Model Driven Software Development

Herausforderung: Entwicklungsmethodik und technisches Umfeld

Referent:
Christoph Schmidt-Casdorff

Inhaltsverzeichnis

Language Workbench

- **Werkzeuglandschaft, um DSL-zentrierte Software zu entwickeln**
  - [Martin Fowler 2003]

- **dies umfasst Werkzeuge,**
  - um DSL zu definieren
  - um Editor zu definieren
    - eigentliche Modellierung
    - dynamische Modellvalidierungen
  - um Generatoren zu entwickeln
    - Generierung
    - Modellvalidierung
  - um *model-to-model*-Transformationen zu entwickeln
  - Workflow, um die unterschiedlichen Schritte zusammenzuhalten

Modellierungssprachen und Metamodelle

- **… sind beispielsweise**
  - UML
  - ERM (Entity Relationship Model)
- **… werden durch ein Metamodell beschrieben**
  - abstrakte Syntax der Modellierungssprache
  - statische Semantik
  - dynamische Constraints

Beispiele für Metamodelle

- **XSD**
  - XML-Schema
- **BNF**
  - Backus-Naur-Form
  - textuelle Notation für Sprachen
    - Darstellung kontextfreier Grammatiken
- **(E)MOF**
  - (Extended) Meta Object Facility
  - Metamodellierungssprache der UML
  - mittels MOF ist UML, OCL, … beschrieben
- **UML Profile**
  - Möglichkeit innerhalb von UML eine Domänen zu beschreiben
- **EMF / ECORE**
  - ECORE ist semantisch (im Wesentlichen) zur EMOF äquivalente Metamodell im Eclipse-Umfeld
  - EMF (Eclipse Modelling Framework) enthält Werkzeuge zur Metamodellierung
  - Basis vieler Werkzeuge im Eclipse-Umfeld

Modellierungssprachen und Metamodelle

- **eine DSL ist eine spezifische Modellierungssprache**
  - bedarf einer Meta-(Modellierungs-)Sprache
    - um eigenes Metamodell zu entwerfen
  - ist auf einen bestimmten Problembereich zugeschnitten

Modellierung

- **Modellierung setzt einen geeigneten Editor voraus**
  - möglichst graphisch
  - bei Texteditoren *syntax highlighting, code completion, …*
- **UML Werkzeuge sind Editoren zur Modellierung**
  - für Modelle, welche auf UML basieren
  - Austauschformat zwischen OMG-UML und Eclipse
  - DSL werden durch UML Profile beschrieben
- **unterstützt Syntax und Semantik des Metamodells**
  - Modellvalidierungen
    - statische Semantik (wer darf welchen Pfeil auf wen ziehen)
    - dynamische Constraints
      - vgl. *OCL* et al

Generierung / Transformation

- **setzt auf einem Metamodell auf**
  - i.d.R. werden Objektmodelle über den Metamodellen genutzt
  - Generierungswerkzeug kennt bestimmte Metamodelle
    - nur deren Modelle können interpretiert werden
- **besteht i.d.R. aus mehreren Aufgaben**
  - Validierungen
  - Modell-Anreicherungen (*model-to-model*-Transformationen)
  - Codegenerierungen (*model-to-text*-Transformationen)
    - erzeugen die Generierungsartefakte
    - Code, Deskriptoren, XML-Dateien
  - alle Aufgaben können ggf. mehrfach ausgeführt oder kombiniert werden

Generierungs- /Transformationstechniken

- **Template Engines zur *model-to-text*-Transformation**
  - dynamisch typisierte Engines
    - Velocity, freemaker, JSTL (jexl), …
  - statisch typisierte wie Xpand (openArchitectureWare ►Appendix A)
- **spezielle Transformationssprachen**
  - QVT, MOFScript
- **als Komponente entwickelte M2T/M2M-Transformationen heißen Cartridge**
  - wiederverwendbar
  - Generator setzt sich i.d.R. aus mehreren Cartridges zusammen
  - Bestandteil einer Generierungsplattform, die sich auf eine spezielle DSL stützt
    - siehe AndroMDA, Fornax, … (►Appendix A)

Integration von *Individual Code*

- ***Individual Code* ist nicht-generierter/-modellierter Code,**
  - der in die Zielplattform integriert werden muss
  - der unabhängig von Generierung/Modellierung ist
- **während Generierung/Transformation werden Regeln ausgeführt**
  - welchen Individual Code in Generat einbinden
  - i.d.R. über Benennungsregel
- **beispielweise**
  - Super-/ Subclassbeziehung
  - Interface / Implementierung

Integration von *Individual Code*

- **Szenario : Domänenmodell gerät an die Grenzen der Modellierung**
  - fachliche Logik kann nicht mit Mitteln der DSL beschrieben werden
  - fachliche Logik soll durch *Individual Code* umgesetzt werden
    - wird im Modell z.B. über *Activity Diagrams* beschrieben (UML)
  - Model öffnet sich an definierten Stellen und mit definierter Semantik

- ***extension point* Verfahren**
  - vgl. Eclipse
  - Modell öffnet *extension points* für Individual Code
  - Generator bindet *Individual Code* an *extension points*
  - Verfahren eignet sich sehr gut für Integration von Individual Code auf Modellebene

MDSD Ausführungsumgebung

- **ist ein alternativer Ansatz zur Generierung**

- **Modell wird in einer Ausführungsumgebung ausgeführt**
  - es findet keine explizite Generierung statt
  - Modell und Ausführungsplattform sind aufeinander zugeschnitten
  - siehe *OpenMDX* (►Appendix A)

MDSD und Eclipse

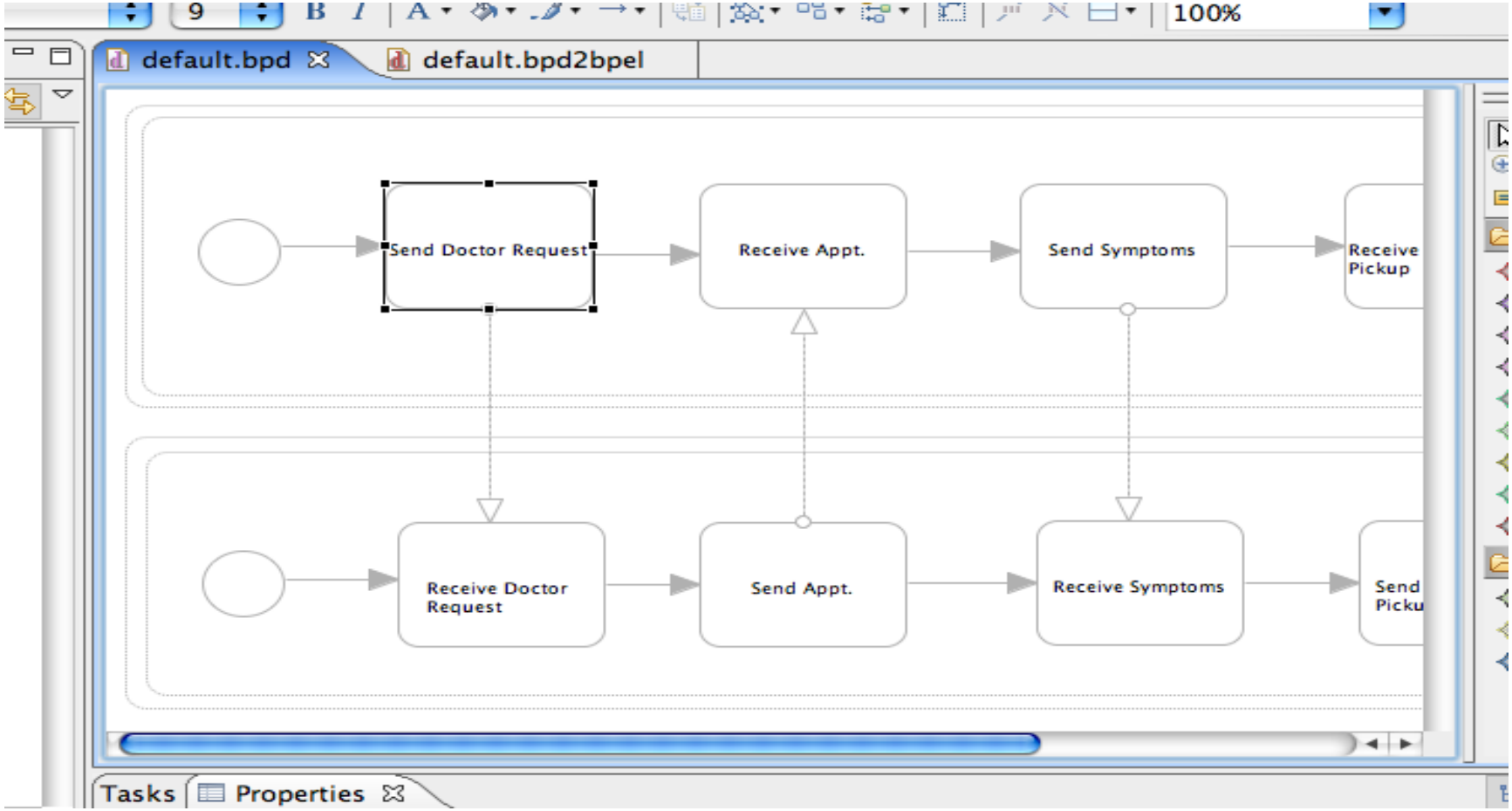
- ***Eclipse Modelling Project***
  - Subprojekt von Eclipse
  - umfasst Werkzeuge für alle Phasen der MDSD

- **Metasprachen und Editoren für Metamodelle (►Appendix A)**
  - erlaubt die Definition neuer Metamodelle auf Basis von ECORE
    - *EMF* liefert Framework zur Editierung
    - *UML nach ECORE* Umwandlung (oAW – uml2ecore)
    - Frameworks zur BNF basierten Definition von DSL
    - Xtext (integriert in *openArchitectureWare*)

MDSD und Eclipse - DSL Editoren

- **Syntax- und kontextbezogene Editoren für textuelle DSLs**
  - Xtext
- **GMF – graphische Editoren auf Basis eines ECORE Modells**
  - wird von Borland aktiv und maßgeblich unterstützt
  - Validierungs-Frameworks lassen sich integrieren

GMF Beispiel



Werkzeuglandschaft

- **Language Workbench**
  - EMF + GMF + openArchitectureWare (►Appendix A)
  - EMF + GMF + AndroMDA 4 (►Appendix A)

- **Generierungswerkzeuge**
  - bieten vorgefertigte DSLs und Generatoren/Transformatoren
  - AndoMDA 3 (►Appendix A)
  - Fornax (►Appendix A)
    - UML, Sculptor

- **MDSD Laufzeitumgebungen**
  - openMDX
  - bietet Modellierung und Laufzeitumgebung zur Ausführung der Modelle
  - ►Appendix A

Entwicklungsprozesse

Es existieren zwei Entwicklungsprozesse :

- **DSL Infrastruktur**
  - Aufbau der DSL Infrastruktur
- **Primärer Entwicklungsprozess**
  - bisheriger Entwicklungsprozess unter Einsatz der MDSD Infrastruktur
- **sollten beide unabhängig voneinander gelebt werden**
- **sind natürlich inhaltlich stark gekoppelt**

DSL Infrastruktur

- **ist ein eigener Entwicklungsprozess**
  - sollte auch so gelebt werden
  - besitzt starke Bindung / Kopplung an *primären* Entwicklungsprozess
- **Aufgaben in Domäne / Modellierung**
  - DSL-Analyse
  - Definition und Aufbau einer Modellierungsumgebung
  - Festlegung der Zielplattform
  - Definition der Architektur- und Designmuster für Generatoren
- **Aufgaben in Generatorenentwicklung**
  - Generatorenentwicklung, Frameworkentwicklung

DSL Infrastruktur

- **erfordert neue Rollen**
  - DSL Analyse
    - Konsistenz der DSL
    - Kommunikation mit Fachseite und Entwicklung
  - Generatorentwicklung
    - neue komplexes Umfeld
    - große Anforderungen an Abstraktionsvermögen
- **erfordert neue Kompetenzen an bekannte Rollen**
  - Architektur
  - Testmanagement und Testkonzeptionen

Architekturzentrierte MDSD

- **Modelle werden um Technologieaspekte angereichert**
  - JEE, Hibernate, Spring, ….
- **MDSD ist auf die Realisierungsphase beschränkt**
  - Artefakte sind
    - Modell
    - zusätzlicher selbst geschriebener Code
- **Generierung beschleunigt die Realisierungsphase**

Domänen-zentrierte MDSD

- **MDSD trifft die Phasen Analyse, Realisierung**
- **Artefakte sind**
  - fachliches Modell (Analyse)
  - zusätzlicher *Individual Code* (Realisierung)

Domänen-zentrierte MDSD

- **Generierung beschleunigt die Realisierungsphase**
- **Designphase degeneriert**
- **Formale Modellierung**
  - verlangsamt die Analyse (zu Beginn)
  - erhöht die Anforderungen an Analytiker
    - keine Realisierung als Puffer
    - Modelle sind ausführbarer
- **Bedarf ständiger Betreuung durch DSL Infrastruktur**
  - DSLs wachsen mit dem Einsatz

Einsatz von domänen-zentrierter MDSD

- **Modell ist kein Artefakt der Realisierung**
  - Modell und Realisierung haben unterschiedliche Lebenszyklen
- **Folgende Fragen sind projektspezifisch zu beantworten:**
  - ? Wer generiert? - Hoheit in Realisierung oder Analyse
  - ? Lebenszyklus der Generate
    - ? Ist Generierung Teil des Buildprozesses?
    - ? Werden Generate versioniert?
- **Best Practice**
  - Pair Modelling
    - zumindest bis sich Analyse an Auswirkungen ihres „Tuns“ gewöhnt hat
  - Konzept zur Integration Individual Codes
    - wird nicht nur durch Generator, sondern durch Modell bestimmt

Entwicklungsprozesse

- **Zwei getrennte Entwicklungsprozesse**
  - für MDSD-Infrastruktur und primärer (bisheriger) Entwicklungsprozess
- **Primärer Entwicklungsprozess**
  - beschäftigt sind Modelle als neue Artefakte
  - benötigt neue Rollen und Phasen
  - standardisierte Prozesse wie RUP und XP sind um MDSD Aspekte erweitert
- **Einsatz von domänen-zentrierter MDSD**
  - ist umfassender
  - hat größeren Einfluss auf Organisationen

www.iks-gmbh.com

Appendix A: Werkzeugübersicht

Standards OMG / Eclipse

- **OMG**
  - MOF – Modellierungssprache für Metamodelle
  - UML – Modellierung
  - OCL – dynamische Constraints
  - QVT – *model-to-model* Transformationen
- **Eclipse – de facto Standards**
  - ECORE - Metamodellierung
    - semantisch gleichwertig zu (E)MOF
    - beide werden sich annähern
  - abgeleitete Standards wie
    - UML2 – UML für Eclipse
    - QVT für Eclipse
    - OCL für Eclipse

Tools im Eclipse-Umfeld - Metamodellierung

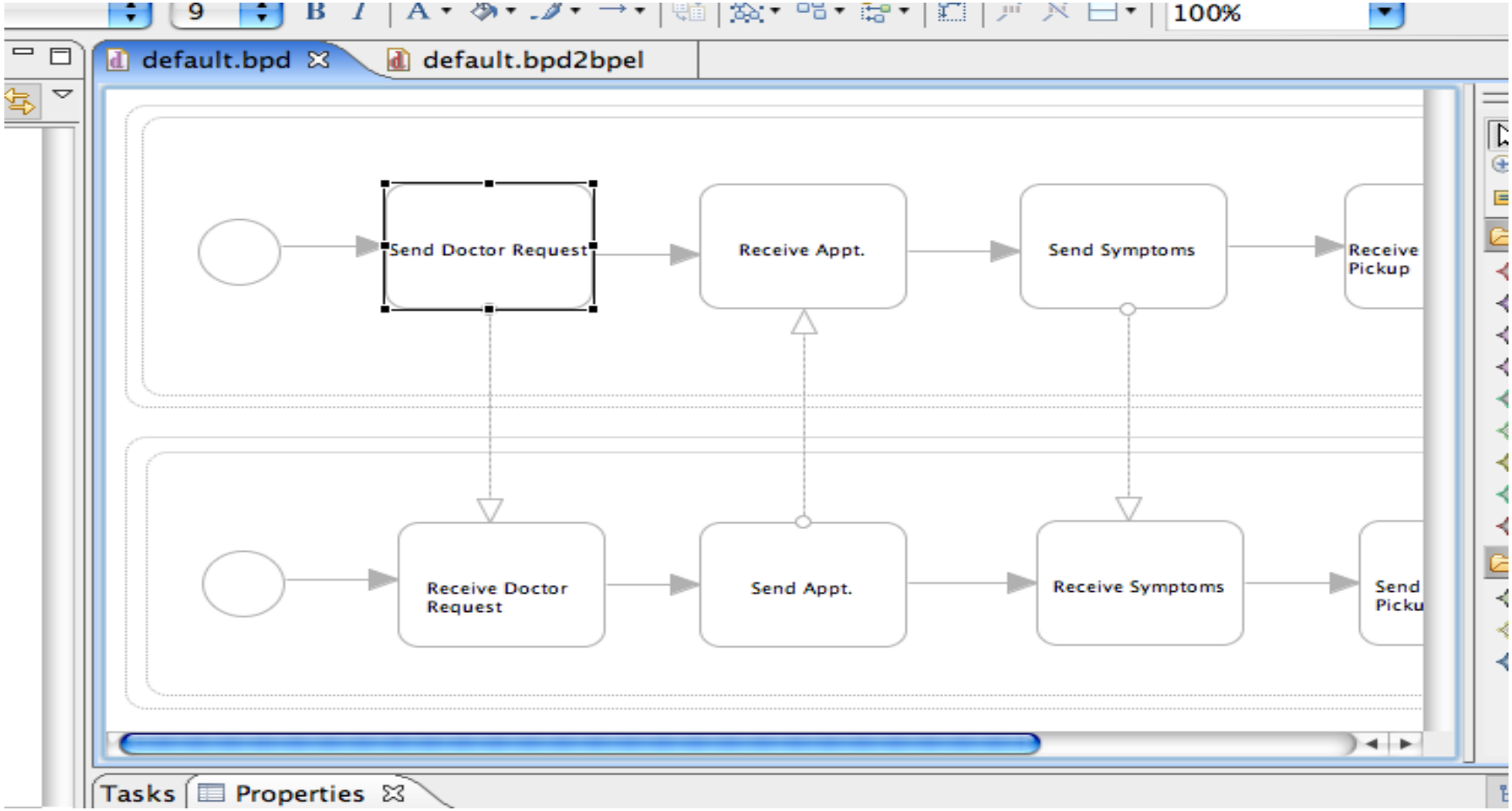
- **Alle MDSD-Entwicklungen unter Subprojekt *Eclipse Modelling Project***

- **Metasprachen und Editoren für Metamodelle**
  - erlaubt die Definition neuer Metamodelle auf Basis von ECORE
    - *EMF* liefert Framework zur Editierung
    - *UML nach ECORE* Umwandlung (oAW – uml2ecore)
  - Frameworks zur BNF basierten Definition von DSL
    - Xtext (integriert in *openArchitectureWare*)

Tools im Eclipse-Umfeld - DSL Editoren

- **Syntax- und kontextbezogene Editoren für textuelle DSLs**
  - Xtext
- **GMF – graphische Editoren auf Basis eines ECORE Modells**
  - wird von Borland aktiv und maßgeblich getrieben
  - Validierung-Frameworks lassen sich integrieren

GMF Beispiel



Tools im Eclipse-Umfeld - Generierung

- **Metamodell-Unterstützungen**
  - Ecore, UML2, ….
- **Generatoren (model-to-text-Transformationen)**
  - Template Engines
    - EMF – JET
    - oAW – Xpand2
  - MOFScript – *model to text* Transformation
- **Validierungen**
    - OCL
    - Check (oAW)
- ***model-to-model* Transformationen**
  - QVT Implementierungen
    - SmartQVT (Teil des Standards)
    - ATL (QVT – Dialekt)

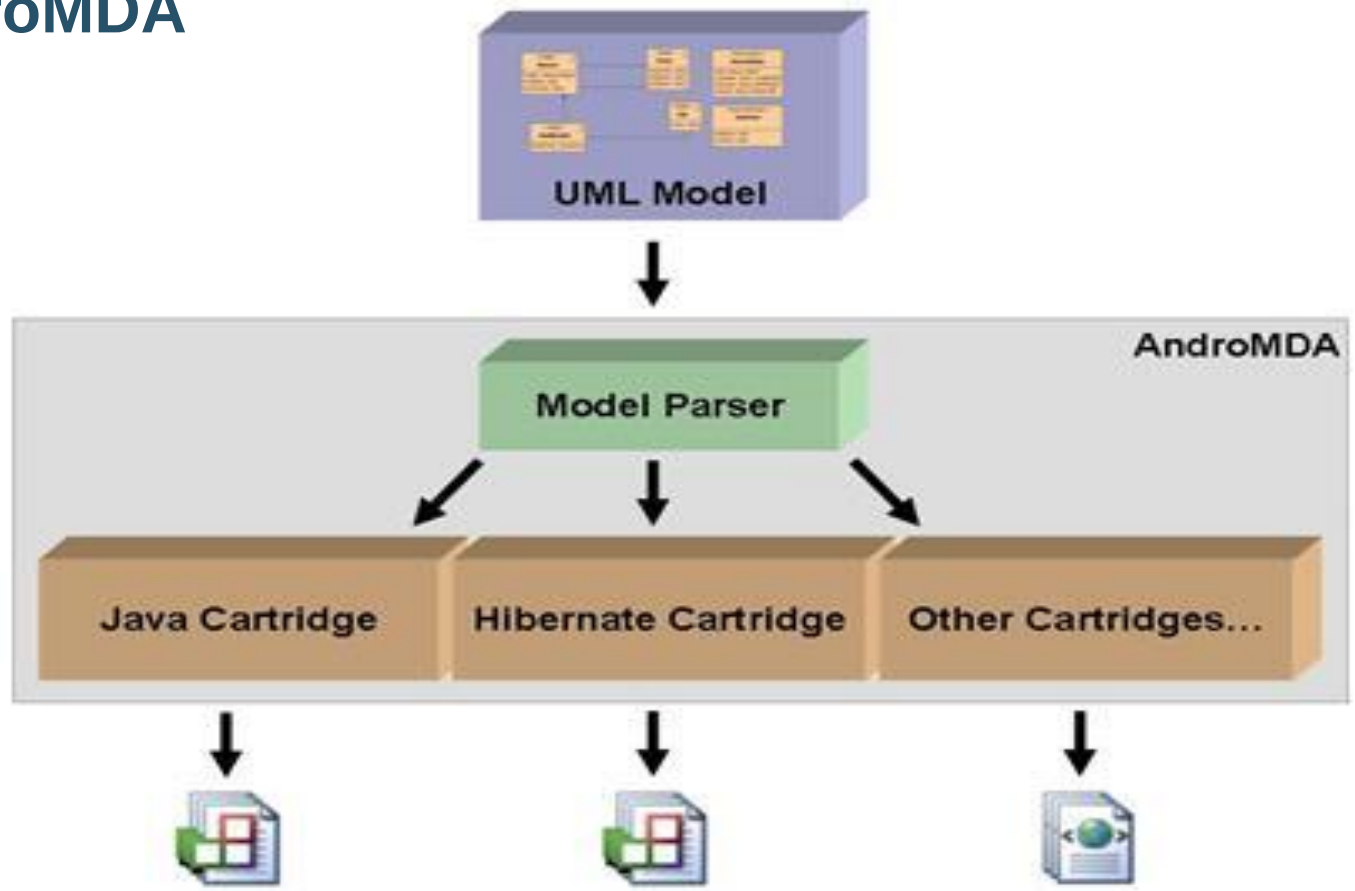
openArchitectureWare (oAW)

- **http://www.eclipse.org/gmt/oaw/**
- **Software-Generator Framework**
- **bietet**
  - Integration vieler Metamodelle
    - EMF, UML, RSA, Xtext (BNF-like)
  - Template Sprachen (*Xpand2*)
  - Model-zu-Model-Transformationen (*Xtend*)
  - bietet Valierungen (*Check*) inkl. Integration in GMF
  - Konzepte zur Integration eigenen Codes in Generatorartefakte (*recipe*)
- **interne Konzepte entsprechen nicht immer Standards**
- **basiert auf dem b+m Generator Framework**

AndroMDA

- **Links**
  - http://galaxy.andromda.org/docs-a4
  - http://galaxy.andromda.org/docs/getting-started/java/index.html
  - http://galaxy.andromda.org/docs-3.2/contrib/birds-eye-view.html
- **Software-Generator Framework**
  - Aktuelles Release 3.2 (A3)
  - Neuentwicklung in Version 4 (Previews verfügbar)
- **AndroMDA 3 setzt Schwerpunkte auf**
  - Einsatz vom UML als Metamodell (*UML Profiles*)
    - kann auf Basis MOF erweitert werden
    - eigenes Objektmodell (*metafacade*) basierend auf *Netbean MDR*
  - Generierung einer JEE-Anwendung
  - Einsatz von vorgefertigten Cartridges
    - Hibernate, EJB, Spring, ….

AndroMDA



Quelle **http://galaxy.andromda.org/**

AndroMDA

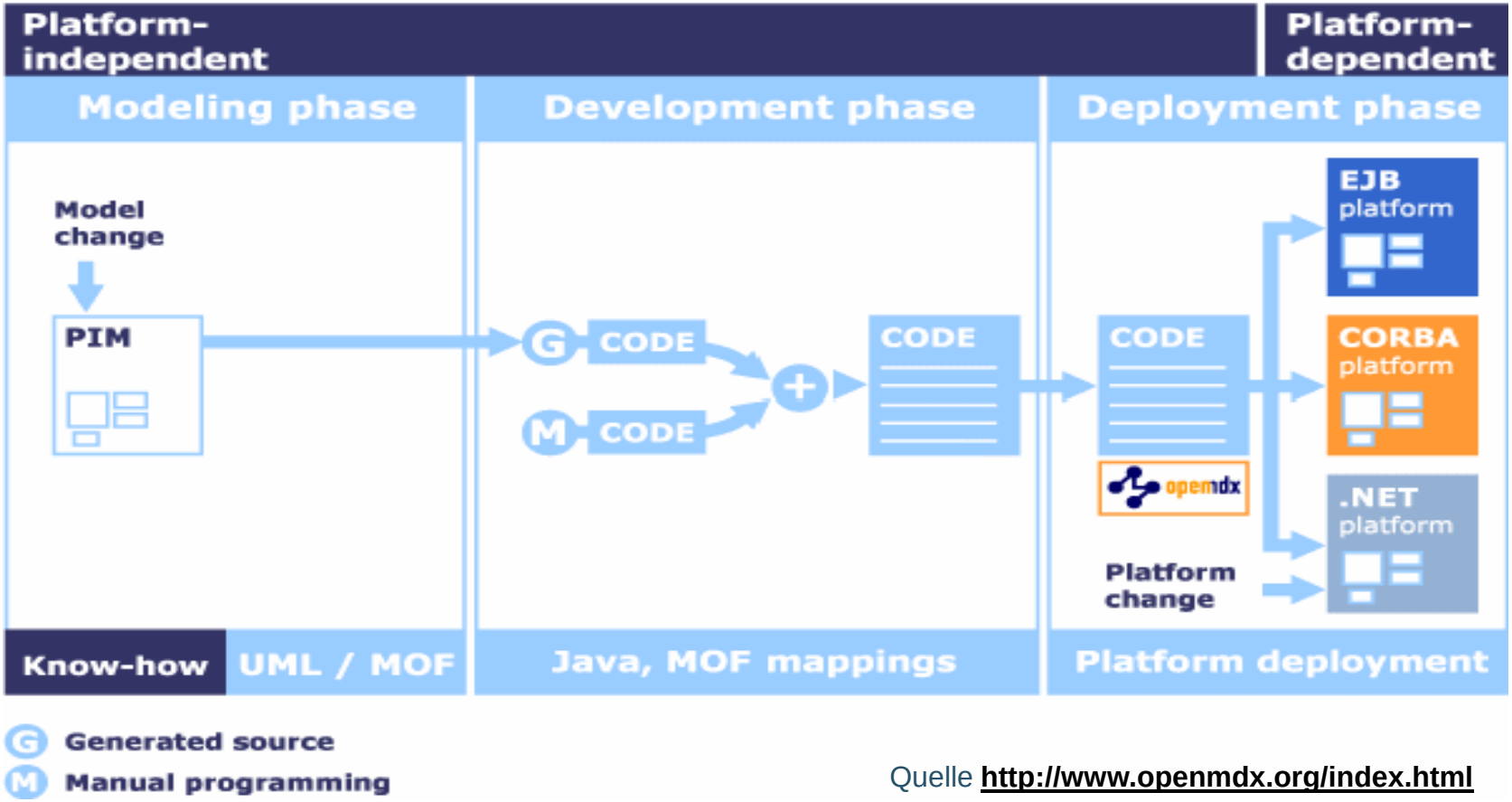
- **AndroMDA 4 bietet**
  - die Definition eigener Metamodelle
    - EMF basierte Metamodelle
    - UML 2 Ecore Converter
  - model-to-model-Transformationen via *ATL*
  - model-to-text-Transformation mittels *MOFScript*
  - … beides Teilmengen von OCL
  - eigenen Workflow konfigurierbar in *Groovy*
  - bessere Integration in *Eclipse*
  - *maven* basierter Entwicklungsprozess
  - nicht abwärtskompatibel zu A3

openMDX

- **http://www.openmdx.org/index.html**

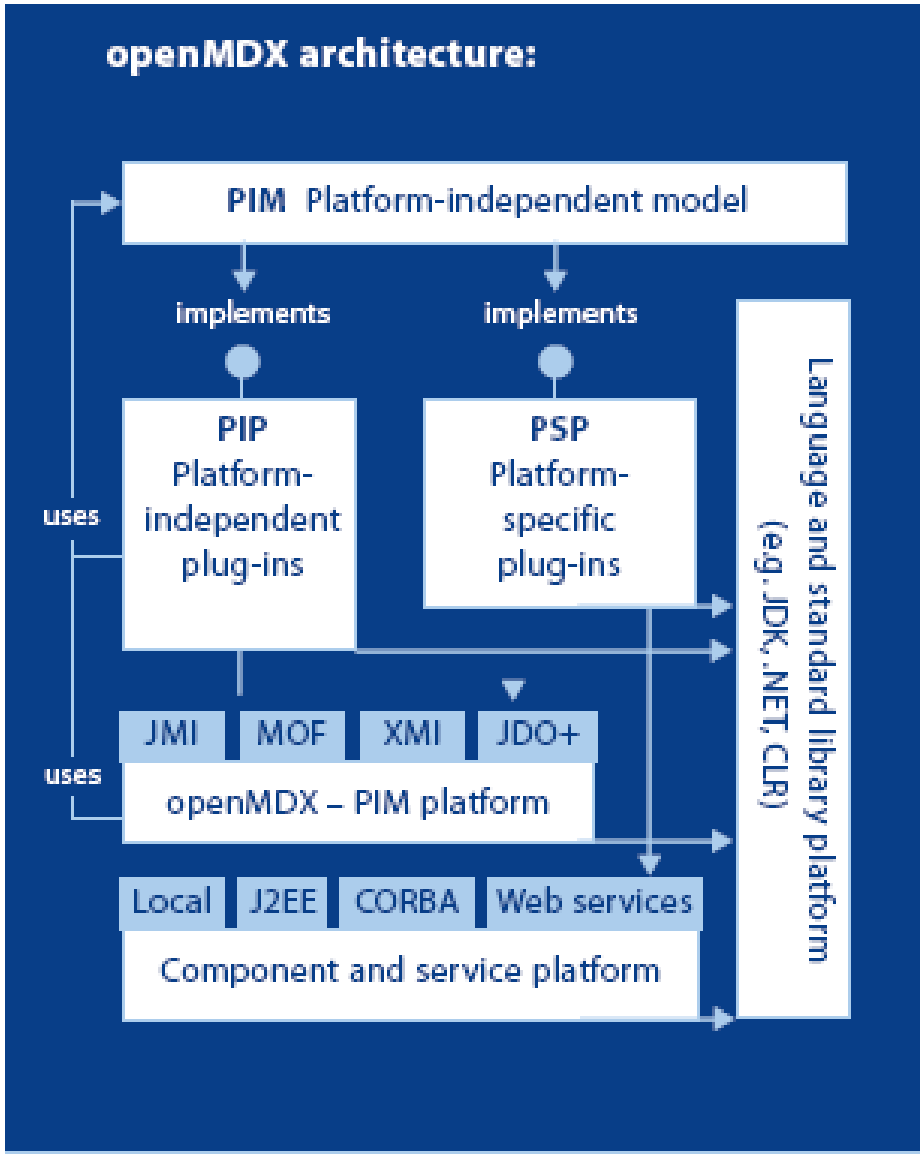
- **Stellt Laufzeitumgebung zur Ausführung von Modellen bereits**
  - kein Generator-Ansatz
- **Implementiert den MDA Standard gemäß OMG**
  - Modellierung erfolgt in MOF
  - basiert auf JMI-Binding
- **Generische Plattform für verteilte Objekte**
  - abstrahiert von Standards wie JEE, CORBA, etc.
  - Plattform-unabhängige Logik
- **Code wird in Plug-Ins bereitgestellt**
  - es existieren generische Plug-Ins für P*ersistence*, V*erteilung* etc. …

openMDX Entwicklungsprozess



Quelle **http://www.openmdx.org/index.html**

openMDX architecture



Quelle **http://www.openmdx.org/index.html**

Fornax

- **http://fornax-platform.org/cp/display/fornax/Fornax**

- **Fornax bietet DSL**
  - inklusive Metamodelle, Editoren, Cartridges
  - basiert auf EMF / oAW Plattform
  - *maven* basierter Entwicklungsprozess

- **Fornax - UML2**
  - unterschiedliche Cartridge für EJB, Spring, Hibernate, Grails
  - Metamodelle basieren auf (Eclipse-) UML2

Fornax - Sculptor

- **http://www.theserverside.com/tt/articles/content/ProductivityWithSculptor/article.html**
- **ist ein Plattform zur Codegenerierung**
- **besitzt eine textuelle DSL auf Basis oAW/XText**
- **basiert auf Konzepten des *Domain Driven Designs***
- **liefert Cartridges für gängige Frameworks**
  - Spring, Hibernate und Java EE

UML Tools

- **IBM Rational Software Architect, Borland Together Architect, Magic Draw, Enterprise Architect usw.**
  - bieten i.d.R. Export in EMF-Umfeld
    - genauer Export in EMF UML2 2.x
    - somit steht das EMF-Umfeld für Generatoren zur Verfügung
  - bieten teilweise eigene Schnittstelle auf Modell an
  - haben teilweise UML2 2.x als internes Metamodell
    - Together Architect 2008
  - bieten teilweise integriertes OCL
    - Editor
    - Ausführungsumgebung
    - Dynamische Constraints in UML-Profilen
  - graphische Editoren basieren teilweise auf GMF
  - bieten teilweise Unterstützung für QVT
- **UML- und EMF-Welt wachsen zusammen**