Model Driven Architecture –

Ein neuer Ansatz für Wiederverwendung?

Markus Völter, voelter@acm.org, www.voelter.de
Eberhard Wolff

Es gibt in der Softwareentwicklung einen heiligen Gral: die Wiederverwendung. Jede neue Technologie die etwas auf sich hält, hat am Anfang versprochen, jetzt endlich könne man dieses Ziel erreichen. Objektorientierung hatte hier Vererbung als Mechanismus und die Komponententechnologie hat sich sowieso nur dieses Thema auf die Fahnen geschrieben und in den Mittelpunkt ihrer Anstrengungen gestellt.

Ein zweiter Aspekt, der die Softwareentwicklung stark geprägt hat, ist die „Separation of Concerns", auf Deutsch etwa Trennung der Zuständigkeiten. Dieser Begriff geht auf Dijkstra zurück, einem der bedeutendsten Informatiker überhaupt: Es geht darum, verschiedene, konzeptionell unabhängige Teile eines Systems auch unabhängig voneinander zu implementieren.

Bei typischen Business-Anwendungen bedeutet dies, dass die Business Logik von den technischen Anteilen des Systems (also Transaktionshandling, Sicherheit, Ausfallsicherheit, Skalierbarkeit, ...) getrennt implementiert werden sollen. Dies hat verschiedene Vorteile: Zum einen lassen sich vor allem die technischen Anteile in Form von Applikationsservern wiederverwenden, zum anderen können sich Entwickler auf das konzentrieren was sie am besten können: die einen auf die Businesslogik, die anderen auf die Implementierung der technischen Aspekte. Davon verspricht man sich natürlich eine entsprechende Produktivitätssteigerung, weil man als Anwendungsentwickler die technische Infrastruktur, also einen Applikationsserver, einfach zukaufen kann.

Allerdings gibt es bei diesem Ansatz ein zentrales Problem: Damit die Businesslogik im Rahmen eines Applikationsservers läuft, muss man sie richtig in sogenannten Komponenten verpacken. Damit eine Komponente im Applikationsserver laufen kann, muss sie bestimmte Schnittstellen implementieren und über andere, wohl definierte Schnittstellen auf ihre Umgebung zugreifen. Nun sind zwar diese Schnittstellen standardisiert, aber nur bezüglich einer Art von Applikationsserver: J2EE Applikationsserver haben ein anderes Komponentenmodell als z.B. COM+ oder auch CORBA/CCM. Wiederverwendbare Businesslogik in Form von Komponenten ist also immer nur auf einer bestimmten Komponenteninfrastruktur lauffähig – und die verfügbaren Technologien entwickeln sich bekanntlich schnell weiter. Das zentrale Problem besteht also darin, dass man Businesslogik nicht ohne weiteres unabhängig von der zugrundeliegenden technischen Infrastruktur beschreiben und implementieren kann.

Zur Lösung dieses Problems wird nun die Model Driven Architecture ins Rennen geschickt. Der Vorschlag kommt von der OMG (Object Management Group); alleine dieser Name ist schon Grund genug, sich mit dem Thema zu beschäftigen. Mit CORBA und der UML hat die OMG in zwei sehr unterschiedlichen Bereichen bereits nicht nur technisch elegante Ansätze vorgestellt, sondern ihnen auch zur Marktdominanz verholfen: CORBA im Bereich der Middleware für verteilten Systeme und der UML im Bereich der Modellierung objekt-orientierter Systeme. Hinter der OMG stehen 800 Mitglieder, wobei praktisch alle großen Namen aus der IT Industrie vertreten sind. Dabei verwendet die OMG für ihre Vorhaben einen offenen Standardisierungsprozess und ist somit herstellerunabhängig.

Die Idee der MDA ist es, Businesslogik in Form von abstrakten Modellen zu spezifizieren, und diese dann zum größten Teil automatisiert auf verschiedene technische Plattformen zu mappen.

Ein Modell ist dabei eine formale Repräsentation eines Teils einer Funktionalität. Üblicherweise werden solche Modelle mittels UML beschrieben, es kann aber auch eine Schnittstellendefinition oder normaler Programmcode sein. Entscheidend ist nur, dass das Modell formalisiert ist und damit nicht nur eindeutig beschrieben, sondern auch durch Tools verarbeitet werden kann.

Um nicht bei jedem Projekt mit der Modellierung ganz vor vorne anfangen zu müssen beruht jedes Businessmodell auf einem sogenannten Core Model. Es sind verschiedene Core Models vorgesehen, die Bereiche wie medizinische Anwendungen, Anwendungen im Telekomunnikationssektor oder im Finanz- oder Versicherungswesen abdecken. Ein Core Model definiert dabei die Grundabstraktionen eines Systems der entsprechenden Domäne.

Der Entwickler beginnt also mit der Modellierung der Businesslogik eines Systems unter Zuhilfenahme eines Core Models. Da dieses Modell nicht von einer konkreten Implementierungstechnologie oder Programmiersprache abhängt, spricht man dabei von einem Plattform Independent Model (PIM). Der eigentliche Clou bei der MDA ist nun, dass dieses PIM mehr oder weniger automatisch in ein Modell überführbar ist, welches das PIM auf einer bestimmten technischen Plattform (also z.B. EJB oder COM+) lauffähig macht – ein sogenanntes Plattform Specific Model (PSM). Weitere Abbildungen, zum Beispiel um die Features eines bestimmten EJB Applikationsservers zur Performanzsteigerung ausnutzen zu können, sind auch möglich, man spricht dann von einem PSM nach PSM Mapping.

Natürlich sind solche Abbildungen nicht trivial. Das Zielmodell muss dabei schließlich mehr Details enthalten als das Ausgangsmodell. Dazu definiert die MDA Abbildungsregeln, die ein PIM auf ein PSM, oder ein PSM auf ein anders PSM abbilden. Um die Spezifikation dieser Regeln erst zu ermöglichen, sind die Core Models nötig. Ein Core Modell kann zum Beispiel das Konzept eines „Business Prozess“ oder einer „Business Entity“ einführen, auf der die Prozesse arbeiten. Für Instanzen dieses Typs werden dann

Abbildungsregeln definieren. Diese Regeln sind abhängig von der verwendeten Technologie und können unterschiedliche Implementierungen der abstrakten Konzepte wählen, je nach dem welche Performance und welcher Einsatzkontext verwendet werden soll. Es kann daher also möglich sein, dass der Entwickler dem System bei der Abbildung von PIM auf PSM etwas unter die Arme greifen muss. Dies kann entweder dadurch geschehen, dass er explizit angibt, wie was gemappt werden soll, oder er erweitert die Business Entity im PIM um Attribute, die dem System diese Entscheidung ermöglichen.

Also lässt sich die Anwendungsentwicklung von der Analyse über Design und Implementierung (ggfs. sogar Test) konsistent mittels Modellen beschreiben. Um die Konsistenz dieser verschiedenen Modelle sicher zu stellen, müssen natürlich Änderungen über die verschiedenen Abstraktionsebenen verfolgbar sein, so dass man zum Beispiel herausfinden kann, welche Änderung in einem PIM-Analysemodell zu welchen Änderungen in einem PSM geführt hat. Dies ist schon deshalb wichtig, damit nur jene Teile des Systems neu generiert werden, die tatsächlich neu generiert werden müssen.

Es dürfte klar sein, dass die Ziele der MDA nur mit Hilfe ausgereifter Tools erreichbar sind. Abgesehen von der Möglichkeit, UML Modelle zu erstellen, muss dieses Tool auch in der Lage sein, mit Metamodellen (also z.B. den Core Models) umzugehen und diese auch anzupassen. Das Tool muss dazu auf der OMG MOF (Meta Object Facility) basieren – derzeit #beileibe keine Selbstverständlichkeit. Neben der MOF-Unterstützung ist vor allem auch ein entsprechendes Repository notwendig. In diesem Repository müssen alle Metamodelle, Modelle und Instanzen gespeichert werden. Idealerweise sollte es auch Versionsmanagement unterstützen, sowie die Abgängigkeiten zwischen den verschiedenen Modellen und deren Elementen modellieren. Als Format für ein solches Repository bietet sich XMI an, das XML-basierte Format für auf der MOF basierende Datenstrukturen.

Was bedeutet die MDA nun für den Entwickler? Anwendungen können zunächst völlig abstrakt von einer konkreten Middleware entwickelt werden. Dadurch ist man unabhängig von ein konkreten Middleware und dies ist ein deutlicher Vorteil, denn die Idee, dass sich ein Unternehmen auf **eine** Middleware festlegt, hat sich als nicht praxistauglich herausgestellt. Während ohne MDA eine Wiederverwendung über verschiedene Middleware Technologien nicht möglich ist, kann man relativ leicht in einem MDA Modell Teile eines anderen Modells wiederverwenden (funktionale Wiederverwendung). Da die Business Logik auf einer Ebene modelliert ist, die von der technischen Realisierung unabhängig ist, wird auch die Wiederverwendung neuen Middleware-Platformen erleichtert: Endlich kann die Businesslogik von einem EJB System in ein COM+ oder .NET System übernommen werden. Es lässt sich außerdem ein System mit viel geringerem - in der Theorie sogar ohne - Portierungsaufwand auf eine andere Middlewareplattform portieren. Damit kann man die Lebensdauer der Systeme deutlich verlängern: Ein kompletten Umschreiben aus rein technischen Gründen ist nicht mehr notwendig.

Um die MDA mit bereits vorhandenen Systemen einsetzen zu können, muss entsprechend das PSM, in diesem Fall meistens konkreter Programmcode, in ein PIM überführt werden, es muss also eine Art Reverse Engineering vorgenommen werden. Dies ist zwar konzeptionell Teil der MDA, läßt sich aber sicherlich nicht automatisch durchführen. Damit lassen sich die Vorteile der MDA also nur ohne zusätzlichen Aufwand nutzen, wenn man in einem Projekt von Anfang an nach diesem Paradigma gearbeitet hat.

Letztendlich geht man mit der MDA aber einen großen Schritt in Richtung vollständiger Separation of Concerns: Der Business Entwickler muss noch nicht einmal mehr den konkreten Code sehen, sondern kann bei seinen

Modellen bleiben, während Plattformspezialisten die technische Infrastruktur und die Umsetzung der abstrakten Modelle auf diese Infrastruktur zur Verfügung stellen.

Zusammenfassend ist MDA ein guter Ansatz trotz der Heterogenität in den jetzigen IT Welten der Unternehmen Wiederverwendung und Konzentration auf die Business Logik technisch möglich zu machen. Jedoch gehen damit deutlich gesteigerte Anforderungen an die verwendeten Werkzeuge einher. Und hier hat die Entwicklung erst begonnen, so dass die Funktionalität dieser Tools im Moment eher beschränkt ist. Außerdem ist dieses neue Vorgehen nur dann effizient implementierbar, wenn man MDA möglichst an vielen Stellen im Unternehmen plaziert und diese Technologie wohl organisiert einführt - sie ist ein deutlicher Unterschied zu den bisherigen Ansätzen. Letztendlich ist Wiederverwendung aber auch und gerade eine Frage der Organisation und des Vertrauens unter den Entwicklern.

Die heiligen Gral der Wiederverwendung ist also immer noch schwer zu erreichen, aber es gibt einen neuen Ansatz, der zudem den Vorteil der höherer Abstraktion bietet – der klassischen Methode für das bewältigen von Komplexität.