Vorlesung (WS 2014/15)

Softwarekonstruktion

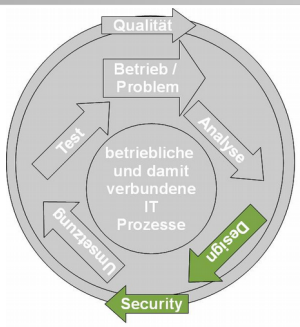
Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

1.5: Eclipse Modeling Framework (EMF)

v. 01.12.2014

- Modellgetriebene SW-Entwicklung
  - Einführung
  - Modellbasierte Softwareentwicklung
  - OCL
  - Ereignisgesteuerte Prozesskette (EPK)
  - Petrinetze
  - Eclipse Modeling Framework (EMF)
- Qualitätsmanagement
- Testen



Inkl Beiträge von Markus Bauer, Florian Lautenbacher, Stephan Roser.

Literatur:
V. Gruhn: **MDA - Effektives Software-Engineering**. (s. Vorlesungswebseite)
- Kapitel 8.2

Literatur:

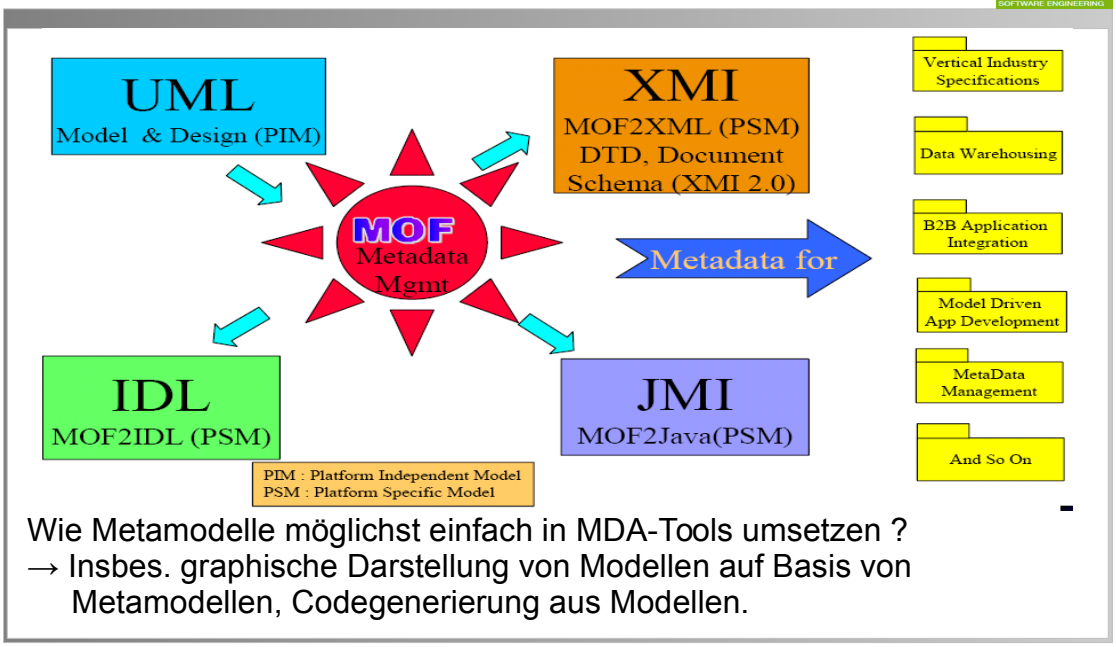
V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129
- Kapitel 8.2

D. Steinberg: **EMF – Eclipse Modeling Framework**
http://www.ub.tu-dortmund.de/katalog/titel/1403033
- Kapitel 2 – Introducing EMF

Einleitung Eclipse Modeling Framework

- **Vorheriger Abschnitt:** Grundlage und Techniken für modellbasierte Softwareentwicklung.
- **Dieser Abschnitt:** Technische Grundlage dafür: Eclipse Modeling Framework
  - Vorstellung der Standards → EMF, GEF, GMF

Motivation

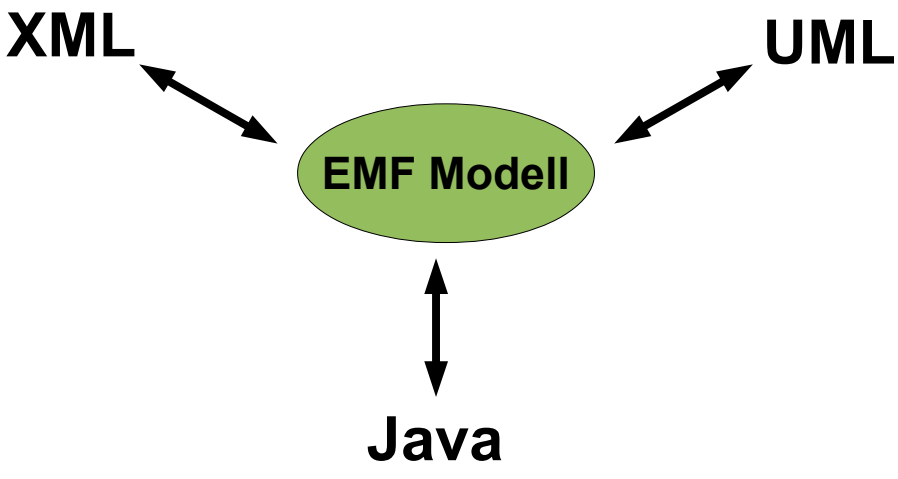


Wie Metamodelle möglichst einfach in MDA-Tools umsetzen ?
→ Insbes. graphische Darstellung von Modellen auf Basis von Metamodellen, Codegenerierung aus Modellen.

D. Steinberg: **EMF – Eclipse Modeling Framework**
http://www.ub.tu-dortmund.de/katalog/titel/1403033

- Abschnitt 2.1 – Unifying Java, XML and UML (S.12-14)

Metamodelle aus Java-Klassen, UML-Diagrammen und XML-Dateien importierbar.



Modelle auf unterschiedlichem Wege erstellbar:

- Aus annotierten Java-Klassen.
- Aus XML-Dokumenten.
- Aus Modellierungstools wie Rational Rose.
- Direkt mithilfe EMF Ecore Baum-Editors.

Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 8.2.4 (S.291-295)
- Abbildung 8.21 (S.293)

D. Steinberg: **EMF – Eclipse Modeling Framework**
http://www.ub.tu-dortmund.de/katalog/titel/1403033

- Abschnitt 2.1 – Unifying Java, XML and UML (S.12-14)
- Abbildung 2.2 (S.14)

EMF – Überblick

EMF.EMOF:

- Teil der MOF 2.0-Spezifikation (Essential MOF).

EMF.Ecore: Core EMF-Framework beinhaltet Meta-Model:

- **Um Modelle zu beschreiben.**
- **Laufzeitunterstützung** für Modelle inkl. Benachrichtigung bei Änderungen,
- **Persistenzunterstützung** durch Standard XML-Serialisierung,
- **API** um EMF-Modelle generisch zu verändern.

EMF.Edit:

- Generische und wiederverwendbare Klassen, um **Editoren** für EMF-Modelle zu erstellen.

EMF.Codegen:

- EMF Code-Generierungsframework: kann den für einen Editor für EMF-Modelle benötigten Code generieren.

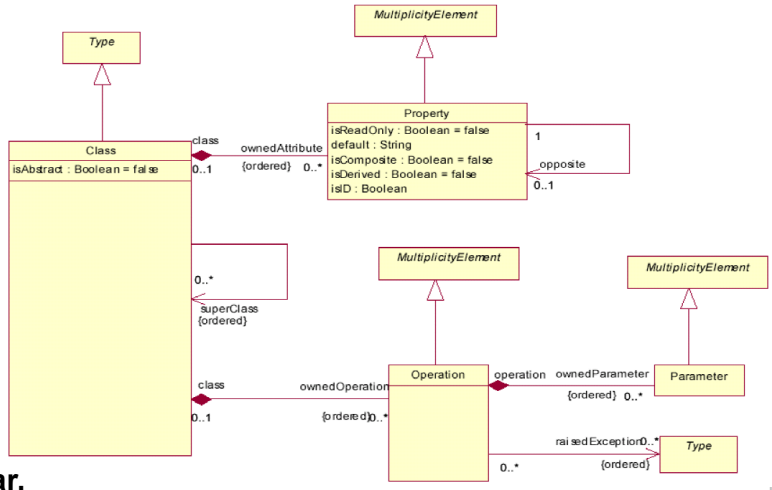
Essential MOF (EMOF)

EMOF:

- Teil von MOF 2.0
- Zur **Definition von einfachen Metamodellen**.
- Nutzt OO-Konzepte.

MOF 2.0 verwendet UML 2.0-Klassen-Diagramme.

- **Metamodell mit UML-Tools erstellbar.**



- **MOF 2.0 definiert Complete MOF** (CMOF) mit zusätzlichen Eigenschaften.

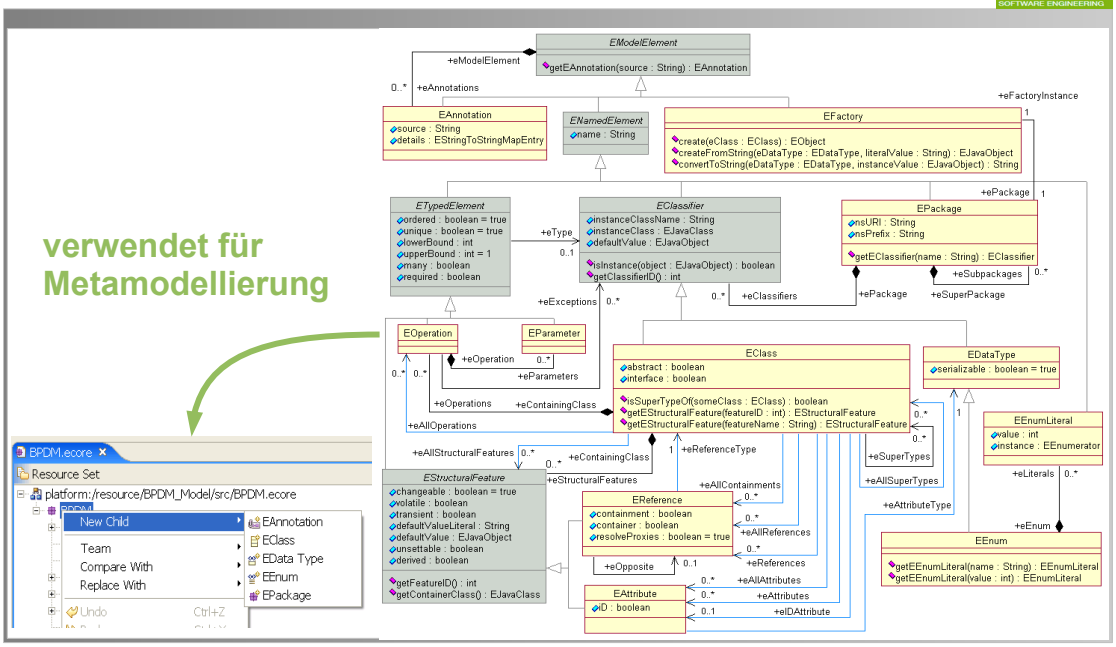
Beispiel: vereinfachtes Metamodell für Klassendiagramme (vgl. Teil 1.2 Folie 29 !)

Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 3.3.2 – MOF 2 (S.87-92)
- EMOF (S.89-90)
- CMOF (S.90-91)

Ecore – Das Kern-Metamodell für EMF



Literatur:

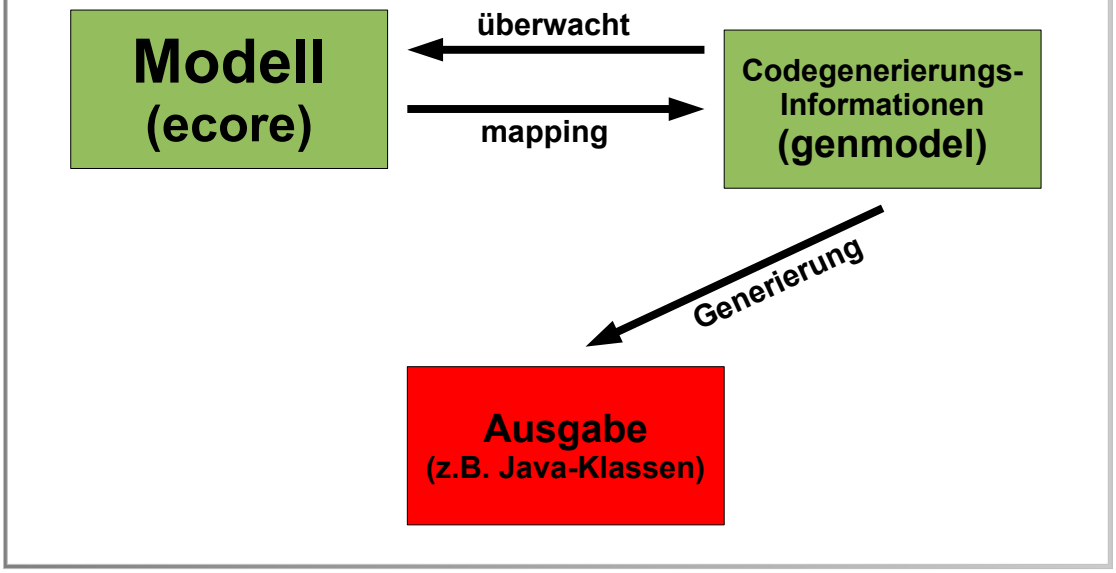
V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 8.2.4 (S.291-295)
- Abbildung 8.22 – Ecore Kernel (S.293)

D. Steinberg: **EMF – Eclipse Modeling Framework**
http://www.ub.tu-dortmund.de/katalog/titel/1403033

- Abschnitt 2.3.1 – The Ecore (Meta) Model (S.17-19)
- Abbildung 2.3 (S.17)

Abhängigkeit Modell- und Generierungsdateien



Literatur:

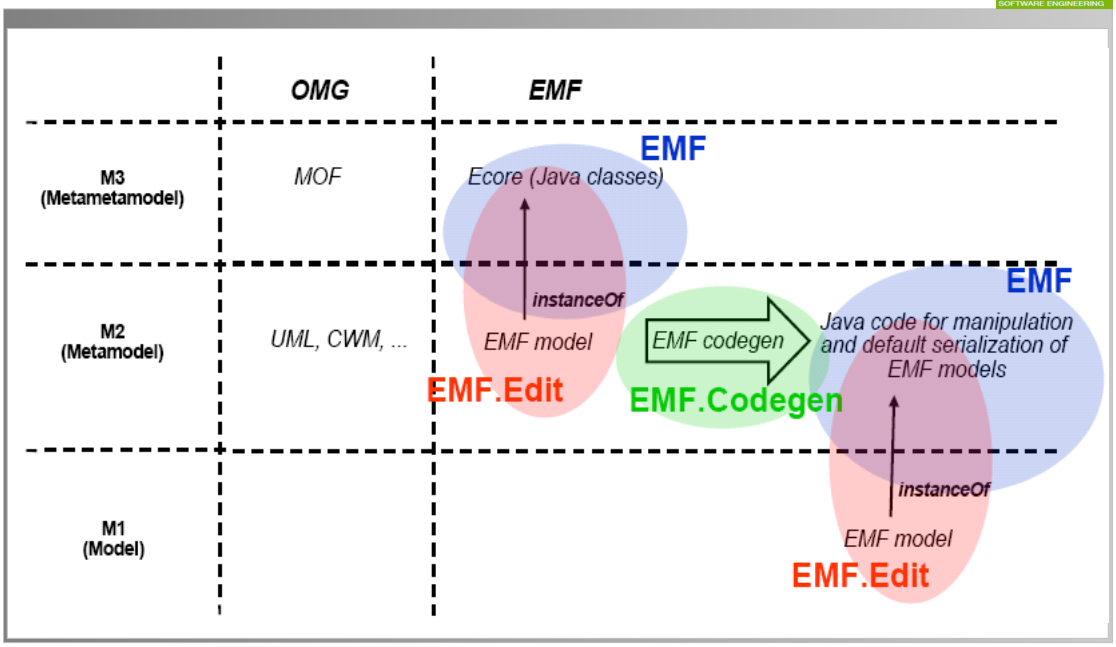
V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 8.2.4 (S.291-295)
- Abbildung 8.23 – Beispiel (S.294)

D. Steinberg: **EMF – Eclipse Modeling Framework**
http://www.ub.tu-dortmund.de/katalog/titel/1403033

- Abschnitt 2.4 – Generating Codes (S.23-29)

EMF – Überblick über Edit und Codegen



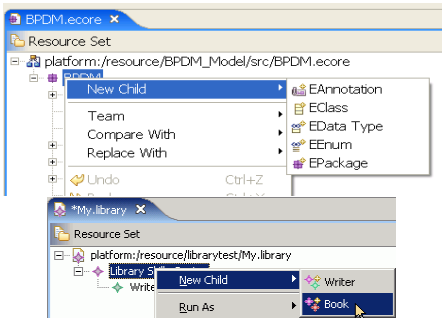
D. Steinberg: **EMF – Eclipse Modeling Framework**
http://www.ub.tu-dortmund.de/katalog/titel/1403033

- Abschnitt 2.4 – Generating Codes (S.23-29)
- Abschnitt 3.1 – Editing EMF Models (S.42-46)
- Abschnitt 3.1.2 – EMF.Edit Support (S.45-46)

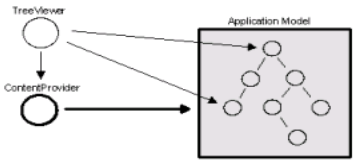
EMF.Edit – EMF.Codegen

EMF.Edit

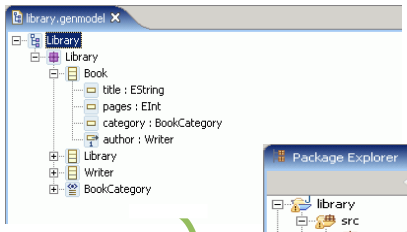
Modellierungseditor



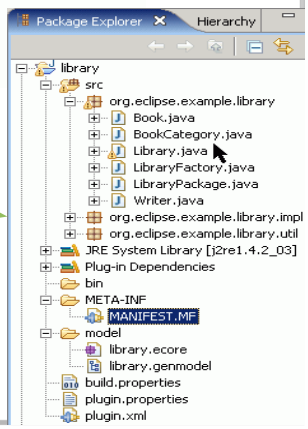
Content Provider, etc.



EMF.Codegen



generate



D. Steinberg: **EMF – Eclipse Modeling Framework**
http://www.ub.tu-dortmund.de/katalog/titel/1403033

- Abschnitt 2.4 – Generating Codes (S.23-29)
- Abschnitt 3.1 – Editing EMF Models (S.42-46)
- Abschnitt 3.1.2 – EMF.Edit Support (S.45-46)

EMF Zusammenfassung

UML:

- EMF Ecore beschäftigt sich mit **Klassenmodellierungsaspekten** der UML.
- UML 2.0 Metamodell: In EMF Ecore implementiert.

MOF:

- Meta-Object Facility definiert konkrete Untermenge von UML.
  → Beschreibung der **Modellierungskonzepte** innerhalb Repository.
- Vergleichbar mit Ecore.
- Ecore vermeidet einige komplexe Elemente von MOF.
  → Fokus auf Tool-Integration als Management von Metadaten-Repositories.

XMI:

- Zur **Serialisierung von Modellen**.
- Verwendung von EMF-Modell und Ecore selbst.

MDA:

- EMF unterstützt Hauptkonzept der MDA.
  → **Modelle** für **Entwicklung / Generierung** (nicht nur Dokumentation).

D. Steinberg: **EMF – Eclipse Modeling Framework**
http://www.ub.tu-dortmund.de/katalog/titel/1403033

- Abschnitt 2.1 bis 3.1.2 (S.11-46)

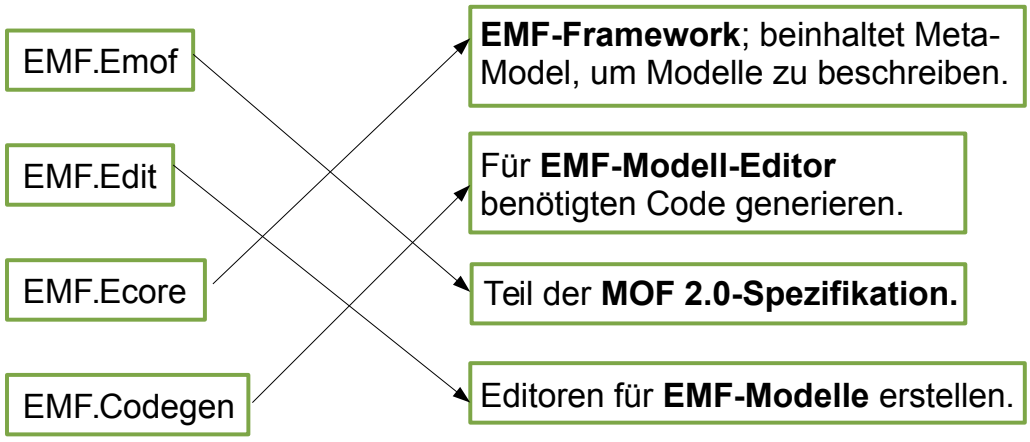
Diskussionsfrage: EMF-Bestandteile

Welche **Aussagen** passen zu den angegebenen **Begriffen** ?

EMF.Emof	**EMF-Framework**; beinhaltet Meta-Model, um Modelle zu beschreiben.
EMF.Edit	Für **EMF-Modell-Editor** benötigten Code generieren.
EMF.Ecore	Teil der **MOF 2.0-Spezifikation.**
EMF.Codegen	Editoren für **EMF-Modelle** erstellen.

Diskussionsfrage: EMF-Bestandteile

Welche **Aussagen** passen zu den angegebenen **Begriffen** ?



Was ist GEF?

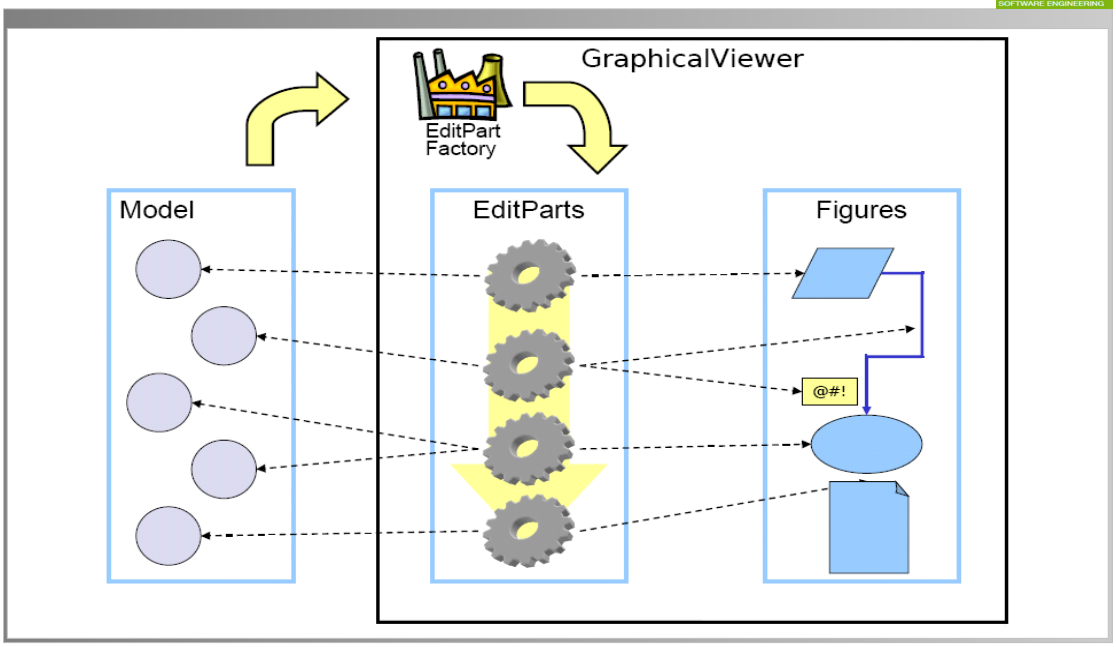
- Framework: **Modelle graphisch darstellen.**
- **Interaktion** mit Modell:
  - Verarbeitung von Benutzereingaben durch Maus und Tastatur.
  - Interpretation der Eingaben.
  - Möglichkeiten Modell zu verändern.
  - Änderungen rückgängig machbar (undo/redo).
- **Workbench Funktionen:**
  - Aktionen und Menüs.
  - Toolbars.
  - Keybindings.
- **Plugin** von Eclipse.
- Baut auf **Model-View-Controller Pattern** auf.
- **Ziel:** Wiederverwendete Funktionalitäten nicht jedesmal neu entwickeln.

Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**

http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 8.2.3 – GEF (S.289-291)

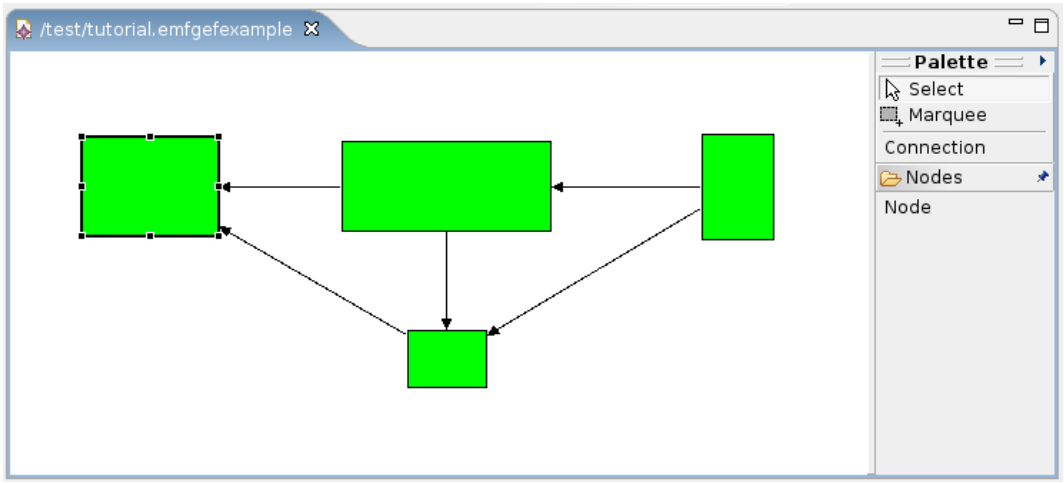


Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 8.2.3 – GEF (S.289-291)
- Abbildung 8.20 – MVC in GEF (S.290)

Beispiel - Editor



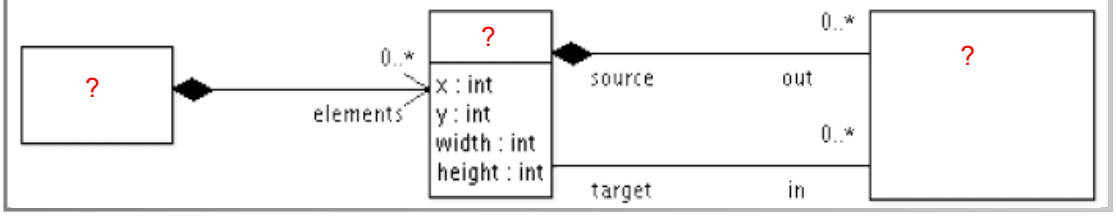
Beispiel - Features

- **Erstellen eines neuen Diagramms** mittels Wizard.
- Öffnen eines existierenden Diagramms.
- **Speichern von Änderungen.**
  - auch als neues Dokument („speichern als“).
- **Palette mit Selektionstools** und Elementen.
- **Erstellen von Knoten** (*node*).
- Erstellen von Verbindungen (*connection*) zwischen Knoten.
- **Löschen von Knoten** und Verbindungen.
- Verschieben von Knoten.
- Ändern der Größe von Knoten.
- Alle **Veränderungen rückgängig machbar** (*undo*) und
- wiederherstellbar (*redo*).

Diskussion: Metamodell für Beispiel-Editor ?

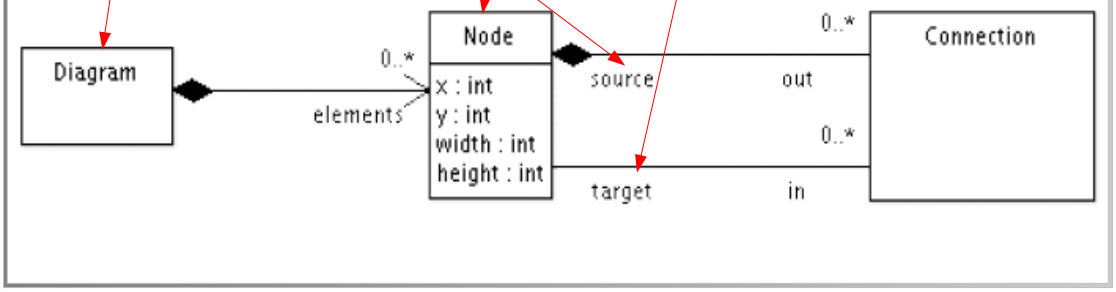
Was gehört an die fehlenden Stellen im unten abgebildeten Metamodell ?

- Erstellen eines neuen **Diagramms** mittels Wizard.
- Öffnen eines existierenden Diagramms.
- Speichern von Änderungen.
  - auch als neues Dokument („speichern als“).
- Palette mit Selektionstools und Elementen.
- Erstellen von **Knoten** (***node***).
- Erstellen von **Verbindungen** (***connection***) zwischen Knoten.
- Löschen von **Knoten** und Verbindungen.
- Verschieben von Knoten.
- Ändern der Größe von Knoten.
- Alle Veränderungen rückgängig machbar (*undo*) und
- wiederherstellbar (*redo*).



Diskussion: Metamodell für Beispiel-Editor ?

- **Diagram:** Wurzelelement.
- Diagramm enthält Knoten (*node*).
- Knoten besitzen **Quell- und Zielverbindungen** (*source Connection / target Connection*).



Vor- und Nachteile für die Verwendung von EMF mit GEF

Vorteile:

- **Kostengünstige Möglichkeit** für modellbasierte Softwareentwicklung.
- **Effektivität** durch automatische Konsistenzerhaltung der Modellrepräsentanten.
- **Mächtige Codegenerierung** erspart viel stupiden Programmieraufwand.

Nachteile:

- Modellierungssprachschatz nicht mächtig wie UML (**Essential** MOF).
  - Aber meist ausreichend.

EMF / GEF / GMF im Überblick

Eclipse Modeling Framework (EMF):

- Spezifische Realisierung der OMG MOF-Konzepte mit Eclipse und Java.
- Integriert im Eclipse Tools Projekt.

Graphical Editing Framework (GEF):

- Framework zur Darstellung von Modellen.
- Geschieht auf Basis eines EMF-Metamodells oder eigenständig.

Graphical Modeling Framework (GMF):

- Versuch, EMF und GEF zu integrieren.

Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 8.2.1 – Das Eclipse Tools Projekt (S.282-283)

In diesem Abschnitt: Eclipse Modeling Framework (EMF)

- Technische Grundlagen für UML-Werkzeuge und MDA.

Damit Ende des Kapitel 1: Modellbasierte Entwicklung.

Als nächstes: **Softwarequalitätsmanagement** und insbesondere **Softwareverifikation**.

Insbesondere unter Verwendung von Techniken aus Kap. 1 (**Testautomatisierung** durch **Modellbasiertes Testen mit UML**, Einhaltung von **Constraints mittels OCL**).

Anhang
(weitere Informationen zu Nachbereitung)

EMF Allgemeines

Modellierungsframework und Tool zur **Code-Generierung** basierend auf strukturiertem Datenmodell.

Ausgehend von Modellspezifikation in XMI bietet EMF:

- **Tools und Laufzeitunterstützung.**
  → Javaklassen aus Modell erstellen.
- **Adapterklassen:** Einfache Sicht und kommandobasiertes Editieren des Modells.
- Grundlegender **Editor.**

Grundlage für **Interoperabilität** zwischen EMF-basierten Anwendungen.

Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 8.2.4 (S.291-295)

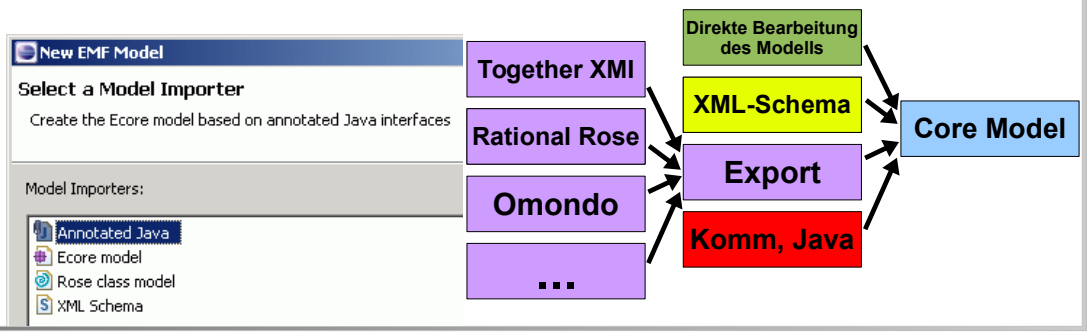
D. Steinberg: **EMF – Eclipse Modeling Framework**
http://www.ub.tu-dortmund.de/katalog/titel/1403033

- Abschnitt 2.1 – Unifying Java, XML and UML (S.12-14)

EMF - Modellimport

EMF **(Meta-)Modelle** wie folgt erstellbar:

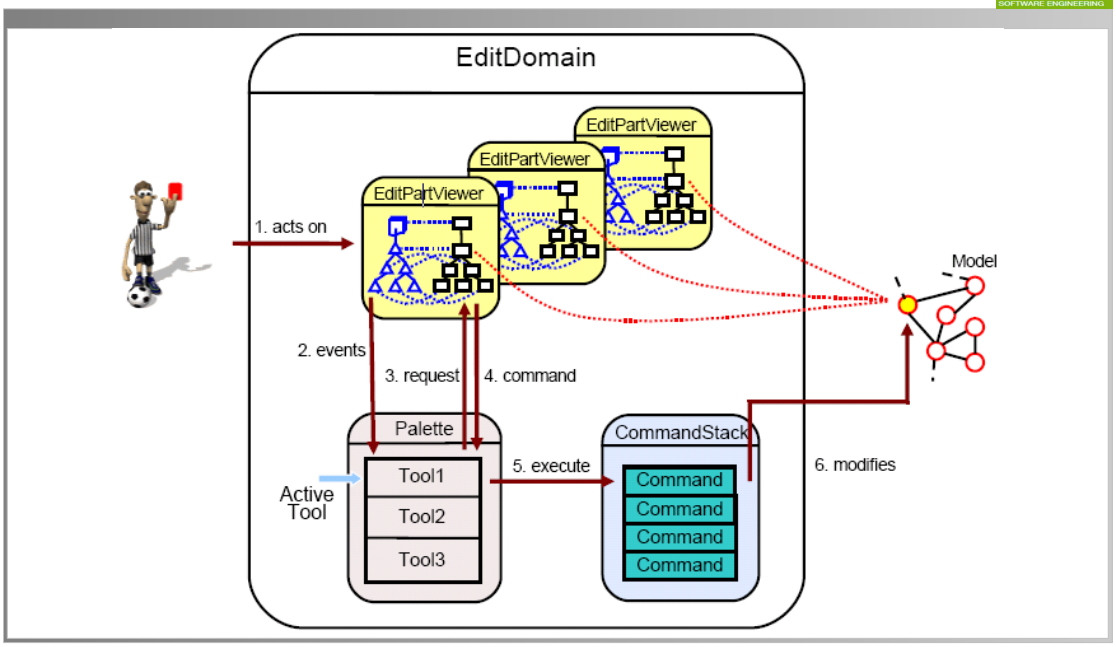
- XMI-Datei **direkt im Texteditor** erstellen (→ Ecore model).
- Verwendung eines **Modellierungstools** wie bspw. Rational Rose und Export als XMI-Dokument (→ Rose class model).
- Annotierte Java-Klassen und **Interfaces einlesen** (→Annotated Java).
- **XML-Schema verwenden:** Modell-Serialisierung beschreiben (→ XML Schema)



D. Steinberg: **EMF – Eclipse Modeling Framework**
http://www.ub.tu-dortmund.de/katalog/titel/1403033

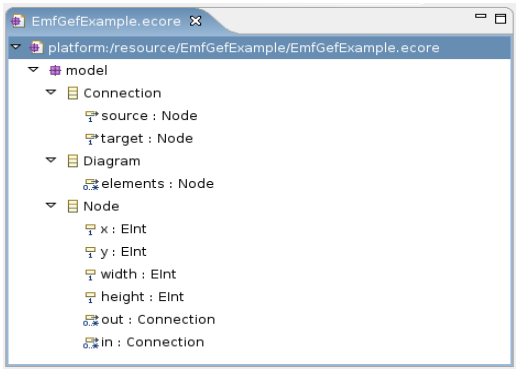
- Abschnitt 2.3.2 – The Ecore (Meta) Model (S.19-20)
- Abschnitt 2.3.3 – XMI Serialization (S.20-21)
- Abschnitt 2.3.4 – Java Annotations (S.21-22)
- Abschnitt 2.3.5 – The Ecore „Big Picture“ (S.22)
- Abbildung 2.5 – Ecore Model (S.22)

The Big Picture



Metamodell UML → Ecore

- **Erstellt mit ArgoUML** (http://argouml.tigris.org).
- **Export** von ArgoUML **als XMI.**
- **Transformation von ArgoUML** XMI nach Ecore XMI mithilfe des Tools *argo2ecore* (http://argo2ecore.sourceforge.net)



EMF Genmodel

Zusätzliche Informationen um Java Klassen zu erstellen.

Allgemeine Informationen:

- Copyright.
- Name des Modells.
- ID des Plugins.

Einstellungen für EMF.Edit:

- Unterstützung zur **Erstellung von Kindelementen** durch Commands.
- Icons.
- Plug-in Klassen.

Einstellungen für EMF Editor.

Template & Merge:

- Automatische Formatierung des Codes.
- **Dynamische Templates:** Java Klassen mithilfe von JET erzeugen.
  → Bei Bedarf anpassbar.

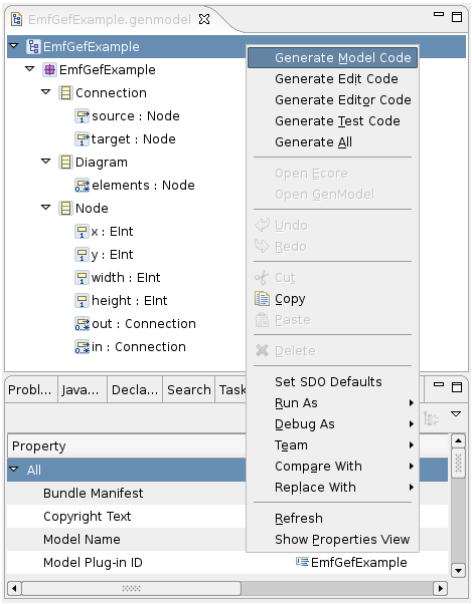
Einstellungen zur Property View.

D. Steinberg: **EMF – Eclipse Modeling Framework**
http://www.ub.tu-dortmund.de/katalog/titel/1403033

- Abschnitt 2.4 – Generating Codes (S.23-29)
- Abschnitt 3.1 – Editing EMF Models (S.42-46)
- Abschnitt 3.1.2 – EMF.Edit Support (S.45-46)

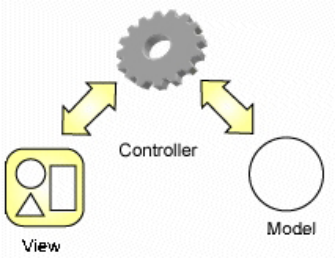
Erstellen des Modells

- **Erstellen des GenModels** aus Ecore Modell.
- **Erstellen des Modells** aus GenModel.
- Wenn man Editor generiert, dann hat man an dieser Stelle einen **Baumeditor**, mit dem man Modell bearbeiten kann.



Model – View – Controller Pattern

- **3 Schichten Modell.**
- Strikte **Trennung der Schichten.**
- Daten in **Modellschicht.**
- Visualisierung der Daten in **Viewschicht.**
- Kommunikation zwischen 2 Schichten in **Controllerschicht.**



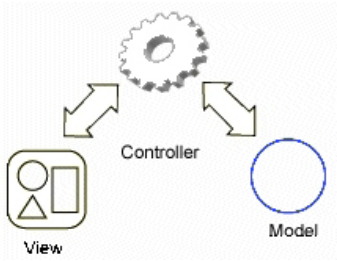
Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 8.2.3 – GEF (S.289-291)
- Abbildung 8.20 – MVC (S.290)

MVC – Pattern: Modellschicht

- Alle **persistente und wichtige Daten** ausschließlich hier gespeichert.
- **Container für Daten.**
- Kennt keine anderen Teile des Programms.
- Teilt **Änderungen** an sich mit über Listener.



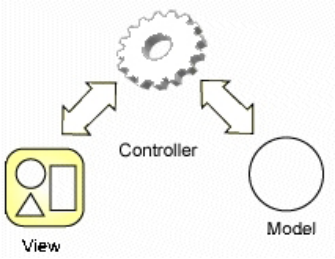
Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 8.2.3 – GEF (S.289-291)
- Abbildung 8.20 – MVC (S.290)

MVC – Pattern: Viewschicht

- **Keine Daten** in Viewschicht.
- **Keine Modelllogik.**
- Kennt keine anderen Teile des Programms.
- **Abbildung der Daten** der Modellschicht.



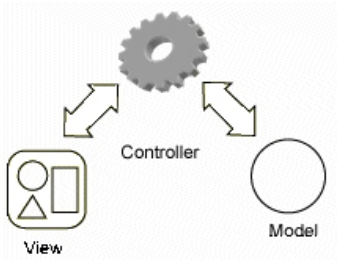
Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 8.2.3 – GEF (S.289-291)
- Abbildung 8.20 – MVC (S.290)

MVC – Pattern: Controllerschicht

- **Verbindung** von Modell- und Viewschicht.
- Leitet **Kommunikation** vom Modell an View weiter.
- **In GEF:** Unterklasse von EditPart.
- Zu jedem EditPart genau **ein Modell und genau eine View.**



Literatur:

V. Gruhn: **MDA - Effektives Software-Engineering**
http://www.ub.tu-dortmund.de/katalog/titel/1223129

- Abschnitt 8.2.3 – GEF (S.289-291)
- Abbildung 8.20 – MVC (S.290)

Diskussionsfrage: Verwendung von MVC-Pattern

Welche konkreten Vor- und Nachteile bietet die Verwendung von MVC-Pattern **im Kontext von GEF ?**

Antwort:

Vorteile:

- Durch Change-update-Mechanismus ist das Model in allen Views immer aktuell visualisiert.

Nachteil:

- Für dasselbe Model sind mehrere View-Controller-Paare vorzusehen.
- Falls sich die Daten sehr oft und schnell ändern, kann es sein, dass das View die Veränderungen nicht schnell genug anzeigen kann.

Weiterführende Literatur

- **GEF Beispiele (im Plug-in enthalten):**
  - Shapes (Einfachstes Beispiel).
  - Logic (Sehr umfangreiches Beispiel).
- **GEF Dokumentation:** http://www.eclipse.org/gef/reference/articles.html
- **GefDescription:** http://eclipsewiki.editme.com/GefDescription
- **EMF Dokumentation:** http://www.eclipse.org/emf/docs.php
- **EMF Übersicht:** http://www.eclipse.org/emf/docs.php?doc=references/overview/EMF.html
- **EMF.Edit Übersicht:** http://www.eclipse.org/emf/docs.php?doc=references/overview/EMF.Edit.html
- **EMF Book: Eclipse Modeling Framework (Overview and Developer's Guide):** http://www.awprofessional.com/content/images/0131425420/samplechapter/budinskych02.pdf
- **Create an Eclipse-based application using the GEF:** http://www-128.ibm.com/developerworks/opensource/library/os-gef
- **Using GEF with EMF:** http://www.eclipse.org/articles/Article-GEF-EMF/gef-emf.html
- **IBM Redbook: Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework:** http://www.redbooks.ibm.com/abstracts/sg246302.html

Quellen entnommen aus

- Hudson, Randy; Shah, Pratik: *Tutorial #23 / GEF In Depth*; http://www.eclipse.org/gef/reference/GEF%20Tutorial%202005.ppt
- EclipseCon 2005 und 2006: Vorträge zu EMF und GEF
- **Beispiele und Tutorials** von Eclipse EMF und GEF

- Motivation und Überblick
- Eclipse Modeling Framework (EMF)
  - EMF-Modellimport
  - EMOF und Ecore
  - EMF Edit & Codegen
- Graphical Editing Framework (GEF)
  - Model-View-Controller (MVC)-Pattern
  - MVC in GEF
  - Weitere Konstrukte: EditPolicies und Commands
- Nutzung von EMF in GEF
  - Einführung eines Beispiels
  - Beispiel in EMF und GEF
- Zusammenfassung und Ausblick

Erzeugte Java Klassen - Modell

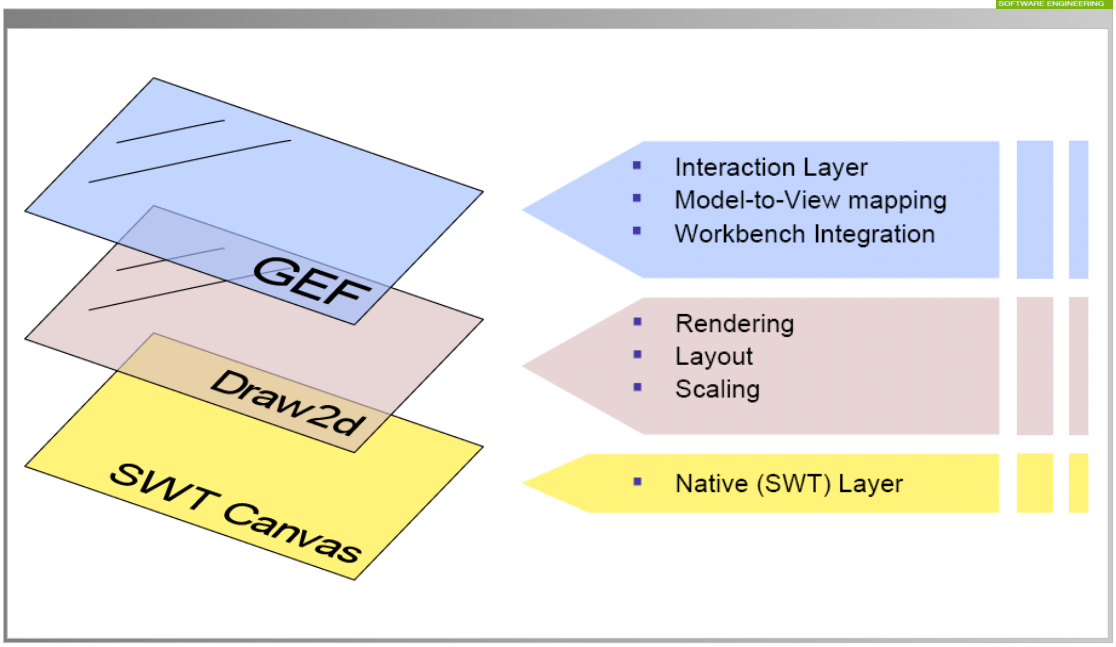
- **Zu jeder Klasse** im Ecore Metamodell:
  - Java Interface.
  - Implementierung im Unterpackage impl.
- **Zu jedem Package:**
  - Eine Package Klasse.
  - Informationen zu Features und Metadaten des Modell.
  - **Factory Klasse:** Bietet Methoden zum Erzeugen neuer Objekte.

Erzeugte Java Klassen - Edit

- **Edit Provider für jede Klasse** im Ecore Metamodell:
  - Informationen zu Kindern und Eltern vom Objekt.
  - Descriptoren zur Erzeugung von Kindern.
  - Commands zur Änderung des Objekts.
  - Informationen zur Erzeugung eines Baumes, der das Modell repräsentiert.
  - Text und Icon zum Objekt.
  - Informationen für Property Sheet.
- **Adapter Factory:**
  - Liefert richtigen Provider zum Objekt.

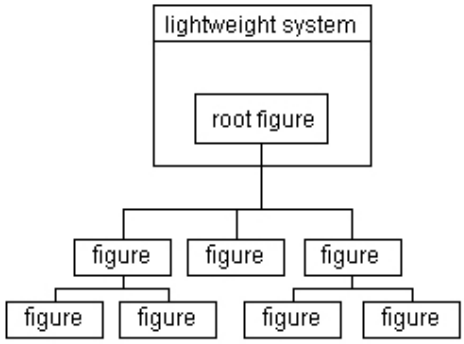
- Motivation und Überblick
- Eclipse Modeling Framework (EMF)
  - EMF-Modellimport
  - EMOF und Ecore
  - EMF Edit & Codegen
- Graphical Editing Framework (GEF)
  - Model-View-Controller (MVC)-Pattern
  - MVC in GEF
  - Weitere Konstrukte: EditPolicies und Commands
- Nutzung von EMF in GEF
  - Einführung eines Beispiels
  - Beispiel in EMF und GEF
- Zusammenfassung und Ausblick

Was ist GEF?


GEF
Draw2d
SWT Canvas
Interaction Layer
Model-to-View mapping
Workbench Integration
Rendering
Layout
Scaling
Native (SWT) Layer

Die Viewschicht in GEF: Draw2D und Figures

- **Darstellung der Modellobjekte** in GEF mit Figures.
- **Figures in Baumstruktur.**
- Anzeige der Figures im lightweight system von Draw2D.
- **Figures zeichnen sich selbst** und rekursiv ihre Kinder.



Die Controllerschicht in GEF: EditParts und EditPartFactory

- EditParts wie Figures in **Baumstruktur.**
- **Drei wichtige Methoden** in EditParts:
  - `createFigure()`**: Erstellen der Figure** zu dieser EditPart.
    → Verbindung Controllerschicht ↔ Viewschicht
  - `refreshVisuals()`**: Aktualisieren der Daten** der Viewschicht mit Daten der Modellschicht.
  - `getModelChildren()`**: Liste von Modellklassen:** Logisch Kinder vom zum EditPart korrespondierenden Modellelement.
- **Verbindung Modellschicht ↔ Controller** über EditPartFactory:
  - Neues Modellobjekt erzeugen.
  - In Factory dazu korrespondierenden EditPart suchen.
  - Verbindung knüpfen.

Kommunikation zwischen den Schichten

- **Ausgangspunkt:** Änderung findet im Modell statt.
- In EMF sendet Objekt bei Änderung **Notification** an alle registrierten Adapter.
- **Adapter:** EditParts.
- **EditParts:** Bei ihren Modellklassen registrieren.
- Dafür zwei **Methoden:**
  - `activate()`: Nach Erzeugung von EditPart, Registrierung beim entsprechenden Modellelement.
  - `deactivate()`: Wenn EditPart aus EditPartBaum ausgehängt wird, entfernt es Adapter aus dem Modellelement.
- Notifications an Methode `notifyChanged(Notification notification)` senden.

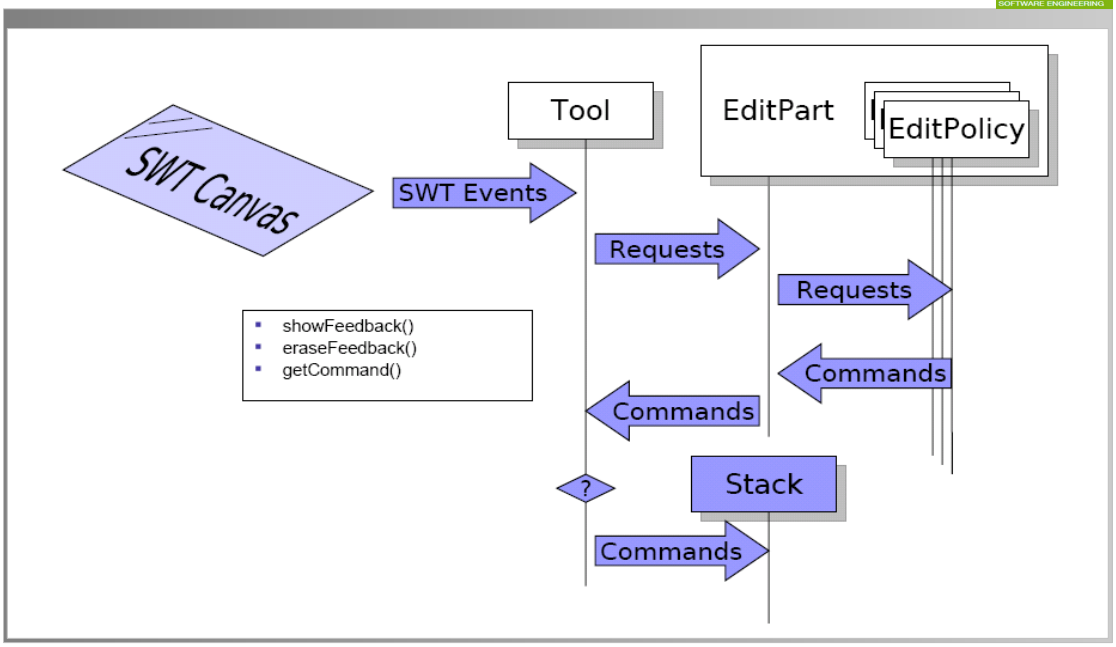
1.3 Eclipse Modeling Framework (EMF) Agenda

- Motivation und Überblick
- Eclipse Modeling Framework (EMF)
  - EMF-Modellimport
  - EMOF und Ecore
  - EMF Edit & Codegen
- Graphical Editing Framework (GEF)
  - Model-View-Controller (MVC)-Pattern
  - MVC in GEF
  - Weitere Konstrukte: EditPolicies und Commands
- Nutzung von EMF in GEF
  - Einführung eines Beispiels
  - Beispiel in EMF und GEF
- Zusammenfassung und Ausblick

Veränderungen im Modell während der Laufzeit: EditPolicies und Commands

- **Festlegen der Aufgaben** eines EditParts über EditPolicies.
- **EditPolicies** bekommen Requests.
- **Requests:** Anfragen des Systems, um Aufgabe auszuführen.
- Mit Informationen eines **Requests Command** erzeugen.

Von Requests zu Commands



Ausführung von Commands

- Im Command **Änderungen im Modell** vornehmen.
  - `execute()`: Bei 1. Ausführung des Commands.
  - `undo()`: Rückgängig machen der Aktionen von execute.
  - `redo()`: Wiederholen der Aktionen nach undo.
- Commands **intern im CommandStack** halten.

EditParts und EditPolicies

- **EditPolicies** in Methode createEditPolicies() von EditPart erzeugen.
- **EditPart:** Verantwortlich View aktuell zu halten.
- EditPolicies behandelt **durch Editieren entstandene Aufgaben:**
  - **Verhindern Einschränkung** durch Einfachvererbung.
  - Übernehmen Aufgaben, die nicht zu EditParts gehören.
  - Erlauben **Bearbeitung dynamisch** zu halten.
  - Werden mithilfe von Roles verwaltet.
  - Behandeln Feedback, Commands, Targeting, etc.
  - **Tipp:** UnexecutableCommand vs. null
- Verwendetes Pattern: **“Pool of Responsibility”**.

GEF Fazit

- **Durchdachte und robuste Struktur.**
- **Viele Funktionalitäten** wie CommandStack bereits implementiert.
- View durch andere **austauschbar.**

Diskussion

- Was wird benutzt um zu **spezifizieren**, welche **commands** auf welche **grafische Elemente** ausführbar sind?
- **Antwort:**
  - **EditParts** benutzen eine Kollektion von **EditPolicy** Instanzen.