Model Driven Architecture

Martin Kempa · Zoltán Ádám Mann

Bei der Model Driven Architecture (MDA) bilden Modelle die zentralen Elemente des Softwareentwicklungsprozesses. Ziel ist es, plattformspezifische Modelle möglichst automatisiert aus plattformunabhängigen Modellen abzuleiten. Dadurch soll der Aufwand der Softwareentwicklung verringert und die Adaptierung an neue Technologien erleichtert werden.

Einführung

Programmcode als textuelle Repräsentation von Software ist die Grundlage, um lauffähige Programme zu erzeugen. Soll die Funktionalität einer Softwarekomponente erweitert werden, so geschieht dies in der Regel durch die Änderung des Quellcodes. Dem gegenüber dienen Modelle als graphische Repräsentation, wie sie beispielsweise in der UML (Unified Modeling Language [7]) notiert werden, im Softwareentwicklungsprozess hauptsächlich zur Spezifikation und Dokumentation. Nur partiell erfolgt eine Generierung von Programmcode aus solchen Modellen, der dann noch um wesentliche Teile manuell zu ergänzen ist. In seltenen Fällen werden Änderungen auf der Quellcodeebene auch auf der Modellebene nachgeführt.

Mit der Model Driven Architecture (MDA) liegt nun ein Standardisierungvorschlag der Object Management Group (OMG) vor, der die Repräsentation von Software von der Programmcodeebene auf die Modellebene heben möchte [2]. Um die Komplexität auf Modellebene zu reduzieren, werden Modelle und Plattformen verschiedener Abstraktionsebenen unterschieden, die aufeinander aufbauen. Die Grundidee des Ansatzes ist nicht neu, sondern folgt dem Grundsatz, dass die Spezifikation einer Komponente unabhängig von der technischen Umsetzung zu beschreiben ist. Neu ist, dass mit MDA aufeinander abgestimmte Spezifikationen und ein Konzept für die modellgetriebene Softwareentwicklung existieren. Dabei kann die Realisierung eines plattformunabhängigen Modells durch Wahl einer Plattform, also der konkreten technischen Umsetzung, teilweise oder vollständig automatisiert erfolgen. Modelle werden durch die Auswahl von Plattformen in weniger abstrakte Modelle transformiert bis letztendlich ausführbarer Programmcode entsteht. Änderungen an der Software erfolgen nun nicht mehr im Programmcode, sondern, so ist das Ziel von MDA, in einem der Modelle. MDA soll die Portierbarkeit und Wiederverwendung von Modellen und dadurch der Software verbessern. Erhofft wird so eine Beschleunigung und damit Kostensenkung in der Entwicklung von Software.

Modelle

Mit der MDA wird das Modell zum zentralen Element des Softwareentwicklungsprozesses. Neben der Spezifikation und Dokumentation von Software werden formale Modelle nun auch zur Definition der Architektur, des Designs und der Implementierung genutzt. Sie beschreiben nicht nur den statischen

DOI 10.1007/s00287-005-0505-2

Dr. Martin Kempa, Zoltán Ádám Mann
sd&m AG
software design & management
Carl-Wery-Str. 42, 81739 München
E-Mail: {martin.kempa, zoltan.mann}@sdm.de

*Vorschläge an Prof. Dr. Frank Puppe <puppe@informatik.uni-wuerzburg.de> oder Dieter Steinbauer <dieter.steinbauer@schufa.de>

Alle „Aktuellen Schlagwörter" seit 1988 finden Sie unter: www.ai-wuerzburg.de/as

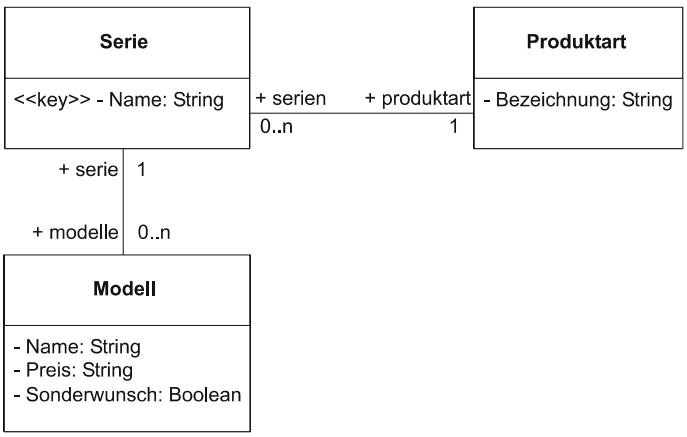

Abb. 1 Beispiel für ein PIM

Teil einer Softwarekomponente auf Modellebene, sondern auch den dynamischen Teil, beispielsweise in Form von Zustandsmaschinen. Drei Arten von Modellen werden unterschieden: das *Computation Independent Model*, *Platform Independent Models* und *Platform Specific Models*.

Das *Computation Independent Model (CIM)* beschreibt ein Softwaresystem auf fachlicher Ebene. Es liegt in einer Sprache vor, die für die fachlichen Anwender des Systems verständlich ist, und dient zum Diskurs zwischen Softwarearchitekten und Anwendern über Leistungsumfang und Anforderungen. Das CIM legt fest, was ein Softwaresystem leistet. Es definiert nicht, wie es dies leistet oder wie das System strukturiert ist.

MDA kennt als weiteren wichtigen Begriff, das Konzept der Plattform. Mit einer *Plattform* werden abgeschlossene Softwarekomponenten oder -technologien bezeichnet, die über spezifizierte Schnittstellen verwendet werden können, ohne dass die benutzende Komponente die Implementierung der von der Plattform bereitgestellten Funktionalität kennen muss. Die Plattform stellt technische Dienste bereit, ohne die die Software nicht funktionieren kann. Ein *Plattform-Modell* ist die Repräsentation einer Plattform auf Modellebene.

Das *Platform Independent Model (PIM)* modelliert die Funktionalität einer Komponente unabhängig von der gegebenen Plattform. Damit enthält ein PIM also den Teil eines Systems, der sich beschreiben lässt, ohne die endgültige Zielplattform zu kennen.

Das *Platform Specific Model (PSM)* kennt eine spezielle Plattform und realisiert ein PIM, wofür die von der Plattform bereitgestellten Schnittstellen genutzt werden.

Beispiel: Wenn das System basierend auf der Java-2-Enterprise-Edition (J2EE)-Plattform zu realisieren ist, ist das Plattform-Modell die Beschreibung von J2EE, das PIM die Beschreibung des Systems ohne J2EE-spezifische Details, während das PSM ein mit J2EE-spezifischen Details angereichertes Modell ist, aus dem schon der Programmcode generiert werden kann. Abbildung 1 zeigt ein kleines Beispiel für ein PIM, wie es bei einem Automobilhersteller vorkommen könnte. Abbildung 2 zeigt den Ausschnitt aus dem zugehörigen J2EE-spezifischen PSM, der der PIM-Klasse „Serie“ entspricht. (Das vollständige PSM enthält zusätzlich zum abgebildeten Teil ähnliche Strukturen für die weiteren Klassen des PIMs.)

Da in MDA Plattformen auf unterschiedlichen Abstraktionsebenen vorkommen können, ist es möglich, dass ein PSM der einen Ebene ein PIM der anderen Ebene ist. In solch einem Fall beziehen sich die beiden Ebenen auf verschiedene Plattformen.

Die Unterscheidung der verschiedenen Modellebenen hat zur Folge, dass Änderungen an der Software immer auf der Ebene vorgenommen werden müssen, auf der die relevante Information definiert wurde.

Transformationen zwischen Modellen

MDA sieht vor, PIMs in PSMs zu transformieren (siehe Abb. 3). Eine solche Transformation kann manuell, semiautomatisch oder vollautomatisch erfolgen. Jede Transformation besteht aus der Anwendung von Transformationsregeln, sogenannten Mappings, die für die automatische und semiautomatische Ausführung formal definiert sind. Die Transformation reichert die Information des PIMs durch zusätzliche Information der ausgewählten Plattform und der verwendeten Transformationsregeln an. Neben statischen Erweiterungen kann hier auch technisches, durch die Wahl der Plattform motiviertes dynamisches Verhalten hinzukommen. Ein

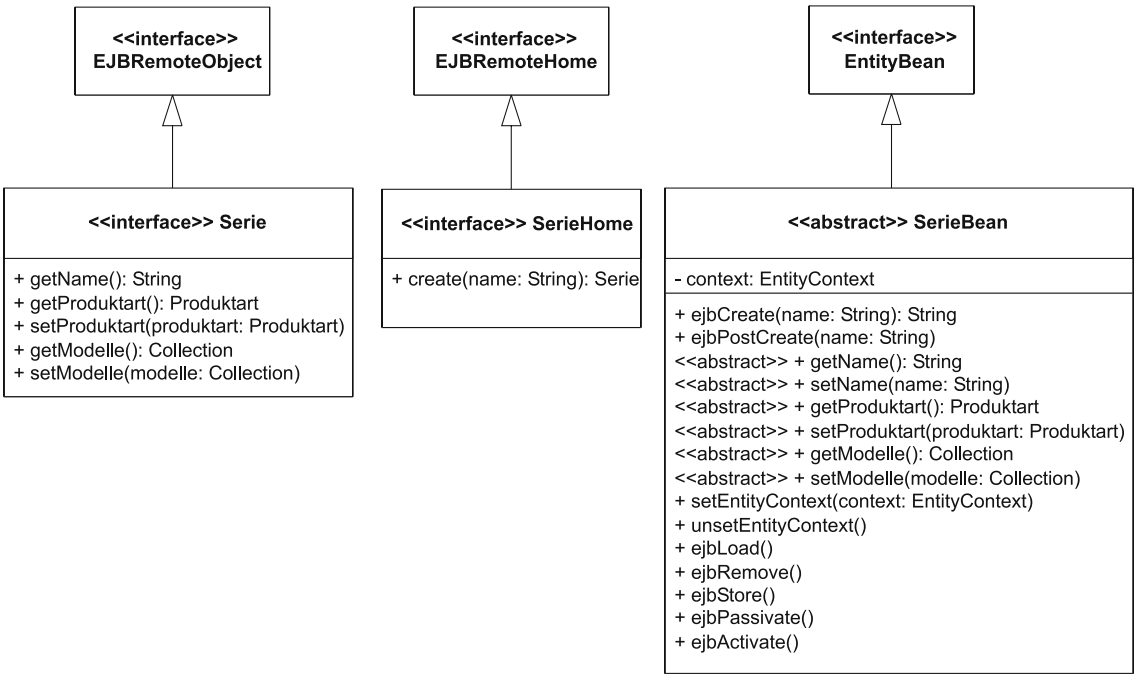

Abb. 2 Beispiel für ein PSM

Record of Transformations protokolliert die angewendeten Transformationsregeln, was als Grundlage zur Synchronisation zwischen PIM und PSM im Entwicklungsprozess dienen kann.

Transformationsregeln werden in Model Type Mapping und Model Instance Mapping und einer Kombination von beiden, den Combined Type and Instance Mappings, unterschieden.

Model Type Mappings sind Transformationsregeln, die auf der Ebene der Sprachkonstrukte der Modellsprache definiert werden. Beispielsweise könnte ein Model Type Mapping für das Entity-Relationship-Modell (ER-Modell) als PIM-Sprache und Java als PSM-Sprache so aussehen, dass alle Entity-Typen eines konkreten ER-Modells auf Java-Klassen abgebildet werden. Dabei ist es erlaubt, dass die Abbildung auf Eigenschaften von Instanzen der PIM-Sprachkonstrukte verweist. So könnten Entity-Typen ein spezielles Attribut benötigen, um nach einem bestimmten Model Type Mapping transformiert zu werden.

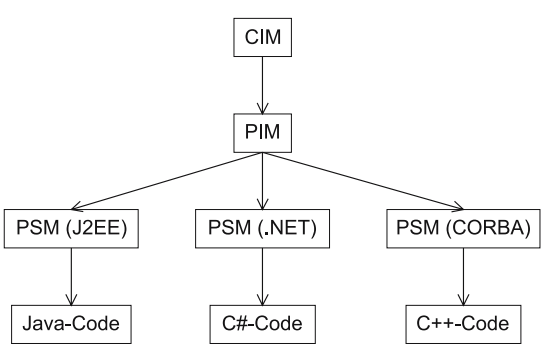

Abb. 3 Transformationen in MDA

Bei *Model Instance Mappings* definieren die Regeln, dass konkrete Instanzen des PIMs auf konkrete Instanzen im PSM abgebildet werden. Deshalb benötigt die Anwendung einer solchen Regel die Identifikation der zu transformierenden Instanzen durch *Markierungen*. Markierungen sind Plattformspezifisch, weil sie nur für die Transformation benötigt werden, und deshalb nicht Teil des eigentlichen PIMs. Mit diesen Transformationsregeln ist auch die Abbildung in Entwurfsmuster möglich.

Standardisierung und Werkzeugunterstützung

MDA setzt im Prinzip auf beliebigen Modellierungssprachen auf. Die Konstrukte der Modellierungssprache selbst können in einer Metamodellierungssprache definiert werden; die OMG hat für diesen Zweck die MOF (Meta Object Facility [3]) definiert. Die MOF legt generische Konstrukte fest, wie z. B. den Begriff des Objekts, der Vererbung oder der Assoziation. Mit dieser Infrastruktur können dann Modellierungssprachen definiert werden. Die aktuelle Version ist MOF 2.0.

Als Modellierungssprache ist vor allem UML vorgesehen, und hat sich auch in der Praxis erfolgreich durchgesetzt. UML kann mit Hilfe von

UML-Profilen auch branchen- oder projektspezifisch angepasst werden. Die OMG hat bereits einige UML-Profile (z. B. für „Enterprise Distributed Object Computing“) selbst spezifiziert.

Die wohl wichtigste Neuerung von UML 2.0 gegenüber der letzten Version 1.5 mit Bezug auf MDA ist die Erweiterung und Verbesserung der Konstrukte zur detaillierten Modellierung des dynamischen Verhaltens. Zudem wurde auch die Bedeutung von OCL (Object Constraint Language) in der Version 2.0 deutlich erweitert. Früher diente OCL ausschließlich zur Spezifikation von Bedingungen (Vor- und Nachbedingungen, Invarianten), nun ist sie eine Sprache für die Beschreibung allgemeiner Ausdrücke, sowohl in Modellen als auch in Metamodellen und Transformationsregeln. Sogar einfache Anfragemethoden können in OCL definiert werden.

Die Basis für die Interoperabilität verschiedener Werkzeuge wird durch ein standardisiertes Format zum Austausch der Modelle geschaffen. Für diesen Zweck hat die OMG XMI (XML Metadata Interchange) eingeführt, ein XML (eXtensible Markup Language)-Dialekt zur Notation von Modellen. Die MOF-Spezifikation definiert auch die Abbildung von MOF-Konstrukten auf XMI-Elemente. Dadurch ist die Darstellung beliebiger MOF-konformer Modelle, darunter auch UML-Modelle, in XMI möglich.

Mit der Standardisierung von MOF, UML und XMI ist das Thema Modellierung relativ weit fortgeschritten. Für Transformationen dagegen gibt es im Moment noch keine Standards, aber die OMG hat das Problem erkannt, und versucht es mit der neuen Spezifikation QVT (Query/Views/Transformations) anzugehen [6]. Dementsprechend ist die Palette von als „MDA-fähig“ bezeichneten Werkzeugen in Bezug auf die Modellierung relativ homogen, aber sehr heterogen bezüglich der Unterstützung von Transformationen. Für die Modellierung wird in den verfügbaren Werkzeugen fast ausschließlich UML verwendet. Es gibt jedoch Unterschiede bezüglich des unterstützten Umfangs und der Version von UML.

Die meisten Werkzeuge nutzen proprietäre Formate für die Speicherung der Modelle, bieten aber XMI als zusätzliche Export/Import-Schnittstelle an. Leider ist die Portabilität dieser XMI-Darstellungen zwischen verschiedenen Werkzeugen noch sehr eingeschränkt, weil viele Werkzeughersteller XMI nicht vollständig implementieren.

Die Werkzeuge unterscheiden sich vor allem in den unterstützten Transformationsschritten. Einige Werkzeuge generieren nur ein Codegerüst aus einem Modell, die eigentliche Logik muss von Hand eingebracht werden. Andere Werkzeuge können durchaus komplexe Anwendungen erzeugen, sind aber nur für eine bestimmte Plattform (z. B. unterstützt OptimalJ nur die J2EE-Plattform [5]). Nur wenige unterstützen die Generierung komplexer Anwendungen für eine erweiterbare Menge von Plattformen; ArcStyler ist ein Beispiel dafür [1]. In solchen Fällen wird ein proprietäres Format für die Definition von Transformationsregeln verwendet. Weiterhin kann bei vielen Werkzeugen beobachtet werden, dass sie nur eine Teilmenge der von der OMG vorgesehenen Modellierungsebenen benutzen: Ein CIM kommt fast nie vor, und oft erfolgt die Generierung des Codes gleich aus dem PIM, ohne die Zwischenstufe eines PSMs, wie z. B. bei openMDX [4].

Versprechungen und Probleme

Befürworter sehen in MDA den nächsten großen Fortschritt der Softwareindustrie mit folgenden Vorteilen:

- Modelle auf einer hohen Abstraktionsebene erlauben die kompakte Darstellung komplexer Systeme. So hilft ein PIM dabei, die ohnehin komplizierten fachlichen Abläufe eines großen Softwaresystems zu verstehen und zu organisieren, ohne technische Details mitbetrachten zu müssen.
- Die Technik ändert sich rasant, was viele Softwaresysteme vorzeitig veralten lässt. MDA dagegen erlaubt eine rapide Adaptierung an neue Technologien, indem aus dem PIM ein neues, der neuen Technologie entsprechendes PSM abgeleitet wird.
- Die Standardisierung der benutzten Modellierungssprachen und Metamodellierungsformate ermöglichen eine nahtlose Integration zwischen Werkzeugen unterschiedlicher Hersteller.
- Das CIM und gewissermaßen auch das PIM ermöglichen eine effiziente Kommunikation mit dem Kunden über das zu erstellende Softwaresystem.
- Die (zumindest größtenteils) automatische Generierung des PSMs und schließlich des Programmcodes beschleunigen den Entwicklungsprozess deutlich. Dabei werden gerade die stereotypen Programmieraufgaben durch Wiederverwendung eliminiert, was auch die Wahrscheinlichkeit von Programmierfehlern minimiert.

Neben diesen Vorteilen trifft MDA auch auf Skepsis, insbesondere bei folgenden Punkten:

- Wenn nicht das gesamte Programm generiert werden kann (was vermutlich nur in speziellen Fällen möglich ist), müssen nachträglich Änderungen von Hand gemacht werden, und es ist ein schwieriges Unterfangen, ein nicht selbst geschriebenes großes Programm zu ändern. In dieser Hinsicht taucht die Frage auf, welche Qualität das generierte Programm bezüglich Strukturierung, Lesbarkeit und Wartbarkeit aufweist.
- Es ist kompliziert, manuelle Erweiterungen und Änderungen in generierten Artefakten konsistent mit den Änderungen auf Modellebene zu halten.
- Die Performanz des generierten Codes sowie der Generierung selbst könnte ein Problem sein, da die Optimierungsmöglichkeiten am Code durch das Verfahren der Generierung eingeschränkt sind.
- Die OMG arbeitet zwar schon seit Jahren an den grundlegenden Spezifikationen, aber die Interoperabilität von verschiedenen Werkzeugen funktioniert nur bedingt, zumal verschiedene Hersteller grundsätzlich verschiedene Vorstellungen von MDA haben.
- Während die Modellierung der *Struktur* eines Systems weit verbreitet ist, ist die Modellierung des *dynamischen* Verhaltens eher problematisch. Für die erfolgreiche Generierung der Anwendungslogik wäre diese aber nötig. UML 2.0 enthält zwar viele Konstrukte dazu, diese sind aber auf der gleichen Abstraktionsebene wie eine normale Programmiersprache, so dass das Vorgehen eigentlich gar nicht mehr Modellieren, sondern Programmieren ist, nur graphisch statt textuell. Noch dazu ist diese Art von „graphischer Programmierung" nicht einmal überschaubarer als die herkömmliche, textuelle Programmierung.

Wer in dieser Debatte Recht behalten wird, lässt sich heute noch nicht sagen. Jedenfalls ist klar, dass die Grenze der Generierbarkeit sowie die am besten geeignete Modellierungsebene noch erforscht werden müssen. Zu den Herausforderungen der nächsten Jahre zählen die Überprüfung, ob die gewählten Modellebenen für die Repräsentation komplexer Softwaresysteme angemessen sind, die Definition genügend allgemeiner Transformationsregeln in einer formalen Sprache, die nicht nur effektive Software erzeugen, sondern auch wieder verwendbar sind, und die Etablierung geeigneter Tools, die für eine Verbreitung von MDA in der industriellen Softwareentwicklung sorgen.

Literatur

1. http://www.io-software.com/products/arcstyler_overview.jsp
2. Miller, J., Mukerji, J.: MDA Guide Version 1.0.1, OMG (2003)
3. Object Management Group. Meta Object Facility 2.0 Core Final Adopted Specification, 2004. http://www.omg.org/cgi-bin/doc?ptc/03-10-04
4. http://www.openmdx.org
5. http://www.compuware.com/products/optimalj
6. Object Management Group. Request for Proposal: MOF 2.0 Query/Views/Transformations RFP, 2002. http://www.omg.org/docs/ad/02-04-10.pdf
7. Object Management Group. Unified Modeling Language 2.0 Infrastructure Final Adopted Specification, 2004. http://www.omg.org/cgi-bin/doc?ptc/2003-09-15