Vorlesung (WS 2013/14)

Softwarekonstruktion

Prof. Dr. Jan Jürjens

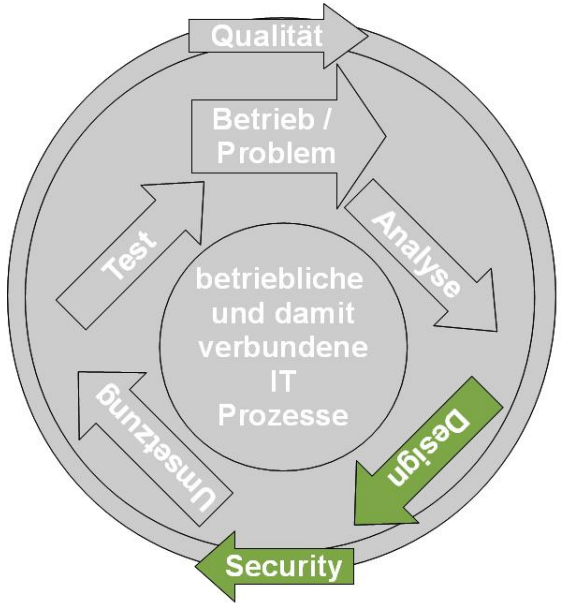
TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

1.3: Eclipse Modeling Framework (EMF)

v. 22.11.2013

Einordnung
1.3 Eclipse Modeling Framework (EMF)

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



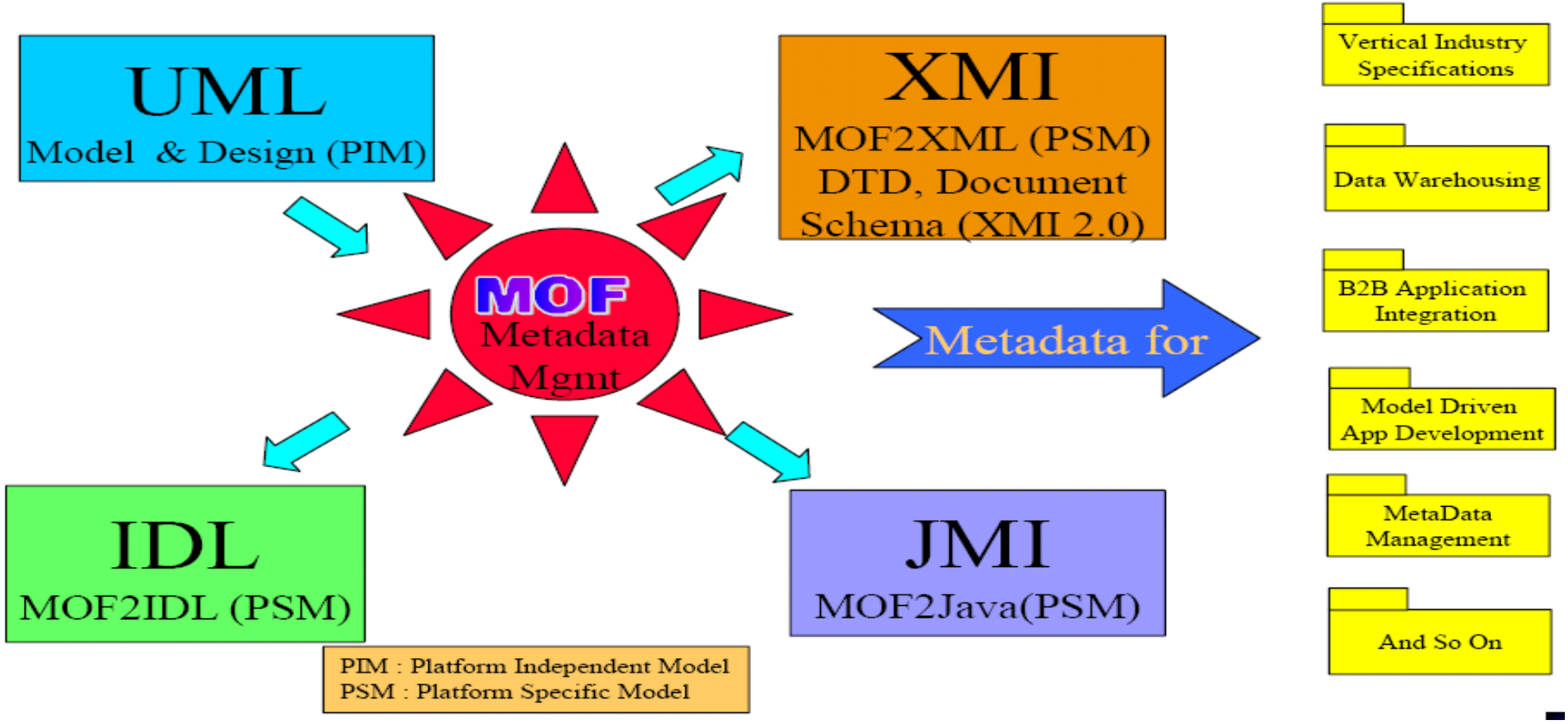
Inkl Beiträge von Markus Bauer, Florian Lautenbacher, Stephan Roser.

Literatur:

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

- Kapitel 8.2

Motivation



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

Standards im Überblick

- **OMG Standards:**
  - Model-Driven Architecture (MDA) zur modellgetriebenen Software-Entwicklung.
  - UML und andere OMG-Modellierungsnotationen (z.B. Business Process Model and Notation (BPMN))
- **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.

1.3 Eclipse Modeling Framework (EMF) Agenda

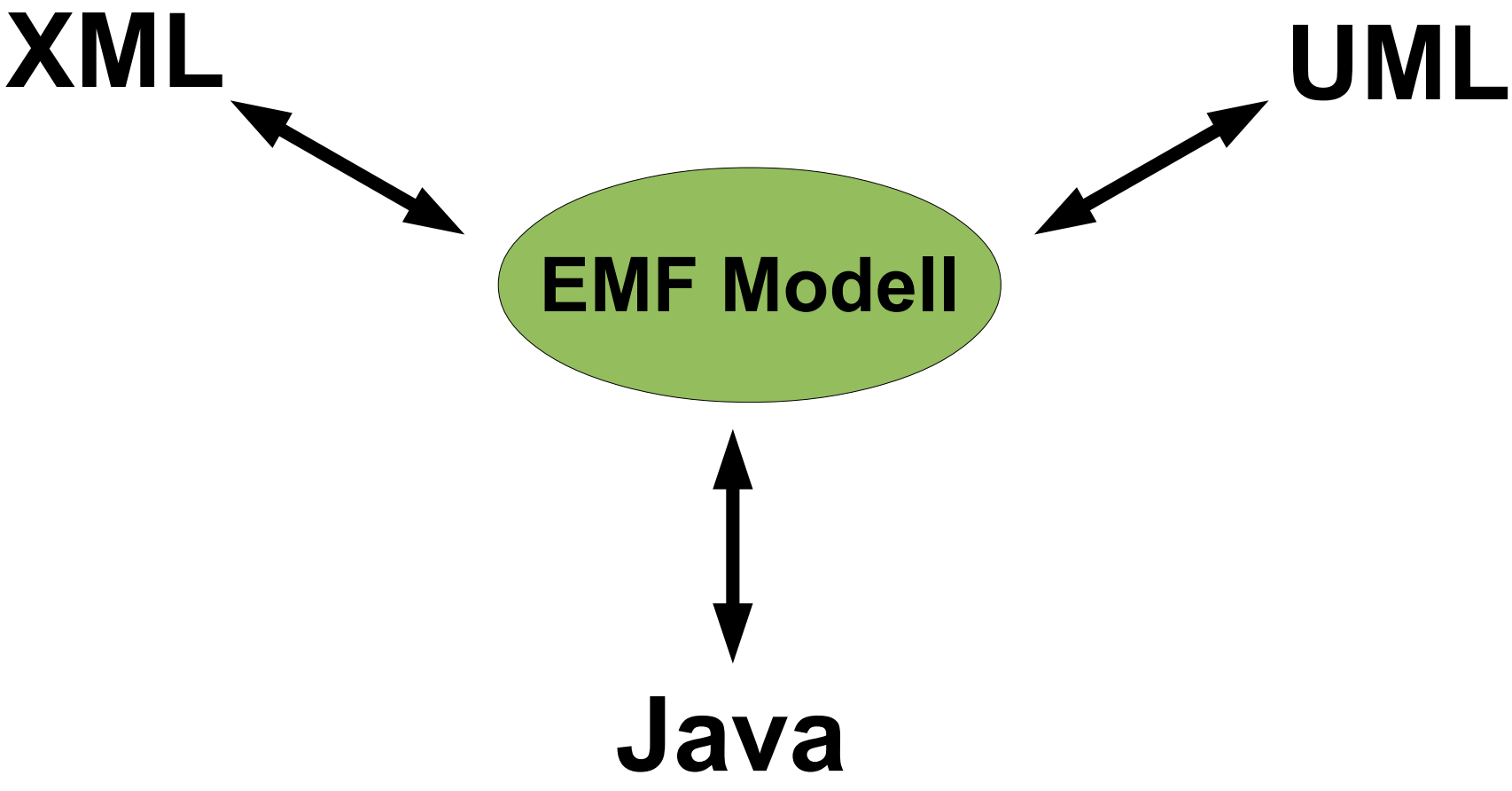
- Motivation und Überblick
- Eclipse Modeling Framework (EMF)
  - EMF-Modellimport
  - EMOF und Ecore
  - EMF Edit & Codegen
- Graphical Editing Framework (GEF)
  - Model-View-Controller (MVC)-Pattern
  - MVC in GEF
- Nutzung von EMF in GEF
  - Einführung eines Beispiels
  - Beispiel in EMF und GEF
-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

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.**
- Modelle auf **unterschiedlichem Wege** erstellbar:
  - Aus annotierten Java-Klassen.
  - Aus XML-Dokumenten.
  - Aus Modellierungstools wie Rational Rose.
  - Direkt mithilfe EMF Ecore Baum-Editors.
- Grundlage für **Interoperabilität** zwischen EMF-basierten Anwendungen.

EMF - Modellimport

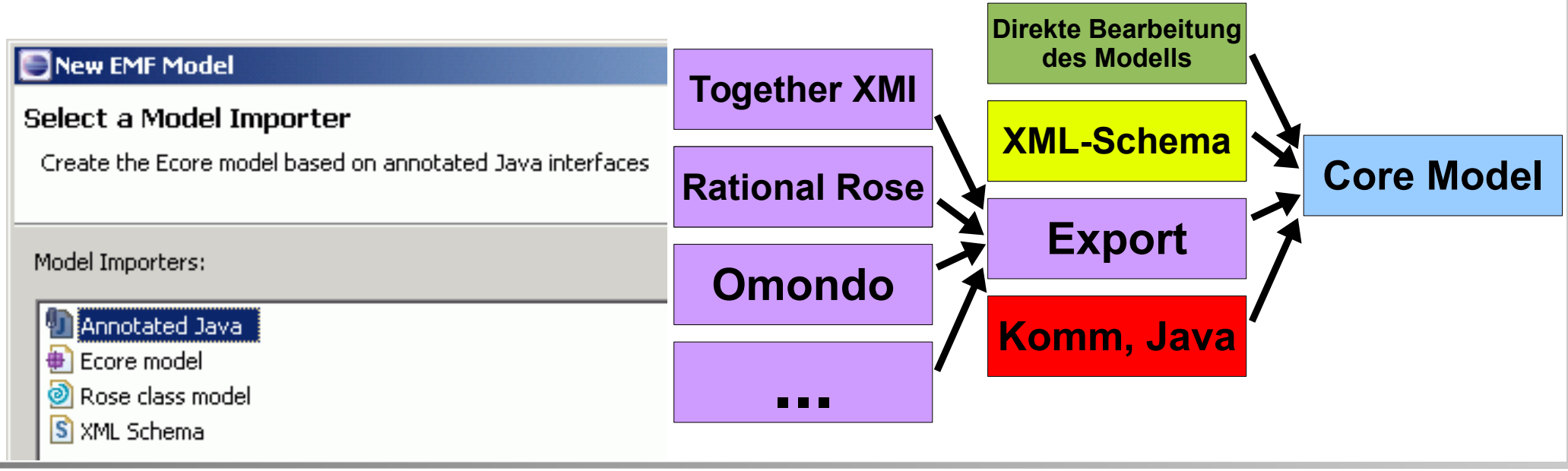
Metamodelle aus Java-Klassen, UML-Diagrammen und XML-Dateien importierbar.



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)



EMF – Überblick

- **EMF.EMOF:**
  - Teil der MOF 2.0-Spezifikation (Essential MOF).
- **EMF.Ecore:**

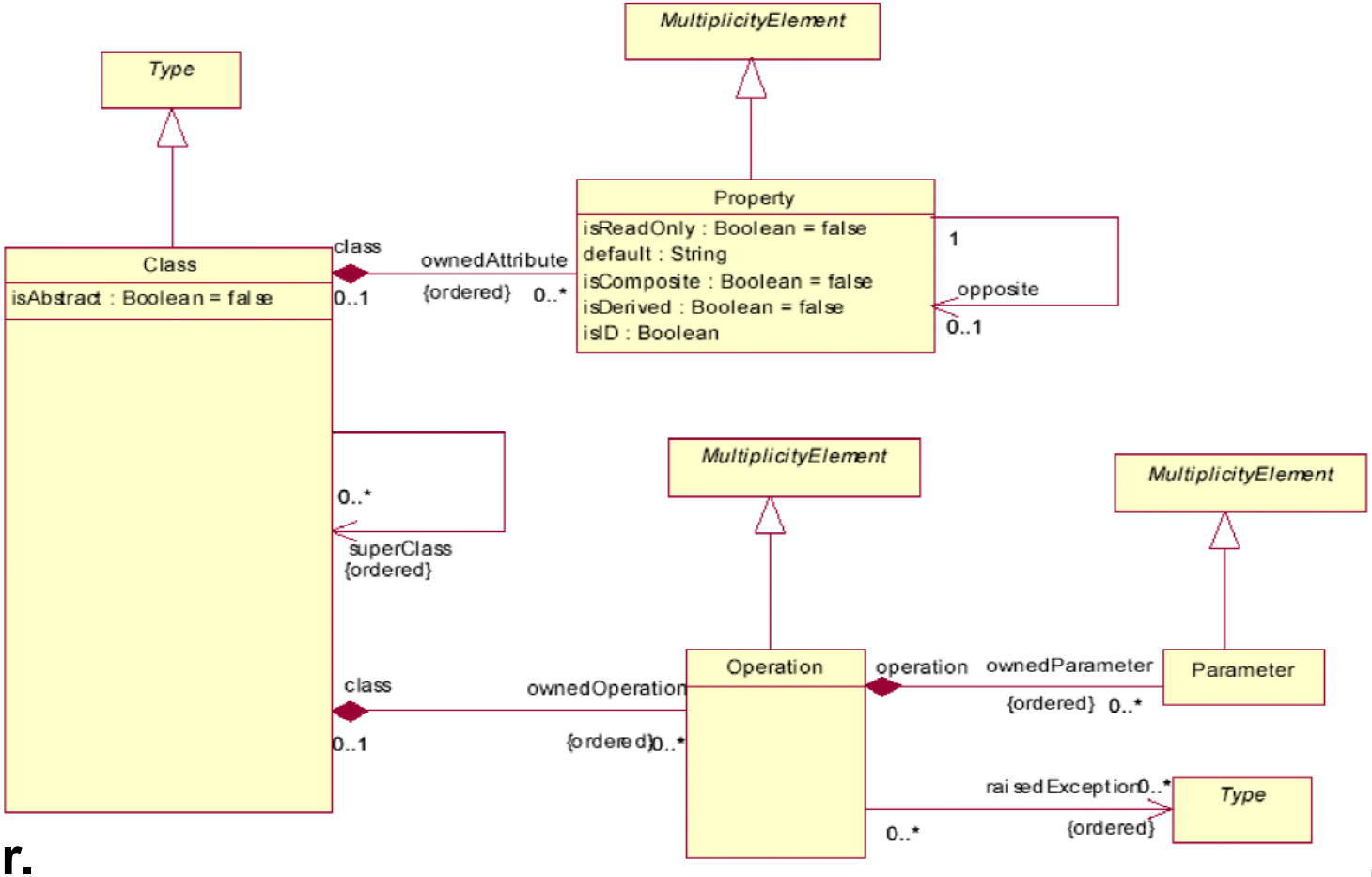
  Core EMF-Framework beinhaltet Meta-Model:
  - **Um Modelle zu beschreiben.**
  - **Laufzeitunterstützung** für Modelle inkl. Benachrichtigung bei Änderungen,
  - **Persistenzunterstützung** durch Standard XML-Serialisierung,
  - **API** um EMF-Modelle generisch zu verändern.
- **EMF.Edit:**
  - Generische 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.



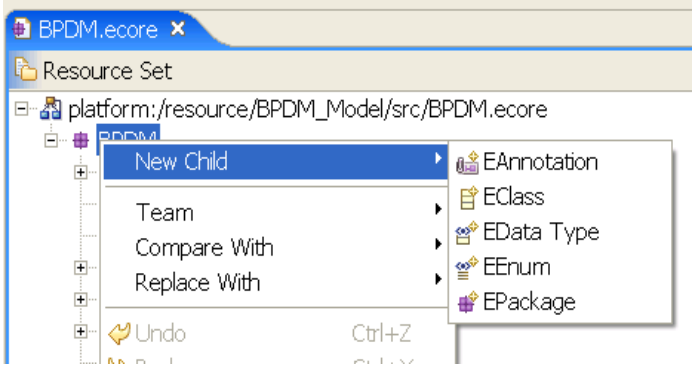
- **Metamodell mit UML-Tools erstellbar.**
- **MOF 2.0 definiert Complete MOF** (CMOF) mit zusätzlichen Eigenschaften.

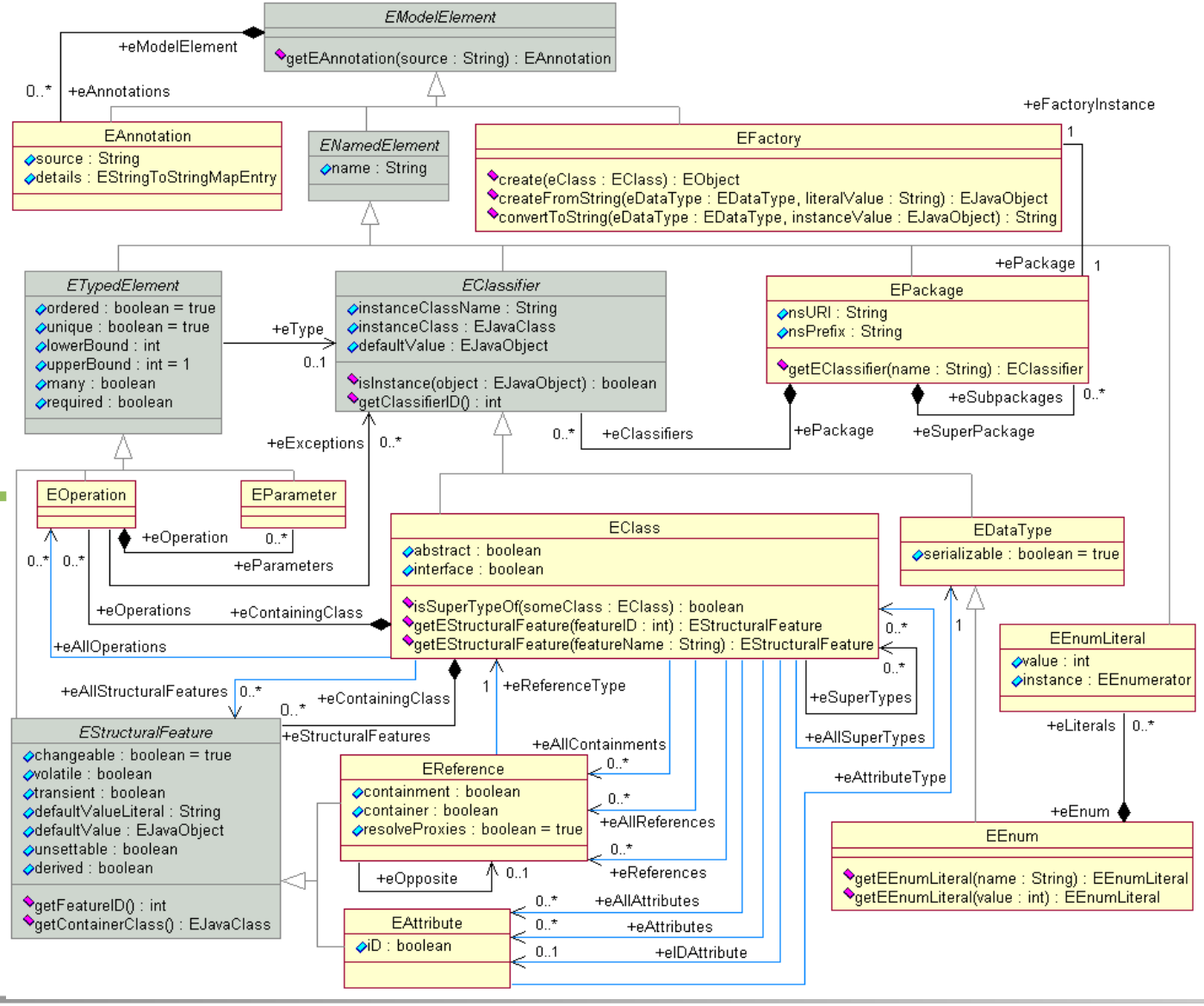
Beispiel: vereinfachtes Metamodell für Klassendiagramme (vgl. Teil 1.2 Folie 29 !)

Ecore – Das Kern-Metamodell für EMF

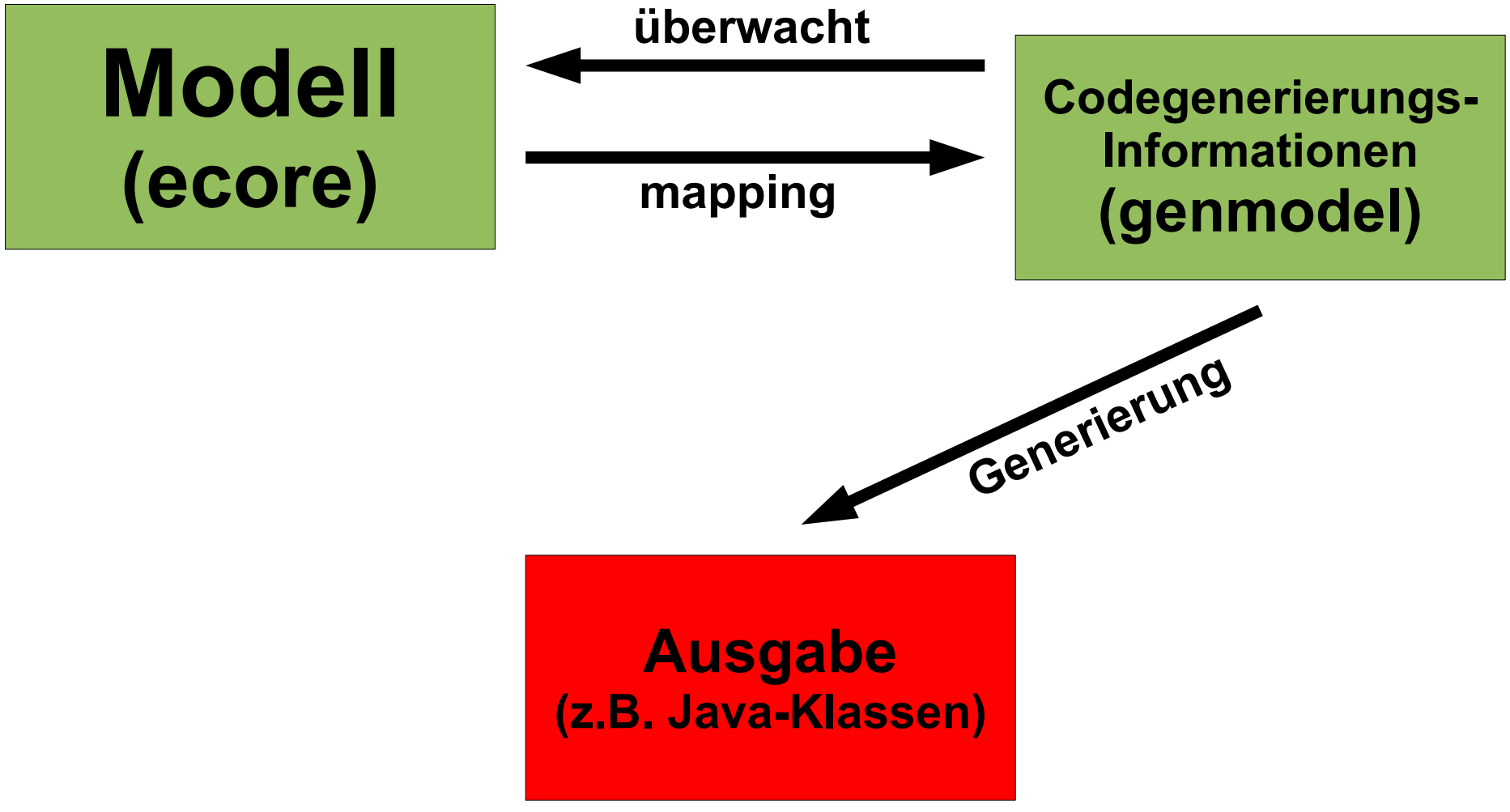
LEHRSTUHL 14
SOFTWARE ENGINEERING

verwendet für Metamodellierung





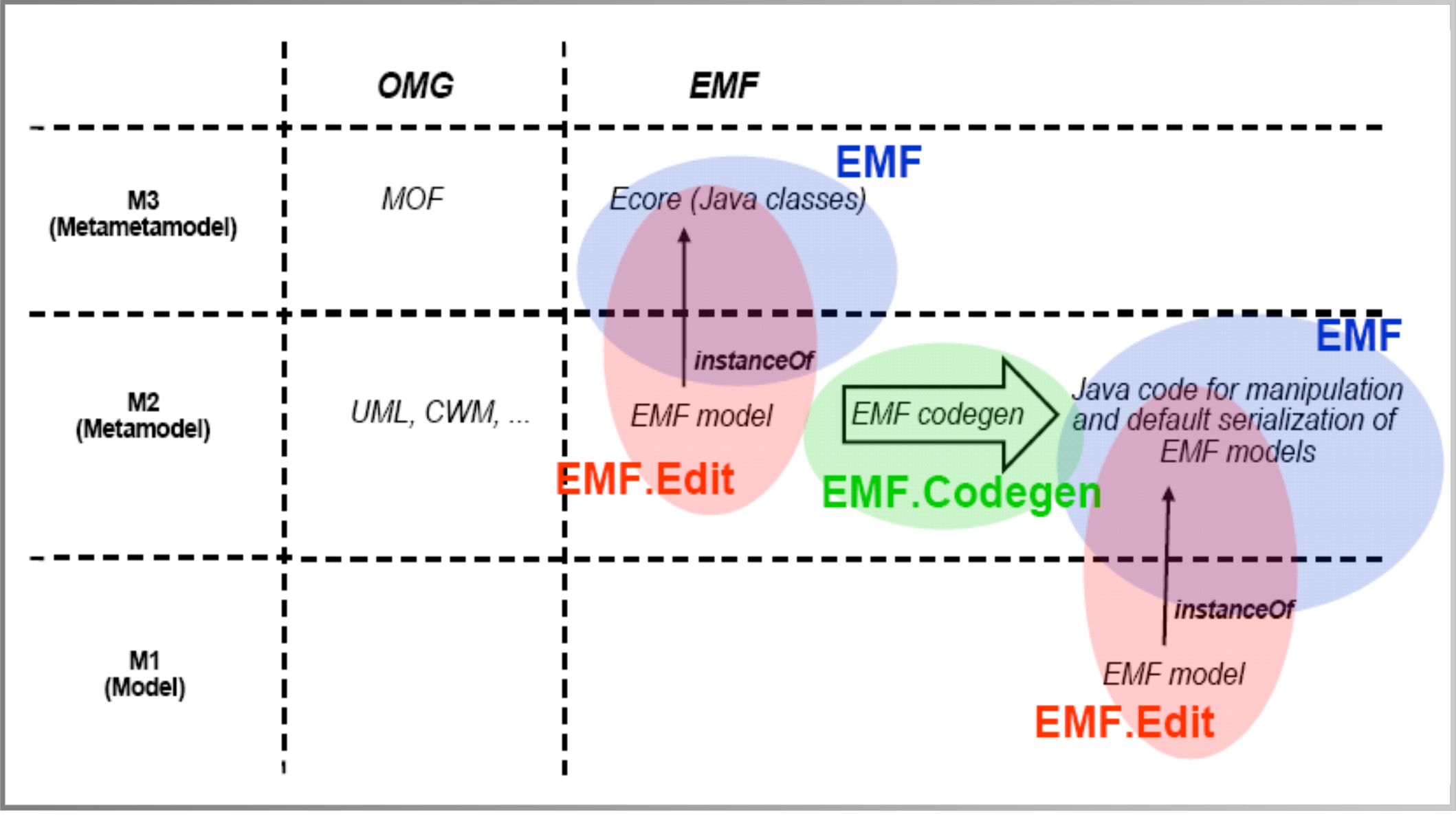
Abhängigkeit Modell- und Generierungsdateien



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.

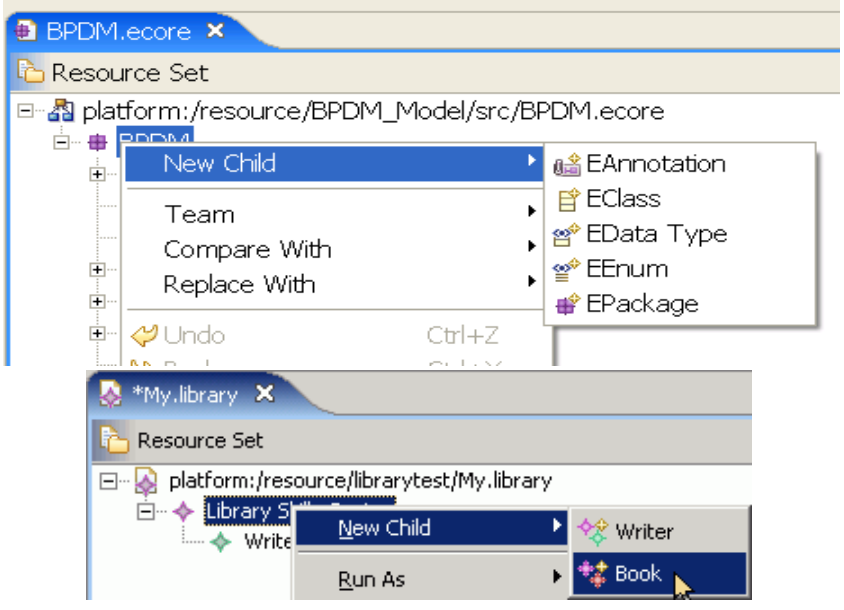
EMF – Überblick über Edit und Codegen



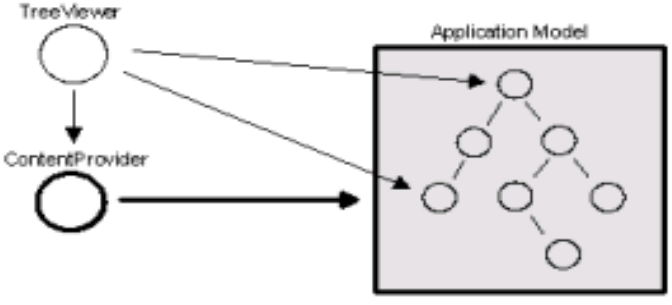
EMF.Edit – EMF.Codegen

EMF.Edit

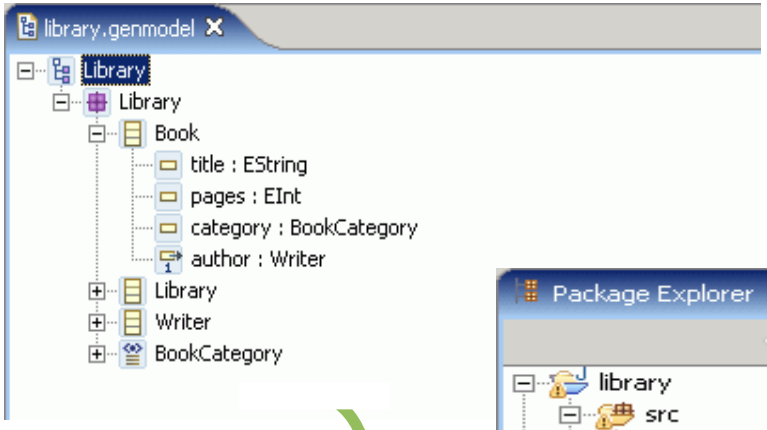
Modellierungseditor



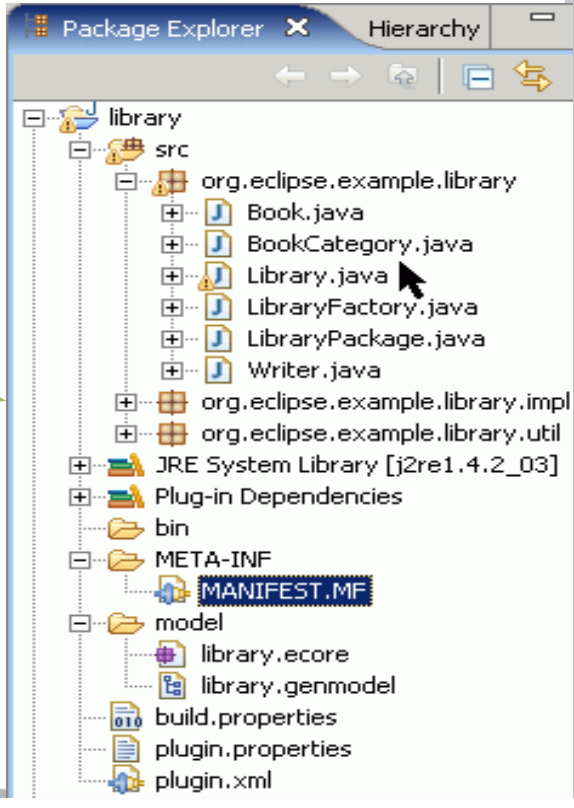
Content Provider, etc.



EMF.Codegen



generate



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).

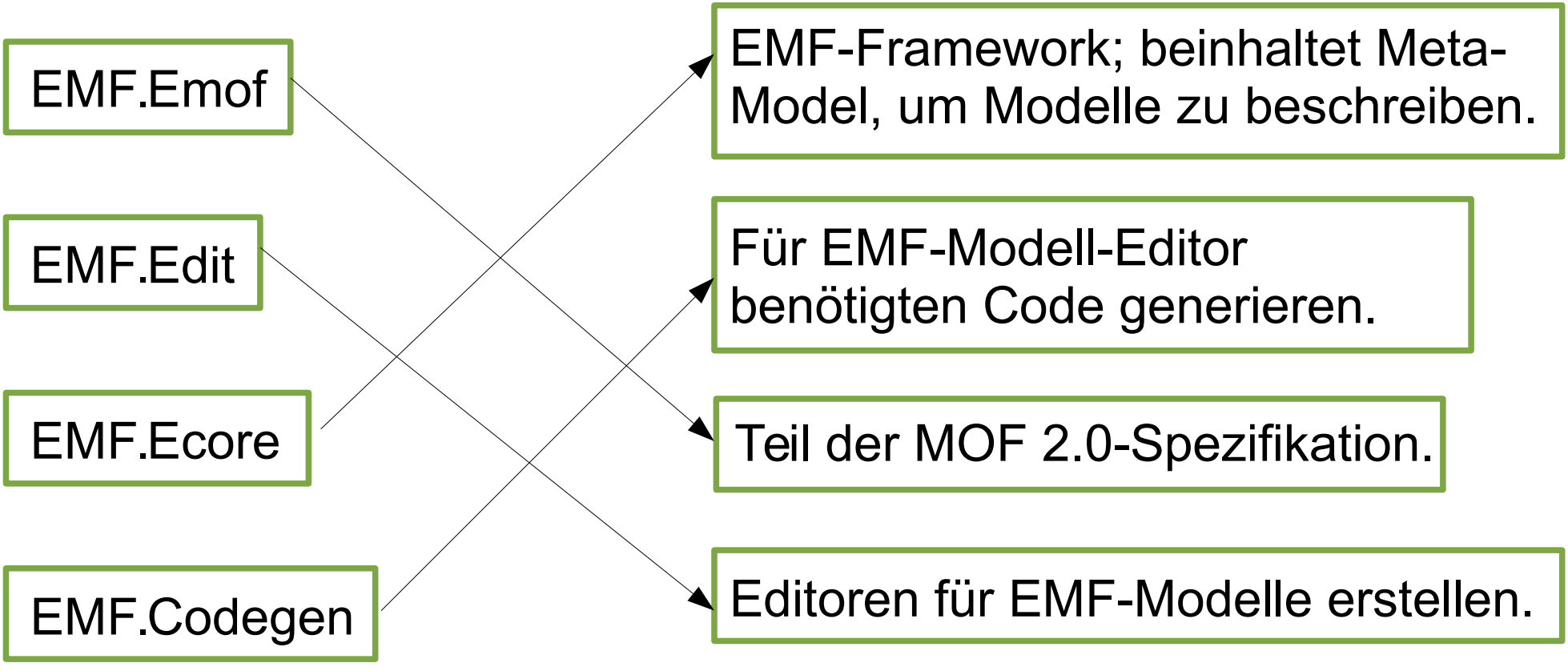
Diskussionsfrage: EMF-Bestandteile

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** ?



1.3 Eclipse Modeling Framework (EMF) Agenda

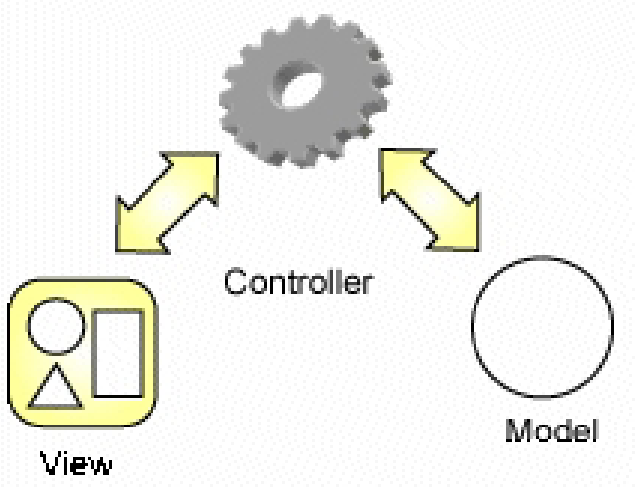
- Motivation und Überblick
- Eclipse Modeling Framework (EMF)
  - EMF-Modellimport
  - EMOF und Ecore
  - EMF Edit & Codegen
- Graphical Editing Framework (GEF)
  - Model-View-Controller (MVC)-Pattern
  - MVC in GEF
- Nutzung von EMF in GEF
  - Einführung eines Beispiels
  - Beispiel in EMF und GEF
-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

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.

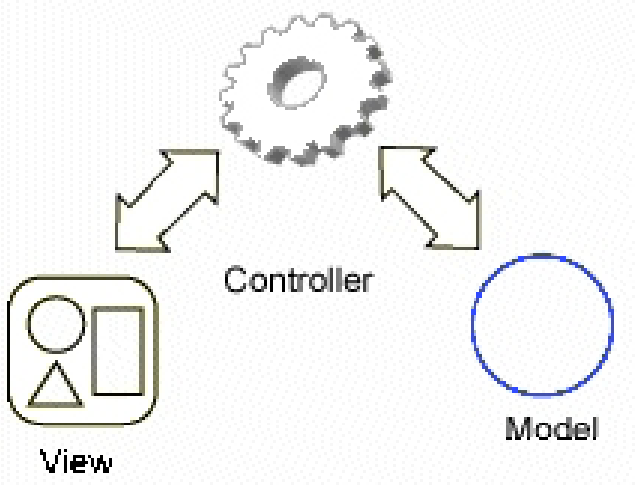
Model – View – Controller Pattern

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



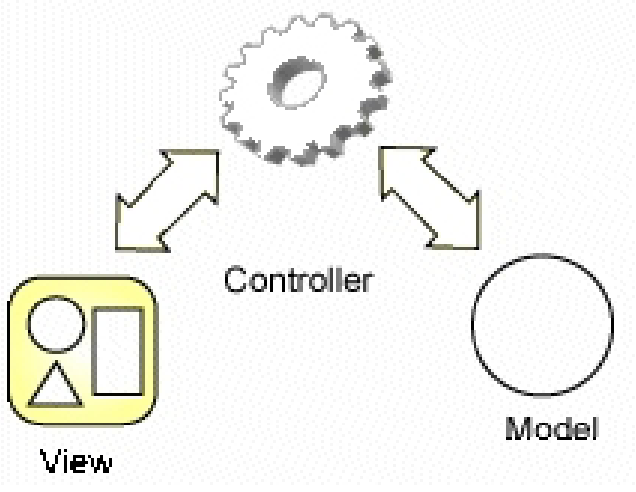
MVC – Pattern: Modellschicht

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



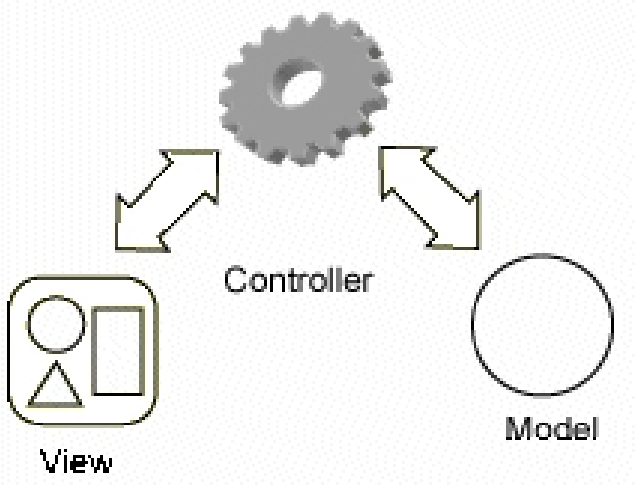
MVC – Pattern: Viewschicht

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



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.**



Diskussionsfrage: Verwendung von MVC-Pattern

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

Antwort:

Vorteile:

-

Nachteil:

-

-

Diskussionsfrage: Verwendung von MVC-Pattern

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

Antwort:

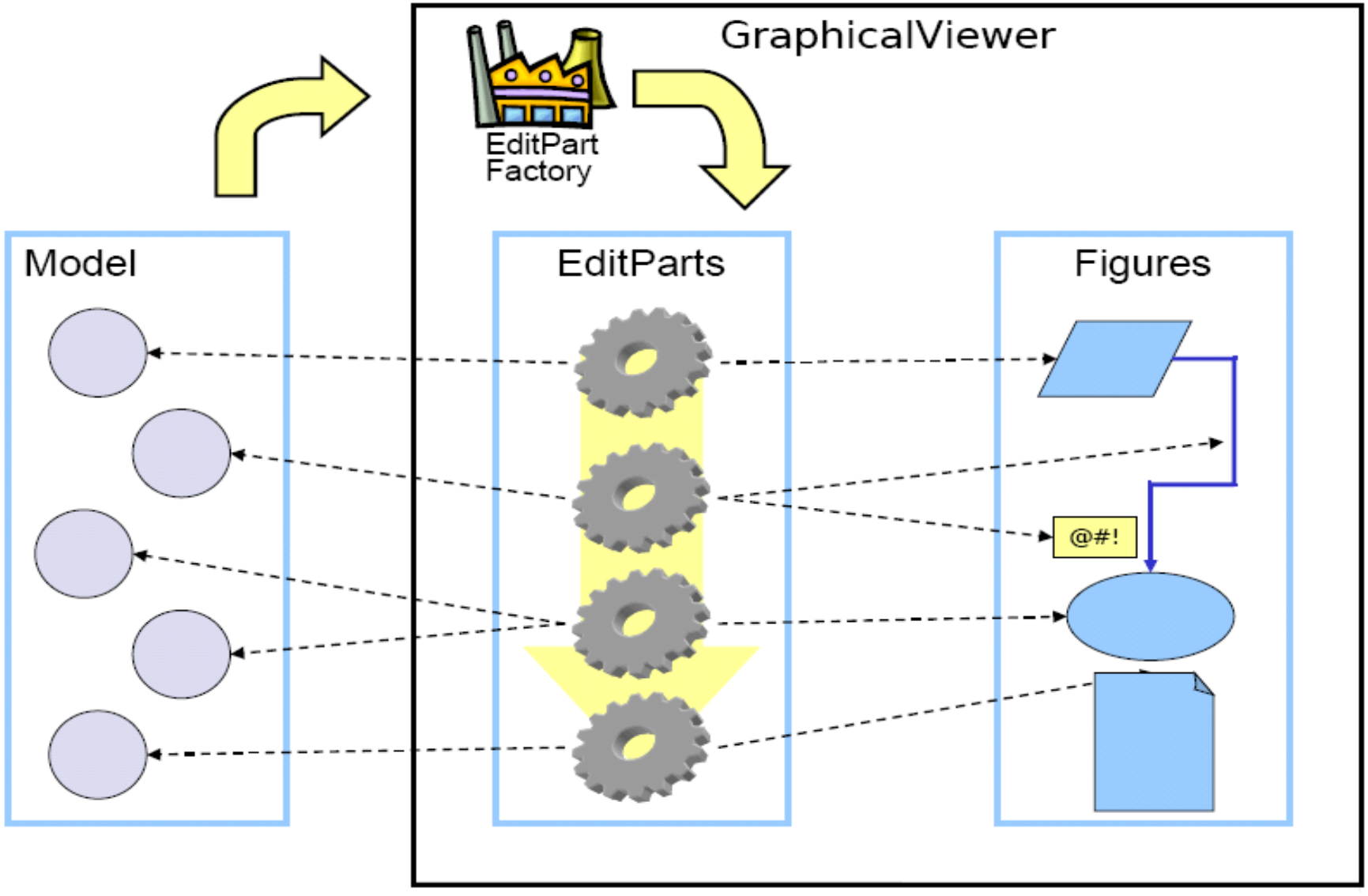
Vorteile:

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

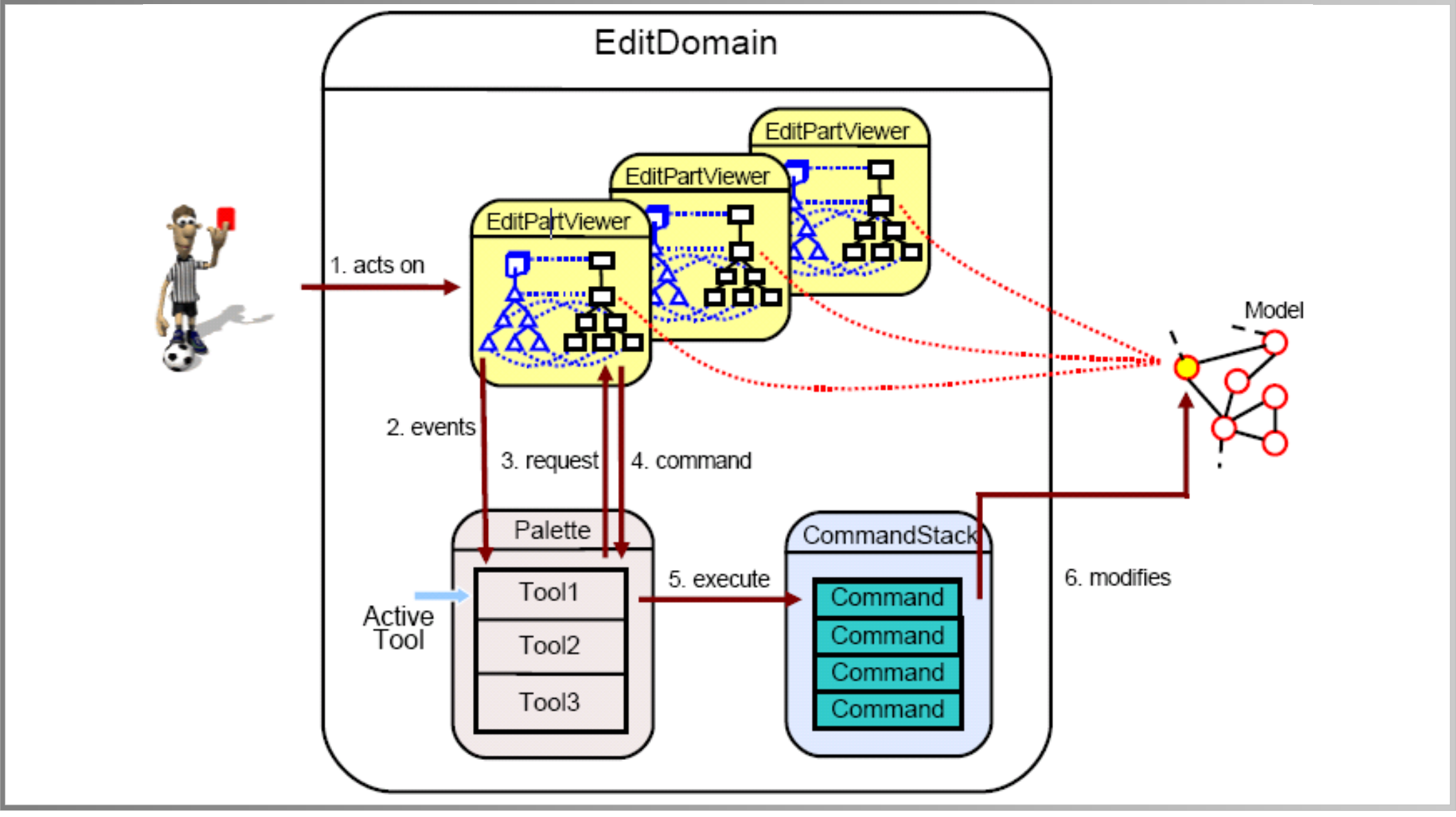
Nachteil:

- 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.

MVC in GEF: EditPartFactory



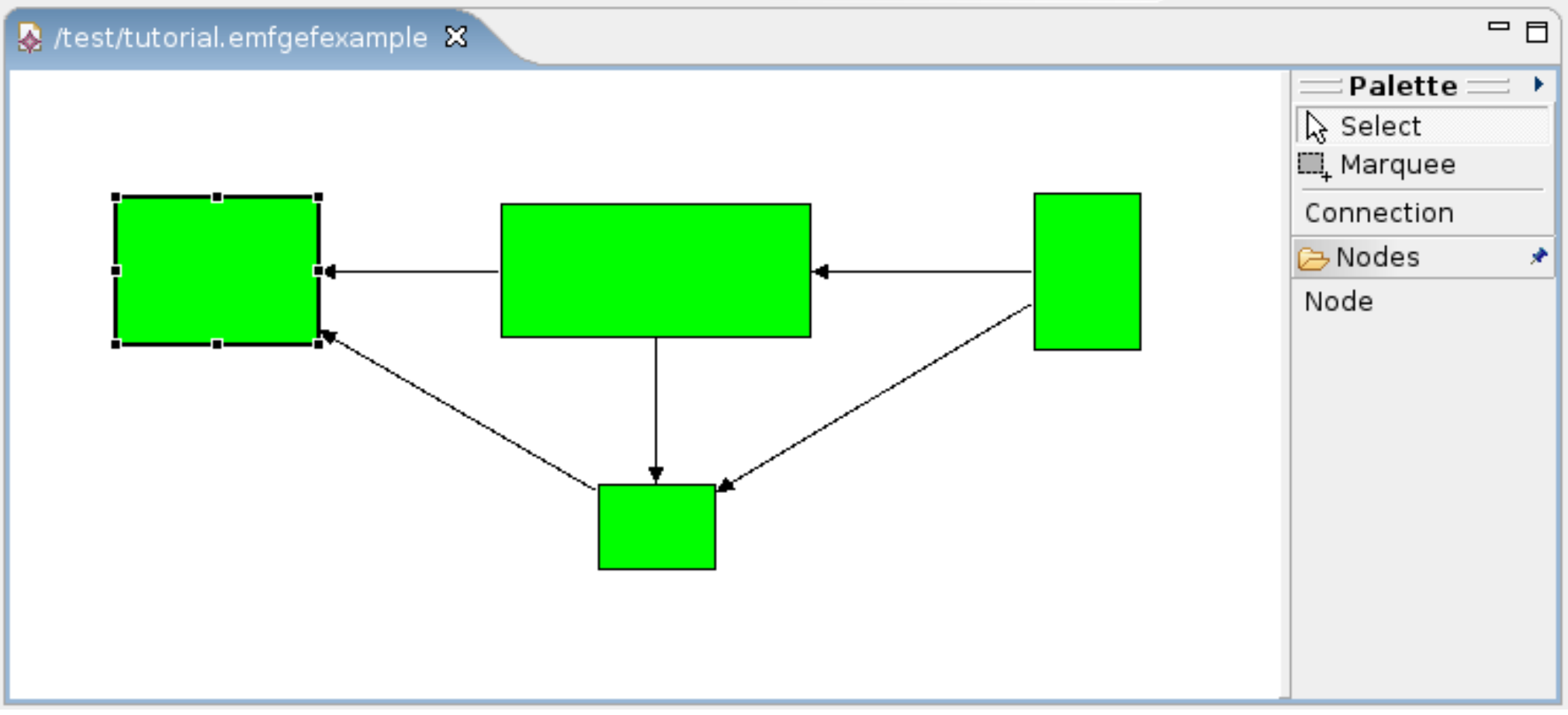
The Big Picture



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
- Nutzung von EMF in GEF
  - Einführung eines Beispiels
  - Beispiel in EMF und GEF
- Zusammenfassung und Ausblick

Beispiel - Editor



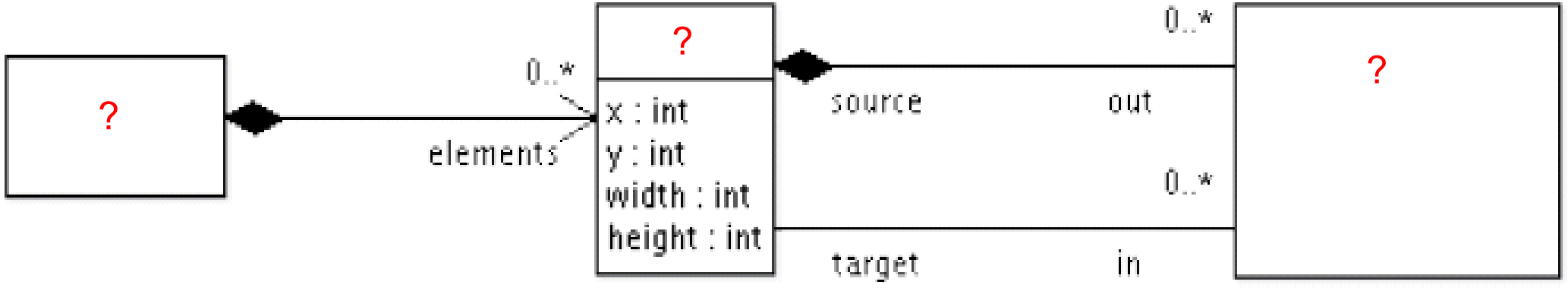
Beispiel - Features

- **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 ?

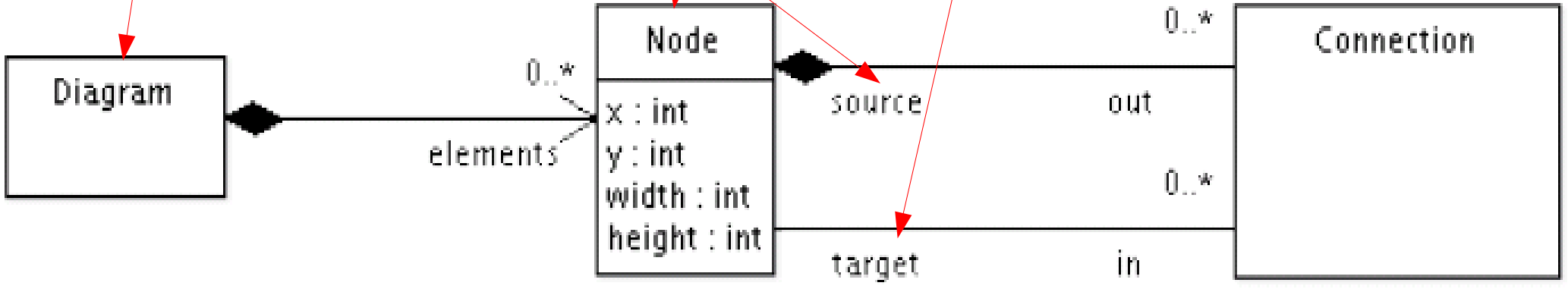
Was gehört an die fehlenden Stellen im unten abgebildeten Metamodell ?

- 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*).



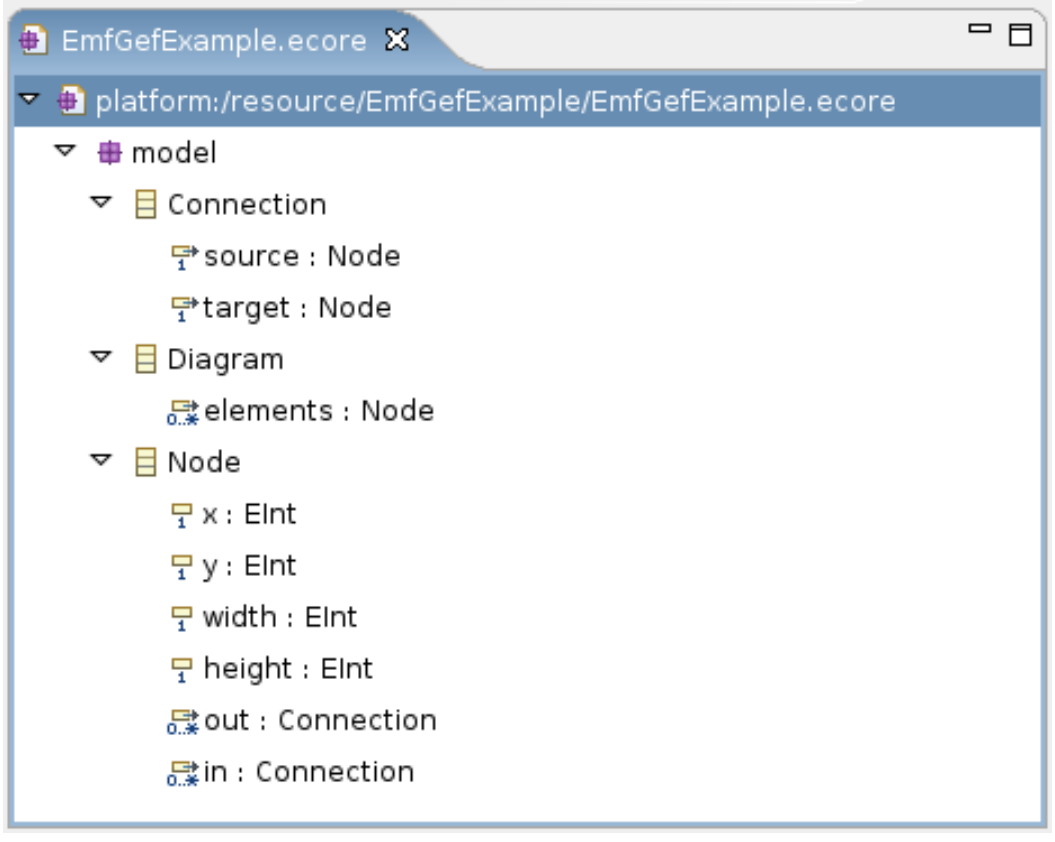
Diskussion: Metamodell für Beispiel-Editor ?

- Erstellt mit ArgoUML (http://argouml.tigris.org).
- **Diagram:** Wurzelelement.
- Diagramm enthält Knoten (*node*).
- Knoten besitzen **Quell- und Zielverbindungen** (*source Connection / target Connection*).



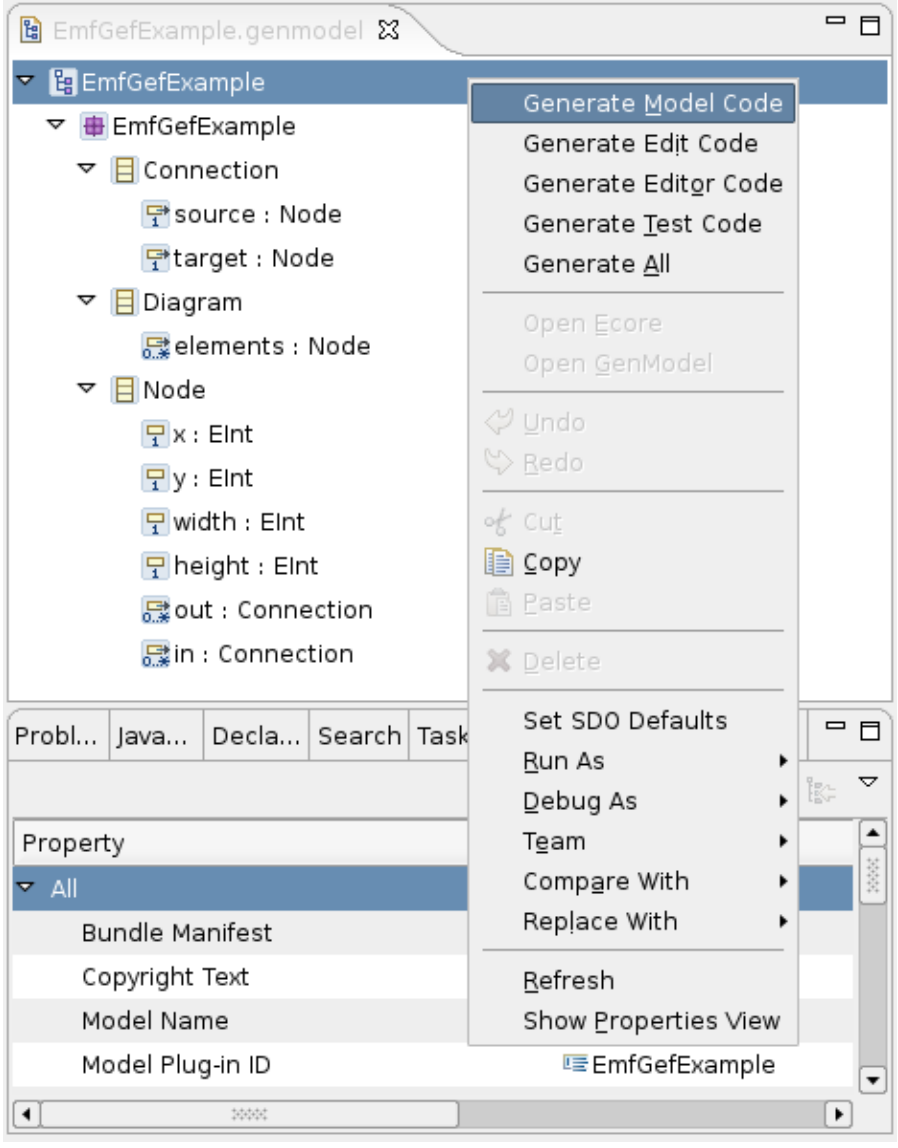
Metamodell UML → Ecore

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



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.



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
- Nutzung von EMF in GEF
  - Einführung eines Beispiels
  - Beispiel in EMF und GEF
- Zusammenfassung und Ausblick

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.

1.3 Eclipse Modeling Framework (EMF): Zusammenfassung und Ausblick

In diesem Abschnitt: Eclipse Modeling Foundation (EMF)

- Technische Grundlagen für UML-Werkzeuge und MDA.

Damit Ende des Kapitel 1: Modellbasierte Entwicklung.

Bauen darauf später noch auf: Modellbasierte Entwicklung sicherer Software (**Kapitel 4)**.

Aber zunächst (ebenfalls als **Grundlage für Kapitel 4**):
Softwarequalitätsmanagement (Kap. 2) und insbesondere **Softwareverifikation (Kap. 3)**.

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

Anhang
(weitere Informationen zu Nachbereitung)

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

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

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.

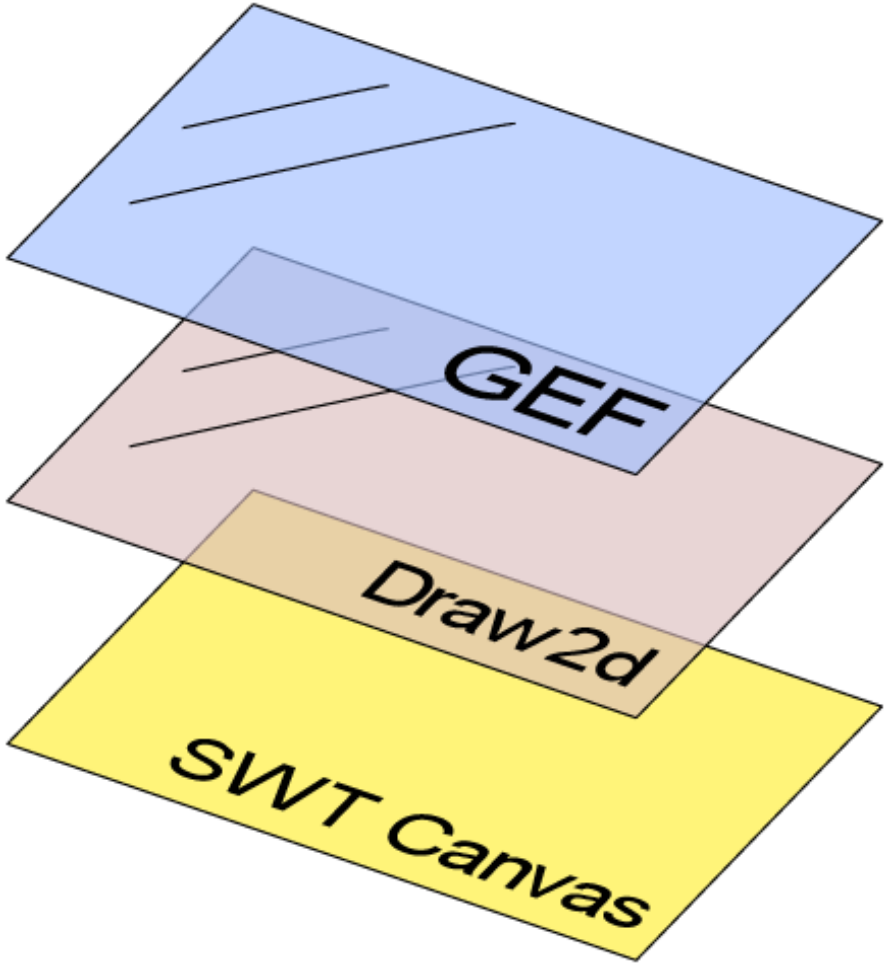
Erzeugte Java Klassen - Edit

- **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.

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

Was ist GEF?



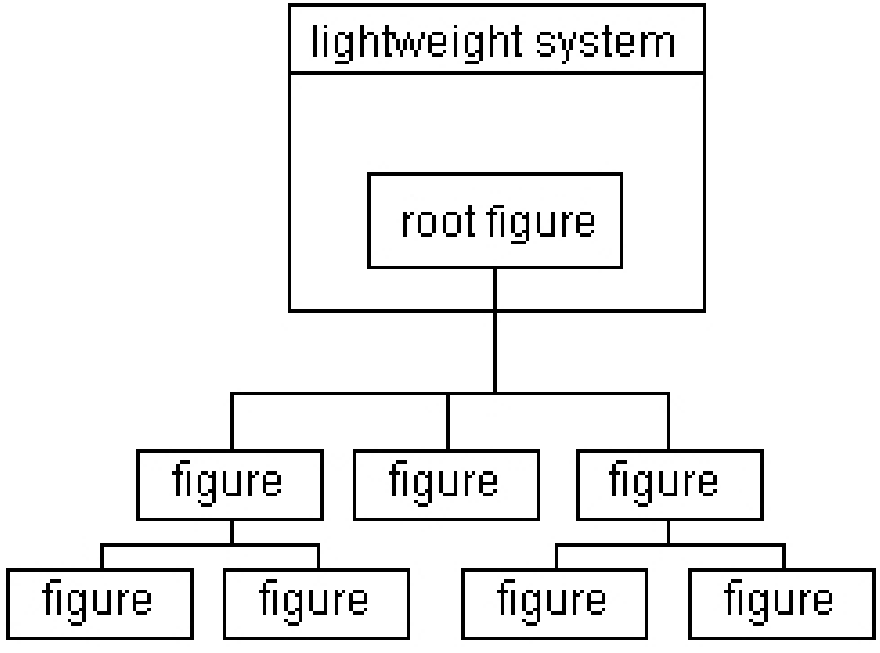
- Interaction Layer
- Model-to-View mapping
- Workbench Integration

- 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

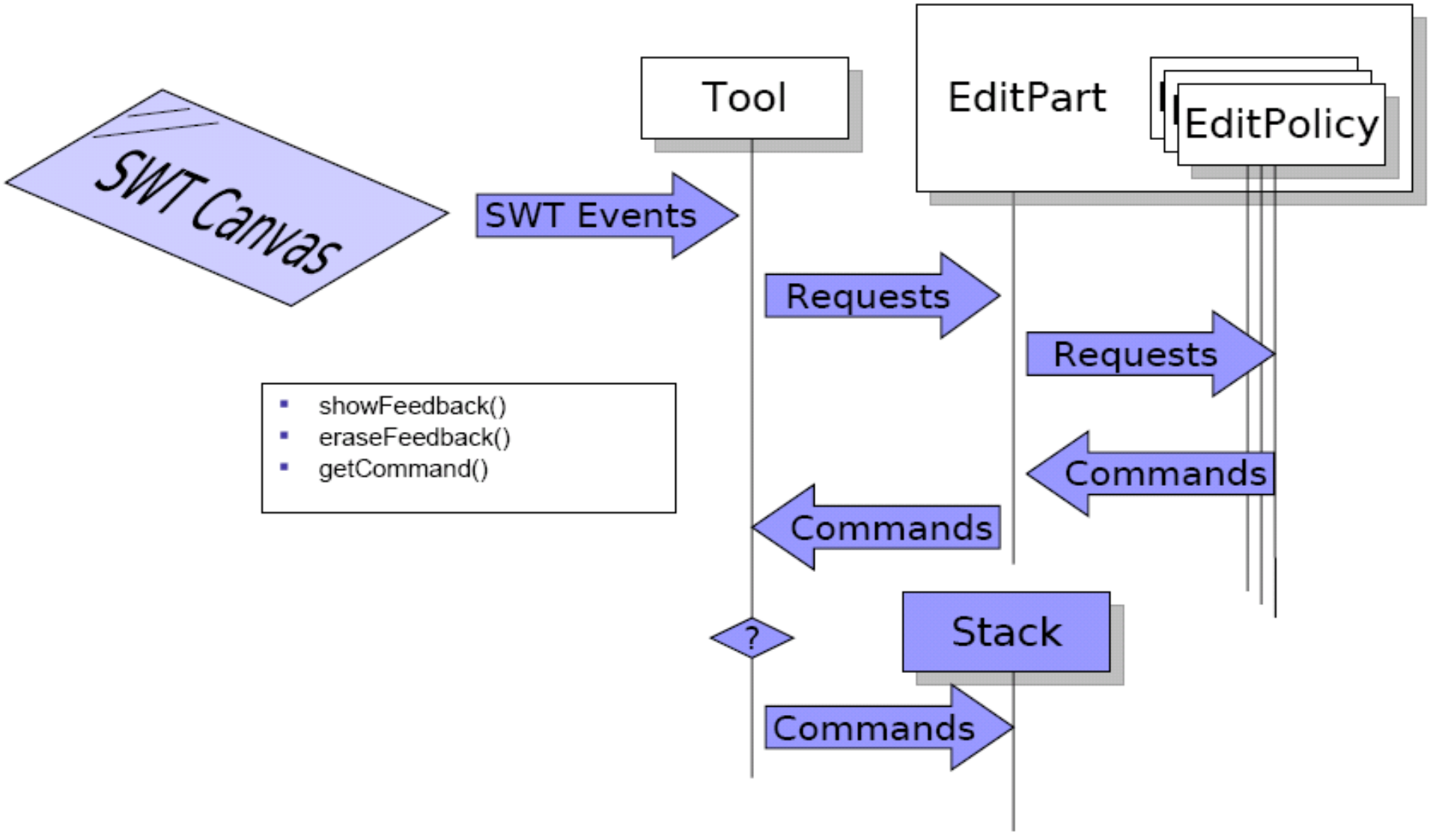
- Motivation und Überblick
- Eclipse Modeling Framework (EMF)
  - EMF-Modellimport
  - EMOF und Ecore
  - EMF Edit & Codegen
- Graphical Editing Framework (GEF)
  - Model-View-Controller (MVC)-Pattern
  - MVC in GEF
  - Weitere Konstrukte: EditPolicies und Commands
- Nutzung von EMF in GEF
  - Einführung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

LEHRSTUHL 14
SOFTWARE ENGINEERING

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.

Modell: Commands

- Jedes erstellbare Objekt benötigt ***CreateCommand:***

```
public void execute(){
      diagram.getElements().add(newNode);
        }


public void undo(){
   diagram.getElements().remove(newNode);
     }
```

Modell: Commands

- Jedes löschbare Objekt benötigt ***DeleteCommand:***

```
public void execute(){
   source = connection.getSource();
        target = connection.getTarget();
             connection.setSource(null);
             connection.setTarget(null);
               }


public void undo(){
   connection.setSource(source);
      connection.setTarget(target);
        }
```

Modell: Commands

- Jedes änderbare Objekt benötigt ein ***Command:***

```
public void execute(){
    oldBounds = new Rectangle(node.getX(), node.getY(),
           node.getWidth(), node.getHeight());
           redo();
         }

public void redo(){
    node.setX(newBounds.x);
    node.setY(newBounds.y);
    node.setWidth(newBounds.width);
    node.setHeight(newBounds.height);
  }

public void undo(){
    node.setX(oldBounds.x);
    node.setY(oldBounds.y);
    node.setWidth(oldBounds.width);
    node.setHeight(oldBounds.height);
  }
```

View: Figures

- In **EditParts** erzeugt.
- **Komplexere Figures** als eigene Unterklassen implementiert.
- **Node:**

```
protected IFigure createFigure(){
    IFigure f = new RectangleFigure();
    f.setOpaque(true);
    f.setBackgroundColor(ColorConstants.green);
    return f;
 }
```

- **Connection:**

```
protected IFigure createFigure(){
    PolylineConnection connection =
        (PolylineConnection) super.createFigure();
        connection.setTargetDecoration(new PolygonDecoration());
        return connection;
    }
```

Controller: EditPartFactory

- **EditParts** in **EditPartFactory** erzeugt und mit Modellobjekt verbunden.

```
public EditPart createEditPart(EditPart context, Object model)
{     EditPart editPart = null;
      if (model instanceof Diagram){
         editPart = new DiagramEditPart();
      } else if (model instanceof Node){
         editPart = new NodeEditPart();
      } else if (model instanceof Connection){
         editPart = new ConnectionEditPart();
      }
   editPart.setModel(model);
   return editPart;
   }
```

Controller: Adapter

- EditParts müssen *org.eclipse.emf.common.notify.Adapter* implementieren. → Auf **Änderungen aus EMF Modell** reagieren.
- **Registrierung als Adapter** zum entsprechenden Modellobjekt:

```
public void activate(){
   super.activate();
   getTarget().eAdapters().add(this);
   }

public void deactivate(){
   super.deactivate();
   getTarget().eAdapters().remove(this);
   }
```

Controller: Reaktion auf Notifications


```
public void notifyChanged(Notification notification){
     int featureId = notification.getFeatureID(Node.class);
     switch (notification.getEventType()){
          case Notification.SET:
          case Notification.UNSET:
               switch (featureId){
                    case EmfGefExamplePackage.NODE__X:
                    case EmfGefExamplePackage.NODE__Y:
                    case EmfGefExamplePackage.NODE__WIDTH:
                    case EmfGefExamplePackage.NODE__HEIGHT:
                    refreshVisuals();
                    break;
               }
               break;
          case Notification.ADD:
          case Notification.ADD_MANY:
          case Notification.REMOVE:
          case Notification.REMOVE_MANY:
          switch (featureId){
               case EmfGefExamplePackage.NODE__IN:
               refreshTargetConnections();
               break;
          case EmfGefExamplePackage.NODE__OUT:
               refreshSourceConnections();
               break;
     }
     break;
}}
```

Controller: Hierarchie

- ***Nodes* als Kindelemente** von *Diagram*:

```
protected List getModelChildren(){
     return ((Diagram) getModel()).getElements();}
```

- **Connections** zwischen *Nodes:*

```
protected List getModelSourceConnections(){
     return ((Node) getModel()).getOut();}

protected List getModelTargetConnections(){
     return ((Node) getModel()).getIn();}
```

Controller: EditPolicies - DiagramEditPart


```
protected void createEditPolicies(){

   // disallows the removal of this edit part from its parent
   installEditPolicy(EditPolicy.COMPONENT_ROLE,
      new RootComponentEditPolicy());

   // handles constraint changes
   //(e.g. moving and/or resizing) of model
   // elements and creation of new model elements
   installEditPolicy(EditPolicy.LAYOUT_ROLE,
      new DiagramXYLayoutEditPolicy());
}
```

Controller: EditPolicies - DiagramXYLayoutEditPolicy


```
protected Command getCreateCommand(CreateRequest request){
    Object childClass = request.getNewObjectType();
    if (childClass == Node.class){
        Node newNode = (Node) request.getNewObject();
        Command command = new NodeCreateCommand(newNode,
            (Diagram) getHost().getModel());
        return command.chain(new NodeSetConstraintCommand(newNode,
        (Rectangle) getConstraintFor(request)));
    }
    return null;
}

protected Command createChangeConstraintCommand(ChangeBoundsRequest
    request, EditPart child, Object constraint){
    if (child.getModel() instanceof Node && constraint instanceof
        Rectangle){
            return new NodeSetConstraintCommand((Node)
                child.getModel(), (Rectangle) constraint);
    }
    return super.createChangeConstraintCommand(request, child,
    constraint);
}
```

Controller: EditPolicies - NodeEditPart


```
protected void createEditPolicies(){

    // allow removal of the associated model element
    installEditPolicy(EditPolicy.COMPONENT_ROLE,
        new ComponentEditPolicy(){
            protected Command getDeleteCommand(GroupRequest request){
                return new NodeDeleteCommand((Node)
                    getHost().getModel());
            }
        });

    // allow the creation of connections
    installEditPolicy(EditPolicy.GRAPHICAL_NODE_ROLE,
        new NodeGraphicalNodeEditPolicy());
}
```

Controller: EditPolicies - NodeGraphicalNodeEditPolicy


```
protected Command getConnectionCreateCommand(CreateConnectionRequest
    request){
        Node node = (Node) getHost().getModel();
        ConnectionCreateCommand command = new
            ConnectionCreateCommand(request, node);
        request.setStartCommand(command);
        return command;
}

protected Command getConnectionCompleteCommand(
    CreateConnectionRequest request){
        ConnectionCreateCommand command =
            (ConnectionCreateCommand) request.getStartCommand();
        command.setTarget((Node) getHost().getModel());
        return command;
}
```

Editor: Laden eines Modells

```
String fileName = ((IfileEditorInput) getEditorInput()).
    getFile().getFullPath().toString();
URI resourceUri = URI.createPlatformResourceURI(fileName);

// Create a resource set
ResourceSet resourceSet = new ResourceSetImpl()

// register plugin extension to XMI serialization
resourceSet.getResourceFactoryRegistry().getExtensionToFactoryMap().
put(EmfGefExampleCreationWizard.DEFAULT_EXTENSION,
    new XMIResourceFactoryImpl());

// Create a resource for this file.
modelResource = resourceSet.createResource(resourceUri);

// load model
modelResource.load(Collections.EMPTY_MAP);
Diagram diagram = modelResource.getContents().get(0);
```

Editor: Speichern des Modells


```
try{
   modelResource.save(Collections.EMPTY_MAP);
   getCommandStack().markSaveLocation();
} catch (IOException e){
   // ...
}
```

Editor: Erstellen eines neuen Modells im Wizard


```
IFile newFile = createNewFile();
ResourceSet resourceSet = new ResourceSetImpl();
resourceSet.getResourceFactoryRegistry().
   getExtensionToFactoryMap().put(DEFAULT_EXTENSION,
      new XMIResourceFactoryImpl());
URI fileURI = URI.createPlatformResourceURI(
      newFile.getFullPath().toString());
Resource resource = resourceSet.createResource(fileURI);
Diagram rootObject = EmfGefExampleFactory.eINSTANCE.
   CreateDiagram();
resource.getContents().add(rootObject);
try{
   resource.save(Collections.EMPTY_MAP);
} catch (IOException e){
}
```

Editor: Weitere Komponenten

- **AbstractUIPlugin:**
  - Instanz des Plugins.
- **ActionBarContributor:**
  - Actions für ToolBar.
  - Actions für Menüs.
  - Im Beispiel: Undo, Redo, Delete.
- **ContextMenuProvider:**
  - Kontextmenü im Editor.
  - Im Beispiel: Undo, Redo, Delete.
- **Plugin.xml:**
  - Name, Version, ID des Plugins.
  - Durch Abhängigkeiten benötigte Plugins.
  - Registrierung des Wizards.
  - Registrierung des Editors.