

# MODELLGETRIEBENE ENTWICKLUNG MIT EINER SERVICEORIENTIERTEN ARCHITEKTUR ZUR HOST-INTEGRATION

Model Driven Architecture wird mittlerweile in vielen Projekten erfolgreich eingesetzt. Informationen zentral in einem Modell zu pflegen, bietet viele Vorteile, und daraus entsprechende Code-Artefakte zu erzeugen, ist ebenso sinnvoll. Gerade im Java-Enterprise-Umfeld (J2EE) bietet es sich an, viele Artefakte generativ aus einer einzigen Quelle zu erzeugen; dort müssen Implementierungsklassen, Interfaces, Deskriptoren und Datenbankschemata zueinander passen. Durch die Vielzahl der abhängigen Teile hat sich MDA mittlerweile im J2EE-Umfeld verbreitet und etabliert. Diesen Ansatz aber auch für die Entwicklung auf Host-Systemen einzusetzen, ist derzeit weniger verbreitet. Aber auch dort gibt es abhängige Teile – Unterprogramme (UPROs), die Copystrecken verwenden; HPROs, die UPROs aufrufen, etc. So bietet sich auch hier ein großes Potenzial für einen modellgetriebenen, generativen Ansatz. innoQ hat in mehreren Projekten Erfahrungen gemacht, welche Besonderheiten beim Einsatz von modellgetriebener Entwicklung (Model Driven Development, MDD) für Host-Systeme zum Tragen kommen. Die gemeinsamen Erfahrungen werden beispielhaft dargestellt.

## Einleitung

Eine moderne J2EE-Architektur bedingt eine Vielzahl an Dateien für jedes modellierte Geschäftsobjekt und jeden Anwendungsfall, alleine auf dem Applikationsserver. Der Einsatz von MDD im Allgemeinen und Code-Generatoren im Speziellen ermöglicht die automatische Erzeugung zumindest eines großen Teils dieser Dateien. Hierzu werden aus den Informationen des Design-Modells mit Hilfe von Abbildungsvorschriften (in Form von Templates) die Dateien der Implementierung erzeugt. Das entbindet die Anwendungsentwicklung weitgehend von der eintönigen und fehleranfälligen manuellen Erstellung gleichförmiger Dateien. Dies wiederum spart massiv Entwicklungszeit und erhöht die Qualität. Daher ist MDA in Projekten mit J2EE-Architekturen inzwischen sehr weit verbreitet.

Wenn es darum geht, Host-Systeme in eine heterogene Infrastruktur mit z.B. Applikationsserver und Web-Clients zu integrieren, ist der Einsatz von MDA bislang weniger üblich. Je nach eingesetzter Middleware bietet sich aber auch hier ein lohnenswertes Potenzial für den Einsatz

von MDA (Model Driven Architecture) bzw. MDD (Model Driven Development), die hier synonym verwendet werden. Anhand eines realen Einsatzszenarios soll gezeigt werden, wie eine weitgehend automatisierte Integration von Host-Systemen in eine J2EE-Umgebung möglich ist.

## Ansatz

Ausgangspunkt in einem Großprojekt eines deutschen Finanzdienstleisters war das geplante Re-Engineering der Altsysteme. Da dieses in einer komplexen Anwendungslandschaft nicht ad hoc funktioniert, musste

### Akronyme

CSV	Comma Separated Value
HPRO	Hauptprogramm
IDL	Interface Definition Language
JSP	Java Server Pages
J2EE	Java 2 Enterprise Edition
MDA	Model Driven Architecture
MDD	Model Driven Development
ORB	Object Request Broker
UML	Unified Modeling Language
UPRO	Unterprogramm
XMI	XML Metadata Interchange

## ▶ die autoren



Frank Bruch

(E-Mail: frank.bruch@innoq.com) ist Senior Consultant bei der innoQ Deutschland GmbH. Er beschäftigt sich mit der rationellen Softwareentwicklung unter Einsatz von MDA und Code-Generatoren und arbeitet als Berater in Projekten an der Konzeption und Umsetzung verteilter Architekturen.



Marcel Tilly

(E-Mail: marcel.tilly@innoq.com) ist Senior Consultant bei der innoQ Deutschland GmbH. Sein Arbeitsschwerpunkt ist derzeit die Konzeption und Umsetzung von Service-orientierten Architekturen sowie der Einsatz generativer Softwareentwicklung in Projekten.

eine Architektur geschaffen werden, die eine schrittweise Migration auf eine neue Zielarchitektur und Technologie gestattet. Hierzu sollten die Altsysteme, die allesamt auf dem Host liefen, zunächst gekapselt und über einen J2EE-Server „ansprechbar“ gemacht werden. Durch die Kapselung bekommen die einzelnen Host-Programme nach außen eine Service-Schnittstelle. Wenn die entstandenen Services dann anwendungsübergreifend, sowohl als Modellprimitive als auch mit ihren Implementierungen, wiederverwendet werden, würde man heute von einer „serviceorientierten Architektur“ sprechen.

Außerdem sollte modellgetrieben entwickelt werden, damit auf Basis des Modells und unter Verwendung eines geeigneten Generators ein Wechsel auf eine andere Technologie möglich ist. Denn ein großer Vorteil des modellgetriebenen

Ansatzes ist, dass der Aufwand bei Architekturänderungen zentral gesteuert werden kann. Wird dabei ein Generator verwendet, der template-basiert arbeitet, kann ein großer Teil der Migrationsaufwände einfach durch einen Wechsel der Templates reduziert werden.

## Anbindung

Die Kommunikation zwischen Applikationsserver und Host erfolgt über CORBA-Schnittstellen. Als Übertragungsprotokoll kommt jedoch kein binäres IIOP, sondern XML, definiert durch einen proprietären ORB, zum Einsatz. Dieser Ansatz erfordert für jeden Service eine ganze Reihe zu erzeugender Dateien. Dies sind auf dem J2EE-Server:

- Transferobjekt (Java-Bean),
- CORBA-Stub etc.,
- XML-Builder,
- XML-Parser.

Analog dazu auf dem Host:

- Copystrecken<sup>1)</sup>,
- CORBA-Skeleton etc.,
- XML-Builder,
- XML-Parser.

Wenn die XML-Dateien zu irgendeinem Zeitpunkt, und sei es nur während der Entwicklung, validiert werden sollen, kommt z.B. noch eine DTD bzw. ein XML-Schema hinzu.

Die Nutzung der Standards CORBA und XML ermöglicht den Aufruf der „Host-Services“ auch von bei der Implementierung nicht besonders berücksichtigten oder geplanten Plattformen aus. Die Datenstrukturen, Copystrecken, werden jeweils auf XML abgebildet, um die Host-Anbindung zu erreichen. Für den Aufruf wird eine passende XML-Nachricht verschickt und als Ergebnis eine XML-Nachricht empfangen.

Die XML-Strukturen, die als Nachrichten zum Host geschickt werden, entsprechen inhaltlich den Transferobjekten [PAT], die von der Java-Schnittstelle zum Client geschickt werden, und diese entsprechen wiederum direkt den Copystrecken, die auf dem Host verarbeitet werden.

Für fast alle benötigten Artefakte gilt, dass Änderungen an einem Artefakt meist

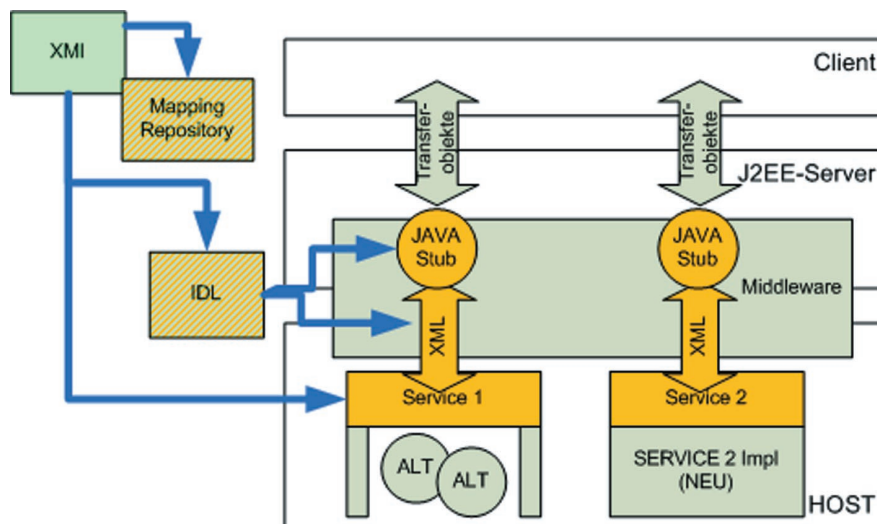


Abb. 1: Architektur und Generierung

weitere Anpassungen an anderen Stellen bedingen. Deshalb bietet es sich an, nur an einer zentralen Stelle – im Modell – diese Änderung vorzunehmen, und über einen generativen Prozess diese Änderung in alle abhängigen Teile zu propagieren [TRE].

## Modellierung

Für die modellgetriebene Entwicklung müssen die Service-Schnittstellen nach der fachlichen Analyse jeweils modelliert werden, d.h. sowohl die Operationen, die ausgeführt werden können, als auch die Datenstrukturen, die transferiert werden.

Grundsätzlich ist auf den sorgfältigen Einsatz eines unternehmensspezifischen UML-Profiles zu achten, also die eindeutige Definition und konsistente Verwendung von Stereotypen und Tagged Values zur Kennzeichnung bestimmter Typen von Modellelementen. Das UML-Profil ist nämlich die Basis für die modellgetriebene Generierung. Ein MDA-konformer Code-Generator entscheidet aufgrund dieser Angaben, für welche Modellelemente welche Artefakte wie generiert werden müssen. Dazu gehören z.B. Stereotypen für Komponenten und Services sowie üblicherweise eine Anzahl Tagged Values, um u.a. bestimmte Sonderfälle zu kennzeichnen. Zwar könnte man nun über Tagged Values auch plattformspezifische Daten wie z.B. Implementierungsnamen ins Design-Modell einpflegen, hiervon ist jedoch abzuraten. Die Vermischung von plattformunabhängigen Design- und plattformabhängigen Implementierungsinformationen auf Modellebene verringert Flexibilität und Wartbarkeit. Für die Ablage der plattformspezifischen Zusatzinformationen, die bei der Abbildung vom Design zur Implementierung benötigt wer-

den, gibt es bessere Möglichkeiten, die weiter unten beschrieben werden.

## Generierung

Der eingesetzte Generator, iQgen [IQG], ist template-basiert und ermöglicht somit eine einfache und schnelle Einflussnahme auf die generierten Artefakte. Die Übergabe der Design-Informationen aus dem UML-Modell an den Generator erfolgt über das standardisierte XML-Export-Format XMI. Der Generator bietet eine Schnittstelle, um in den Templates, die als JSP-Dateien erstellt werden, auf die exportierten Modellelemente zuzugreifen. Selbstverständlich ist es möglich, ein iterativ inkrementelles Vorgehen zu unterstützen. Bereits von einem früheren Generierungslauf vorhandene, manuell angepasste Artefakte werden mit neu generierten Artefakten verschmolzen, indem so genannte geschützte Bereiche bei einer erneuten Generierung aus dem alten in das neue Generat übernommen werden. **Abbildung 1** zeigt die Zusammenhänge der einzelnen Elemente bei der Generierung. Die blauen Pfeile symbolisieren jeweils die Generierung von Artefakten.

Um die CORBA-Artefakte für beide beteiligten Plattformen mit entsprechenden ORB-spezifischen Generatoren erzeugen zu können, muss zunächst eine IDL-Beschreibung der entsprechenden Schnittstelle erstellt werden. Diese wird sinnvollerweise direkt aus dem Design-Modell generiert. Damit der Host mit den XML-Strukturen arbeiten kann, basiert die Generierung der Service-Schnittstellen ebenfalls auf dem XMI-Modell-Export. Somit ist es vorteilhaft, die IDL-Beschreibung und die Service-Schnittstelle vom selben Generator erzeugen zu lassen. ▶

<sup>1)</sup> Der Begriff Copystrecke bzw. Copybook bezeichnet Datenstrukturdefinitionen in der Cobol-Programmiersprache. Eine Copystrecke bzw. ein Copybook entspricht konkret einer Datei mit Variablendefinitionen.

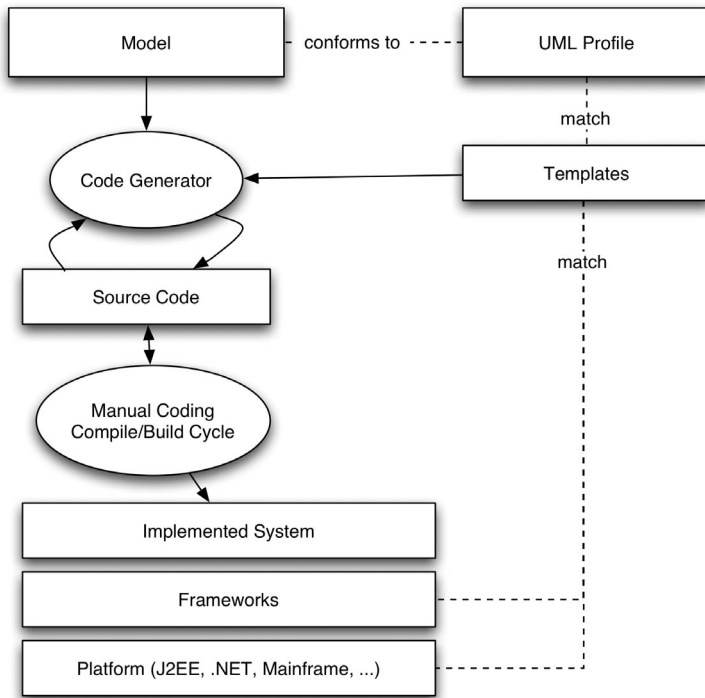


Abb. 2: Generierung mit iQgen

Der allgemeine Prozess der Generierung mit einem template-basierten Code-Generator ist in **Abbildung 2** schematisch dargestellt.

## Repository

Ein kritischer Punkt bei der Abbildung der Informationen aus dem Design-Modell in die Implementierung sind Typen und Namen. Mit Typen sind hier sowohl fachliche Datenstrukturen bestehend aus mehreren atomaren Datenelementen als auch die technischen Datentypen der atomaren Datenelemente selbst gemeint. Die Datenstrukturen gehören, bei entsprechender sorgfältiger Analyse und Design, zur zentralen Fachlichkeit eines Unternehmens. Um eine einheitliche Verwendung und damit Kompatibilität verschiedener Schnittstellen zu gewährleisten, sollte also bei jeder Nutzung eines entsprechenden fachlichen Typs dieselbe Datenstruktur verwendet werden. Hierfür reicht unter Umständen die Definition der Struktur im fachlichen UML-Modell nicht aus.

Eine ähnliche Problematik gibt es bei den technischen Datentypen. Hier muss sichergestellt werden, dass eine einheitliche Abbildung der im UML-Modell plattformunabhängig definierten Typen auf die konkrete Zielplattform erfolgt. Ähnliches gilt für die Namen. Die in einem UML-Modell fachlich getrieben formulierten Namen müssen für die Verwendung in einem Cobol-Programm im Normalfall gekürzt werden. Auch braucht es eine kla-

re Vorschrift zur Definition von Programmnamen.

Sowohl atomare Datentypen als auch Namen lassen sich – zumindest theoretisch – algorithmisch eindeutig ermitteln. In der Praxis scheitert ein solcher Ansatz aber schnell an unabhängigen Vorgaben wie z. B. bereits verwendeten Namen oder Richtlinien bzw. Prozessen, die sich nicht durch einen automatischen Algorithmus abbilden lassen.

Eine mögliche Lösung ist die Übernahme der technischen Namen und Typen für die Zielplattform in das UML-Modell. Damit wäre dieses aber nicht mehr plattformunabhängig. Auch ließe sich nur schwer sicherstellen, dass z. B. kein Name doppelt verwendet wird. Eine bessere Variante ist die Einrichtung eines unternehmensweiten Repository mit Schnittstellen sowohl für die manuelle Pflege als auch den automatischen Zugriff während der Generierung. Im einfachsten Fall kommt man hier vielleicht noch mit einem Satz Property- bzw. CSV-Dateien aus, die in eine Versionsverwaltung eingepflegt werden. Skalierbarer und mächtiger ist aber eine Datenbanklösung.

In unserem Beispielprojekt wird sowohl für die Generierung der IDL-Beschreibung als auch zur Generierung der Service-Schnittstellen zusätzlich zum XMI-Modell-Export auf ein Mapping-Repository zugegriffen. Grundsätzlich werden in diesem Mapping-Repository technische Informationen gepflegt, die im Modell

nicht gepflegt werden können oder sollten. Konkret wird hier abgelegt, welcher fachliche Datentyp auf welchen IDL-Typen und auf welchen Cobol-Typen abgebildet wird. Außerdem werden in dem Mapping-Repository die technischen Namen für die Generierung für das Host-System gepflegt. Durch diese Auslagerung der technischen Informationen in das Mapping-Repository ist es gelungen, das Modell plattformunabhängig zu halten.

Beide Repository-Varianten, sowohl der Zugriff auf zusätzliche Dateien als auch eine zentrale Datenbank, sind mit iQgen möglich und bereits mehrfach erfolgreich umgesetzt worden.

Zusammengefasst nochmals die zentralen Entwicklungsschritte

- Modellierung der Service-Schnittstellen und Datenstrukturen,
- Erstellung eines Repository mit Abbildungen von Typen und Namen,
- Entwicklung von Generator-Templates für die Zielplattformen,
- Generierung aus dem Modell unter Nutzung des Repository.

## Fazit

Der gezeigte Ansatz der serviceorientierten Anbindung von Host-Systemen hat sich bereits bei mehreren Kunden als sinnvoll und tragfähig herausgestellt. Zudem ist es gelungen, über einen modellgetriebenen Ansatz sehr anwendungsspezifisch zu modellieren, ohne dabei die Zielplattform zu berücksichtigen. Dadurch war es möglich, sowohl die für die Host-Implementierung notwendigen Datenstrukturen als auch die zur Anbindung an die J2EE-Plattform notwendige Implementierung generativ zu erzeugen. Somit konnte Entwicklungszeit gespart, die Produktivität erhöht und gleichzeitig Flexibilität in Bezug auf die Zielarchitektur erhalten werden. ■

## Links

[IQG] iQgen, MDA-Generator, siehe: <http://www.innoq.com/iqgen>

[PAT] Core J2EE Patterns - Transfer Object, siehe: <http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html>

[TRE] A. Ditze, P. Ghadir, S. Tilkov, Trennung von fachlicher und technischer Komponentenarchitektur im Sinne der MDA, siehe: [http://www.sigs-datacom.de/sd/publications/pub\\_article\\_show.htm?&AID=1105&TABLE=sd\\_article](http://www.sigs-datacom.de/sd/publications/pub_article_show.htm?&AID=1105&TABLE=sd_article)