiert das übergebene Modell und stößt dann eine Transformation des Modells in ein cartridge-spezifisches Modell an. Dieses Modell basiert auf einem internen Metamodell der Cartridge, das stark an die generierten Artefakte angelehnt ist. Die Abbildung des internen Modells auf den generierten Code durch den Generator ist dadurch nahezu eine 1:1 Transformation. Die Xpand-Templates, die in oAW zur eigentlichen Generierung eingesetzt werden, können daher sehr einfach gehalten werden.

Zur Transformation dient in oAW die Sprache Xtend. Xtend ist eine funktionale Sprache mit einer an OCL angelehnten Syntax. Funktionalität, die die Transformation vom Kontext der Cartridge benötigt, wird als Xtend-Funktion ohne echte Implementierung definiert. Die so definierten Funktionen bilden die Generierungs-Context-Schnittstelle der Cartridge. So lange diese Funktionen nicht vom Cartridge-Nutzer mit einer echten Implementierung versorgt werden, bricht die Transformation in der Cartridge mit einer Fehlermeldung ab.

Die Schnittstelle Artefakt-Manager wird ebenfalls über Xtend-Funktionen realisiert und bietet Zugriff auf das interne Modell der Cartridge.

Die konkrete Umsetzung dieser Schnittstellen zeigen wir am Beispiel der Komposition von Data- und Hibernate-Cartridge im AWK-Generator. Der AWK-Generator ist ein fertiger Generator für die Capgemini sd&m Entwicklungsplattform [LNH07]. Er dient als Proof of Concept für die vorgestellten Konzepte und zeigt beispielhaft die Komposition von Cartridges. Abbildung 5 zeigt den Generierungsablauf des AWK-Generators.

Die Data-Cartridge generiert Entitäten, Datentypen und DataAcessObjects technologieneutral als reine Java-Klassen. Die Hibernate-Cartridge generiert anschließend die notwendigen Mapping-Dateien für Hibernate. Durch diese Trennung könnte die Hibernate-Cartridge in einem konkreten Projekt etwa durch eine Toplink-Cartridge ersetzt werden. Die von der Hibernate-Cartridge des MDD-Toolkits benötigten Informationen über die in der Data-Cartridge generierten Entitäten werden in ihrer Generierungs-Context-Schnittstelle folgendermaßen definiert:

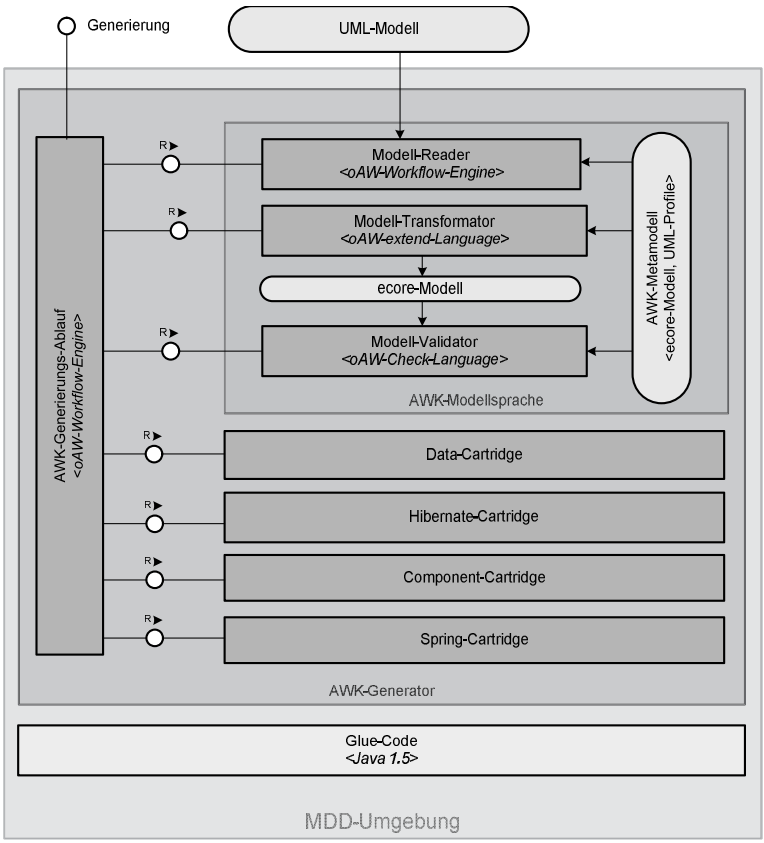

Abbildung 5: AWK-Generator Innensicht

```
/* resolve entity references */
cached String pltRef(Entity this):
                        unsupportedExtension("pltRef");
```

Die Standardimplementierung mittels `unsupportedExtension()` bricht die Transformation mit einer Fehlermeldung ab.

Um die Cartridge zu verwenden muss, der AWK-Generator eine andere Implementierung der Funktion einbinden. Dies geschieht über Aspect-Oriented Programming (vgl. [EFH+08] Kapitel 5.5.10 und 5.6.9). Im Generator wird für die Funktion `pltRef()` ein `around`-Advice angelegt. In diesem wird die Funktione durch den Aufruf der Xtend-Funktion `entityRef()` aus der Artefakt-Manager-Schnittstelle der Data-Cartridge implementiert. Diese liefert die notwendige Information zurück, so dass der Aufruf von `pltRef()` im Transformationsschritt der Hibernate-Cartridge die Referenz auf die gerade behandelte Entität liefert.

```
extension hibernate::gencontext;
extension data::artefactmanager;

/* resolve entity references */
around hibernate::gencontext::pltRef(Entity this):
                        entityRef(this.type);
```

Der AWK-Generator verknüpft also die Schnittstelle Artefakt-Manager der Data-Cartridge mit der Schnittstelle Generierungs-Context der Hibernate-Cartridge und komponiert diese zu einem lauffähigen Ganzen.

3.2 Anpassung des generierten Codes

Neben der Komposition der Cartridges ist der Generator auch dafür zuständig, den von den Cartridges generierten Code an projektspezifische Anforderungen anzupassen.

Zur einfachen Anpassung von Namen hat sich in den Projekten die Definition über Properties bewährt. Jede Cartridge definiert einen festen Satz von Properties mit Standardwerten. Diese können vom Generator überschrieben werden indem eine Datei mit Namen `global.properties` für die globalen Properties und eine cartridge-spezifische Datei (etwa `hibernate.properties`) im Klassenpfad abgelegt werden. Wird in der Datei `global.properties` die Property `pkg.base=com.capgemini-sdm` definiert, so beginnen die Packagenamen aller zu generierenden Artefakte mit dem Präfix `com.capgemini-sdm`.

Um Umfangreiche Änderungen am generierten Code durchzuführen, bietet oAW die Möglichkeit auch in ein Xpand-Generierungs-Template mittels AOP einzugreifen. Dies hat sich jedoch in der Beta-Phase des MDD-Toolkit nicht bewährt. Ein Eingriff mittels AOP benötigt detaillierte Kenntnis der Templates und damit der Interna einer Cartridge. Zusätzlich müssen die Templates für jeden angedachten Eingriff mittels AOP vorbereitet sein, wodurch die Templates unnötig aufgebläht werden.

Als wesentlich flexibler und sauberer herausgestellt hat sich die Möglichkeit, eine Schnittstelle zum internen Modell einer Cartridge zu bieten. Repräsentiert das interne Modell den generierten Code sehr nah, so sind die Templates nur noch eine 1:1 Übersetzung des Modells in den Code. Änderungen am Code können jetzt über Zugriff auf das interne Modell geschehen. Dazu wird als letzer Schritt der M2M-Transformation in einer Cartridge immer die Xtend-Funktion `postProcess()` mit dem internen Modell als Argument aufgerufen. Diese Funktion wird als weiterer Teil der Generierungs-Context-Schnittstelle definiert. Sie liefert in ihrer Standardimplementierung das Modell ohne Veränderung zurück. Um dieses Verhalten zu ändern, muss der Generator wieder AOP einsetzen und dadurch seine eigene Transformation in das Cartridge injizieren.

Diese Möglichkeit wird von Projekten ausgiebig genutzt, um Code zu verändern. Beispielsweise generiert das Data-Cartridge pro modellierter Entität genau ein DataAcessObject (DAO). In einem konkreten Projekt sollte aber pro Package nur ein DAO existieren,

das die Operationen für alle Entitäten des Packages enthält. Dies konnte durch eine zusätzliche Transformation des Cartridge-Modells verändert werden, die DAOs eines Packages wurden vor der eigentlichen Generierung zu einem DAO zusammengezogen.

Einige Transformationen konnten als typische Pattern für einzelne Cartridges erkannt werden. Sie werden im nächsten Release als Pattern-Transformationen in die Cartridges aufgenommen und in der Schnittstelle Artefakt-Manager als Xtend-Funktionen zur Verfügung gestellt.

4 Fazit

Capgemini sd&m Research ist es gelungen mit dem MDD-Toolkit einen Baukasten für die Nutzung modellgetriebener Entwicklung zu erstellen.

Wir stellen Modellierungssprachen, Modellierungseditoren, eine integrierte Generierungsumgebung, sowie Cartridges zum Bau von Generatoren zur Verfügung.

In der Version 1.1 vom Dezember 2008 enthält das Toolkit folgende Bausteine:

MDD-Eclipse-Distribution Eine Eclipse 3.3 Distribution mit allen benötigten MDD-Plugins, insbesondere oAW 4.3, EMF 2.3, M2Eclipse.

Maven-Repository Ein Maven-Repository mit allen benötigten Bibliotheken. Auch die im MDD-Toolkit enthaltenen Cartridges werden als Bibliotheken im Maven Repository zur Verfügung gestellt.

Integration Modellierungseditoren Adapter für die Modellierungswerkzeuge Enterprise Architect und MID Innovator.

Quasar-Metamodell Das Metamodell einer Anwendung basierend auf der Referenz-Architektur Quasar.

UML-Profile UML-Profile, die die Konzepte des Metamodells für die Modellierung in Enterprise Architect und Innovator bereitstellen.

Data-Cartridge Generiert Java-Klassen für fachliche Entitäten und Datentypen, sowie DAO-Klassen mit CRUD-Operationen.

Component-Cartridge Generiert die fachlichen Schnittstellen und den Code-Rahmen für die fachlichen Komponenten.

Hibernate-Cartridge Generiert das OR-Mapping für Hibernate-Persistence.

Spring-Cartridge Generiert Spring-Konfigurationen für die Komposition der fachlichen Komponenten zu einem Anwendungskern. Die Integration der fachlichen Komponenten mit den technischen Komponenten der Ziel-Plattform erfolgt ebenfalls über die generierte Spring-Konfiguration.

EJB2.1-Cartridge Generiert alle Java-Artefakte von Session-Beans, die Deployment-Deskriptoren und einen generischen Service-Locator zum Aufruf der Beans.

EJB3.0-Cartridge Generiert Java-Artefakte für Session-Beans und Entitäten mit Annotationen.

AWK-Generator Generator als Komposition mitgelieferter Cartridges, um alle architektonischen Aspekte eines Anwendungskerns zu generieren.

Durch die Komponentenarchitektur mit festen Schnittstellen können Cartridges beliebig kombiniert und wiederverwendet werden.

Der Chef-Architekt eines Projektes kann durch die weitgehenden Konfigurationsmöglichkeiten der Cartridges die generierten Artefakte anpassen, um an Kundenwünsche und Besonderheiten des Projektes angepasste Ziel-Plattformen und Architekturen zu unterstützen.

Das MDD-Toolkit wurde Ende Juni 2008 veröffentlicht und wird zur Zeit in 2 Großprojekten pilotiert. Ein weiteres Projekt setzt eine Vorabversion des AWK-Generator seit Ende 2007 ein.

Die Projekte waren in der Lage vorhandenen Cartridges, wie das Data-Cartridge und das EJB 2.1 Cartridge zu nutzen und für Ihre Zwecke zu konfigurieren. Eines der Großprojekte hatte die Designphase bereits abgeschlossen und besaß ein umfangreiches Designmodell in Enterprise Architekt (EA) mit eigenen UML-Profilen. Für dieses Projekt konnte die Modell-zu-Modell-Transformation im EA-Adapter innerhalb eines Tages so angepasst werden, dass das Modell zur Generierung nicht verändert werden musste.

Eines der Projekte benötigte für die Zielplattform EJB 3.0 eine neue Cartridge. Diese wurde mit 2 Bearbeiterwochen Aufwand erstellt und ist seit Release 1.1 fester Bestandteil des MDD-Toolkit.

In zukünftigen Versionen des MDD-Toolkits sollen weitere Cartridges ausgeliefert werden. Insbesondere werden zur Zeit für den Client-Anteil einer Anwendung Modellierungssprachen und Cartridges entwickelt, die im nächsten Release Mitte 2009 ausgeliefert werden.

Literatur

[And] AndroMDA. http://www.andromda.org.

[Boh06] Matthias Bohlen. The AndroMDA 4.0 architecture. http://galaxy.andromda.org/index.php?option=com_content&task=blogsection&id=10&Itemid=78, 2006.

[Den93] Ernst Denert. Dokumentenorientierte Software-Entwicklung. In *Informatik-Spektrum 16*, Seiten 159–164, 1993.

[EFH+08] Sven Efftinge, Peter Friese, Arno Haase, Clemens Kadura, Bernd Kolb, Dieter Moroff, Karsten Thoms und Markus Völter. *openArchitectureWare User Guide, Version 4.3*, 2004-2008.

[EFK08] Sven Efftinge, Peter Friese und Jan Köhnlein. Best Practices für modellgetriebene Softwareentwicklung. *Java Magazin*, 5 2008.

[HO07] Martin Haft und Bernd Olleck. Komponentenbasierte Clientarchitektur. In *Informatik-Spektrum, Band 30, Heft 3*. Gesellschaft für Informatik, Juni 2007.

[HSS05] Bernhard Humm, Ulf Schreier und Johannes Siedersleben. Model-driven development - hot spots in business information systems. In A. Hartmann und D. Kreische, Hrsg., *Proceedings of the European Conference on Model Driven Architecture (ECMDE-FA 2005), Lecture Notes in Computer Science 3748*, Seiten 103–114, Berlin Heidelberg, 2005. Springer-Verlag.

[Hum04] Bernhard Humm. Technische Open Source Komponenten implementieren die Referenzarchitektur Quasar. In Olaf Zukunft Helmut Eirund, Heinrich Jasper, Hrsg., *ISOS 2004 - Informationsysteme mit Open Source, Proceedings GI-Workshop*, Seiten 77–87, Berlin, 2004. Gesellschaft für Informatik.

[LNH07] Martin Lehmann, Robert Nikonowicz und Martin Haft. Eine Entwicklungsplattform für die architekturzentrierte Erstellung betrieblicher Anwendungssysteme auf Basis von Open-Source. In *GI Jahrestagung 2007, Workshop ISOS*, 2007.

[OAW] openArchitectureWare. The leading platform for professional model-driven software development. http://www.openarchitectureware.org.

[Sch93] Gero Scholz. Maßgeschneiderte Software-Generatoren. In *Proceedings ONLINE 1993 Congress VI - C636*, 1993.

[Sie03] Johannes Siedersleben. Quasar: Die sd&m Standardarchitektur. Teile 1 und 2. Bericht, sd&m Research, 2003.

[Sie04] Johannes Siedersleben. *Moderne Softwarearchitektur*. dpunkt Verlag, Bonn, 2004.

[SVE07] Thomas Stahl, Markus Völter und Sven Efftinge. *Modellgetriebene Softwareentwicklung. Techniken, Engineering, Management*. d.punkt Verlag, 2. Auflage, Mai 2007.

[VG07] Markus Völter und Iris Groher. Product Line Implementation using Aspect-Oriented and Model-Driven Software Development. In *11th International Software Product Line Conference*, Seiten 233–242. IEEE Computer Society, 2007.