Model-Driven-Architecture (MDA) im Kontext von Vorgehensmodellen

Roland Petrasch, Florian Fieber, Oliver Meimberg, Sandra Morlok

Technische Fachhochschule Berlin – Fachbereich VI
Luxemburger Straße 10, 13353 Berlin
petrasch@tfh-berlin.de, fieber@tfh-berlin.de
oliver.meimberg@form4.de, sandra.morlok@web.de

Abstract: Bei der Verwendung von Vorgehensmodellen wie dem *V-Modell® XT*[1] stellen sich in Verbindung mit der *Model Driven Architecture (MDA)*[2] der *Object Management Group (OMG)* verschiedene Fragen bzgl. der Rollen, Aktivitäten und Produkte. So ist z.B. zu klären, ob die Metamodellierung im Einklang mit den Aktivitäten des V-Modells im Bereich Systementwurf und Prüfung steht und welche Bausteine ggf. noch zusätzlich definiert werden müssen. Zwar lässt sich an dieser Stelle kein umfassendes und abschließendes Konzept für die Verbindung des V-Modell XT mit MDA vorstellen, aber erste Überlegungen für eine weiterführende Arbeit sind erfolgt, z.B. im Bereich der Aktivitäten.

1. Einführung

Dieser Beitrag stellt einerseits MDA sowie die zugehörigen Entwicklungsprozesse vor, andererseits erfolgt eine Einordnung in das V-Modell XT [VMXT]. Diese Betrachtungen sind für Projekte, die nach MDA entwickeln wollen „überlebenswichtig", da sich zum Teil gravierende Unterschiede zu klassischen Methoden der Software-Entwicklung ergeben. So ist beispielsweise bei der Rollenbesetzung darauf zu achten, dass Modellierer zum Einsatz kommen, die über entsprechende Qualifikation in Bereichen wie *Formalisierung, Modelltransformation, Metamodellierung* etc. verfügen. Weiterhin ist die Tool-Kette entscheidend für den Projekterfolg, da zahlreiche Schritte im Rahmen von MDA ausschließlich (semi-)automatisch ablaufen. So soll wie bei anderen generativen Ansätzen auch, ein Großteil des Programm-Codes aus den Modellen erzeugt werden anstatt diese manuell zu entwickeln.

Ein weiterer wichtiger Bereich ist die *Qualitätssicherung*: Der Test von generierten Modellen gegen die Ursprungsmodelle würde lediglich den Generator überprüfen, nicht jedoch die Modelle selbst. Daher müssen Qualitätssicherungsmaßnahmen an das Modellparadigma von MDA angepasst werden, z.B. sind die Transformatoren zu testen, was

---

[1] Das V-Modell® XT ist urheberrechtlich geschützt, © Bundesrepublik Deutschland, 2004
[2] Model Driven Architecture und MDA sind Trademarks der Object Management Group, USA

nur unter Einbeziehung der Metamodellebene möglich ist. Auch die Qualitätsanforderungen an die Software-Architektur sind präzise zu formulieren.

Dargestellt werden die verschiedenen Entwicklungsschritte für MDA und ein Abgleich mit einem Vorgehensmodell (V-Modell® XT). Diese Konformitätsbetrachtung bezieht sich auf die Produkte, d.h. die Artefakte, die Rollen und die Aktivitäten, kann zu diesem Zeitpunkt allerdings weder alle Aspekte eines Vorgehensmodells vollständig abdecken, noch sämtliche Methoden bzw. Anwendungsdomänen betrachten. Daher wird von einem generischen Verfahren für MDA ausgegangen, ohne dass Branchenspezifika berücksichtigt werden. In Bezug auf Software lehnt sich die hier verwendete Terminologie an verteilte objektorientierte Systeme an, die mit der *Unified Modeling Language (UML)* spezifiziert werden, wobei diese Konkretisierung lediglich dazu dient, allzu abstrakte Aussagen zu vermeiden, und daher als Beispiel zu verstehen ist.

2. MDA und MDA-Vorgehen

Die Idee von MDA ist einfach und bestechend zugleich - aber nicht neu (vgl. [Czar00], [Stah02]): Modelle so präzise und maschinen-lesbar zu entwerfen, dass sich die Entstehung der Software auf jeder Abstraktionsebene – zumindest teilweise - automatisieren lässt bis die gewünschte (konkrete) Architektur des Produktes entstanden ist. Keine künstliche Barriere, z.B. durch inkompatible Betriebssysteme, sollte der Entwicklung von Software-Systemen entgegen stehen. Im Zentrum stehen dabei Modelle und deren Transformation bis zum (fertigen) Software-System, d.h. zum Code (vgl. [MDA], [Klep03], [Mell04]). Die Vision von MDA ist daher mit den Begriffen Interoperablität, Integration, Vielfalt und Koexistenz verbunden[3].

Sehr stark vereinfacht stellt Abb. 1 die Grundidee von MDA dar: Ein plattformunabhängiges Modell (PIM) wird transformiert, wobei ein plattformabhängiges Modell (PSM) entsteht, welches z.B. der Code sein kann (Model-to-Code-Transformation). Dabei ist ein PIM nur relativ zum PSM plattformabhängig, d.h. ein PSM kann selbst wieder zum PIM werden, wenn eine weitere Transformation erfolgt.

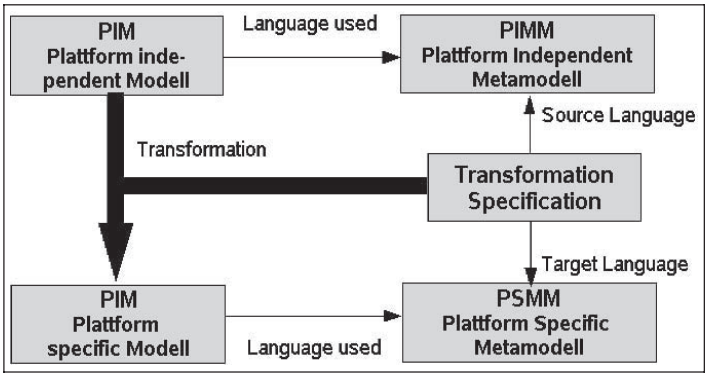

Abb. 1: Eine PIM-PSM-Transformation im Rahmen von MDA (in Anlehnung an [MDA, S. 3-9]

Die o.g. Darstellung einer Transformation auf der Basis von Metamodellen ist allerdings nur eines von zahlreichen Szenarien der MDA. So gibt es beispielsweise auch pattern-orientierte Transformationen. Dennoch wird eines deutlich: Formalisierung spielt eine wichtige Rolle, da u.a. Formale Sprachen bei der Modellierung für eine ausreichende Präzision sorgen, um die Modelle automatisch weiterverarbeiten zu können, z.B. muss sich ein Fachklassenkonzept als PIM durch eine Transformation für die Plattform *Java 2 Enterprise Edition* transformieren lassen, damit verteilte Komponenten wie *Enterprise Java Beans (EJB)* generiert werden können, womit dann die für das Projekt konkrete Zielarchitektur generativ durch den Transformator entsteht[4]. Das Ergebnis einer Transformation kann wie bereits erwähnt Code sein, der ggf. erweitert werden muss. Insofern bleiben viele der „klassischen" Aktivitäten einer Software-Entwicklung auch bei MDA bestehen, z.B. Modulimplementierung. Einige dieser Aktivitäten werden jedoch unbedeutender, andere kommen hinzu, z.B. die Erstellung der Transformatoren.

Ein Vorgehensmodell für MDA muss unweigerlich einige Begriffe wie Architektur, Sichten, Metamodellierung, Plattform, Plattformunabhängigkeit und Plattformmodell, Applikation, Modelltransformationen berücksichtigen. Dies gilt für Produkte, Rollen und Aktivitäten.

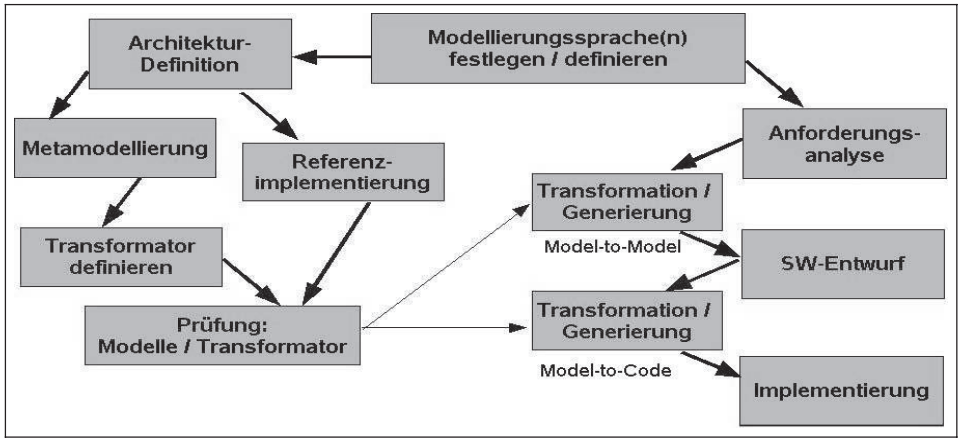

Abb. 2: Aktivitäten im Rahmen von MDA

In Anlehnung an bestehende Vorgehen, die auf generative Software-Entwicklung spezialisiert sind (z.B. der Generative Development Process [GDP]) sei hier ein stark vereinfachter Ablauf vorgestellt, der allerdings einen MDA-spezifischen Bereich hervorhebt: Es gibt zwei „Stränge": a) Die Architekturdefinition mit der Modellierung der plattformunabhängigen und platt-formabhängigen Metamodelle sowie die Definition geeigneter Transformatoren und b) die klassischen Phasen der Software-Entwicklung, die durch Transformationen / Generierung unterstützt werden (s. Abb. 2). Beide Bereiche können

[4] Hier wird auch die relative Plattformunabhängigkeit deutlich: ein Designmodell mit EJBs, das aus einem PIM (Fachmodell) entstanden ist und daher zunächst ein PSM darstellt, ist für weitere Transformationen dann aber selbst ein PIM.

teilweise parallel ablaufen, wobei der gemeinsame Ausgangspunkt die Festlegung der Modellierungssprache(n) ist.

3. V-Modell XT und MDA

An dieser Stelle soll keine allgemeine Einführung über das ansonsten bekannte V-Modell XT erfolgen, sondern eine zusammenfassende Darstellung einiger Aspekte in Bezug auf MDA. Dabei wird von einem Systementwicklungsprojekt eines Auftragnehmers ausgegangen. Als Projektdurchführungsstrategie sei an dieser Stelle ein inkrementeller Ansatz vorausgesetzt.

Es wird deutlich, dass einige Anpassungen an den Prozess notwendig sind: So muss es Entscheidungspunkte für die Bereiche Architektur und Metamodellierung geben (s. Bild 3). Auch innerhalb der Aktivitätsgruppen und Aktivitäten sind Anpassungen und Ergänzungen vorzunehmen. Die Aktivität Systemarchitektur erstellen (im Rahmen der Systementwurfes) beschreibt bereits wichtige Aspekte u.a. für die Architektursichten und die Architekturbewertung und muss nur noch um einige Bereiche der Metamodellierung und der Transformation aus der „MDA-Welt“ ergänzt werden.

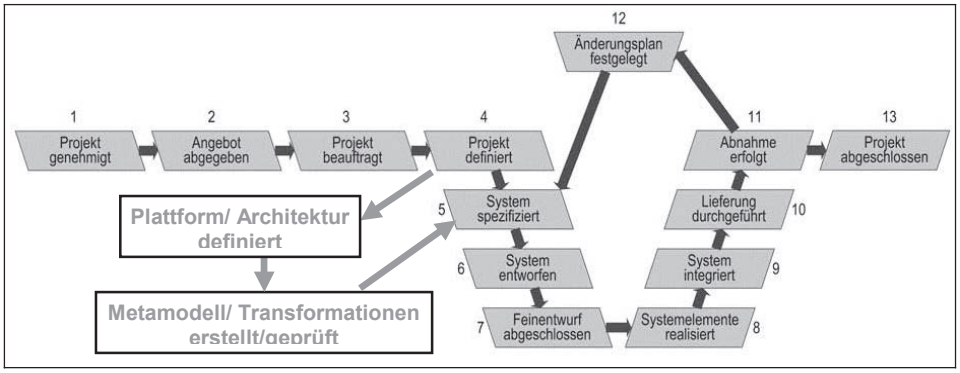

Abb. 3: Projektdurchführungsstrategie Inkrementelle Systementwicklung (AN) [VMXT]

Im Folgenden sei nicht näher auf die Rollenausprägung (verantwortlich, mitwirkend) sondern auf die Rollenbezeichnungen eingegangen. Im V-Modell XT sind 30 verschiedene Rollen definiert, die sich überwiegend problemlos in Verbindung mit MDA verwenden lassen, z.B. Anforderungsanalytiker (AG / AN), Qualitätssicherungsverantwortlicher, SW-Architekt, SW-Entwickler. Allerdings müssen einige Rollen anders definiert oder ausdifferenziert werden. So finden sich in Zusammenhang mit modell-getriebener Software-Entwicklung Rollen wie den Domänenarchitekt, Sprachdesigner, Referenzimplementierer, Plattform-Entwickler oder den Transformationsentwickler [Stah05, S. 340] bzw. Architect, Application Engineer, Middleware Engineer etc. [Fran03, S. 195ff.].

Es finden sich allerdings teilweise Begriffe im V-Modell XT, die in einigen Anwendungsbereichen unüblich sein dürften und sich auf den ersten Blick nur schwer mit der

MDA-Terminologie verbinden lassen: So wird im Vorgehensbaustein Qualitätssicherung (Aktivitätsgruppe Prüfung) von Nachweisakte und im Bereich Entwicklung (Aktivitätsgruppe Systementwurf) von Unterstützungssystem gesprochen. Solche sprachlichen Schwierigkeiten lassen sich jedoch lösen, da sie in vielen Fällen lediglich darin begründet sein dürften, dass keine englischen, sondern deutsche Begriffe verwendet wurden. Ein terminologisches „Mapping“ auf das eigene Begriffssystem ist ohnehin in den meisten Fällen notwendig, so dass dies kein größeres Problem darstellt.

4. Fazit

Eines lässt sich bereits an dieser Stelle feststellen: Prinzipiell ist durch die bausteinartige Struktur des V-Modells XT das „Einbinden“ von MDA möglich, d.h. eine Anpassbarkeit ist genauso gegeben wie die Skalierbarkeit für unterschiedliche Projektgrößen. Es sind einige Ergänzungen notwendig; so ist u.a. ein Ablaufrahmen für MDA-Projekte zu schaffen. Auch ist die Terminologie wie bei jedem Vorgehensmodell auf die spezifischen Begriffe abzubilden.

Als Basis für MDA erscheint daher das V-Modell XT brauchbar, so dass die Bausteine, Aktivitätengruppen und Aktivitäten genauer betrachtet werden können, um die Anpassungen und Ergänzungen vornehmen zu können.

Literaturverzeichnis

[Czar00] Czarnecki, K.; Eisenecker, U.: Generative Programming, Methods, Tools, and Applications. Addison-Wesley, 2000

[Fran03] Frankel, D. S.: Model Driven Architecture. John Wiley & Sons, 2003

[GDP] „b+m Generative Development Process“, b+m Informatik AG, 2003 http://www.architectureware.de/download/b+m_Generative_Development_Process.pdf

[Jeck03] Jeckle, M.; Rupp, C.; Hahn, J.; Zengler, B.; Queins, S.: UML 2 glasklar, Hanser Fachbuchverlag, 2003

[Klep03] Kleppe, A.; Warmer, J.; Bast, W. , „MDA Explained: The Model Driven Architecture: Practice and Promise“, Addison-Wesley, 2003

[MDA] Object Management Group (OMG): MDA Guide. Version 1.0.1, omg/2003-06-01, June 2003

[Mell04] Mellor, S. J.; Scott, K.; Uhl, A.: MDA Distilled. Addison-Wesley Professional, 2004

[MOF2] Object Management Group (OMG): MOF 2.0 – Meta Object Facility Core Specification, version 2.0. Adopted Specification, ptc/03-10-04, Oct. 2003

[Oest04] Oestereich, B.: Objektorientierte Softwareentwicklung - Analyse und Design mit der UML 2. 6. Auflage, Oldenbourg, 2004

[Stah02] Stahl, T.: Generative Softwareentwicklung in der Praxis, In: OBJECTSpectrum 01/2002, http://www.sigs.de/publications/os/2002/01/OS_01_02_stahl.pdf

[UML2] Object Management Group (OMG): UML 2.0 - Unified Modeling Language (UML) Infrastructure Specification. Version 2.0. Final Adopted Specification, ptc/03-09-15, Dec. 2003

[VMXT] Bundesministerien des Innern: V-Model XT, 2005, http://www.kbst.bund.de/V-Modell/-,293/V-Modell-XT.htm