

iks Gesellschaft für Informations- und Kommunikationssysteme mbH

Sponsor der



13.03.2008

iks - Daten und Fakten

- **Gründung** **1989**
- **Sitz** **Hilden**
- **Geschäftsleitung** **Dipl.-Inf. Monika Stoll
Dipl.-Ing. Thomas Kondring
Hartwig Tödter**
- **Team** **ca. 85 Mitarbeiter/innen**
- **Tätigkeitsgebiet** **Individuelle IT-Konzepte und
Softwarelösungen**

Unsere Leistungen (Schwerpunkte)

- **IT-Beratung**
- **Projektleitung**
- **Projektdurchführung / Softwareentwicklung**
 - z.B. als Outsourcing-Partner
 - oder als Ergänzung der Entwickler-Teams unserer Kunden (Know-how-Transfer, Abfangen von Projektspitzen)
- **Schulungen / Coaching**

Technisches Umfeld

- **Schwerpunkt: Java-Technologien**

- Java SE, Java EE, server- und clientseitige Anwendungsentwicklung
- Frameworks
- ...

- **.NET / C#**

- **Business Intelligence**

- **AS/400**

- COBOL, RPG

- **Visual Basic, Delphi, C++ etc.**

Unsere Kunden

International tätige Konzerne sowie große und mittlere Unternehmen aus unterschiedlichen Branchen:

- **Industrie**
- **Versicherungen / Finanzdienstleistungen**
- **Handel**
- **Telekommunikation**
- **Transport / Verkehr / Logistik**
- **Verbände / Vereine / Interessenvertretungen**
- **Dienstleistungsunternehmen**
- **Gesundheitswesen**

Unsere Mitarbeiter

- **ca. 85 Mitarbeiterinnen und Mitarbeiter**
- **Ausbildung u.a.**
 - Informatiker
 - Wirtschaftsinformatiker
 - Mathematiker
 - Ingenieure
- **Einsatz in Kundenprojekten u.a. als**
 - Projektleiter, Analysten
 - IT-Berater, Softwarearchitekten, Anwendungsentwickler
 - Trainer
 - ...

iks im Internet

Weitere Informationen über iks finden Sie auf unserer Website

www.iks-gmbh.com

Und nun viel Spaß beim Vortrag:

**Modellgetriebene Softwareentwicklung:
ein Rundflug**

Modellgetriebene Softwareentwicklung:

ein Rundflug



13.3.2008

Autor:

Christoph Schmidt-Casdorff



Inhaltsverzeichnis

1. MDSD – Motivation und Einführung

- 1.1 Software Dilemma
- 1.2 Grundkonzepte
- 1.3 MDSD Keywords
- 1.4 MDA

2. Werkzeuglandschaft

- 2.1 Language Workbench
- 2.2 Metamodelle und Modellierung
- 2.3 Generierung und Transformation
- 2.4 Übersicht der Werkzeuglandschaft

Inhaltsverzeichnis

3. Entwicklungsprozess

- 3.1 DSL Infrastruktur
- 3.2 architekturzentrierte MDSD
- 3.3 domänen zentrierte MDSD

4. Erfahrungsbericht

- 4.1 Vorstellung
- 4.2 DSLs
- 4.3 Aufwandsbetrachtungen

5. Appendix A

- 5.1 Standards und Werkzeuge

Technologien im OO-Umfeld

Struts Web 2.0 Wicket .Net
Spring Portlets
Component Oriented Web Services
EJBMS
Java Server Faces
AJAX-Mapping
Ajax
Hibernate
Multi Threading

Aktuelles Softwaredilemma

- Jede dieser Architekturen löst ein technologisches Problem, schafft aber folgende neue Probleme:
 - Komplexität der Softwareentwicklung erhöht sich
 - komplexere Anforderungen an Entwickler
 - Komplexität der Software erhöht sich
 - erhöhter Wartungsaufwand
 - Bindung an Technologie verringert Flexibilität
 - eine einmal gewählte Technologie ist nur schwer zu wechseln,
 - aber Technologien sind im Fluss
 - Technologie und Geschäftslogik sind stark gekoppelt
- ... und produziert damit Kosten, Kosten, . . .

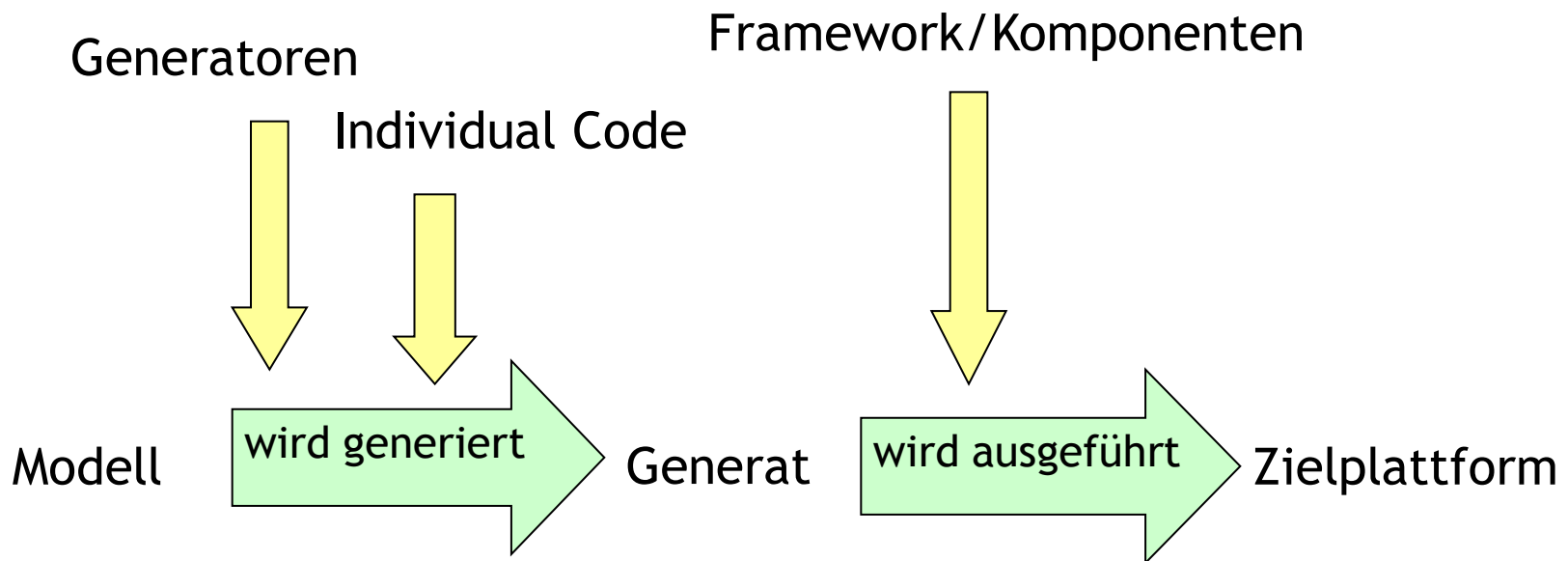
Grundgedanke der Generierung

● Angenommen . . .

- die domänene Logik könnte unabhängig von der einzusetzenden Technologie erstellt werden (==**Modell**)
- spezielle Software (==**Generatoren**) erzeugt aus dem Modell der domänenen Logik den Quellcode (==**Generat**) bzgl. der gewünschten Technologie/Programmiersprache/....
- das Generat trifft auf eine (der domänenen Logik angepasste) Plattform (==**Zielplattform**) aus Frameworks/Komponenten/... innerhalb/mittels derer es ausgeführt wird

● Grundgedanke von **MDSD** (**M**odel **D**riven **S**oftware **D**evelopment)

Grundgedanke ‚Generierung‘ - Skizze



Motivation für MDSD

● Plattformunabhängigkeit

- Generatoren für jede Zielplattform

● Effizienzsteigerung durch Automation

- Softwareerstellung durch Generierung

● Qualitätsverbesserung

- durch Generierung
 - Qualität des generierten Codes wird über Generator gesteuert
 - Architektur- und Designanforderungen werden systematisch eingehalten
- durch Wiederverwendung der Domain Architektur
 - Qualität steigt mit Anzahl der Einsätze einer Domain Architektur

MDSD - Grundgedanke

- **ist allgemein gültig**
- **ist unabhängig von der eingesetzten Systemumgebung**
 - Betriebssystem
 - Programmiersprache
 - Architektur
 - Modellierungssprache
- **allerdings findet sich im OO-/Java-Umfeld eine wachsende Infrastruktur**
 - Modellierungssprachen (UML, EMF,...)
 - Transformationssprachen (QVT,...)
 - Transformationswerkzeuge (Eclipse)
 -

Metapher für Generator/Transformator

- **Compiler dient als Metapher für technische Funktionsweise von Generatoren/Transformatoren**
 - Generator -> Compiler
 - Modell -> Quellcode
 - Generat -> Maschinencode
- **Damit ein Generator ein Modell interpretieren kann, muss dieses**
 - in einer Sprache formuliert sein, die der Generator versteht
 - formal sein (d.h. einer vorgegebenen Syntax entsprechen)

Domäne-spezifische Sprache

- Sprache zur Formulierung von Modellen
 - **DSL** (*domain specific language*)
- ... besitzt eine formale Spezifikation in Form eines Metamodells
 - beschreibt die Sprachelemente
 - **abstrakte Syntax**
- ... kann in unterschiedlichen Formen dargestellt/realisiert werden
 - UML, EMF, textuell
 - **konkrete Syntax**

Problemräume / Domäne

- können durch die Architektur/Technologie motiviert sein

- architektur-zentriert
- Abbildung in die gewählte Technologie
 - Anbindung an Datenbanken (JDBC, Hibernate, ...)
 - Middleware EJB, Spring
- ist der zur Zeit verbreitetere Einsatz von MDSD

- können domänen motiviert sein

- domänen-zentriert
- wird durch die konkrete domänenkeit vorgegeben

DSL

- je abgegrenzter der Problemraum und
- je spezieller die DSL dieses Problemraums ist,
- desto größer ist der Mehrwert durch MDSD
- ▶ Finde abgegrenzten Problemraum und entsprechende DSL
- ▶ In Projekten existieren u.U. mehrere DSLs nebeneinander

Generatoren/Transformatoren

- interpretieren das Modell
- enthalten möglichst viel Wissen um den Problemraum
- setzen ein Regelwerk zur Generierung/Transformation um
- setzen allgemeine Muster für Architektur und Code um

- **Transformatoren** beschreiben *model-to-model*-Transformationen
- **Generatoren** beschreiben Generierung von Artefakten
 - Quellcode, XML-Dateien, HTML-Seiten,
 - Model-to-text-Transformationen
- **Generatoren/Transformatoren können in beliebig vielen Schritten gekoppelt sein**

Zielfplattform

- ist Laufzeitumgebung zur Ausführung der Generate
- je spezifischer die Zielfplattform dem Generat entgegenkommt, desto einfacher wird der Generator
- Beispiele für Zielfplattformen
 - J2EE Container
 - Message Oriented Middleware
 - Datenbank
 - eigenentwickelte Frameworks

Software Systemfamilie

- **Domänen-Architektur (Domain Architecture)**
 - DSL/Metamodell, Modelltransformationen, Zielplattform
 - beschreibt die Umgebung vom Modell zum Softwareprodukt
- **Software-Systemfamilie**
 - Programmgruppe mit gemeinsamen Eigenschaften
 - hier: gemeinsame Domain Architecture

MDA - model driven architecture -

- **ist der Beitrag der OMG (Object Management Group) zu MDSD**
 - ist eine Ausprägung der MDSD
- **definiert 3 unterschiedliche Abstraktionsebenen**
 - CIM – computer independent model
 - PIM – platform independent model
 - PSM – platform specific model
 - ... und einen Prozess, der die 3 Modelle schrittweise ineinander überführt
 - diese 3 Modellabstraktionen haben sich in der Realität nicht bewährt
 - Abgrenzung untereinander ist nur schwer möglich (relative Bewertungen)
 - in der Praxis treten mehr als 3 Modellausprägungen auf
- **definiert Standards im MDSD-Umfeld**
 - (E)MOF, UML, OCL, QVT, XMI
 - CWM – common warehouse metamodel (DSL für dataware housing)



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**
 - welche Individual Code in Generat einbinden
 - i.d.R. über Benennungsregel
- **beispielweise**
 - Super-/Subclassbeziehung
 - Interface / Implementierung

Integration von *Individual Code*

- **Szenario : domänenes Modell gerät an die Grenzen der Modellierung**
 - domänene Logik kann nicht mit Mitteln der DSL beschrieben werden
 - domänene Logik soll durch *Individual Code* umgesetzt werden
 - 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

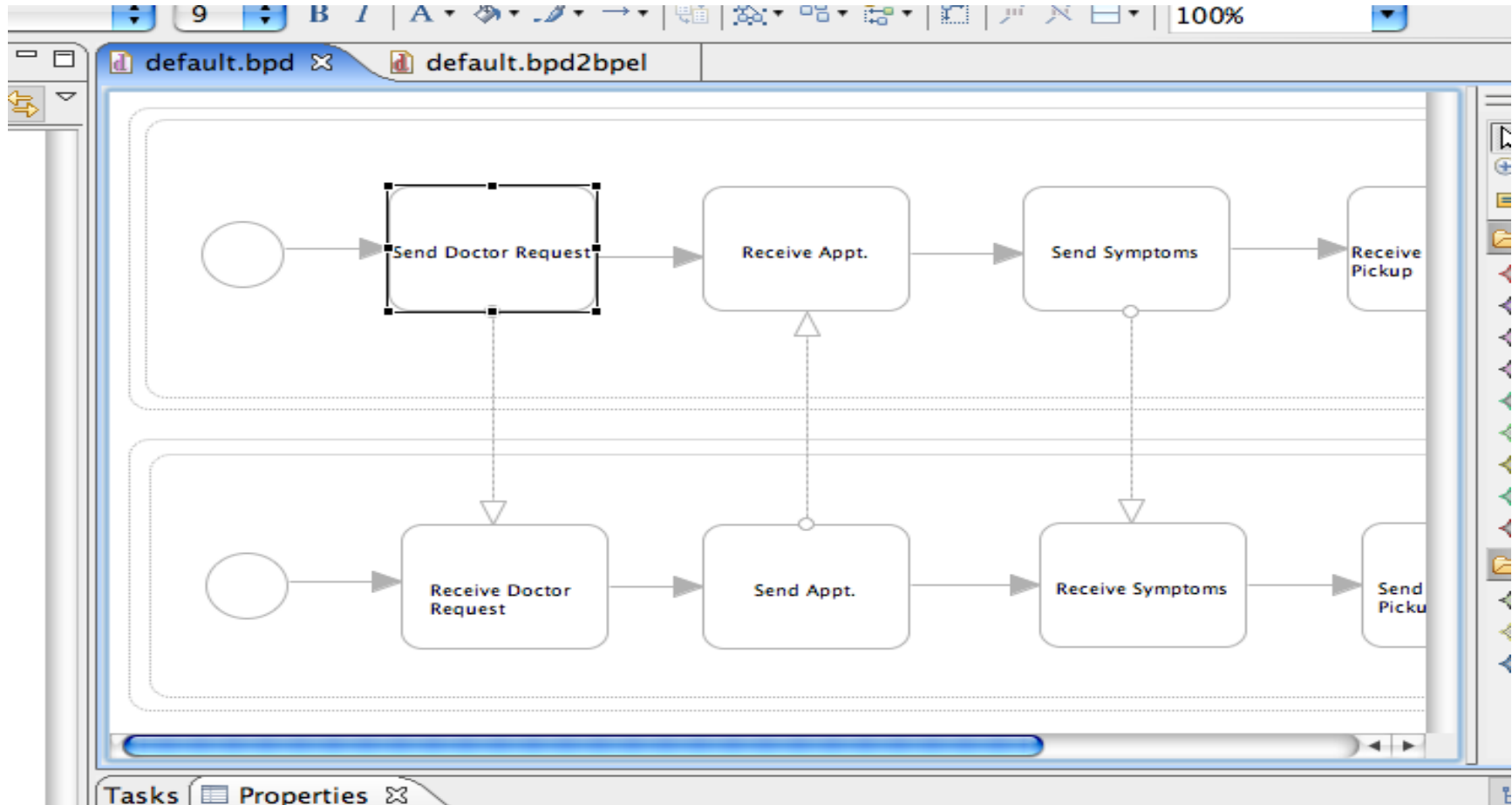
● **Metasprachen und Editoren für Metamodelle (►Appendix A)**

- 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**

Einsatz von domänen-zentrierter MDSD

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

Einsatz von domänen-zentrierter 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 sich mit Modellen als neuen Artefakten
- 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 Organisation



Erfahrungsbericht

- **MDSD wurde bei einem großen Finanzdienstleister umgesetzt**
 - Laufzeit des Projekts > 5 Jahre
- **Änderungsmanagement und Informationsaustausch mit Leistungsträgern**
 - Größtenteils dateibasierte Satzverarbeitung
 - Größenordnung pro Monat:
 - Auflieferung von ca. 0,5 - 1 Mio. Datensätze
 - Versand von bis zu mehreren Mio. Datensätzen an die Leistungsträger

DSLs

- **zentrales Businessmodell**
 - O/R-Mapping
- **WebGUI**
- **Kundenanschreiben**
- **Export / Import zu Dateischnittstellen**
- **Archivierung**

DSLs - Technologie

MetaModel

UML

DSL

UML Profile

Generierung (bisher)

Proprietäre Modellzugriffsschicht

Generierung (zukünftig)

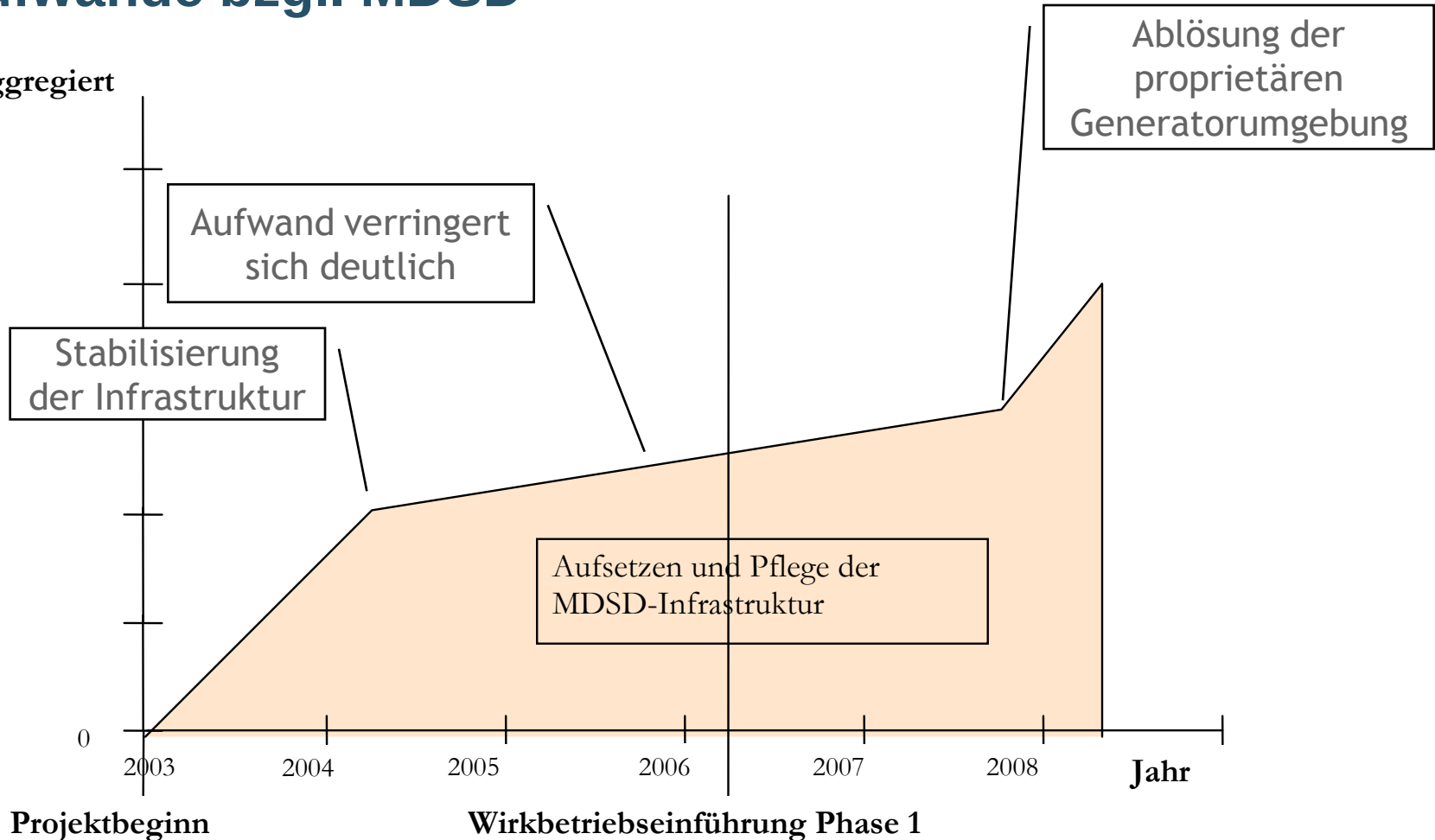
ECORE basierte Modellzugriff

Einsatz von MDSD

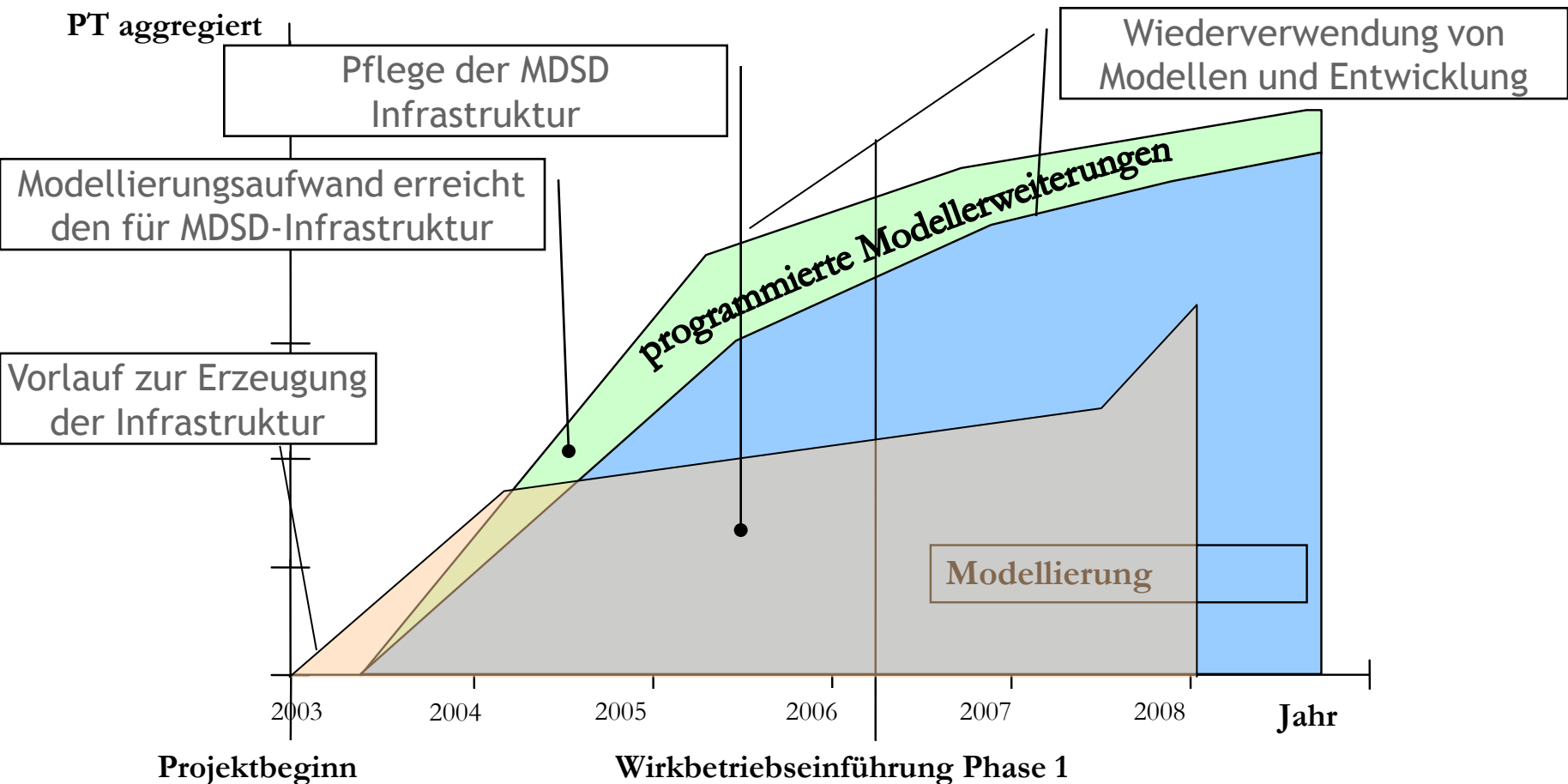
- **Einsatz von MDSD > 40% (bezogen auf Modelle)**
- **Mengengerüste**
 - 10 DSLs
 - derzeit ca. 100.000 Modellelemente, davon < 2% (ca. 1900) programmatisch ergänzt
- **Kleines Entwicklungs- und Modellierungsteam (< 10 Personen)**
 - davon 3 externe Kräfte

Aufwände bzgl. MDSD

PT aggregiert



Aufwände bzgl. MDSD



Aktueller Stand im Projekt

● Stabilität / Verlässlichkeit

- DSL ist seit über zwei Jahr stabil
- Generatoren sind stabil und zuverlässig
- Framework ist bis auf kleinere Änderungen stabil
- Entwicklungsprozess hat sich stabilisiert und bewährt

Risiken

- **Bindung der Generatoren an proprietäres Tool**
- **Komplexität der DSL muss begrenzt bleiben**
 - Gefahr unklarer Modellierungsregeln
 - Gefahr impliziter Abhängigkeiten zwischen Modellierungselementen

Erwartung an MDSD

- + **Erfolgreiches Management der Mengen an Detailinformationen**
- + **Robuste Software durch Framework-Einsatz**
- + **Kurze Reaktionszeiten bei Änderungen der Anforderungen (change requests, bugs)**
 - + Wartung verringert sich um mehr als 50 %
- **Zu erwartende Aufwandsvorteile in zukünftigen Phasen**
 - durch Einsatz der bestehenden Domain Architecture
 - Lernkurve bewältigt
 - Erweiterung der Software-Systemfamilie

Resümee

In unserem Fall und bei unserer Ausgangssituation hat sich der Einsatz von *Modellgetriebener Softwareentwicklung* gelohnt.

Literatur

- **Modellgetriebene Softwareentwicklung. Techniken, Engineering, Management**
 - Stahl, Völter, Efftinge
 - DPunkt Verlag (2007) - ISBN-10: 3898644480
- **Modellgetriebene Softwareentwicklung. MDA und MDSD in der Praxis**
 - Pietrek et al
 - Entwicklerpress (2007) - **ISBN-10:** 3939084115
- **Model Driven Architecture. Applying MDA to Enterprise Computing**
 - David S. Frankel
 - OMG Press (2003) - ISBN-10: 0471319201

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
- abgeleitet Standards wie
 - UML2 – UML für Eclipse
 - QVT für Eclipse
 - OCL für Eclipse

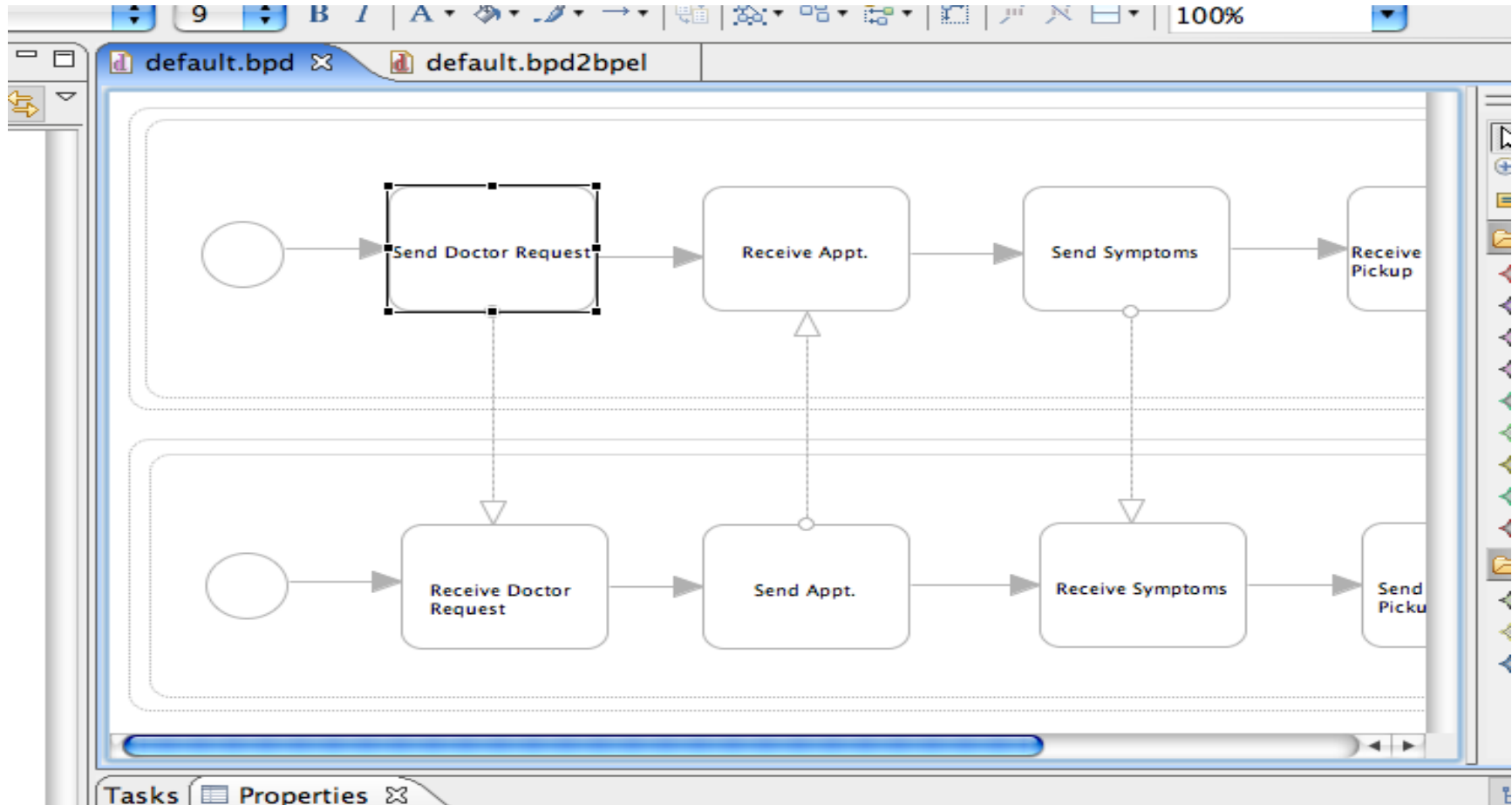
Tools im Eclipse-Umfeld - Metamodellierung

- **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
 - Validierungs-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/oawl>
- **Software-Generator Framework**
- **bietet**
 - Integration vieler Metamodelle
 - EMF, UML, RSA, Xtext (BNF-like)
 - Template Sprachen (*Xpand2*)
 - Model-zu-Model-Transformationen (*Xtend*)
 - bietet Validierungen (*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>

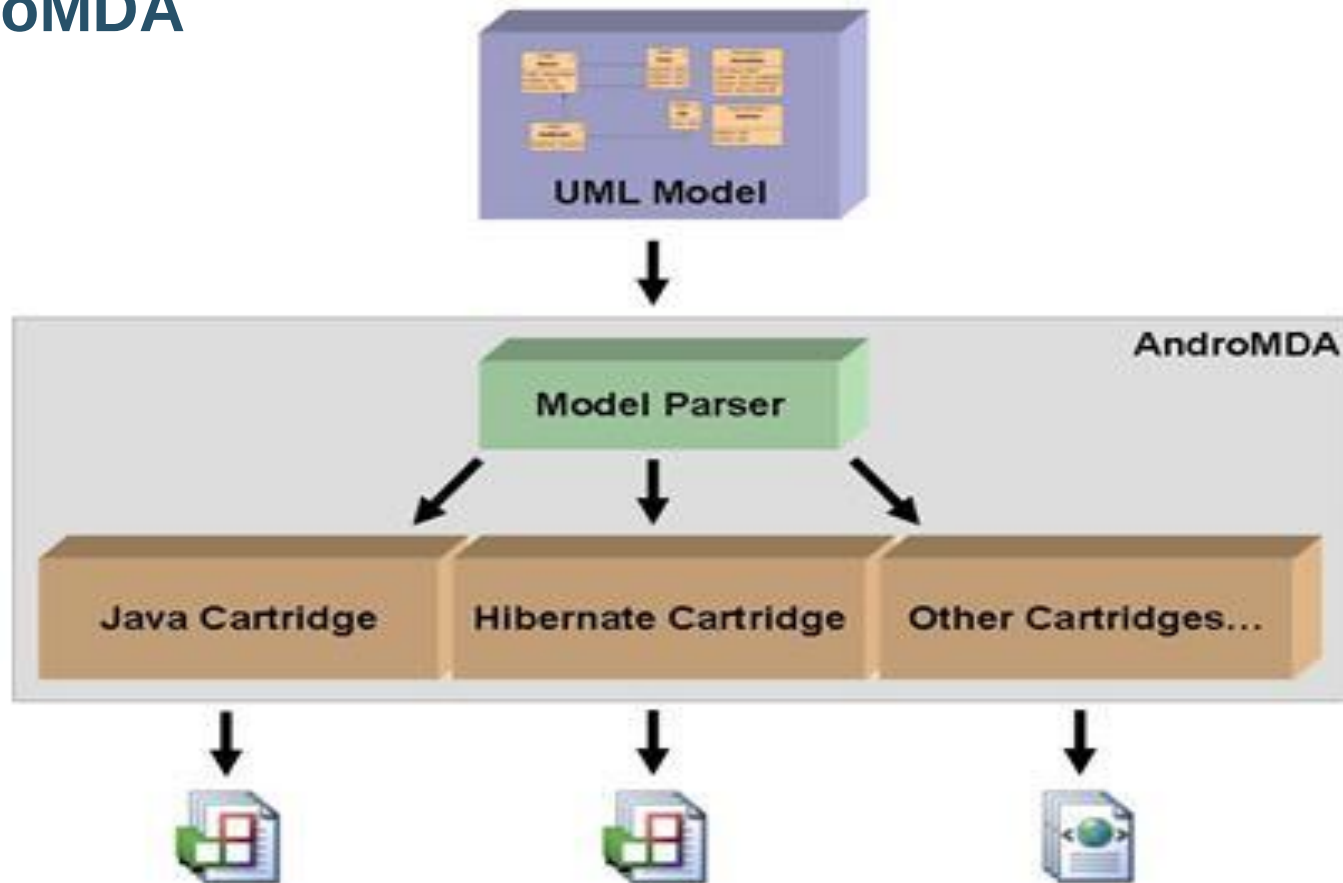
● Software-Generator Framework

- 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
 - eigene Objektmodell (*metafacade*) basierend auf *Netbean MDR*
- Generierung einer JEE-Anwendung
- Einsatz von vorgefertigten Cartridges
 - Hibernate, EJB, Spring,

AndroMDA



Quelle: <http://galaxy.andromda.org/>

AndroMDA

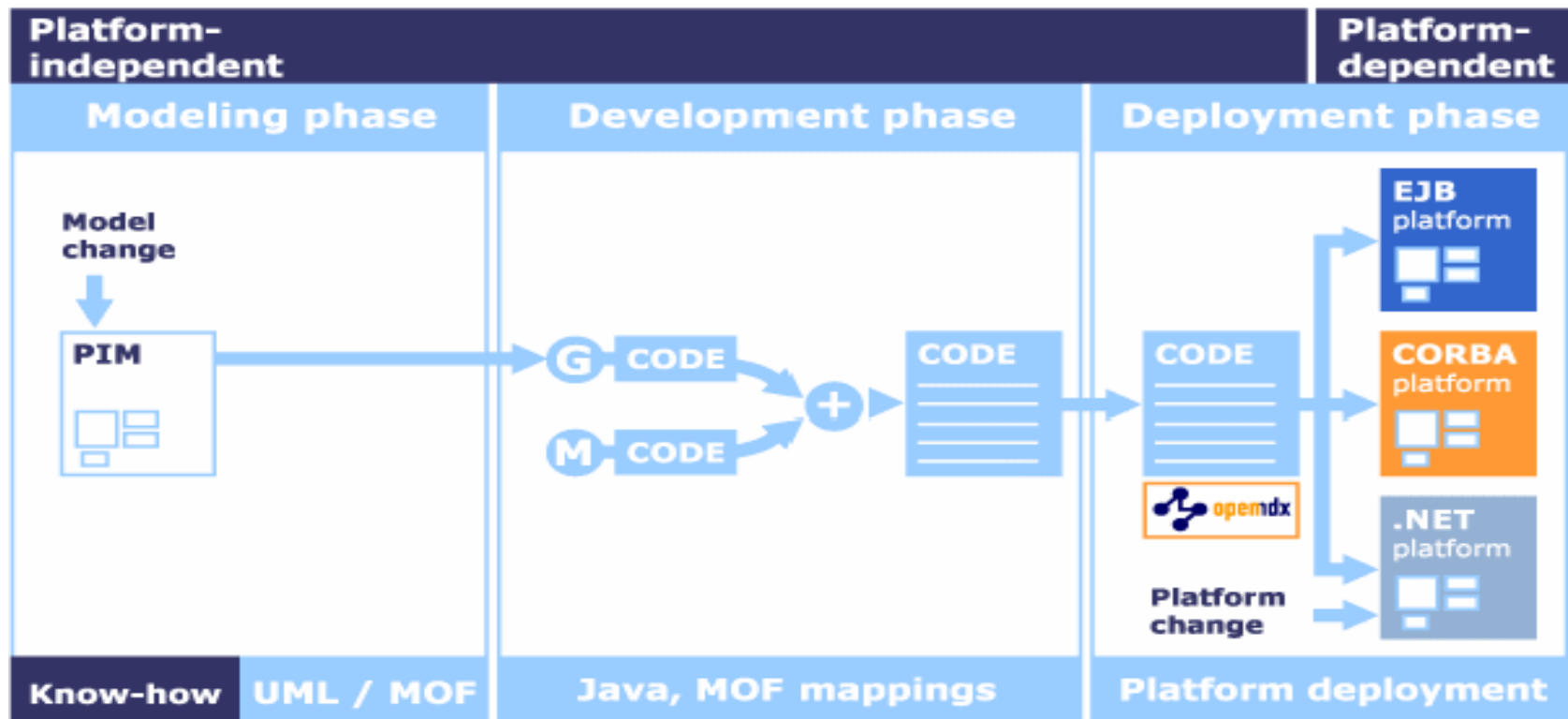
● AndroMDA 4 bietet

- die Definition eigener Metamodelle
 - EMF basierte Metamodelle
 - UML 2 Ecore Converter
- model-to-model-Transformationen via *ATL*
- model-to-text-Transformationen 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 bereit**
 - 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 *Persistence*, *Verteilung* etc...

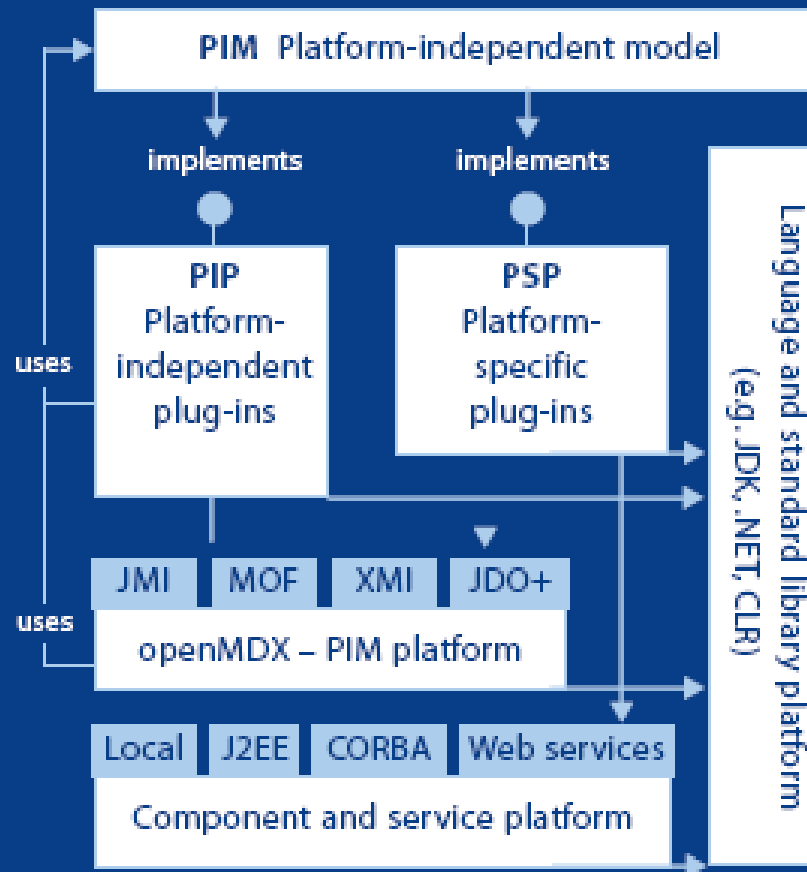
openMDX Entwicklungsprozess



G Generated source
M Manual programming

Quelle: <http://www.openmdx.org/index.html>

openMDX architecture:



architecture

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**

www.iks-gmbh.com