Das Eclipse Modeling Framework (EMF)

SENS
www.SoftwareExperts.de

Dr. Frank Gerhardt
fg@GerhardtInformatics.com

Dieter Moroff
D.Moroff@corag.de

Technik fürs Leben

BOSCH

Dr. Stephan Eberle
Stephan.Eberle@bosch.com

JUGS
java user group stuttgart

Agenda

- **Einführung**

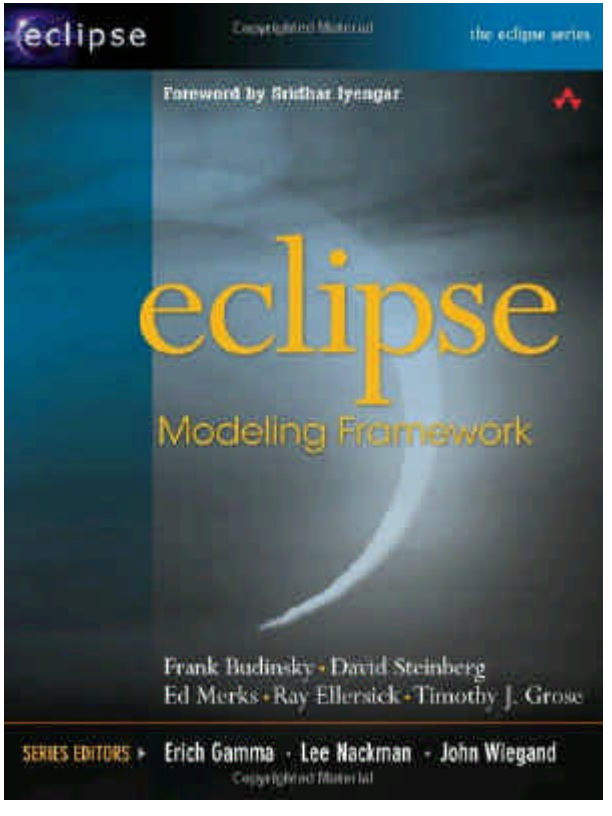
- Eclipse Modelling Framework
  - Demo
  - Modellierung
  - Generierung
  - Runtime
- Objekt-Relationales Mapping

- Zusammenfassung, Fazit

Was ist EMF?

- Ursprünglich von IBM
  - Implementierung von Essential MOF der OMG
- Heute Eclipse.org
- Aktuell EMF 2.2.0

- Demnächst: zweite Auflage des EMF-Buchs



EMF ist kein graphisches Modellierungswerkzeug

Modellierungs-Werkzeuge

- Freie, open source und kostenlose Tools
  - Topcased
  - Omondo EclipseUML
  - eDiagram
  - Argo + argo2ecore
- Kommerzielle Tools
  - IBM Rational Software Architect (ehem. Rose)
  - Borland Together
  - Poseidon
  - MagicDraw (demnächst)

Wo wird EMF verwendet?

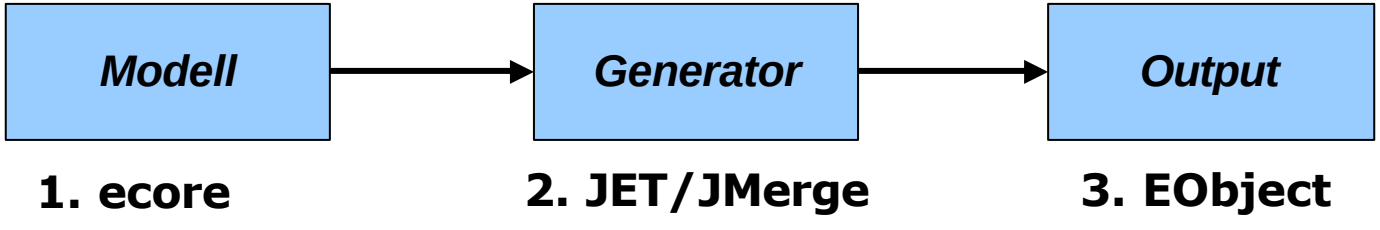
- Eclipse RCP Anwendungen mit komplexem Domänenmodell

- UML2
  - Implementation von UML2 auf Basis von EMF (Analog wie UML auf Basis von MOF spezifiziert wird)
- Graphical Modeling Framework (GMF)
  - Einfaches Erstellen von graphischen Modellen mit GEF und EMF
- Service Data Objects (SDO)
  - Referenzimplementierung der JSR-235 mit EMF für SDO 1.0

- Rational Software Development Platform (RSM, RAD)
- OmondoUML

Was ist EMF?

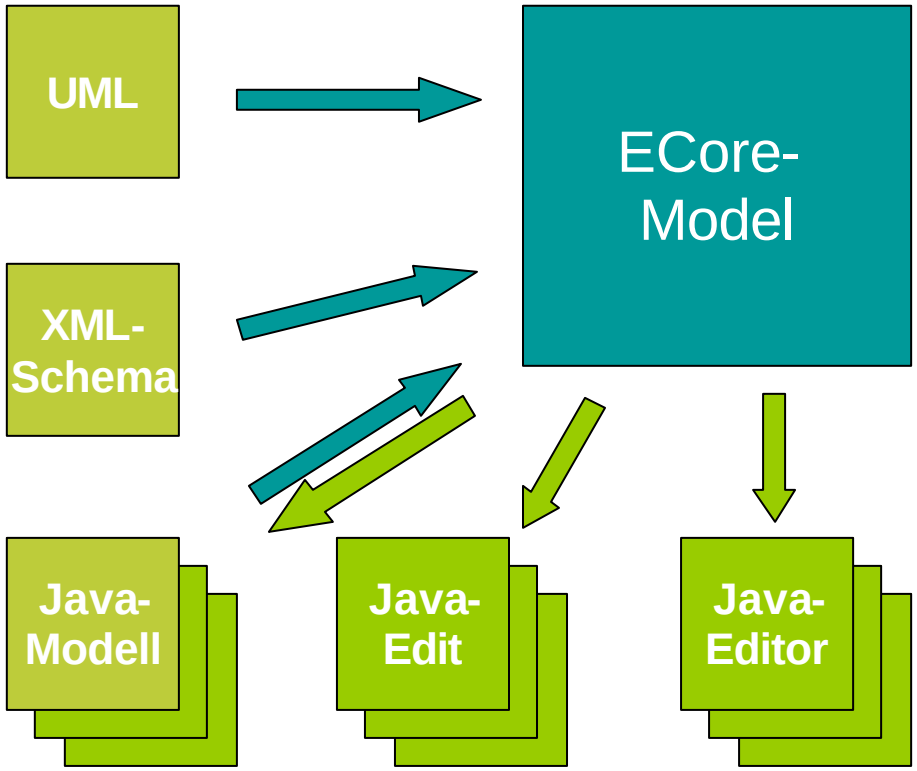
1. Ein Modellierungsspache (Metamodell)
2. Ein Code-Generator (mit Tools und Templates)
3. Ein neues Objekt-Modell für Java-Programme (Laufzeitumgebung)



Warum EMF?

- Code generieren
  - MDSD (Model driven software development)
- Features der Laufzeitumgebung und des generierten Codes nutzen
- Roundtrip, wenn man wieder zurück ins Modell will
- Passt gut zu Eclipse RCP/SDK
- Schnell was machen, Prototyping

Wie wird EMF verwendet?



Drehbuch

- Mdl Datei importieren
- Ecore Editor zeigen, Topcased zeigen
- Gemodel Editor zeigen
- Generieren
- Generierten Code zeigen
- Starten
- Modell anlegen, speichern (XML zeigen), schließen, öffnen, undo/redo

Typisches Szenario

- Erzeugung eines EMF Modells
  - Import UML (z.B. Rational Rose .mdl file)
  - Import XML Schema
  - Import annotated Java interfaces
  - Direkte Erzeugung eines Ecore Modells mit dem EMF Ecore Editor oder Omondo's (freien) EclipseUML grafischen Editor
- Java Source-Code aus Modell generieren
- Iterativ Java Anwendung entwickeln
- Modell verfeinern; Java Source-Code generieren
- Modell-Instanz mit den generierten EMF Modell Editor füllen und testen
- EMF.Edit als Basis verwenden um ein eigenes Userinterface zu bauen

Model Import

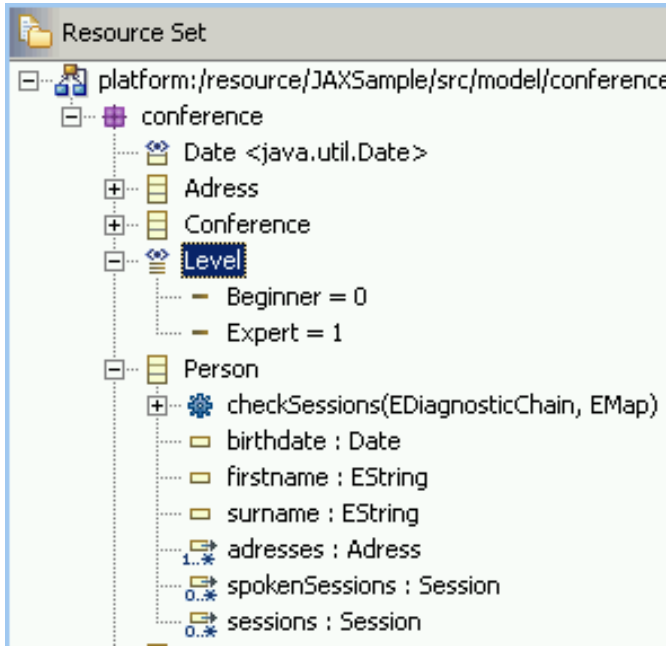
- UML
  - Rational Rose .mdl Dateien direkt unterstützt
- Annotated Java
  - Java Interface für jede Modell Klasse
  - Annotations verwenden **@model** Tag um Modell-Definitionen einzuführen, die sich sonst nicht im Code ausdrücken lassen (z.B. Typ einer Collection)
  - Keine zusätzlichen Tools erforderlich, Round-Trip möglich
- XML Schema
  - Produziert komplizierteres EMF Modell als mit Java oder UML
  - Spezielle Schematas notwendig, kann kein beliebiges Schema verwenden
- Ecore Modell (*.ecore file)
  - Nur Generatormodell wird noch generiert.

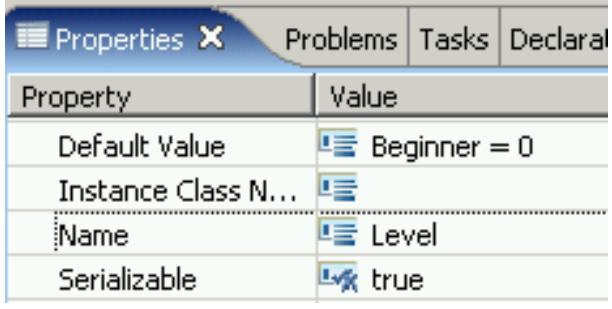
Model Creation

- ECore Modell wird in einem Eclipse Projekt mittels eines Wizards erzeugt, Quellen siehe „Modell Import“
- Erzeugt wird:
  - **modelname.ecore** Datei
    - Ecore Modell im XMI Format
    - Kanonische Form des Modells
  - **modelname.genmodel** Datei
    - Ein Modell um den Generator zu steuern
    - Dekoratiert .ecore Datei
    - EMF Code Generator ist selbst ein EMF .genmodel Editor
  - **.genmodel** und **.ecore** Dateien werden automatisch synchronisiert**.**

Modell Editor

- Ein generierter (und angepasster) EMF Editor für Ecore Modelle
- Modelle können im Treeview zusammen mit dem Propertyview editiert werden
  - Neue Komponenten (EClass, EAttribute, EReference, etc.) werden über die Popup-Menü im Treeview angelegt
  - Namen, etc. werden im Propertyview editiert.

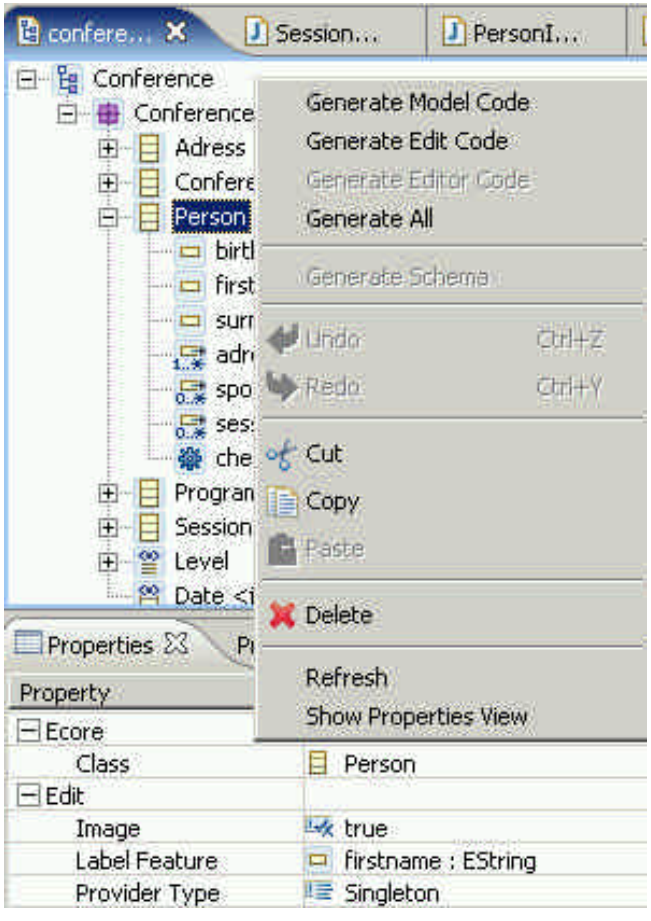




GenModel Editor

Wiederum ein generierter und angepasster EMF .genmodel Editor

- Kontextmenü Eintrag um den Codegenerator zu starten:
  1. Die **Generate Model Code** Aktion generiert den Code des Modells
  2. Die **Generate Edit Code** Aktion generiert den Adapter-Code zur Unterstützung von Viewern
  3. Die **Generate Editor Code** Aktion generiert eine vollständigen Eclipse Editor für das Modell
  4. Die **Generate All** Aktion generiert alles zusammen
- Generation Eigenschaften können im Propertiesview verändert werden



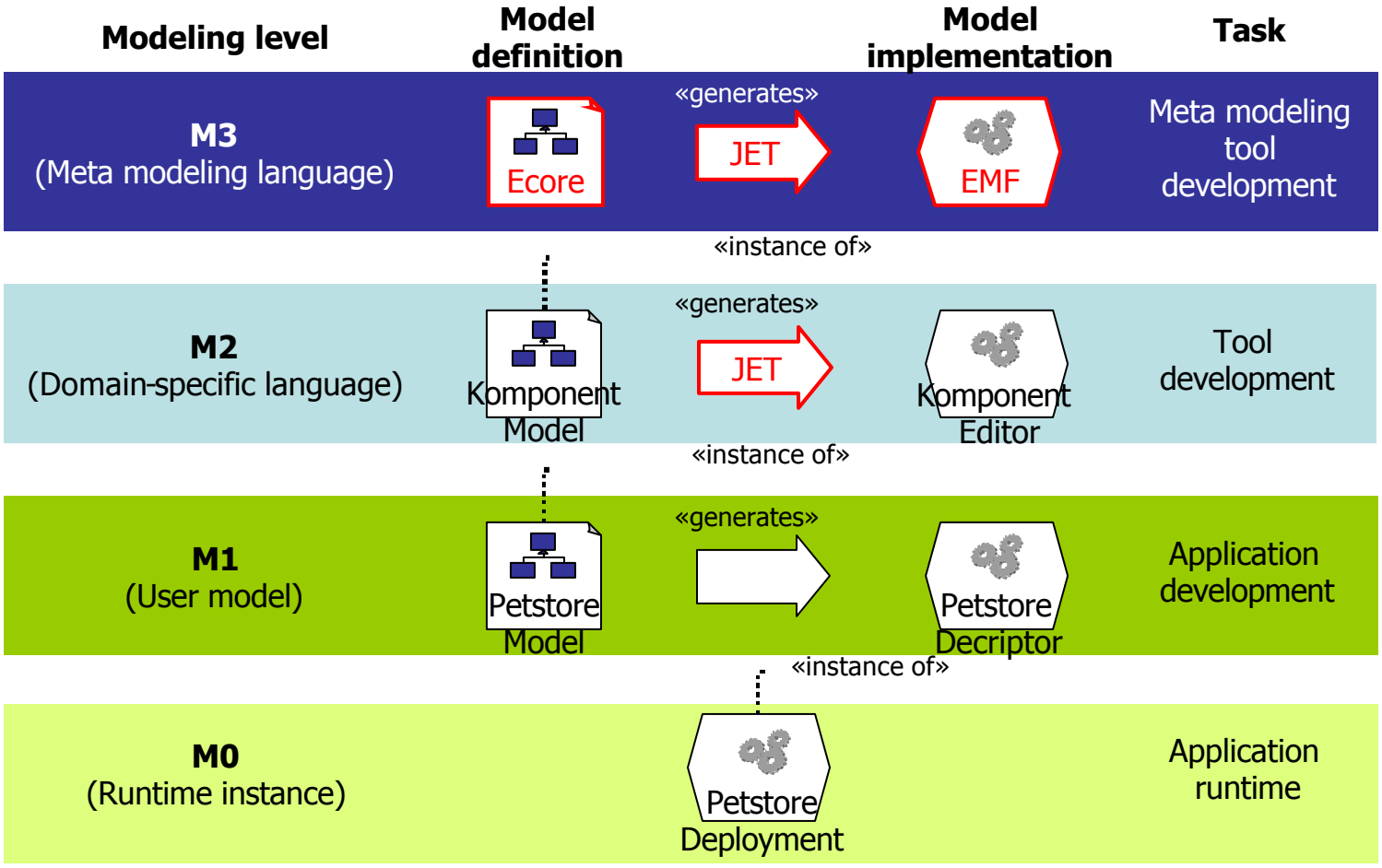
EMF.Codegen - Tools

- GenModel
  - Steuerung des Generators (Dekoratormodell)
- JMerge
  - Mischt neu generierten Code mit verändertem Code und formatiert gemäß Eclipse Code Formatter Preferences
- JET
  - Template Engine mit JSP-ähnlicher Syntax
  - Auch außerhalb des EMF-Generator einsetzbar.
  - Nature ermöglicht automatisches Übersetzen der JET-Templates
  - Editieren mit JSP-Editor (z.B. Lomboz) oder speziellem JET Editor.

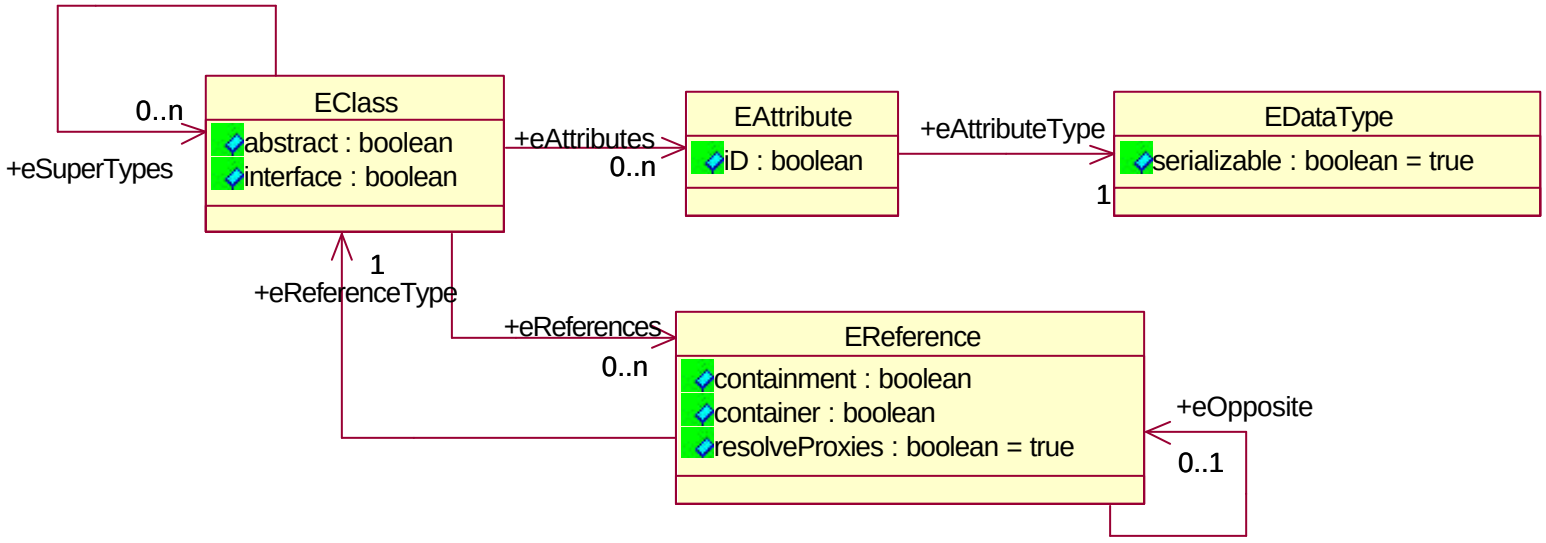
Demo - Tooling

Modellierung

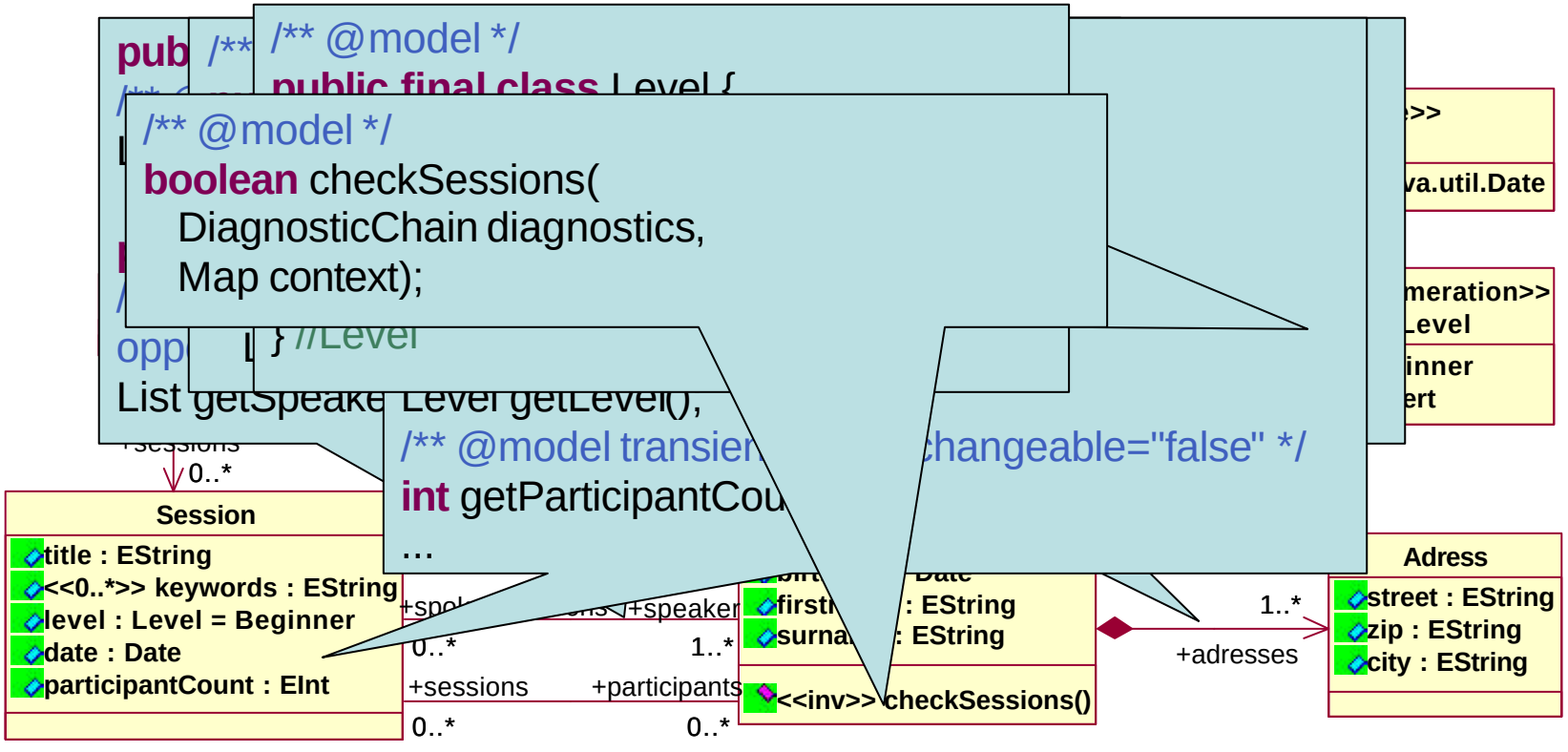
OMG – Modellebenen



EMF Metamodell - ECore



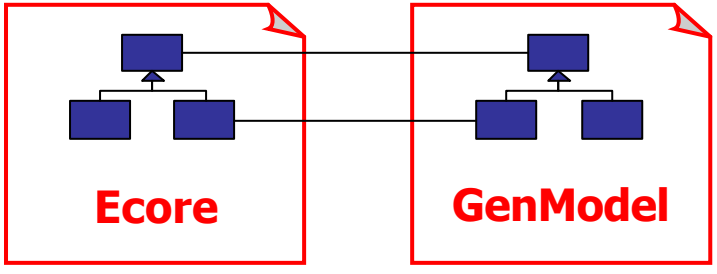
Beispiel eines EMF-Modells



Generierung

EMF - GenModel

- Marking-Model mit den zusätzlichen Generierungsinformation
- Automatische Synchronisation mit Ecore-Modell beim Reload
- GenModel-Editor ist Startpunkt für Generierung

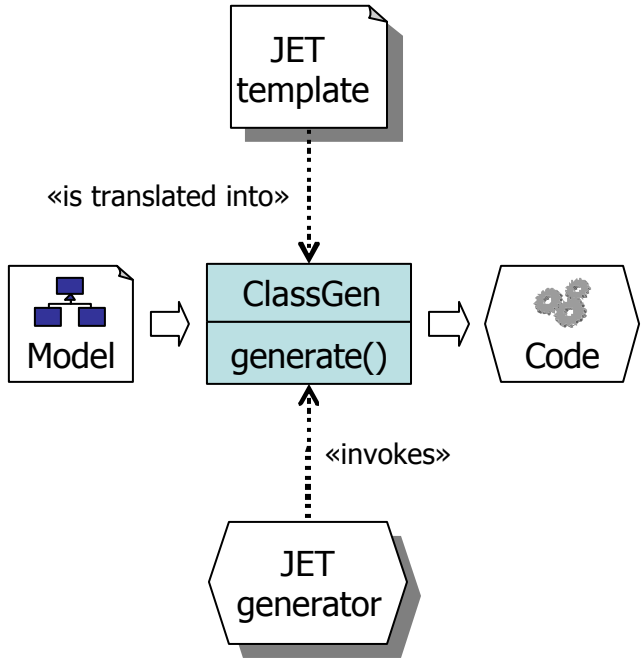


Struktur-information (PIM)

Plattform-information (PSM)

JET – Java Emitter Templates

```
<%@ jet package="org.examples.library" imports="org .. ecore.*" class="ClassGen"%>
<%EClass eClass = (EClass) argument;%>
Name of EClass to be processed: <%=eClass.getName()%>
```



JMerge – Integration von nichtgeneriertem Code

- Code mit @generated NOT wird beim Neugenerieren nicht überschrieben.

```
/**
 * <!-- begin-user-doc -->
 * <!-- end-user-doc -->
 * @generated
 */
public int getParticipantCount() {
// TODO: implement this method to
// return the 'Participant Count' attribute
// Ensure that you remove @generated
// or mark it @generated NOT
throw new UnsupportedOperationException();
}
```

```
/**
 * <!-- begin-user-doc -->
 * Liefert die Anzahl der Teilnehmer
 * <!-- end-user-doc -->
 * @generated NOT
 */
public int getParticipantCount() {
   return getParticipants().size();
}
```

EMF vs. MDA

MDA-Prinzip	EMF-Anwendung
Plattform unabhängige Modelle (PIM)	Ecore Modelle bis aus Verwendung <<datatype>> plattformunabhängig
Plattform spezifische Modelle (PSM)	GenModel enthält Information für Plattform EMF-Runtime
MOF (M3)-Metamodell	ECore implementiert EMOF (Essencial MOF)

→ **EMF verwendet die Prinzipien der MDA auf pragmatische Art und Weise für die Eclipse Plattform**

→ **EMF ist kein universelles MDA / MDSD Framework**

EMF als Basis für MDSD

- EMF zur Definition von Domain Specific Languages (DSL)
- Generierung des DSL-Metamodell mit EMF-Tooling
- Generierung von graphischen Editoren mit dem graphischen Modelling Framework (GMF)
- Erstellen von DSL Instanzen
- Codegenerierung aufgrund der DSL Modelle z.B mit openArchitectureWare (http://www.eclipse.org/gmt/oaw)

Runtime

EMF-Objekte vs. POJOs

- „Angereicherte“ Modellimplementierung
  - Erweitert mächtige Runtime-Framework-Klassen
  - Enthält modellspezifische Runtime-Logik-Anteile
  - Vielfältiger und komfortabler Zugriff auf Modellinstanzen
- Zusätzliche Editierlogikschicht
  - Darstellung von Modellinstanzen
  - Editieren von Modellinstanzen

Runtime-Features Modell

- **Semantikerhaltendes Handling von Referenzen**
  - bidirektionalen Referenzen
  - Containment-Referenzen
- **Change Notification**
  - von Settern im Modell an externe Observer/Adapter
- **Generische Persistenz API**
  - Speichern von Modellen in ein oder mehreren Resourcen
  - Cross-Resource-Referenzen und Lazy Loading
  - Default-Implementierung für Dateien im XML/XMI-Format
- **Effiziente reflektive API**
  - Laufzeitzugriff auf Ecore-Modelldefinition
  - Reflektive Getter & Setter und Factory-Methode

Runtime-Features Edit

- **Content-/Label-Provider** zur Darstellung von Modellinstanzen in Views und Editoren
- **Viewer-Refreshing** bei Änderungen in Modellinstanz
- **Command-basiertes Editieren** von Modellinstanzen mit Undo/Redo-Möglichkeit

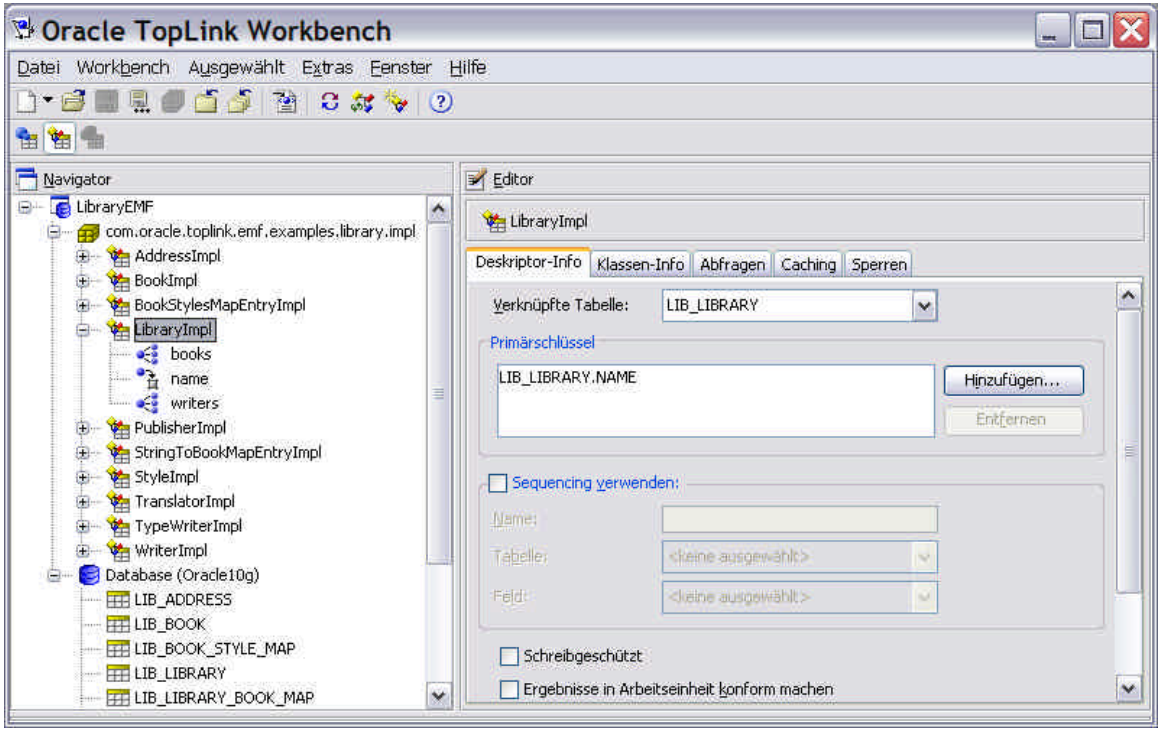
Advanced Runtime-Features

- **Validator** zum Prüfen der Modellkonsistenz
- **Adapter** zum Beobachten von Änderungen und/oder Erweitern des Verhaltens
- **TreeIterator** zum Iterieren über alle Knoten eines Modellbaums
- **CrossReferencer** zum Suchen nach Referenzen auf Modellobjekte
- **ChangeRecorder** zum Aufzeichnen Verarbeiten von Modelldeltas
- **JET-Engine** zur Generierung textbasierter Ausgaben aus Modellinstanzen

O/R-Mapping

O/R Mapping erstellen

Zuordnung von Klassen/Attributen zu Tabellen/Tabellenfeldern mit **Oracle TopLink Workbench**:



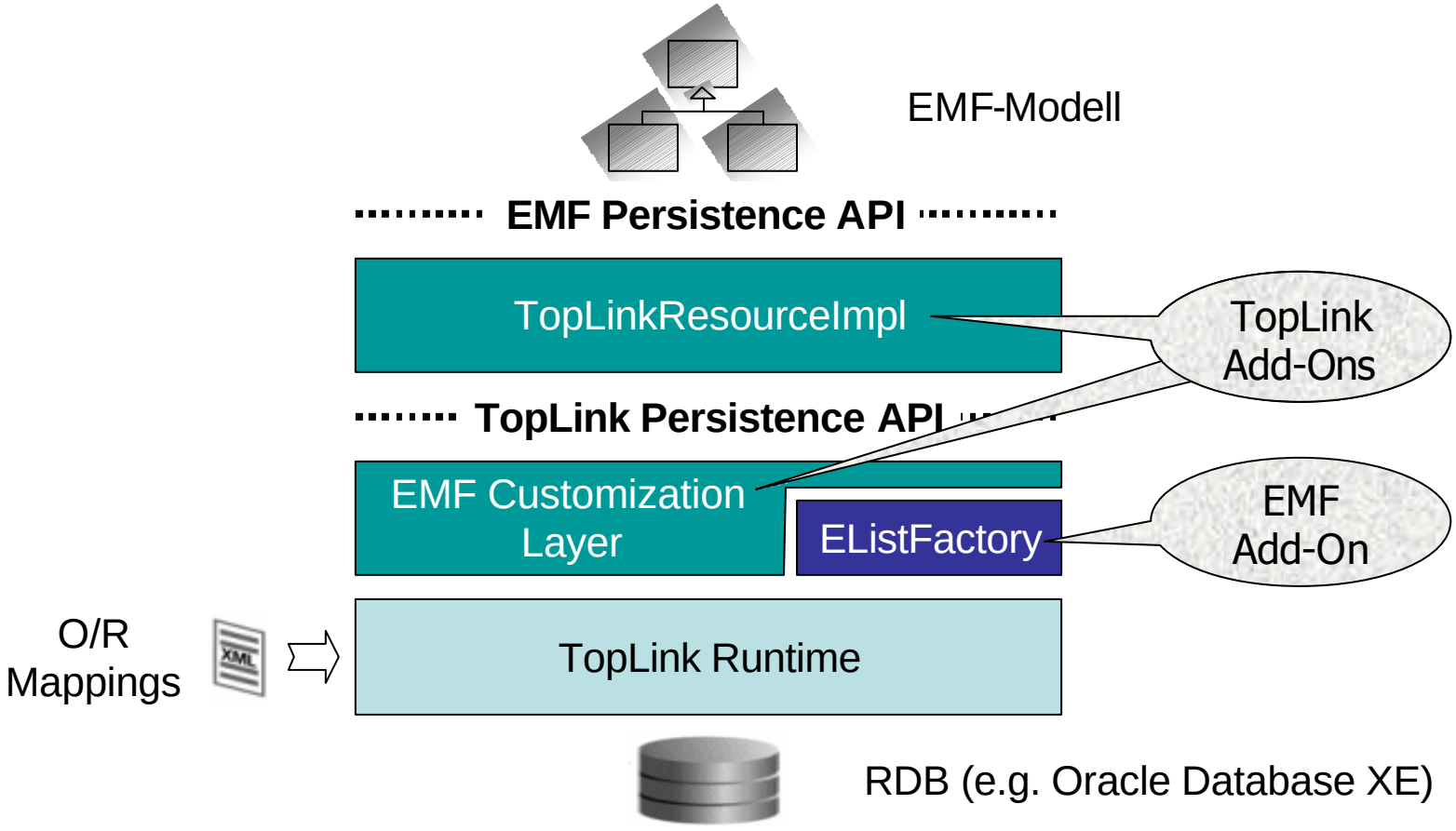
O/R Mapping nutzen

Abfragen/Ändern/Committen von Objekten aus/in Datenbank mit **Oracle TopLink API**:

LibraryWriterTest.java

```
@SuppressWarnings("unchecked")
public void testLibraryWriter() {

  UnitOfWork uow = session.acquireUnitOfWork();

  // read existing library object from database
  ReadObjectQuery query = new ReadObjectQuery(LibraryImpl.class);
  Expression where = new ExpressionBuilder().get("name").equal("JUGSLibrary");
  query.setSelectionCriteria(where);
  Library library = (Library) uow.executeQuery(query);

  // create and add new writer object
  Writer newWriter = LibraryFactory.eINSTANCE.createWriter();
  newWriter.setName("The Java Expert");
  library.getWriters().add(newWriter);

  // save changes to database
  uow.commit();
}
```

O/R Mapping mit EMF



Demo O/R Mapping

Zusammenfassung, Fazit

- Pragmatisch, leistungsfähiges Toolkit für modellbasierte Entwicklung
- Vielenorts aber auch limitiert, z.B.
  - kein Multithreading
  - nur rudimentäre Validierung
  - fehlende Datenbankanbindung
- Erfordert zeitweise Bereitschaft zu Kompromissen bzw. Vornahme von Erweiterungen
- Erfordert modellbasierte Denke, die z.T. noch wenig verbreitet ist

Zusammenfassung, Fazit

- Im Hintergrund teilweise sehr komplexe Architektur von EMF
- Kein universelles erweiterbares MDA Generatorframework
- Sehr gute Verbindung unterschiedlicher Modelle durch modellübergreifende Referenzen
- Idealer Ausgangpunkt für DSLs
- In Zukunft grosses Potential im Zusammenhang mit GMF

Zusammenfassung, Fazit

- Ideale Modellbasis für GEF
- Sehr schnell zu lauffähigen Modellen
- Einfacher Test des Modells mit generierten Editor
- Generierter Code Ausgangsbasis für eigene Editoren
- Echter Roundtrip möglich, nicht nur protected Regions
- Modell wie Source Code behandeln
  - Nicht den generierten Code einchecken
  - Im Nightly Build jedes Mal neu generieren
- Gefahr der Teamaufteilung in Modell-Gurus und Analphabeten

Danke für's Zuhören. Fragen?

www.SoftwareExperts.de

Dr. Frank Gerhardt
fg@GerhardtInformatics.com

Dieter Moroff
D.Moroff@corag.de

Dr. Stephan Eberle
Stephan.Eberle@bosch.com

Resources

- EMF Overview
  http://download.eclipse.org/tools/emf/scripts/docs.php?doc=references/overview/EMF.html
- EclipseCon 2005
  http://www.eclipsecon.org/2005/
  - *Models to Code with the Eclipse Modeling Framework*
    Ed Merks, Dave Steinberg
    http://www.eclipsecon.org/2005/presentations/EclipseCon2005_Tutorial11final.pdf
  - *Mastering Eclipse Modeling Framework*
    Vladimir Bacvanski, Petter Graff
    http://www.eclipsecon.org/2005/presentations/EclipseCon2005_Tutorial28.pdf
- *Agile MDA*, Stephen J. Mellor
  http://www.omg.org/agile/

Lizenz

- Dieser Vortrag ist lizenziert unter EPL mit Ausnahme der verwendeten Logos
- Er enthält Material aus den folgenden Tutorials
  - *Models to Code with the Eclipse Modeling Framework*
    Ed Merks, Dave Steinberg
    http://www.eclipsecon.org/2005/presentations/EclipseCon2005_Tutorial11final.pdf
  - *Mastering Eclipse Modeling Framework*
    Vladimir Bacvanski, Petter Graff
    http://www.eclipsecon.org/2005/presentations/EclipseCon2005_Tutorial28.pdf

Einführung

- Über MDA, Modellieren, Code-Generierung…

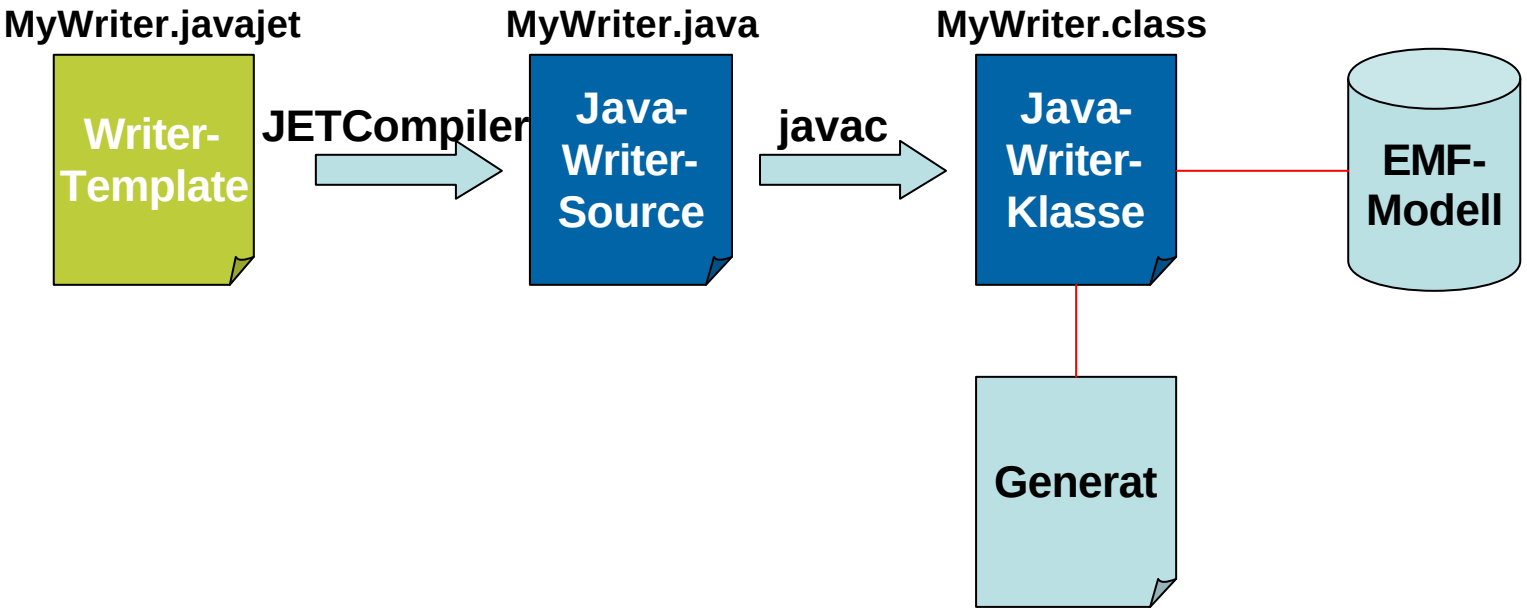
Kontext

- Seit wann?
- Wieviele Downloads?
- Was ist ein Modell
- Wo kommt EMF her
- EMF wird z.B. extensiv in WSAD verwendet
- Was kann man damit machen

Was ist ein EMF Modell?

- Defining an EMF Model
  - UML: "an EMF model is essentially the class diagram subset of UML" p. 14
  - XMI
  - Annotated Java
  - XML Schema
- Generating a Java Implementation

JET – Prinzip: Compilierte Templates



JET - Beispiel

```
<%@ jet package="de.eclipseteam.jet.demo" class="HtmlDocWriter„
    imports="org.eclipse.emf.ecore.* java.util.*" %>
<% EPackage modelPackage = (EPackage)argument; %>
<html>
<head>
<title>Modeldokumentation</title>
</head>
<body>
<h1>Package <%=modelPackage.getName()%></h1>
<% for ( Iterator ic = modelPackage.getEClassifiers().iterator();
   ic.hasNext(); ) {
      EClassifier modelClassifier = (EClassifier)ic.next();
      if ( modelClassifier instanceof EClass ) {
         EClass modelClass = (EClass)modelClassifier;
```

Adapter erstellen und nutzen (1)

- Vorgehen
  - Adapter von AdapterImpl ableiten und implementieren

```
public class MyAdapter extends AdapterImpl {
}
```

  - AdapterFactory von <ModelName>AdapterFactory ableiten und Adapter-Factory-Methoden für betreffende Modellelemente überschreiben

```
public class MyAdapterFactory extends MyModelAdapterFactory {
}
```

  - adapt-Methode aus AdapterFactory aufrufen und betreffendes Modellelement sowie gewünschte Adapterklasse übergeben

```
MyAdapterFactory.INSTANCE.adapt(modelElement, MyAdapter.class);
```

Adapter erstellen und nutzen (2)

- Tipps & Hinweise
  - Falls Adapter des gewünschten Typs bereits existiert, so wird kein neuer erzeugt sondern vorhandener zurückgeliefert
  - Vereinbarung einer gemeinsamen Adapter-Factory-Methode für mehrere Modellelemente möglich, indem Adapter-Factory-Methode des gemeinsamen Superelements überschrieben wird