Workshop „Methodische Entwicklung von Modellierungswerkzeugen“
INFORMATIK 2009 – 39. Jahrestagung der Gesellschaft für Informatik

An ECLIPSE Framework for Rapid Development of Rich-featured GEF Editors based on EMF Models

Tony Modica, Enrico Biermann, Claudia Ermel

Theoretische Informatik – Formale Spezifikation (Prof. Hartmut Ehrig)

Technische Universität Berlin

Graduate Program Human-Centric Communication (H-C3)

Überblick

- Ganz kurz: EMF und GEF
- Ziele
- Was ist MuvitorKit?
- Beispielprojekte
- Aufbau von MuvitorKit-Editoren im Detail
  - Implementation der Komponenten
  - Vorteile
- Animationen mit MuvitorKit
- Demo: RONEditor

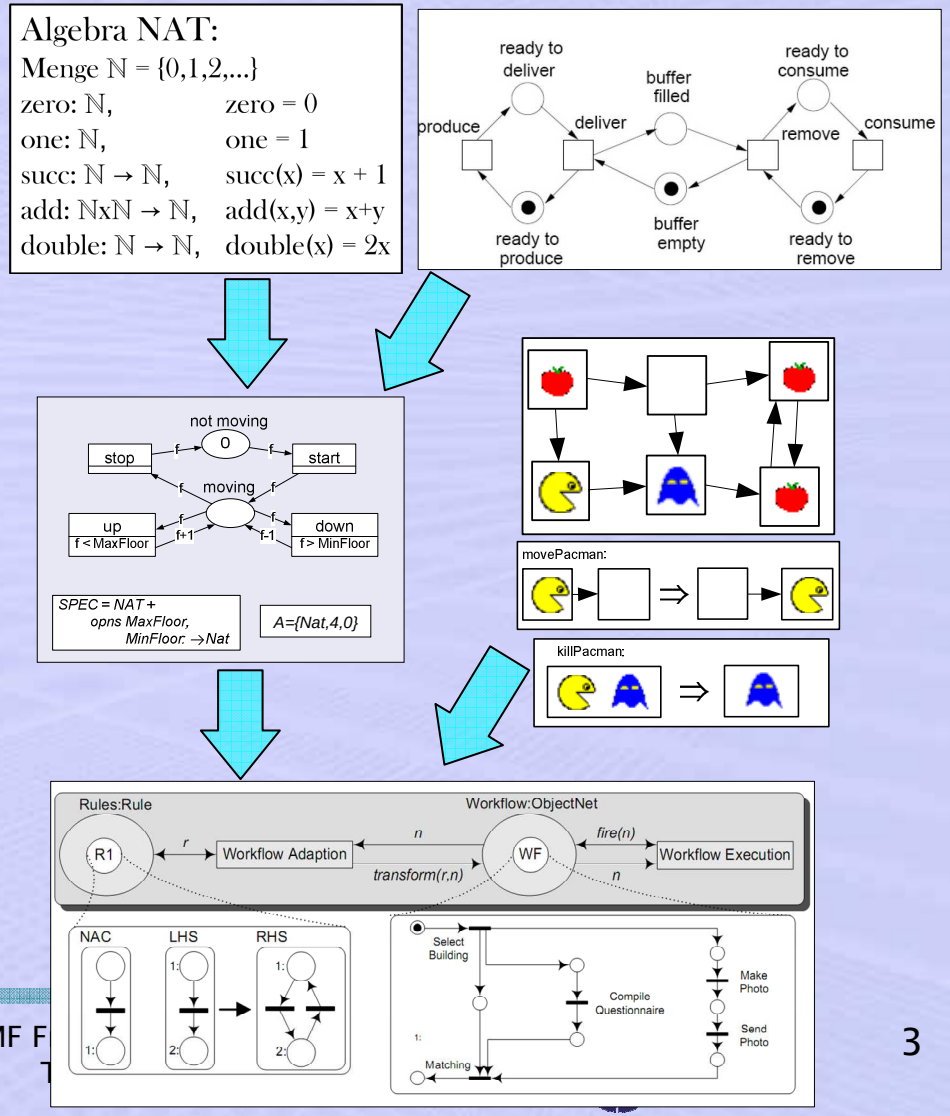
Was machen wir *eigentlich*?

Theorien für formale Modellierung:

- Algebren (Spezifikationen)
- Petrinetze
- Graphtransformation
  - Modelltransformation
- *integrierte Techniken*
- High-level Replacement Systeme (Framework)
  - Kategorientheorie

⇒ Programmierprojekt Visuelle Sprachen (ViLa)

- Graphische Editoren mit Eclipse Modeling Framework (EMF) + Graphical Editing Framework (GEF)



EMF - Überblick

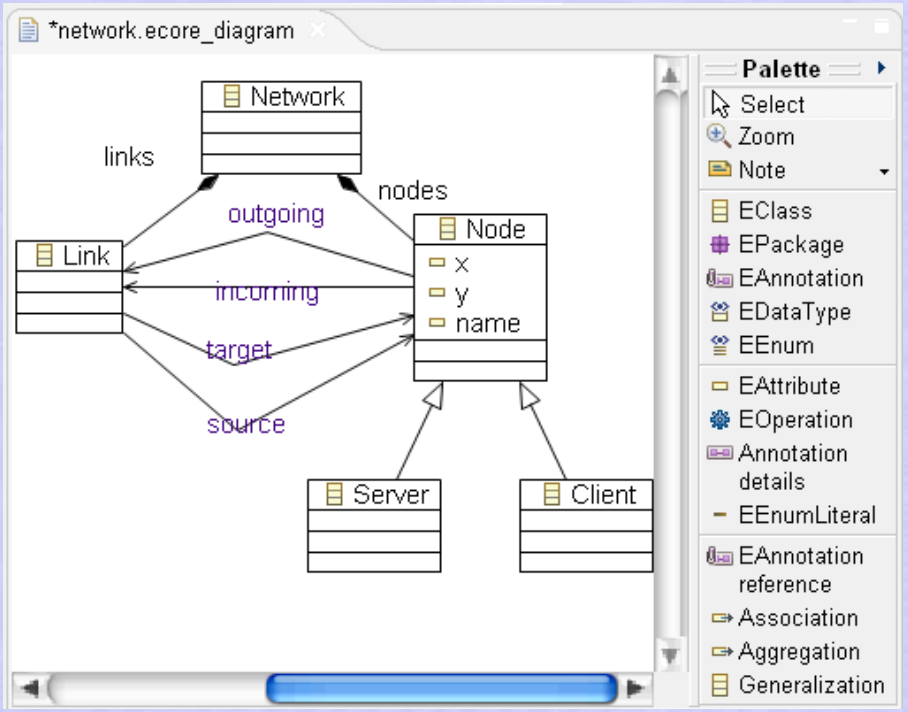
erzeugt aus EMF-Modell instanziierbaren Code

\+ Factory, Reflection, qualifizierte Notifications

```
 * <!-- begin-user-doc -->
public interface Network extends EObject {
     * Returns the value of the '<em><b>Links</b></em>'
    EList getLinks();

     * Returns the value of the '<em><b>Nodes</b></em>'
    EList getNodes();

} // Network
```

Server.java | Node.java

```
 * <copyright>
package networkModel;


 * <!-- begin-user-doc -->
public interface Server extends Node {
} // Server
```



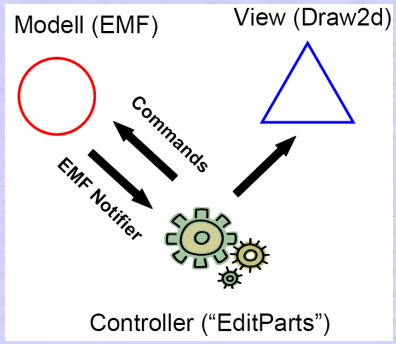

```
 * <!-- begin-user-doc -->
public interface Node extends EObject {
     * Returns the value of the '<em><b>Network</b></em>'
    Network getNetwork();

     * Sets the value of the '{@link networkModel.Node#ge
    void setNetwork(Network value);

     * Returns the value of the '<em><b>Incoming</b></em>
    EList getIncoming();

     * Returns the value of the '<em><b>Outgoing</b></em>
    EList getOutgoing();

     * Returns the value of the '<em><b>X</b></em>' attri
    int getX();

     * Sets the value of the '{@link networkModel.Node#ge
    void setX(int value);

     * Returns the value of the '<em><b>Y</b></em>' attri
    int getY();

     * Sets the value of the '{@link networkModel.Node#ge
    void setY(int value);

     * Returns the value of the '<em><b>Name</b></em>' at
    String getName();

     * Sets the value of the '{@link networkModel.Node#ge
    void setName(String value);

} // Node
```

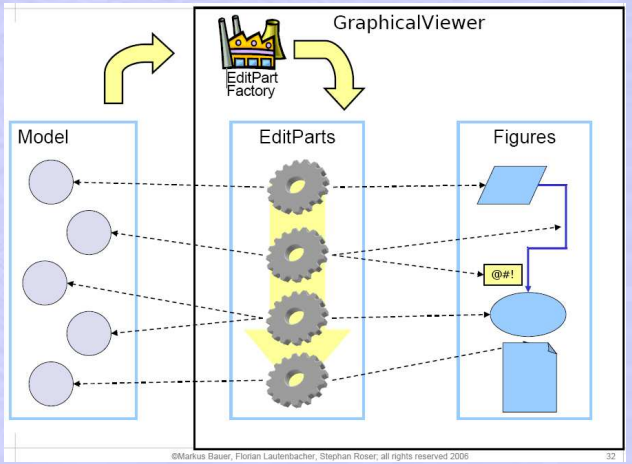
GEF - Überblick

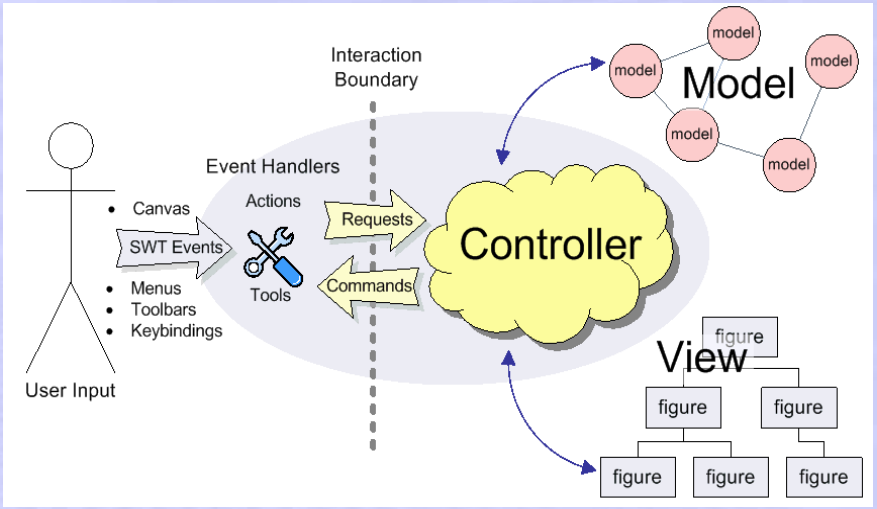
MVC-Prinzip:

- Model (EMF)
  - Daten, Persistenz
  - veränderbar über Commands
- View
  - Modelldarstellung
  - *Figures* (Draw2d), *TreeItems*
- Controller (*EditParts*)
  - Faustregel: ein EditPart pro editierbarem/dargestelltem Modellobjekt
  - EditPartViewers: Panel zur Darstellung des Views eines EditParts (*GraphicalViewer/TreeViewer*)
  - *EditPolicies* zur Bearbeitung von Requests

- Nicht verwechseln mit Eclipse-Views!
  - Allgemein in Eclipse: Editoren und Views sind Workbenchparts
  - Editoren sind Views mit einem "Input"







Was wollten wir erreichen?

Hilfe für ViLa-Studenten

- Einstieg in GEF vereinfachen
- Good Practices vermitteln, unflexible Ad-hoc-Implentierung vermeiden
- generisch implementierte Editorfeatures vorgeben
  - wie Persistenz, Direct Edit, Cut/Copy/Paste, automatisches Layout

  ⇒ im Projekt Konzentration auf modellspezifische Features wie Simulation und Analyse
- Unterstützung und komplexer Modelle mit unterschiedlichen Komponenten
  - Graphen, Petrinetze, Regeln, Aktivitätendiagramme
  - verschachtelt

Multi-View-Editor Framework (**MuvitorKit**)

- konfigurierbare abstrakte Basisklassen
- Hilfsmethoden, zweckorientiert dokumentiert
- Grundannahme: EMF-Modell
  - generische Actions
    - vorkonfiguriertes Kontextmenü und Actionbars
  - EMF-Eigenschaften wie Notificationmechanismus ausnutzen
  - spezielle Editparts
- einheitliches Multi-View-Konzept
  - Views mit mehreren GEF Viewers
  - beliebige graphische Views für Komponenten in einem Editor

    ⇒ flexible Projektplanung durch unabhängige Viewkomponenten

Was MuvitorKit (nicht) ist

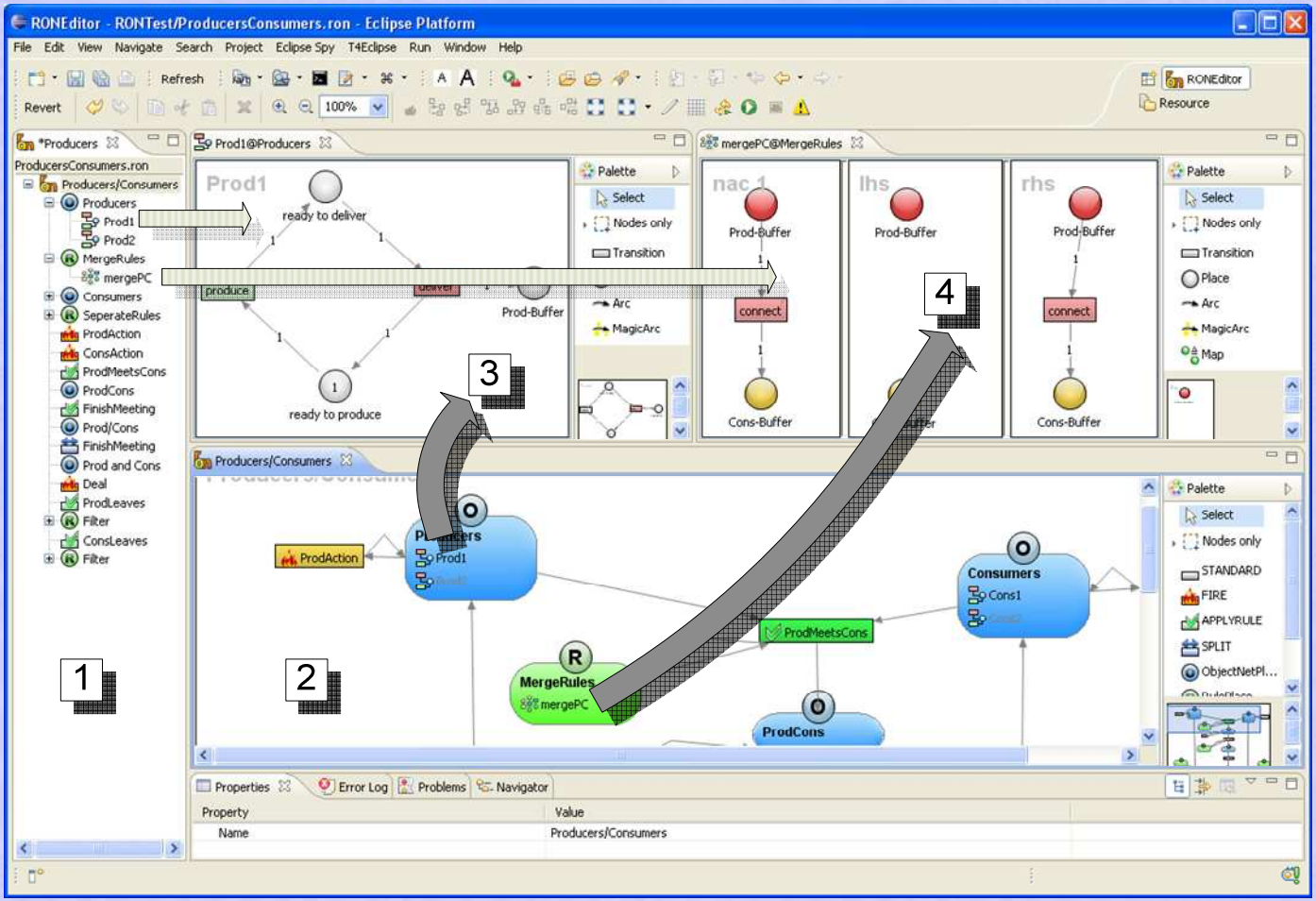
kein GMF-Nachbau

- kein GUI
- keine View/Operations-Spezifikation
- keine Codegenerierung

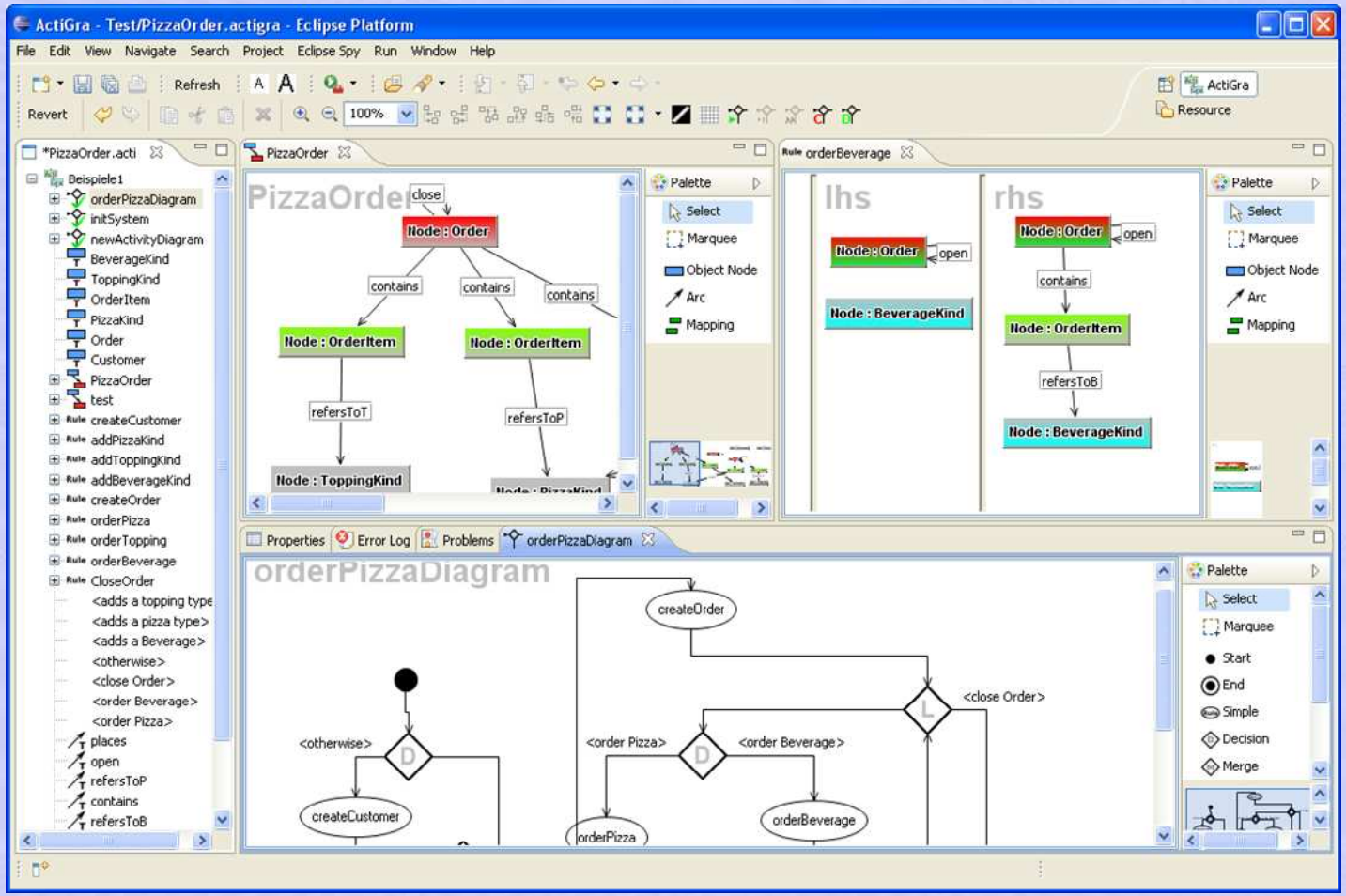
aber:

- dokumentierte Sammlung von bewährtem Code und Vorgehensweisen
  - „Wo fange ich an?“ (selbst, ohne generiertem Code)
  - „Was geht überhaupt und wie sollte ich das machen?“
- schnell zu konfigurierendes Skelett für komplexe Editoren
  - einfache Integration von GEF-basierten Editoren in die Workbench „bis zur GEF-Grenze“
  - viele Features frei Haus dank EMF
  - trotzdem Freiheit bei Implementierung komplexer Funktionen ohne Vorgabe von generiertem (unverständlichen) Code
- Unterstützung für (visuell) heterogene, beliebig verschachtelte Modelle
- Bonus: Package für animierte Commands

Beispiel 1: RON Editor

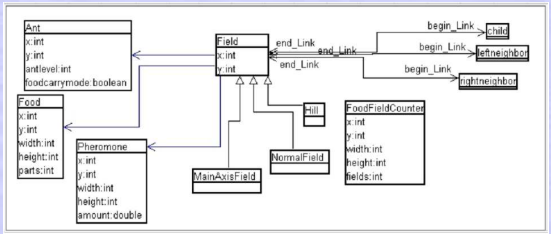


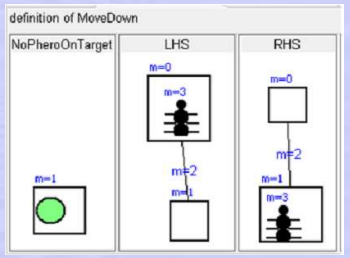
Beispiel 2: ActiGra Editor

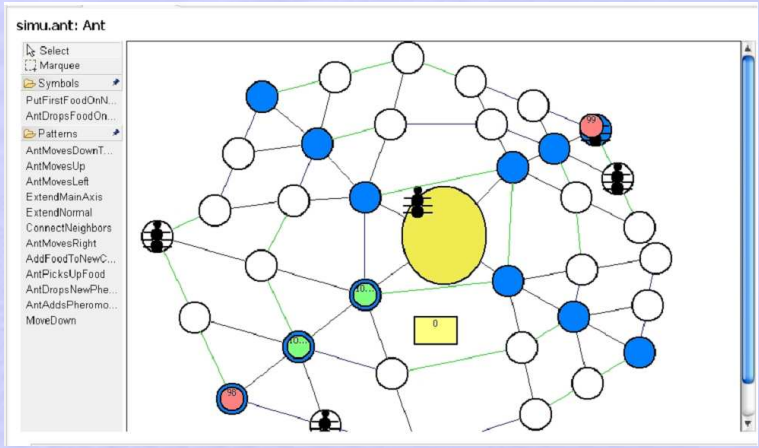


Weitere Projekte

- ViLa Studentenprojekt WS-2009: Editor für EMF-Tiger
  - integriertes Modell für Modelltransformationen in EMF
    - EMF-Klassendiagramme (in ECore-Codierung) + Instanzen
    - Operationen spezifiziert durch EMF-Transformationsregeln
- TIGER2 – *Transformation based Generation of Environments*
  - Neuimplementierung und Erweiterung von TIGER
  - ähnlicher Ansatz wie GMF:
    - Klassendiagramm als Modellsyntax
    - + Definition visueller (konkreter) Syntax
    - aber: Editoroperationen als EMF-Transformationsregeln
    - automatische Generierung von GEF-Editoren
  - MuvitorKit als Grundlange für TIGER2-Spezifikationseditor sowie für generierte Editoren

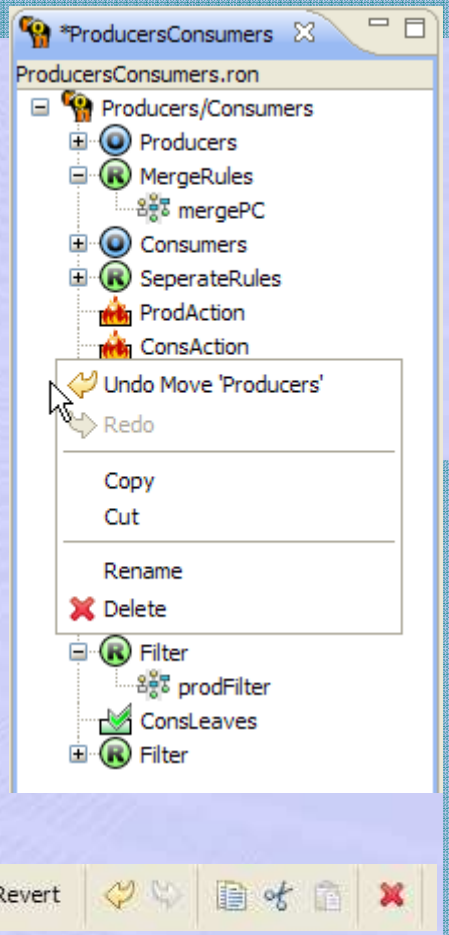






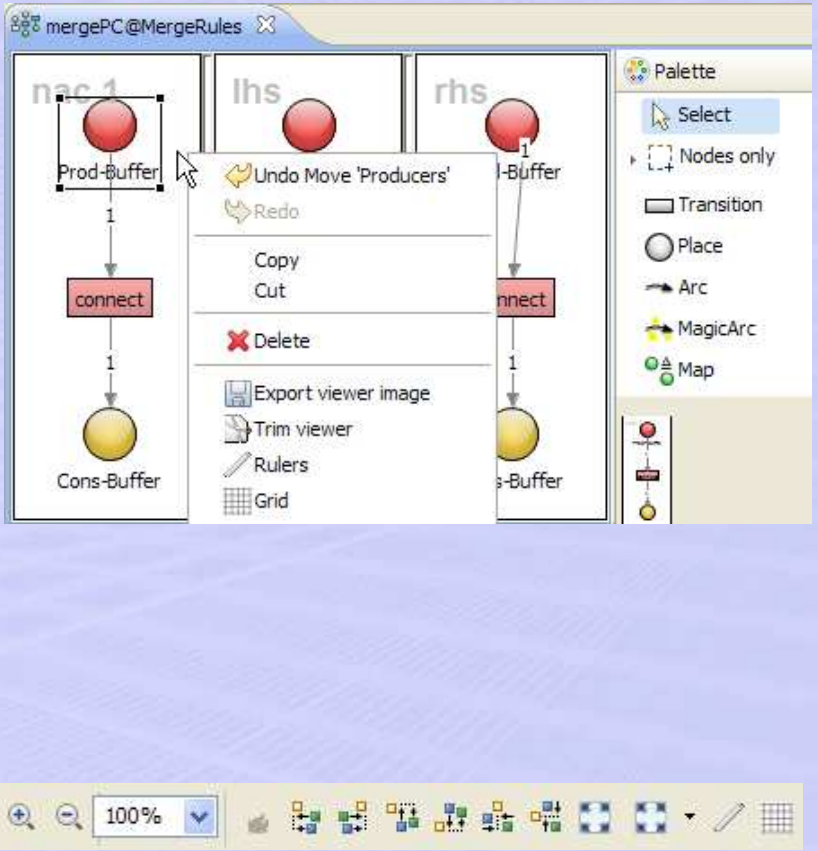
MuvitorTreeEditor im Detail

- zentrale abstrakte Hauptklasse
- erzeugt baumbasierten Editor (*TreeViewer*)
- Implementierung benötigt folgende Infos:
  - „Default“-Modell (für leere oder korrupte Dateien)
  - *EditPartFactory*, bildet Klassen aus EMF-Modell auf EditParts ab
  - *ContextMenuProvider* für den GEF-Baum mit optionalen eigenen *Actions*
  - Zuordnung von EMF-Modellklassen auf MuvitorKit-Views
  - \+ einige Plugin-Informationen in *plugin.xml* (z.B. Dateiendung)
- generische Editor-Actions
- verwaltet EMF-Modell, Workbench-Perspektive und graphische Views
- erlaubt Öffnen und Schließen von graphischen Modellansichten (MuvitorKit-Views)

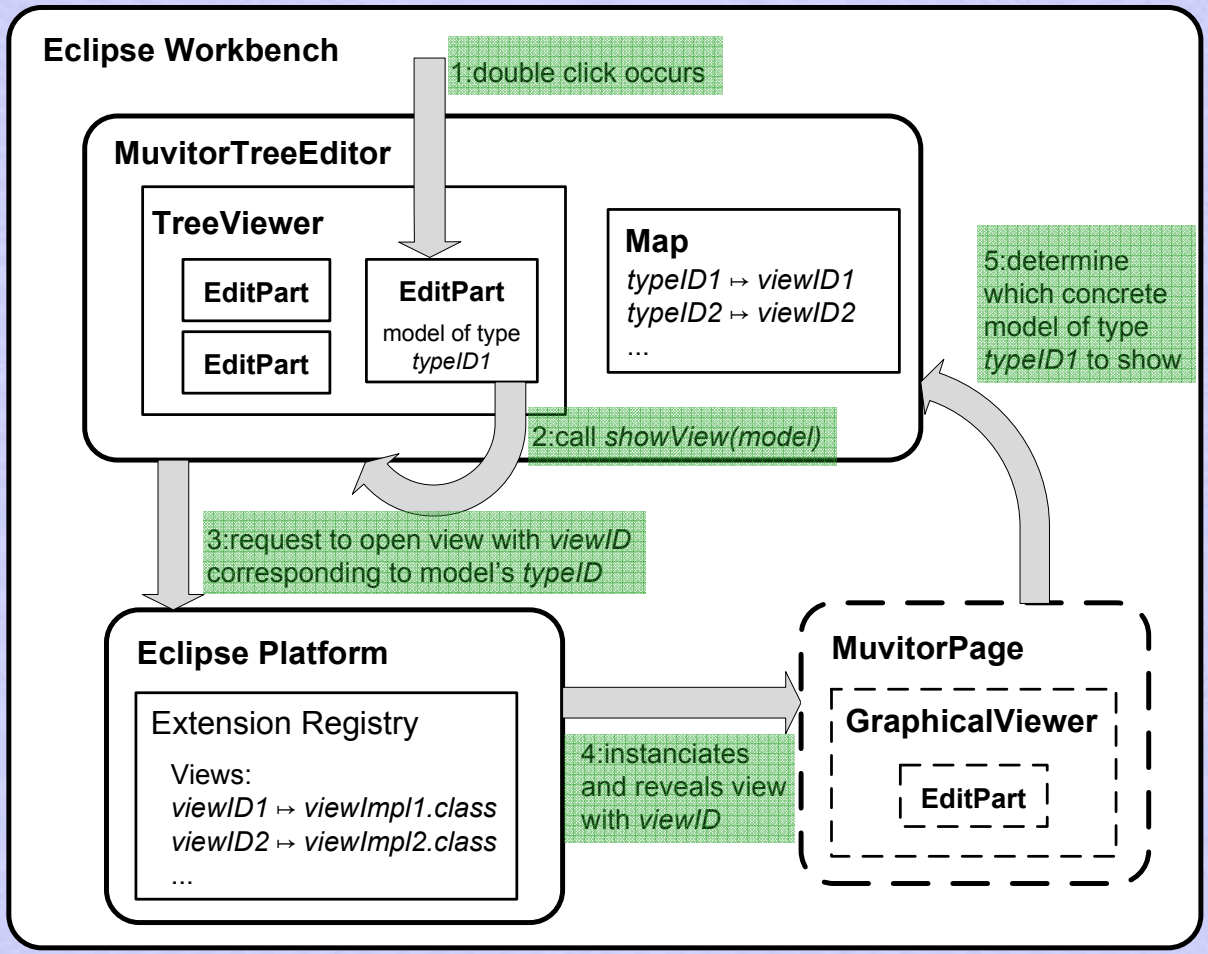


MuvitorPage im Detail

- kurz: Workbench-View mit *GraphicalViewers* (GEF)
- gekoppelt und synchronisiert mit *MuvitorTreeEditor*
- Implementierung ähnlich:
  - „Contents“-Modelle für die Viewers
  - *EditPartFactory*, bildet Klassen aus EMF-Modell auf *EditParts* ab
  - *ContextMenuProvider* mit optionalen eigenen *Actions*
  - GEF-*Palette* mit *Tools*
- generische Editor-Actions

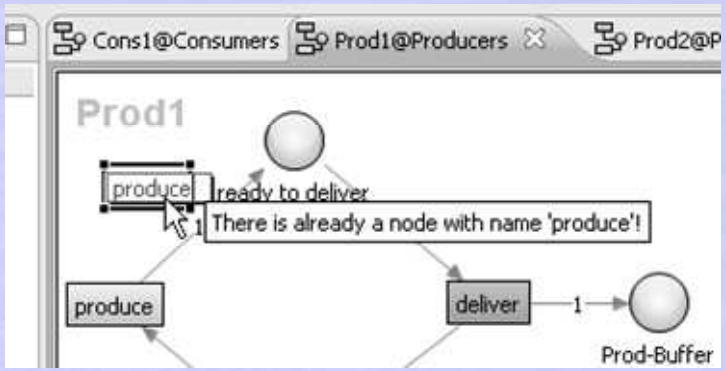


Struktur von MuvitorKit-Editoren



AdapterEditParts im Detail

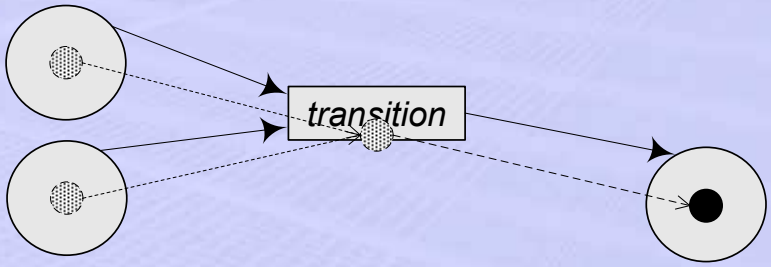
- Varianten: Tree, Graphical, Connection
- unterstützen EMF-*Adapter* (Listener für EMF-Notifications)
- reagieren auf „Open“-Request mit *showView(getModel())*
- generische *PropertySource* für EMF-Attribute im Properties View



- können MuvitorKits *IDirectEditPart* implementieren
  - bestimmt EMF Feature ID für Direct Edit, z.B. *EcorePackage.ENAMED_ELEMENT_NAME*
  - optional Validator für erlaubte Eingaben

Animationen in GEF

- GEF-Animation = Bewegte *Figures*
- GEF liefert schon Klassen *Animator* und *Animation*
  - wozu ein Extra-Package?
- *Animator/Animation* bewegen *Figures* auf direktem Pfad zu neugesetzter Location
  - also: „Alle Figures die gerade umgesetzt wurden, auf gerader Linie zum Ziel animieren“
- funktionale, modellorientierte Spezifikation wünschenswert
  - z.B schalten von Transitionen in Petrinetzen: „Je ein Token von Pre-Places zur Transition bewegen und anschließend je eines auf alle Post-Places“



Animierte Commands mit MuvitorKit

- *AnimatingCommand Commands* erlaubt Spezifikationen wie „Bewege model1 über Punkt (20,20) zu model2, während model3 direkt zu model4 läuft und zwischendurch 4x größer wird“
  - unabhängig von *Figures*, *Views*, Modellzustand
  - ermittelt automatisch zu animierende *Figures* in allen MuvitorKit-Views ⇒ parallele Animation in verschiedenen Viewers
- erlaubt nichtlineare Pfade
  - Sinuskurven, Kreise, Ellipsen, zufälliges Springen und selbstdefinierte
- *Figures* können während Animation unabhängig von anderen gezoomt werden
- können einfach mit modellverändernden verknüpft werden ⇒ automatisches Undo von Animationen

Demo: RONEditor

Schluss

- Programmier-Framework MuvitorKit
  - abstrakte Klassen
    - schnelle Implementierung graphischen Eclipse-Editoren mit vielen generischen Features
    - einheitliches Multi-View-Konzept für heterogene visuelle Modelle
      - zentraler baumbasierter Editor mit speziellen Views und EditParts
  - nutzt Eigenschaften von EMF-Modellen aus
  - funktionsorientierte Animationen als Commands
  - Sammlung weiterer Hilfsklassen
  - fortwährende Weiterentwicklung nach Bedarf
  - RONEditor als Referenzimplementierung
- in Vorbereitung: ausführliches Schritt-für-Schritt-Tutorial bis zum lauffähigen minimalen Editor als Orientierungshilfe