Eclipse Modeling Framework (EMF)

03.02.2010

Mich und Laurent

Agenda

- Motivation
- Grundlagen
- Technologien
- Manipulation
- Ecore
- Genmodell
- Demo
- Persistenz
- Notification
- Ausblick GMF
- Fazit / Quellen

Motivation

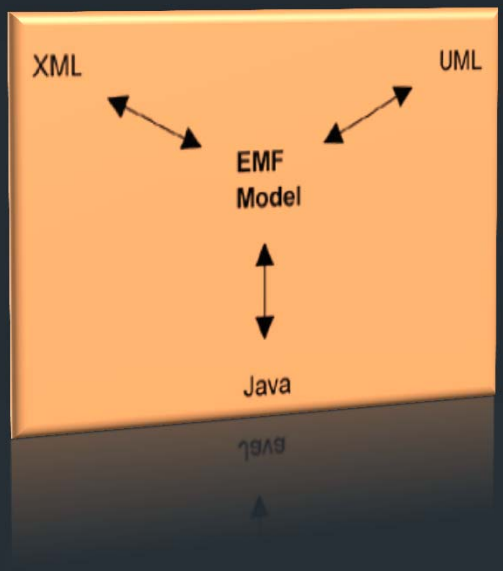
- Soll ich Modellieren oder Programmieren?
  → sowohl als auch!!!
- Produktivitäts-Steigerung durch Code-Generierung (und Fehlerminimierung)
- Software (Modell) leicht erweiterbar
- Annehmlichkeiten eines Frameworks nutzen (Notification, Persistenz, …)

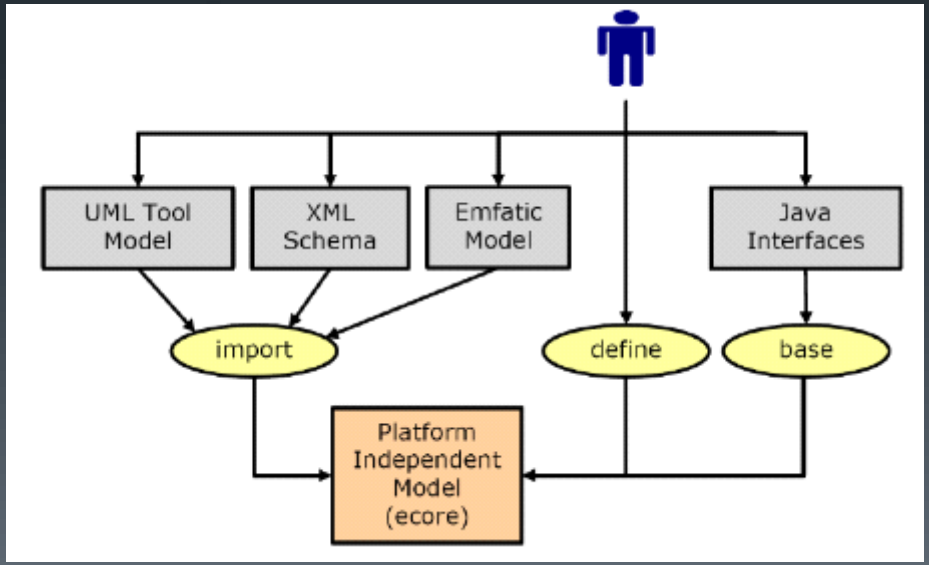
Grundlagen

- EMF ist ein eigenständiges Eclipse-Projekt
- Modellgetriebene Entwicklung
  - Pragmatischer Ansatz von Model-Driven Software Development (MDSD)
- Quelltextgenerierungs-Framework zur Erzeugung diverser Quellcode Artefakte
- Modell-Input/Output:
  - XML-Schema
  - Java-Code mit Annotations
  - UML-Diagramm
  - Baum-Editor
- Modell-Funktionalitäten:
  - Instanzierung
  - Abfragen
  - Manipulieren
  - Serialisieren, validieren, Persistenz
  - Notification für MVC
  - Generieren von JUnit-Code, um den generierten Code zu testen

Technologien

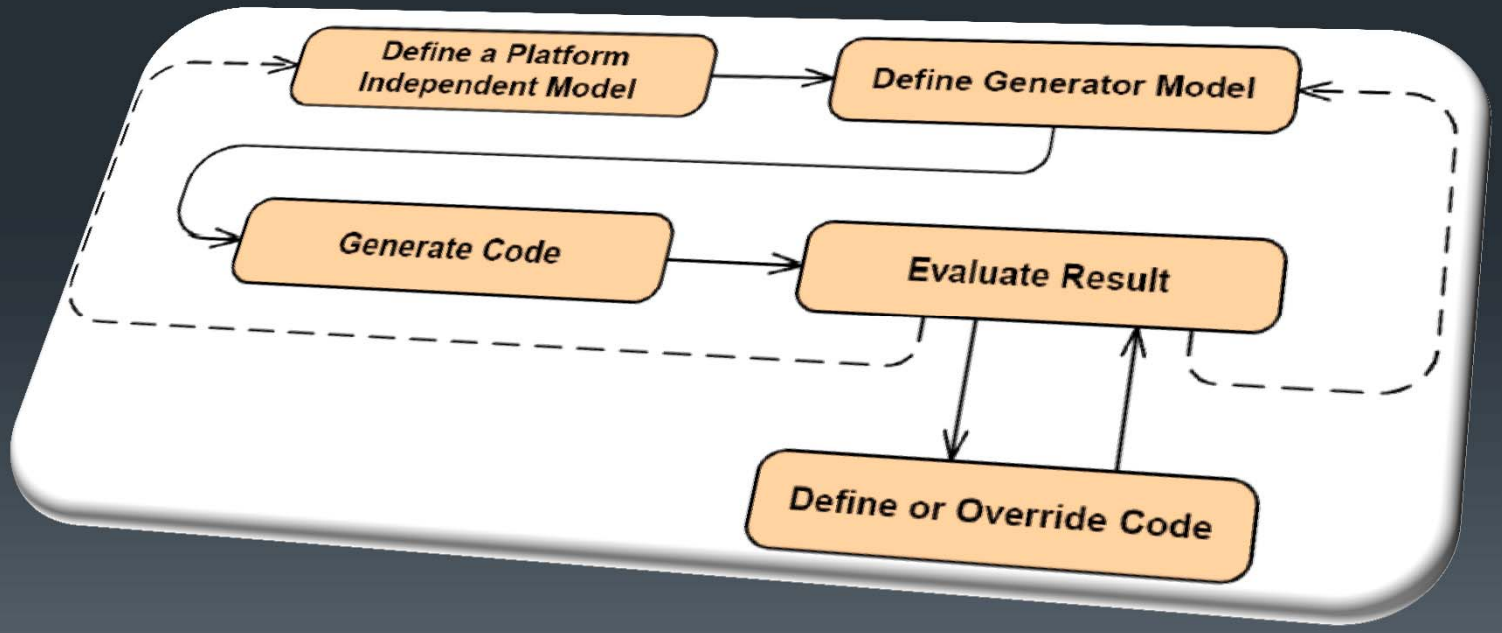
- Vereinigt drei Technologien:
  - XML
  - UML
  - Java
- EMF ist die abstrahierte Notation, welche die Repräsentationen verbindet
- Modellierung des Datenmodells:
  - UML
  - Java mittels Annotations
  - XMI (XML mit vordefinierten Tags)





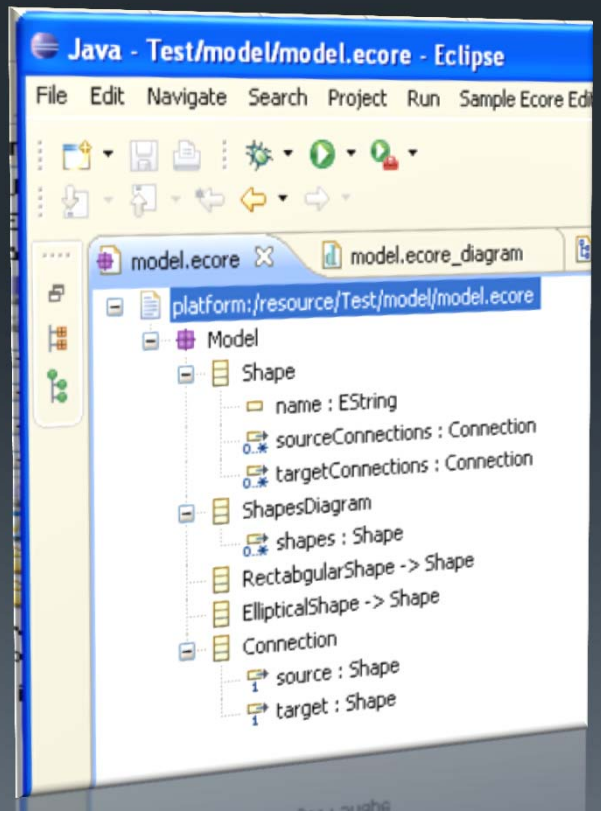
Manipulation

- Veränderung des Datenmodells werden in allen Repräsentation nachgezogen



Begriff «ecore»

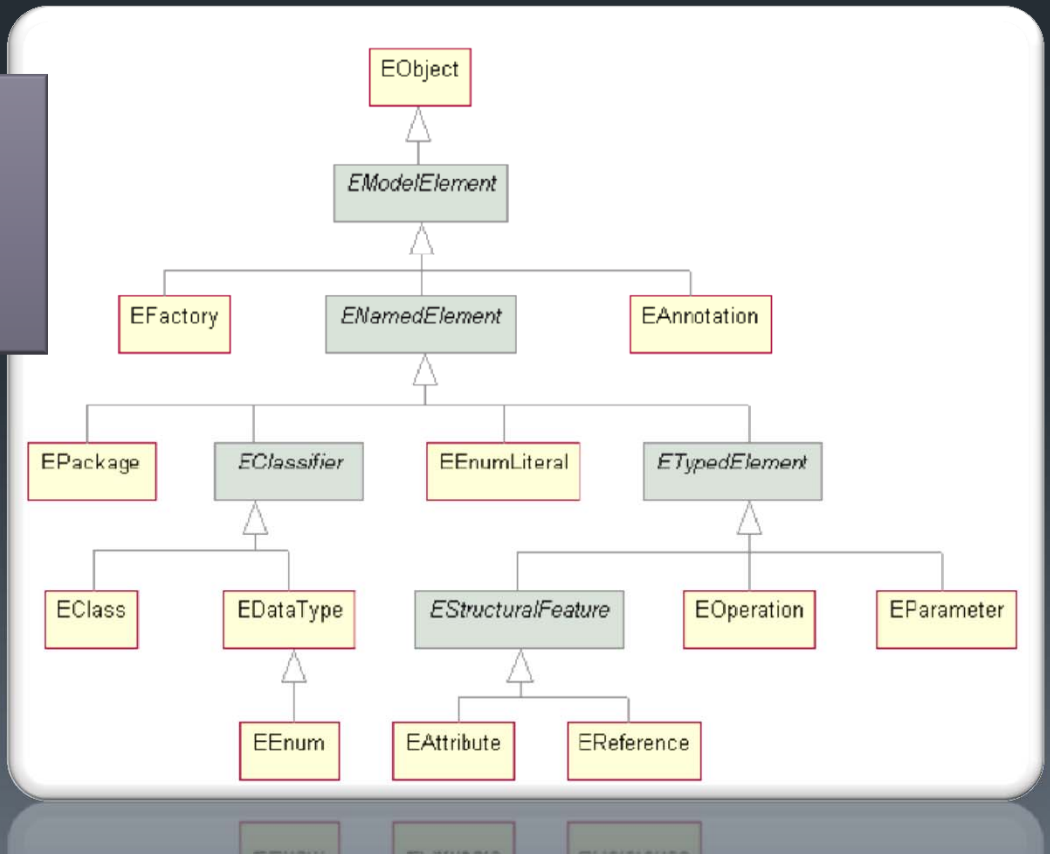
- ecore ist die Meta-Sprache zur Definition des Daten-Modells
- Grundlage für die Quelltexterzeugung
- Eclipse bietet eine Baum-Editor für die Erfassung des ecore-Modells (siehe Bild)
  - Gewisse Eigenschaften können nur im Baum-Editor gesetzt werden
- Salopp betrachtet, entspricht es dem Klassendiagramm
- Konkret: Spezifikation der Daten einer Anwendung (Nutzdaten)
  - Objekte mit Attributen
  - Relationen (Assoziationen) zwischen Objekten
  - Operationen auf Objekten
  - Einfache Bedingungen (z.B. Multiplizität) von Objekten und Relationen



ecore-Metamodell

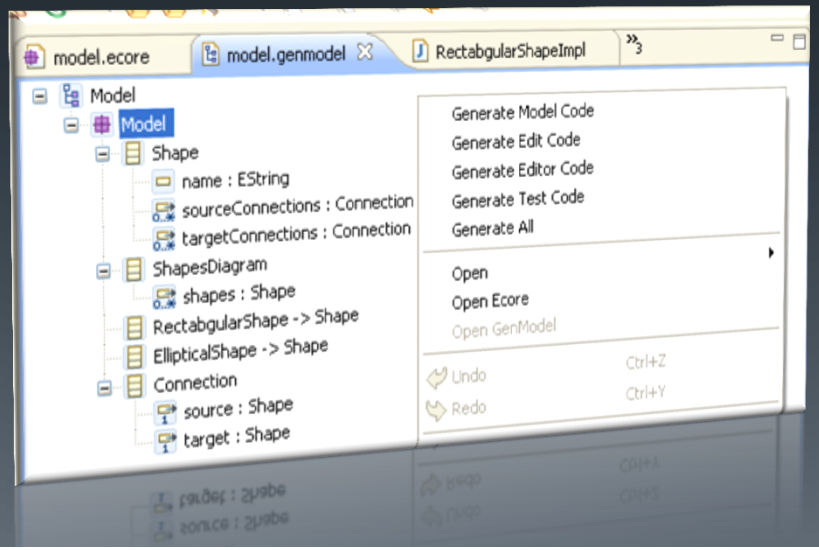
- Daten-Modell unserer Anwendung leitet schlussendlich von EObject ab:
  → primär EClass, EAttribute, EReference
  → Persistenz, Notification, …

```
/**
 * @model abstract="true"
 * @generated
 */
public interface Shape extends EObject
{
  ...
}
```

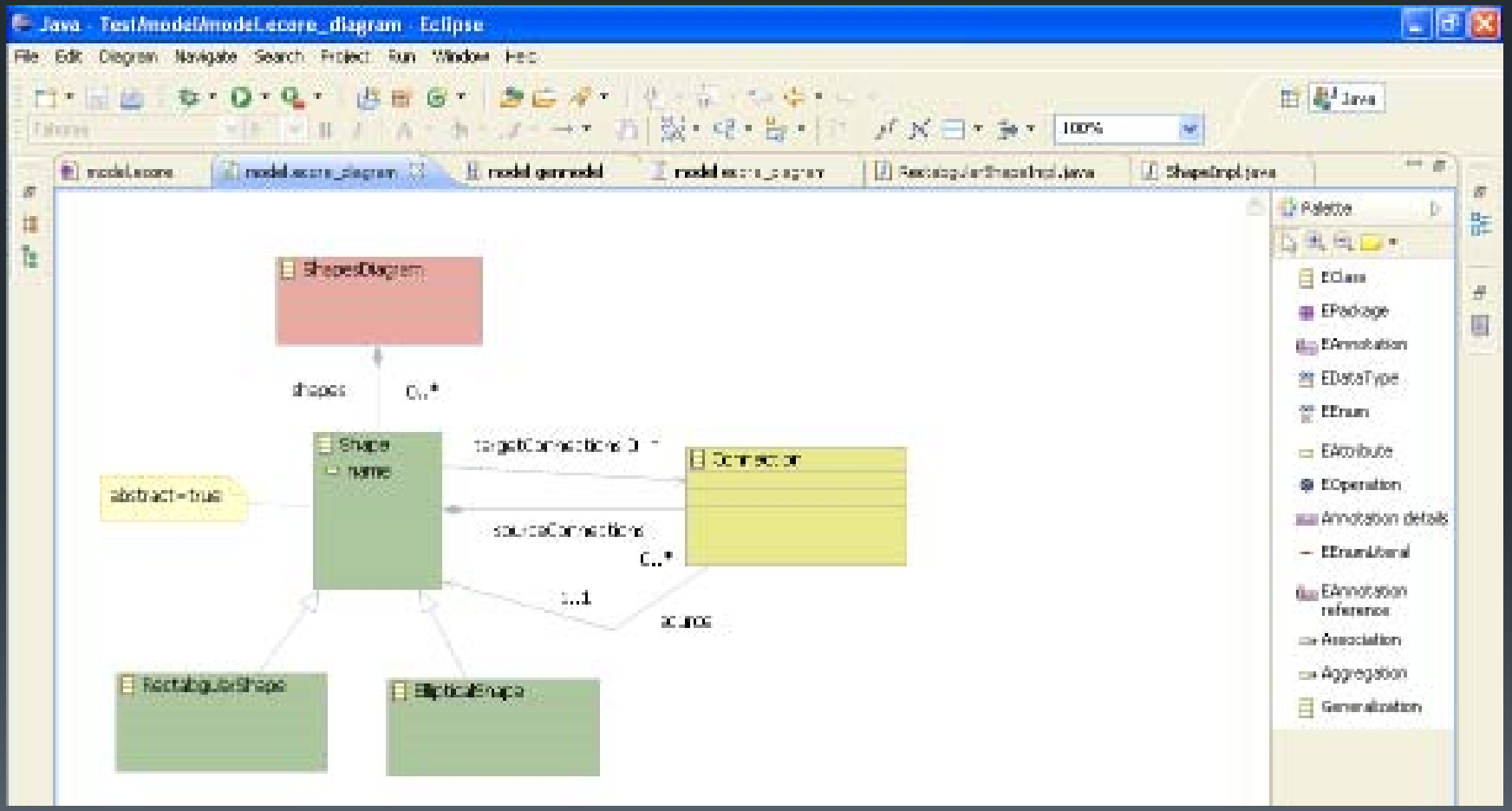


Begriff «genmodel»

- durch genmodel wird die Quelltexterzeugung ermöglicht
- im Gegensatz zur ecore stellt es plattformspezifische (CPU/OS) Informationen zu Verfügung
- Mittels genmodels kann konfiguriert werden, wie der Quelltext generiert werden soll:
  - Angabe der weiter benötigten Package für die Code-Generierung
  - Spezifikation des Base Package
- Vorgang wird meistens durch einen Wizard unterstützt



Demo (Daten-Modell)



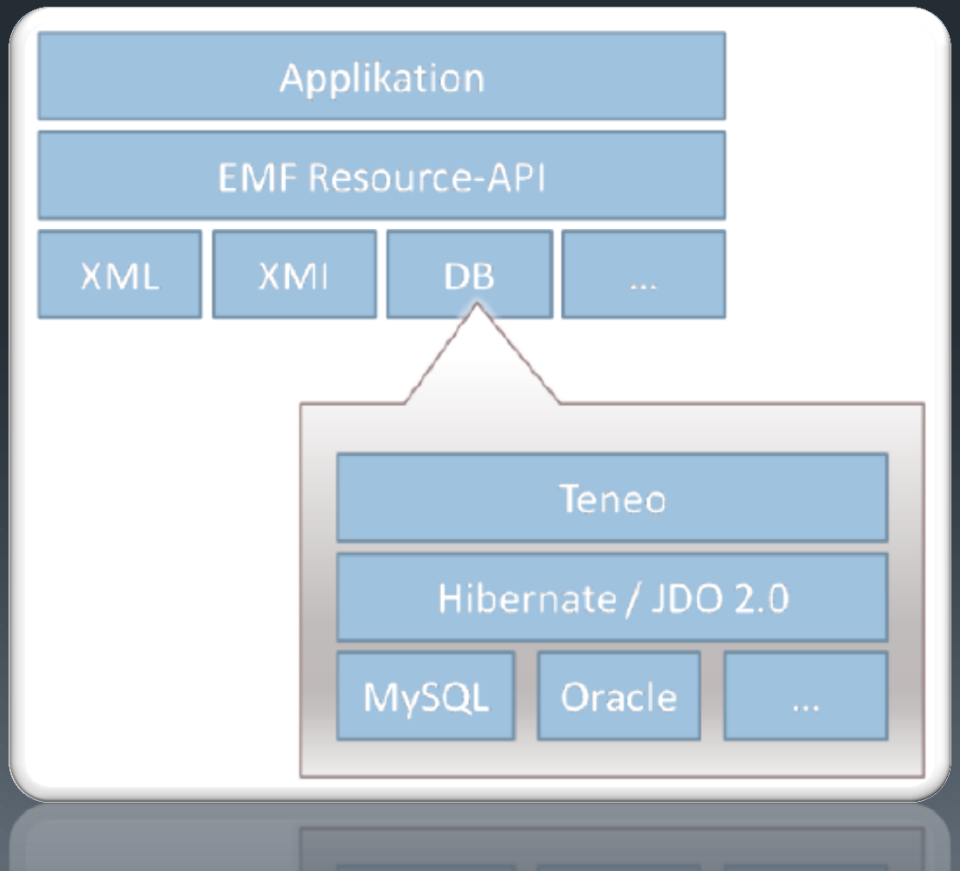
Persistenz

- Standardmässig wird eine dateibasierte Persistenz auf XML-Basis angeboten
- Beispiel Speichern/Laden eines EObject in XMI:

```
// ResourceSet erzeugen
ResourceSet resourceSet = new ResourceSetImpl();

// URI zur Modelldatei (XMI) festlegen
URI fileURI = URI.createFileURI(new File("anyModelFile.xmi").getAbsolutePath());

// Resource für Modelldatei anlegen
Resource resource = resourceSet.createResource(fileURI);

// Ein Objekt vom Typ Vertrag hinzufügen
RectabgularShape rectangle = EvmsFactory.eInstance.createRectabgularShape();
resource.getContents().add(rectangle);

// Inhalt von Resource speichern
resource.save(Collections.EMPTY_MAP);
```

Persistenz mittels Teneo

- Plugin EMF Teneo
- Ermöglicht die Anbindung an eine beliebige relationalen Datenbank
- Nutzt vorhandene O/R-Mapping-Technologien wie Hibernate oder EclipseLink



Notification

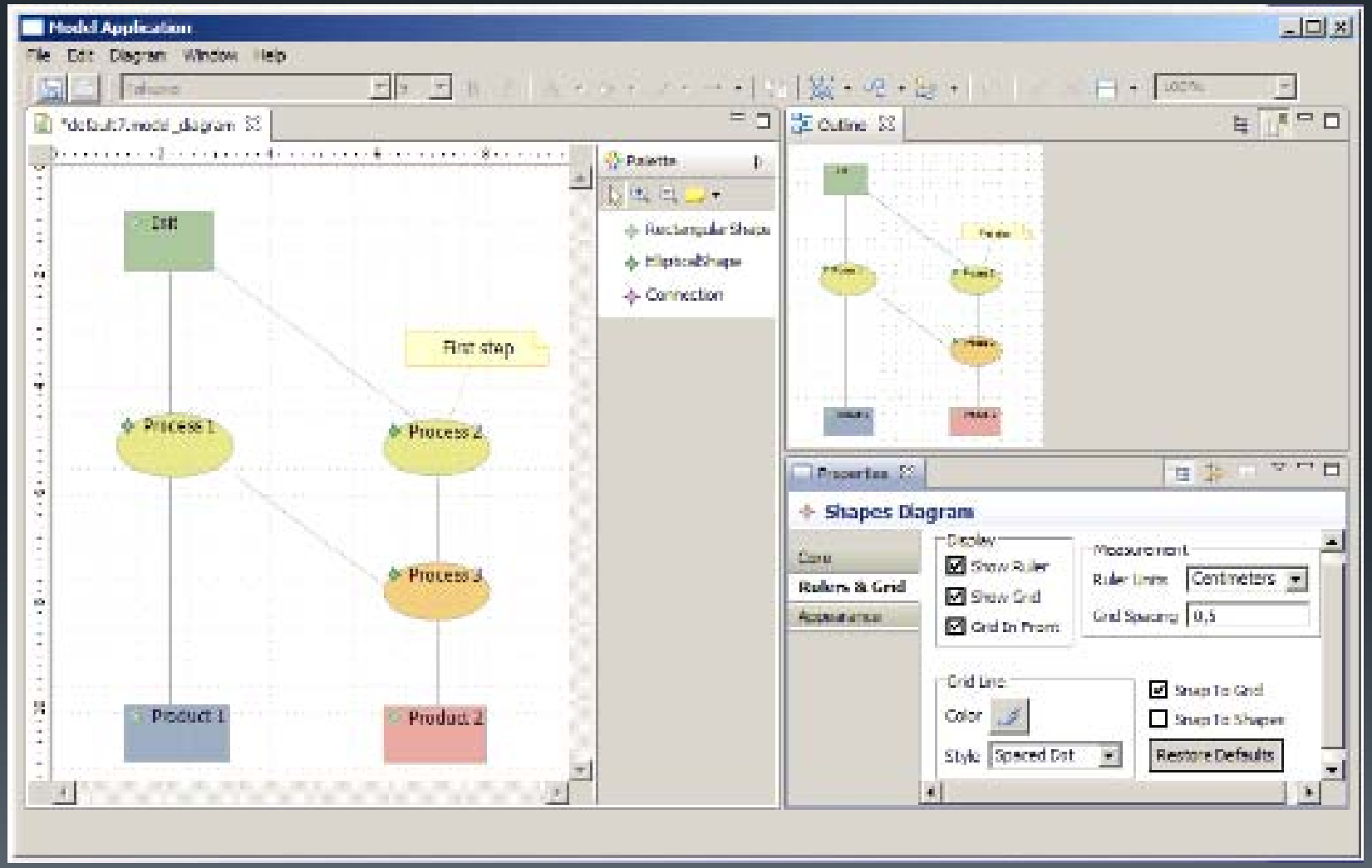
- Ist ein Benachrichtigungs-Mechanismus bei Änderungen eines Objektes
  → Observer-Pattern
- Dient zur Aktualisierung von Views und abhängigen Objekten (MVC)
- Notification ist wichtiger Bestandteil von EMF
- Jedes Objekt des Daten-Modells leitet von EObject ab und implementiert somit automatisch Notifier

```
public interface EObject extends Notifier { ... }

public interface Shape extends EObject { ... }

public abstract class ShapeImpl extends EObjectImpl implements Shape {
...
    /** @generated */
   public void setName(String newName) {
      String oldName = name;
      name = newName;
      if (eNotificationRequired())
        eNotify(new ENotificationImpl(this, Notification.SET,
                ModelPackage.SHAPE__NAME,    oldName, name));
   }
...
```

Ausblick GMF

- Erlaubt einen beliebigen, grafischen Editor «zusammen zu klicken» ohne eigenen Code zu schreiben. Diese Arbeit wird von Generatoren übernommen.
- GMF verwendet EMF um das Daten-Metamodell abzubilden
- GMF greift auf GEF (Graphical Editing Framework) zurück, dieses stellt umfangreiche Klassen für die graphische Darstellung und Werkzeuge für Editoren bereit.

Beispiel GMF



Fazit

- Enge Eclipse-Integration: Generierung von UI-Komponenten, Zusammenarbeit mit anderen Eclipse-Projekten, etwa im Eclipse Graphical Modeling Framework (GMF)
- Man bekommt als Entwickler viel geschenkt, etwa Persistenz und Validierung. Dies steigert die Produktivität beim Entwickeln.
- Wenig Einarbeitungszeit notwendig zum Modellieren aufgrund vieler Möglichkeiten der Modellierung (XML, UML, Java-Annotations)
- Einsatz von Code-Generatoren steigert Produktivität und reduziert Wiederholungen und damit mögliche Fehlerquellen
- Basiert auf offenen Standards und Open-Source-Frameworks
- Nur noch geringfügige Konfigurationen oder manuelle Anpassungen am Quellcode für das Daten-Modell erforderlich
- Erste Erfolge sind schnell erreicht. Komplexe Aufgaben erfordert tiefere Einarbeitung in das Thema → aufwendig
- Diverse Anbindung an EMF vorhanden: GMF, Teneo, …
- Es gelten die allgemeinen Vor-/Nachteile von Frameworks (Einarbeitung, Komfort, Skalierbarkeit, Abstraktion, Standard-Lektüre, …)
- Framework-Wechsel erfordert komplettes Redesign

Quellen

- Web:
  - EMF-Projekt-Homepage: http://www.eclipse.org/modeling/emf/
  - EMF-Tutorial für Eclipse: http://www.vogella.de/articles/EclipseEMF/article.html
  - GMF-Tutorial: www.tm.tfh-wildau.de/vandenhouten/media/GMF-Step-By-Step.pdf
- Bücher:
  - EMF Eclipse Modeling Framework, Addison Wesley, ISBN 10: 0-321-33188-5