

Modellbasierte Softwareentwicklung (MODSOFT)

Part II

Domain Specific Languages

Graphical Notations

Prof. Joachim Fischer /

Dr. Markus Scheidgen / Dipl.-Inf. Andreas Blunk

{fischer,scheidge,blunk}@informatik.hu-berlin.de

LFE Systemanalyse, III.310

Agenda

prolog
(1 VL)

Introduction: languages and their aspects, modeling vs. programming, meta-modeling and the 4 layer model

○
(2 VL)

Eclipse/Plug-ins: eclipse, plug-in model and plug-in description, features, *p2*-repositories, *RCPs*

1.
(2 VL)

Structure: *Ecore*, *genmodel*, working with generated code, constraints with *Java* and *OCL*, *XML/XMI*

➔ 2.
(3 VL)

Notation: Customizing the tree-editor, textural with *XText*, graphical with *GEF* and *GMF*

3.
(4 VL)

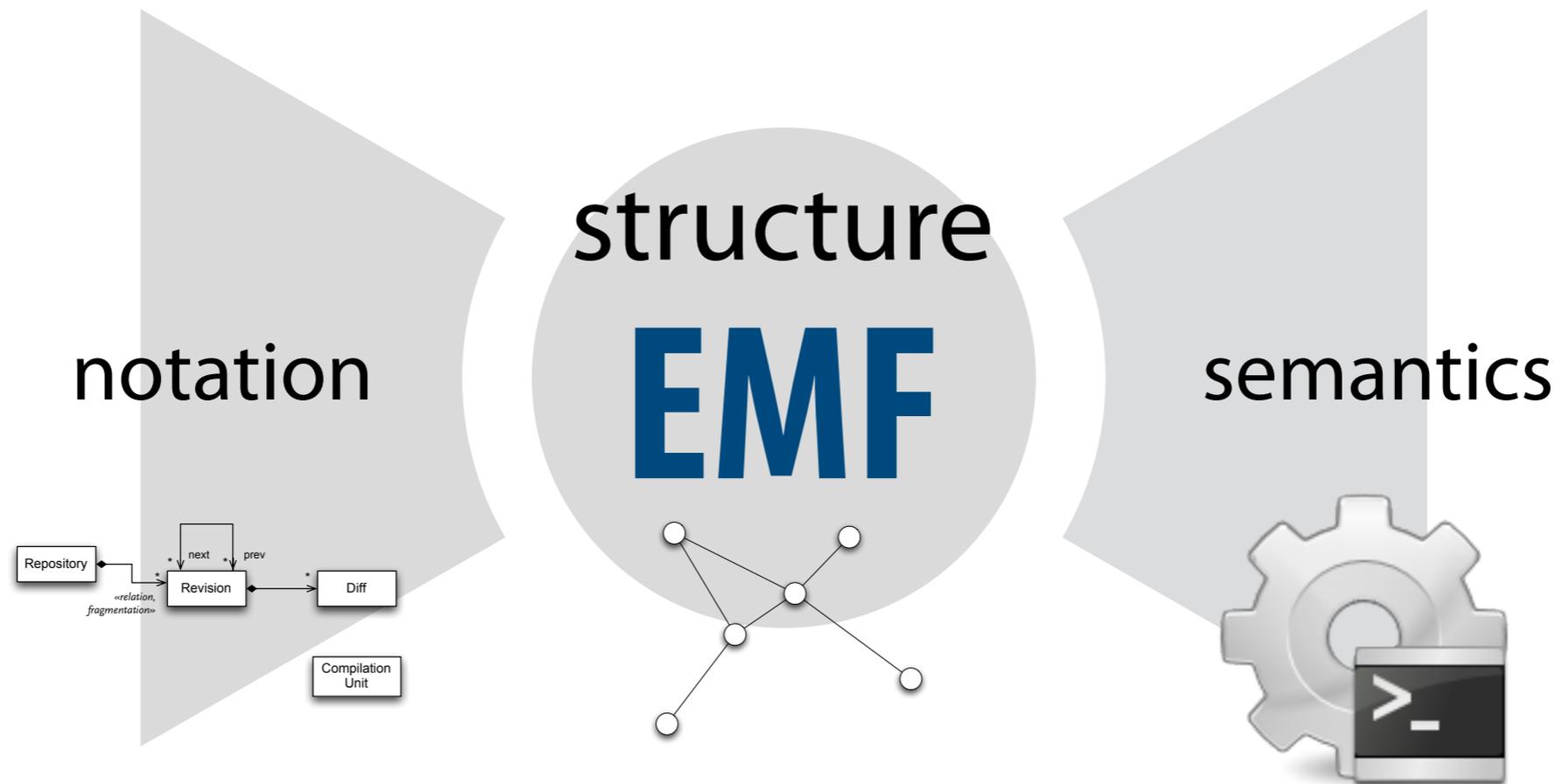
Semantics: interpreters with *Java*, code-generation with *Java* and *XTend*, model-transformations with *Java* and *ATL*

epilog
(2 VL)

Tools: persisting large models, model versioning and comparison, model evolution and co-adaption, modular languages with *XBase*, *Meta Programming System (MPS)*

Previously on MODSOFT

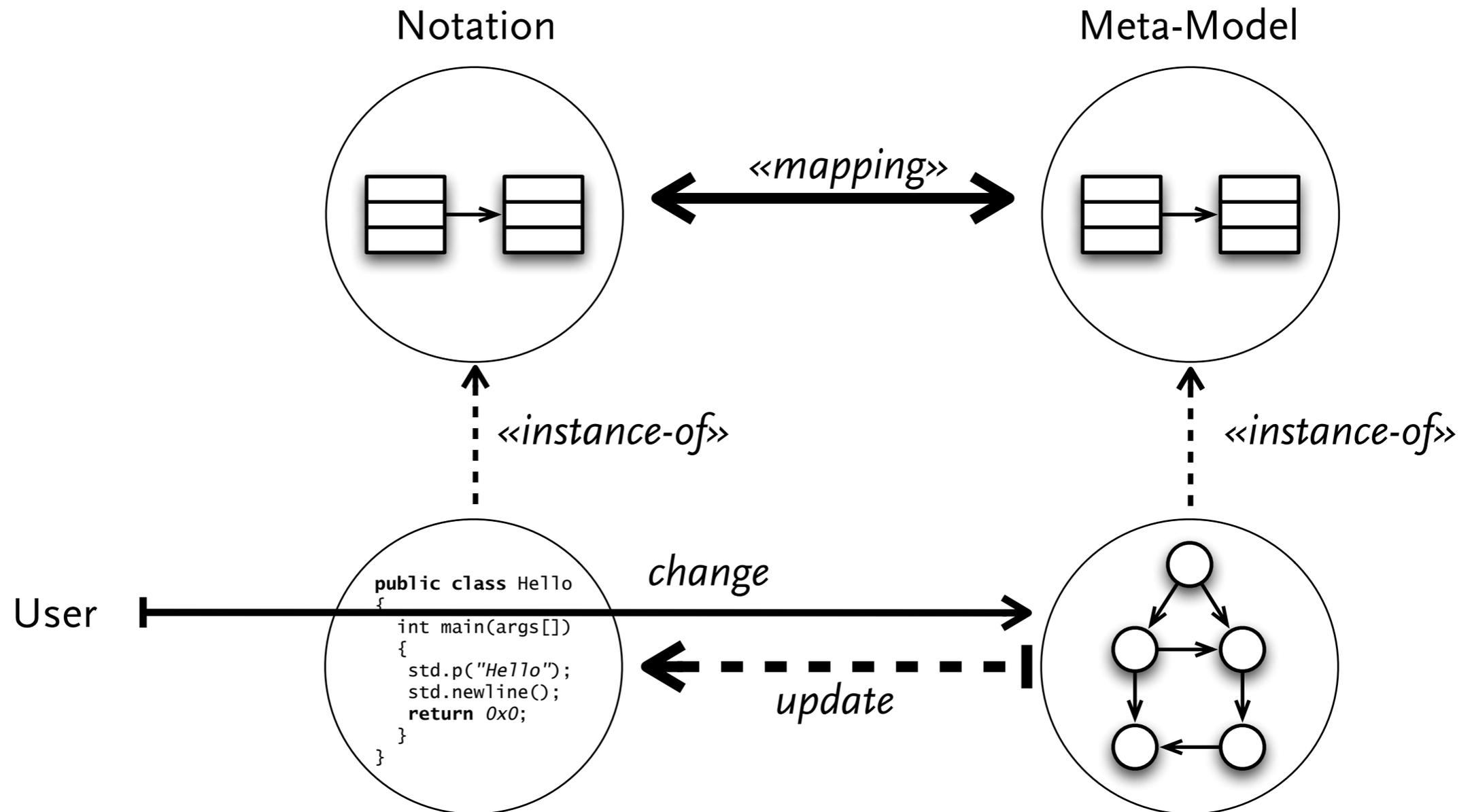
Eclipse Modeling Framework



Graphical Notations

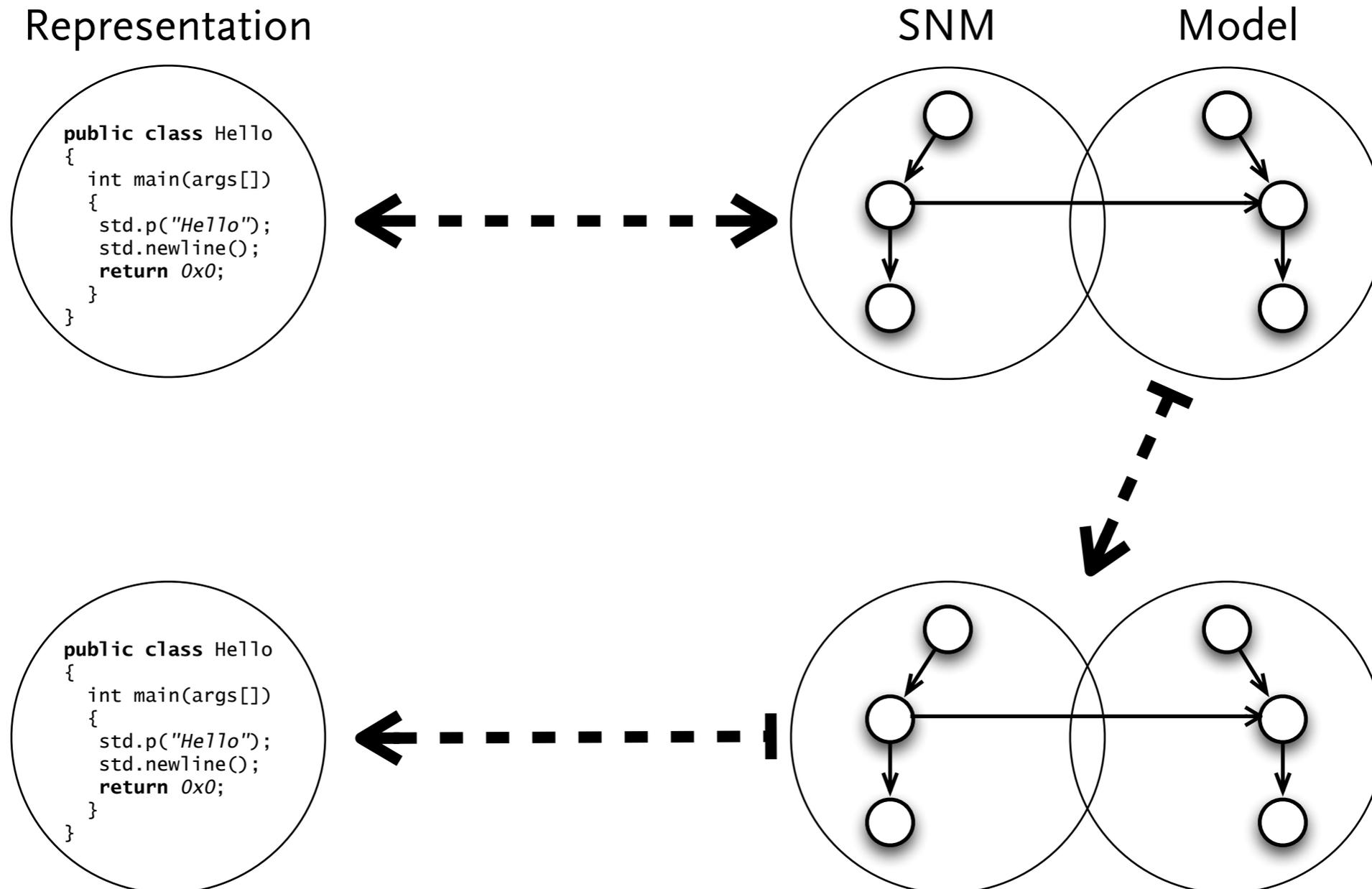
Introduction

Model-View-Controller



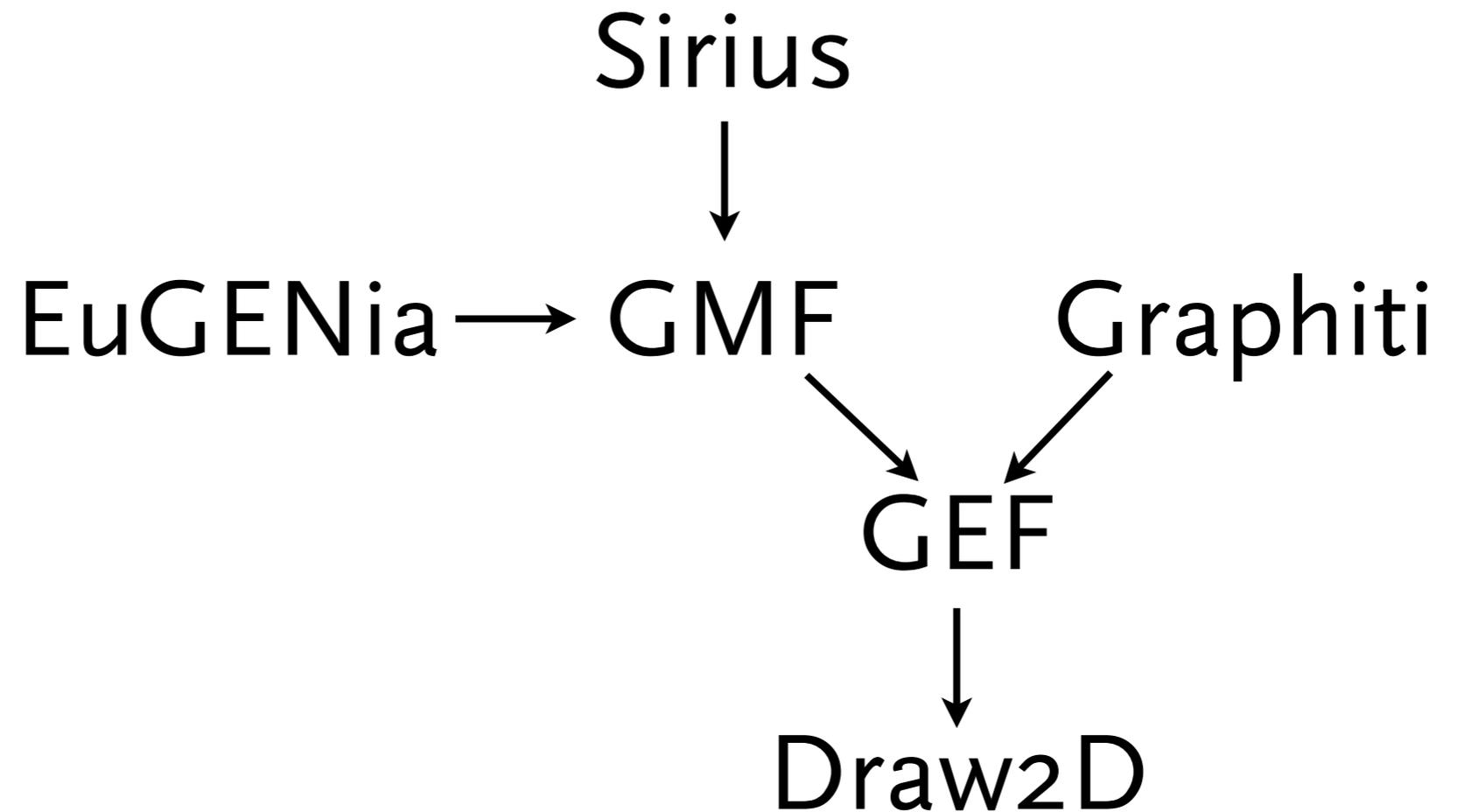
- ▶ the reasonable example of bijective mappings, if secondary notation is part of the model

Representation, Secondary Notation Model + Model



Frameworks

- ▶ Draw2D
- ▶ GEF
- ▶ Graphiti
- ▶ GMF
- ▶ Sirius



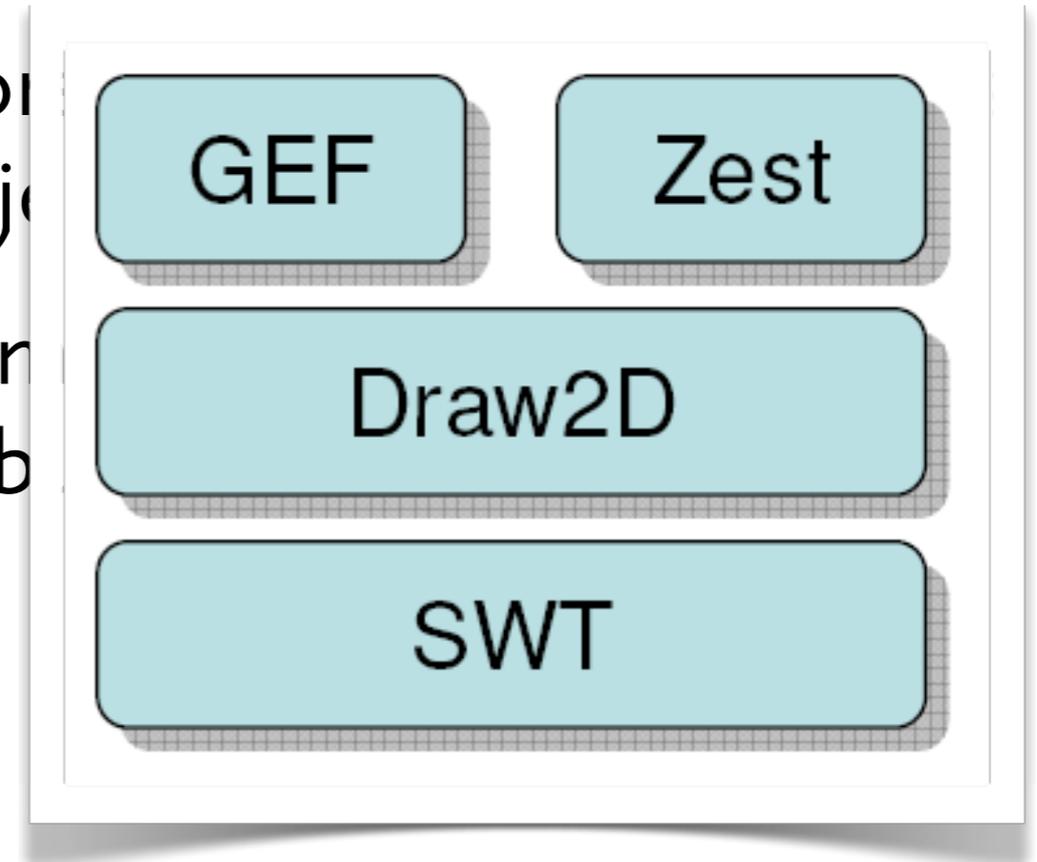
Graphical Editing Framework (GEF)

Graphical Editing Framework

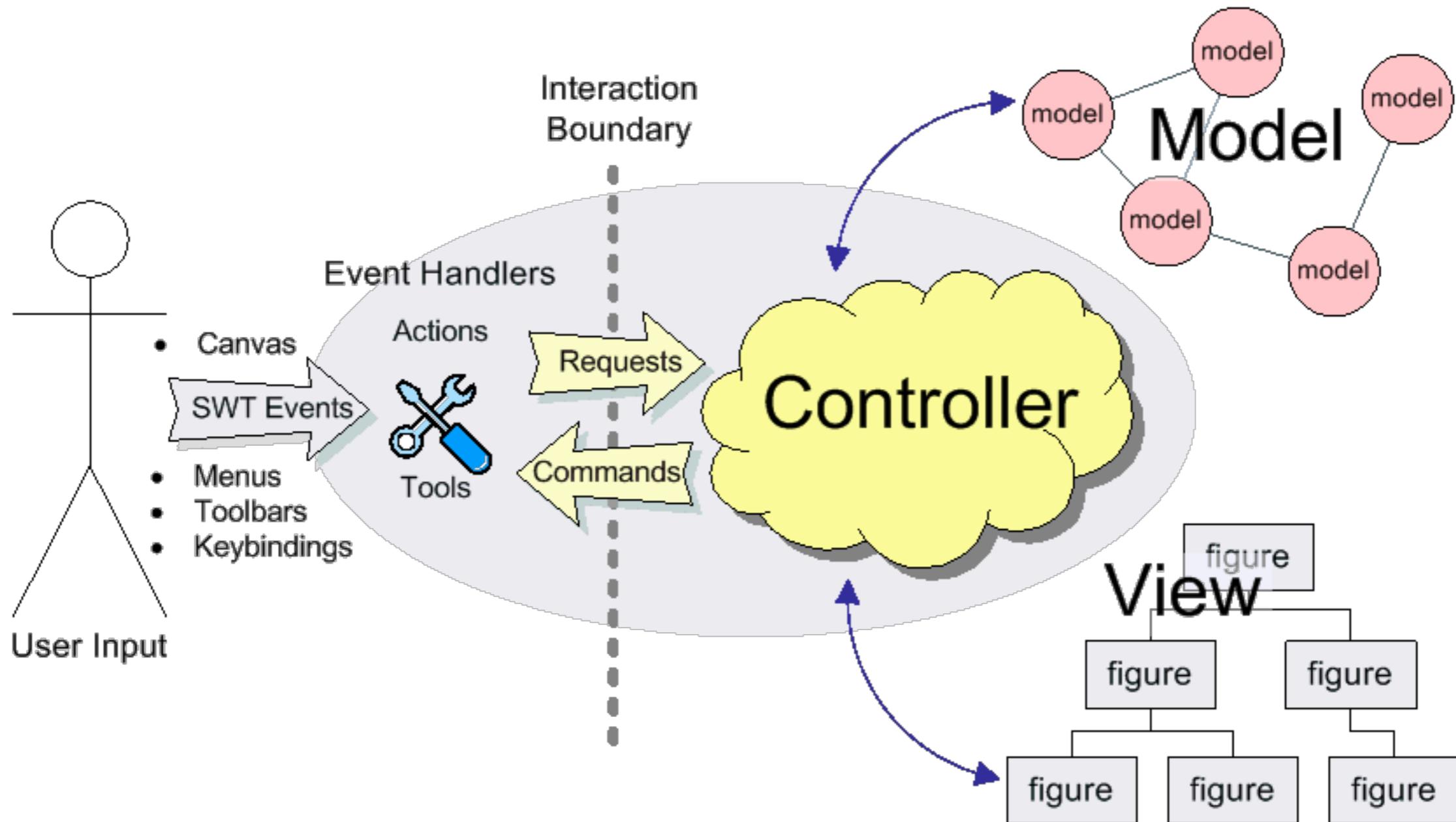
- ▶ Allows you to build MVC-based editors with arbitrary POJOs as models and lightweight graphical objects (Draw2D) as views.
- ▶ Embedded into the eclipse editor framework, workspace integration (copy paste, undo/redo, buttons, menus, outlines, properties, etc.)
- ▶ Abstractions for
 - Rulers & Guides, Grid
 - Snap-to-Geometry, Centered Resize, Match Size
 - Constraint Move and Resize, Cloning, Panning
 - Palette View, Flyout Palette, Palette Stacks
 - Shortest Path Connection Routing

Graphical Editing Framework

- ▶ Allows you to build MVC-based editor models and lightweight graphical objects
- ▶ Embedded into the eclipse editor framework integration (copy paste, undo/redo, block properties, etc.)
- ▶ Abstractions for
 - Rulers & Guides, Grid
 - Snap-to-Geometry, Centered Resize, Match Size
 - Constraint Move and Resize, Cloning, Panning
 - Palette View, Flyout Palette, Palette Stacks
 - Shortest Path Connection Routing



Model View Controller (MVC) Pattern

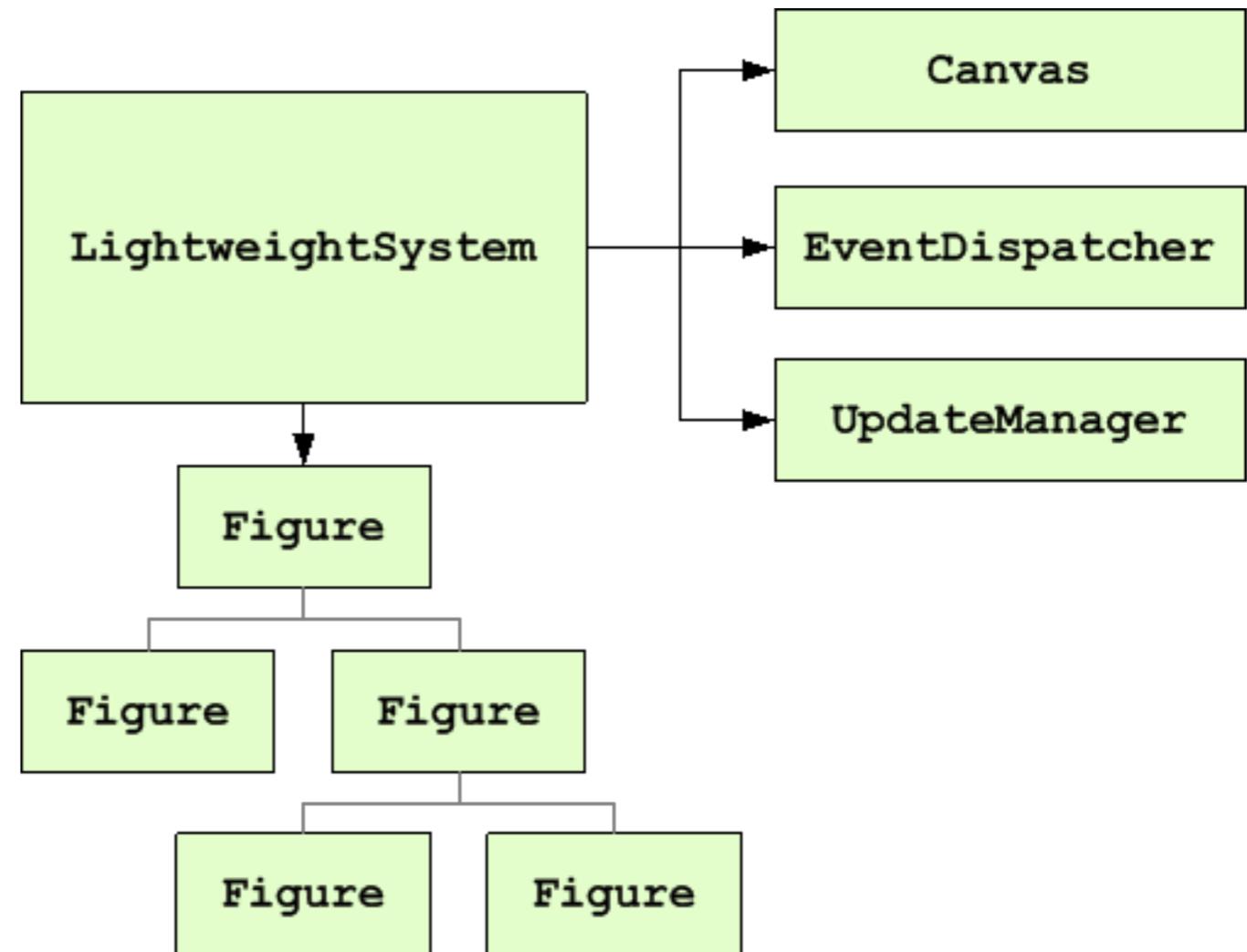


Draw2D

► LightweightSystem

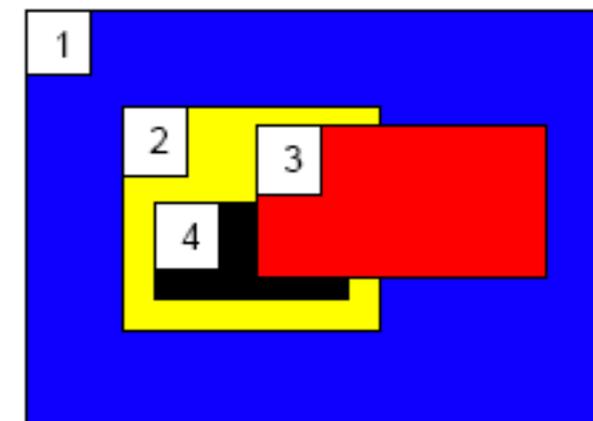
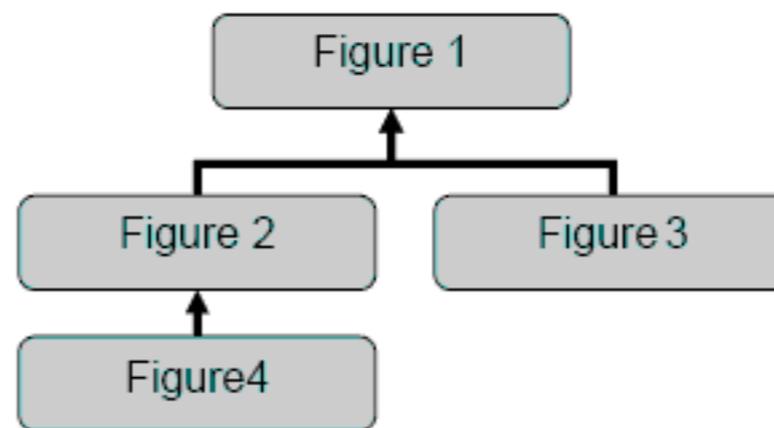
- associates a figure composition with an SWT Canvas
- hooks listeners for most SWT events, and forwards most of them to an EventDispatcher
- which translates them into events on the appropriate figure

► Paint events are forwarded to the UpdateManager, which coordinates painting and layout.

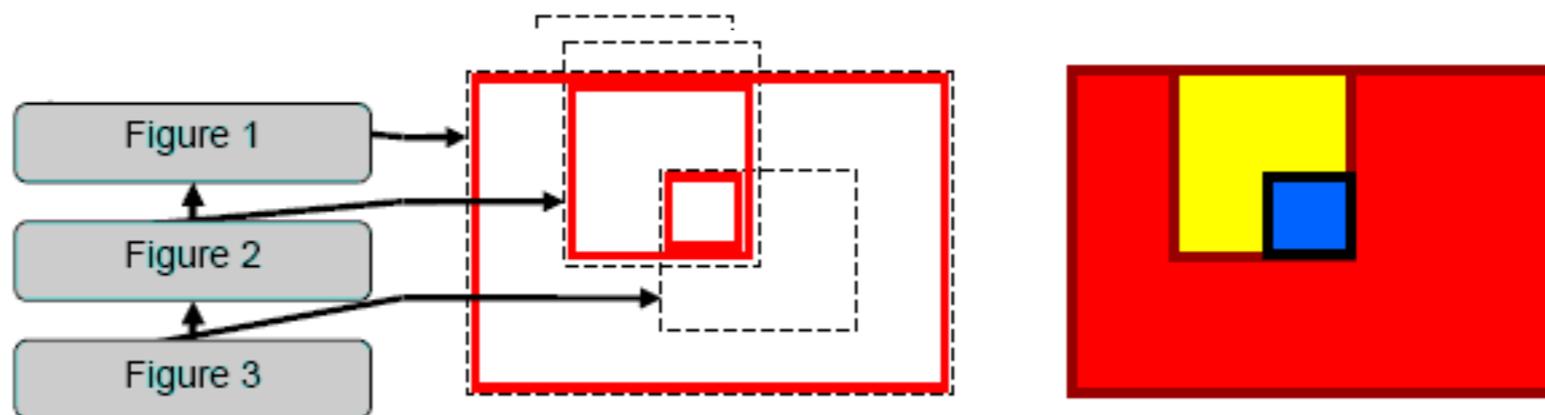


Draw2D – Painting

- ▶ Z-Order
- ▶ Clipping
- ▶ Hit detection

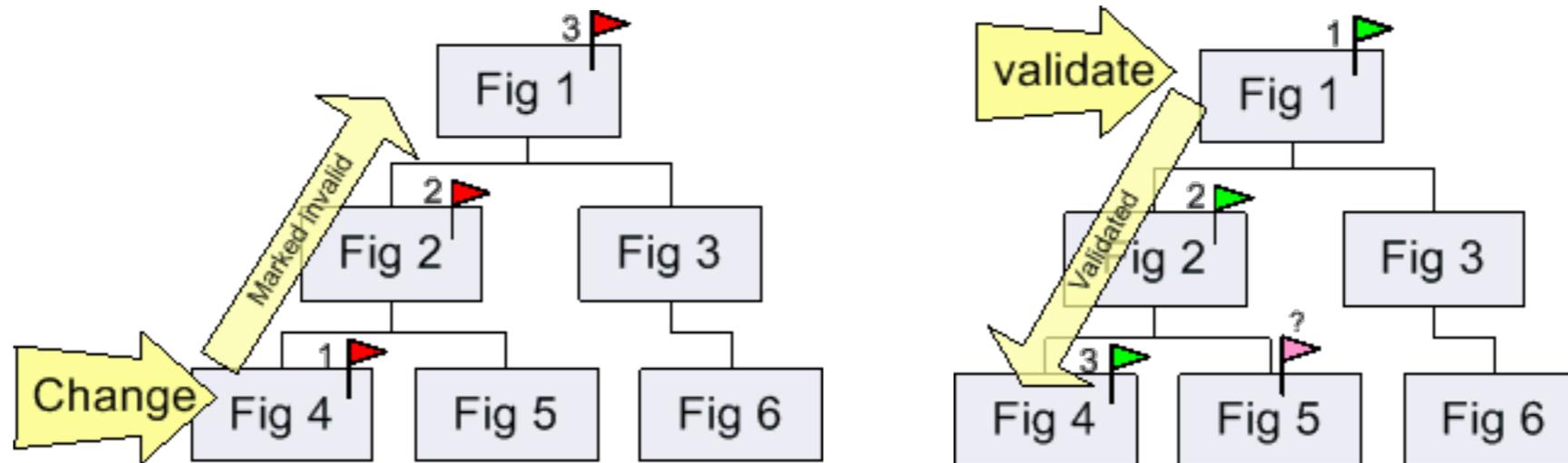


This picture shows a tree of figures and its graphical representation if each figure is painted as a full rectangle.



The Bounds of the figures are represented as dash lines, each figure is painted as a full rectangle with a black border, the clipping area associated with each figure is represented as a red line.

Draw2D – Other Abstractions



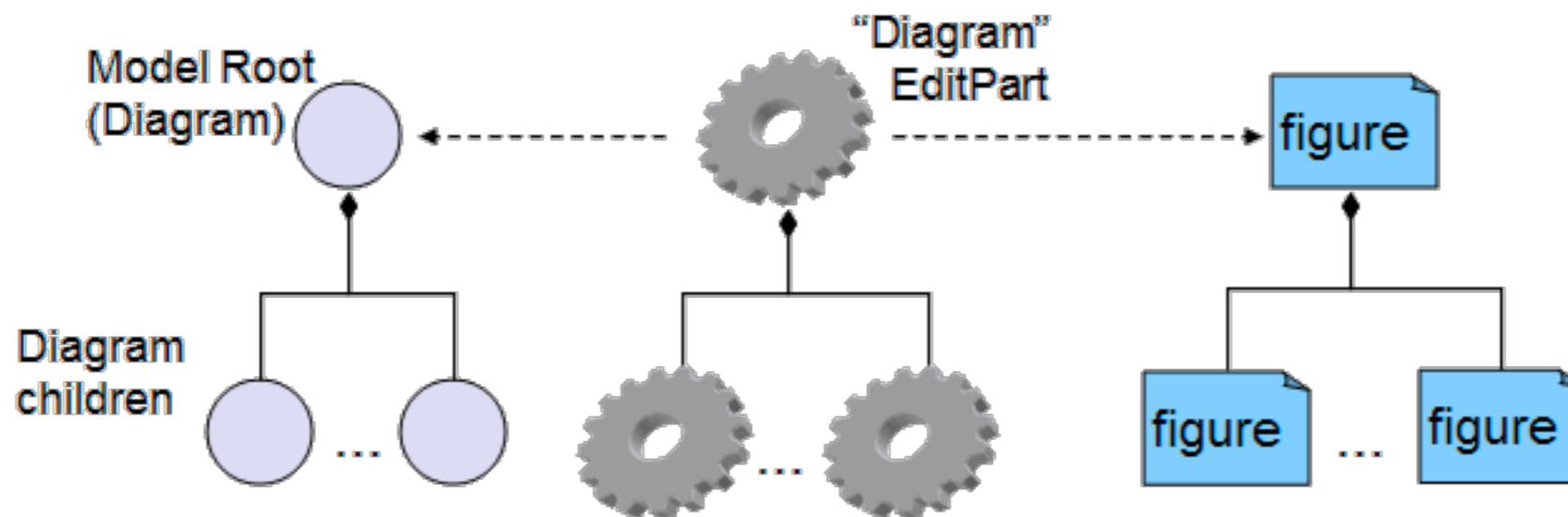
- ▶ Layout
 - invalidation, update manager
- ▶ Connections and routing
- ▶ Coordinate systems

GEF – Core Concepts

- ▶ GraphicalEditor: for building an Editor.
- ▶ Viewer: foundation for displaying and editing your model.
- ▶ EditPart: elements inside a viewer.
- ▶ EditPolicy: restricts possible combination of elements
- ▶ Tool: interprets user input; represents mode.
- ▶ Palette: displays available tools.
- ▶ CommandStack: stores Commands for undo/redo.
- ▶ EditDomain: ties everything together.

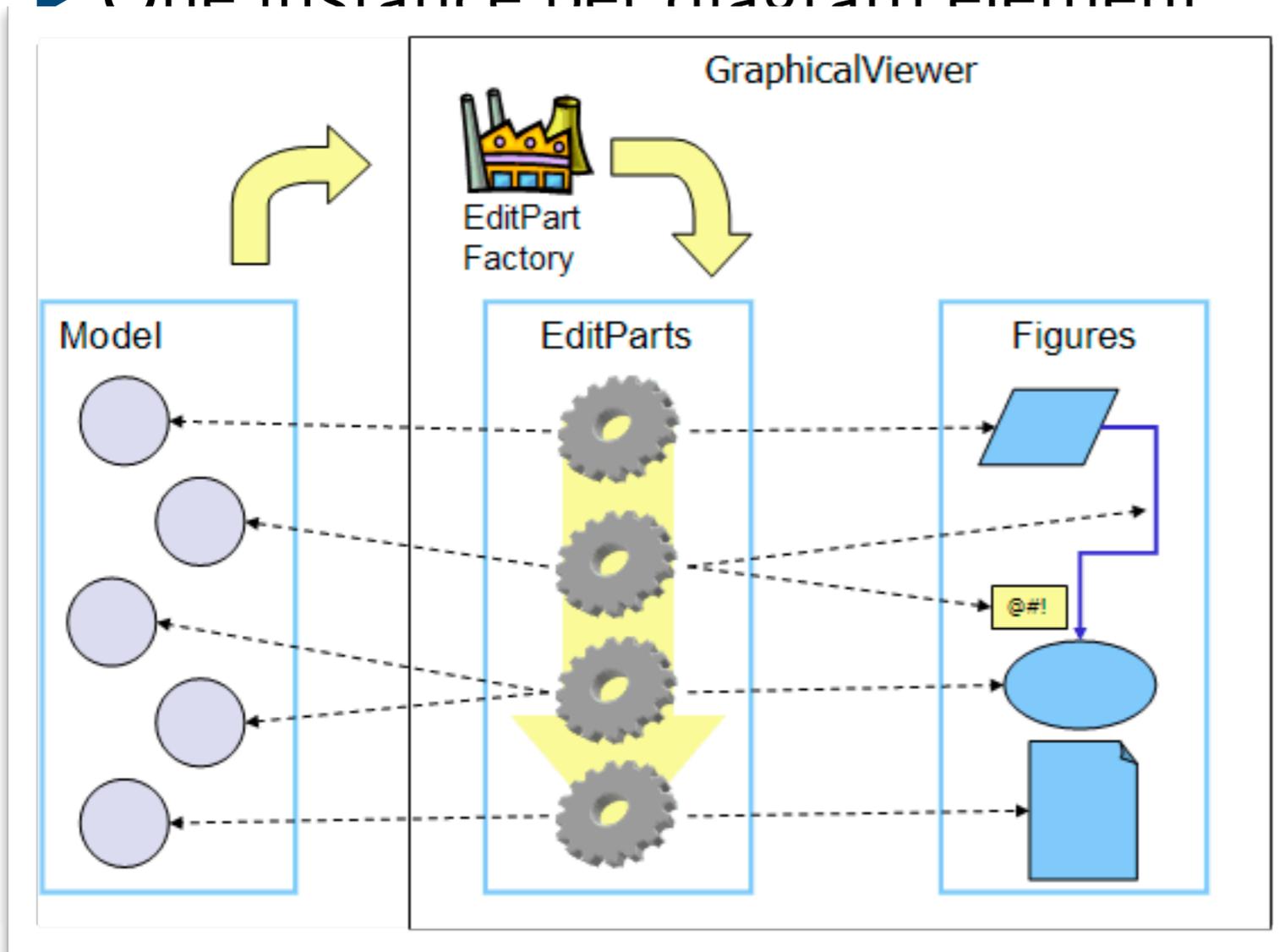
EditParts

- ▶ Controller, they associate models and views (EditPartViewer)
- ▶ One class (EditPart) per diagram element type
- ▶ One instance per diagram element
- ▶ Exception are links: have a figure, but are managed by source and target editor parts

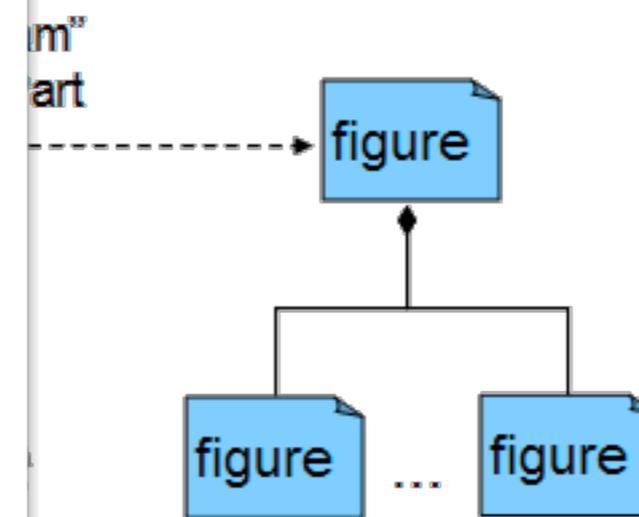


EditParts

- ▶ Controller, they associate models and views (EditPartViewer)
- ▶ One class (EditPart) per diagram element type
- ▶ One instance per diagram element

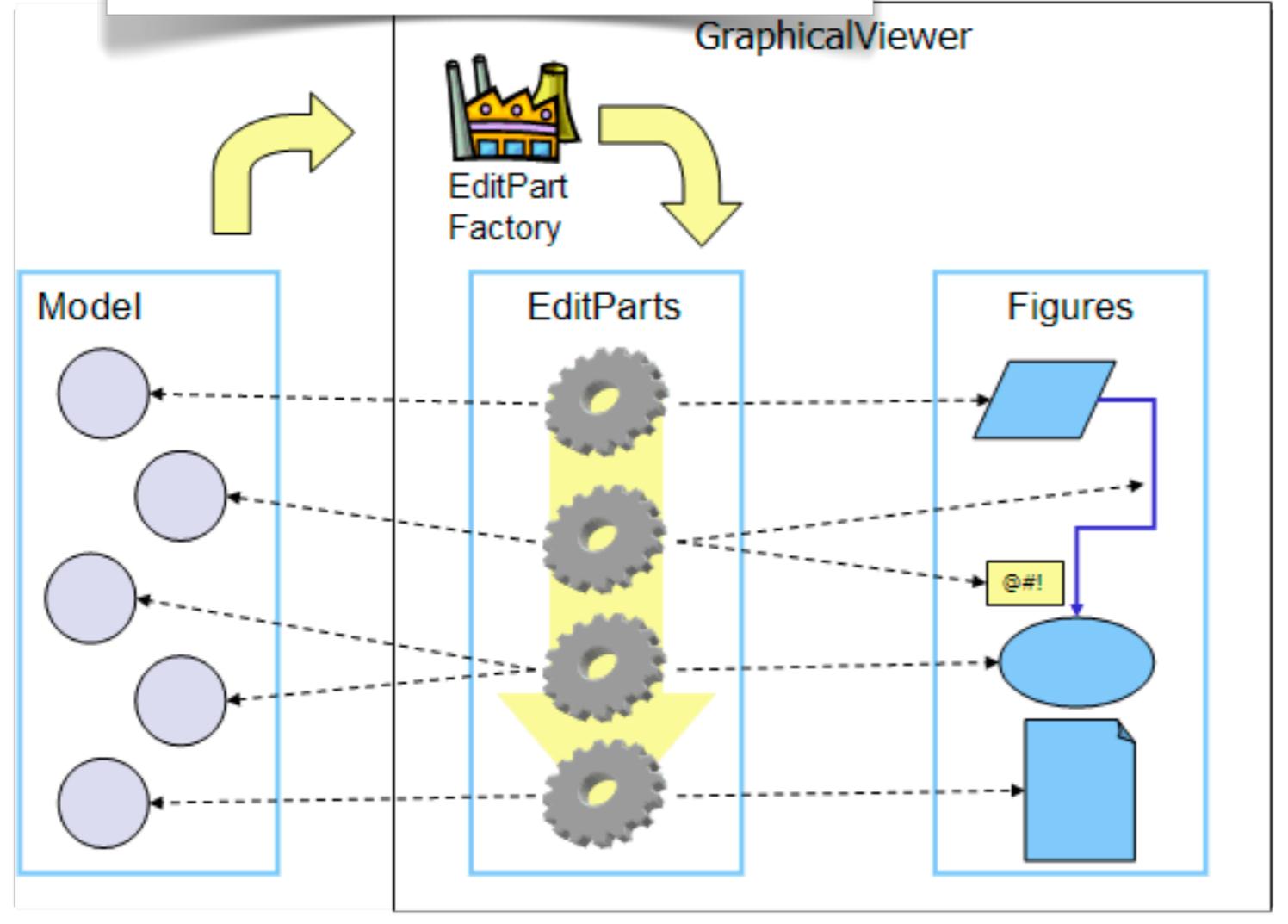
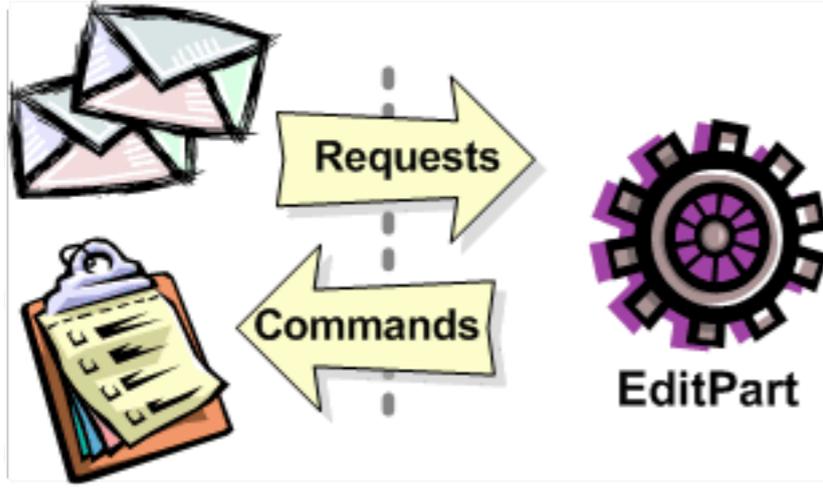


are managed by source

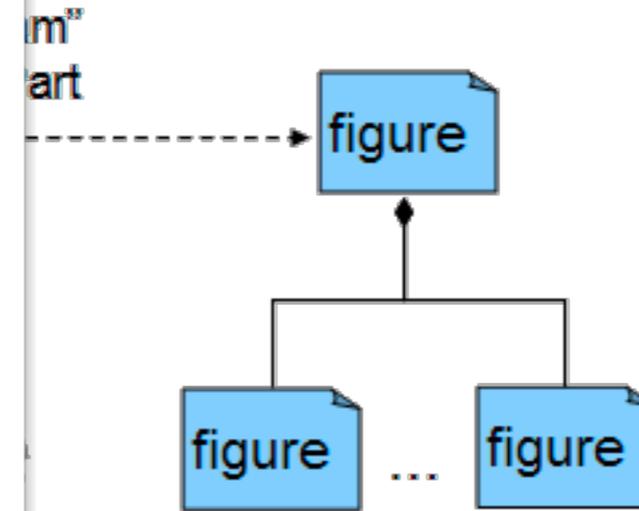


EditParts

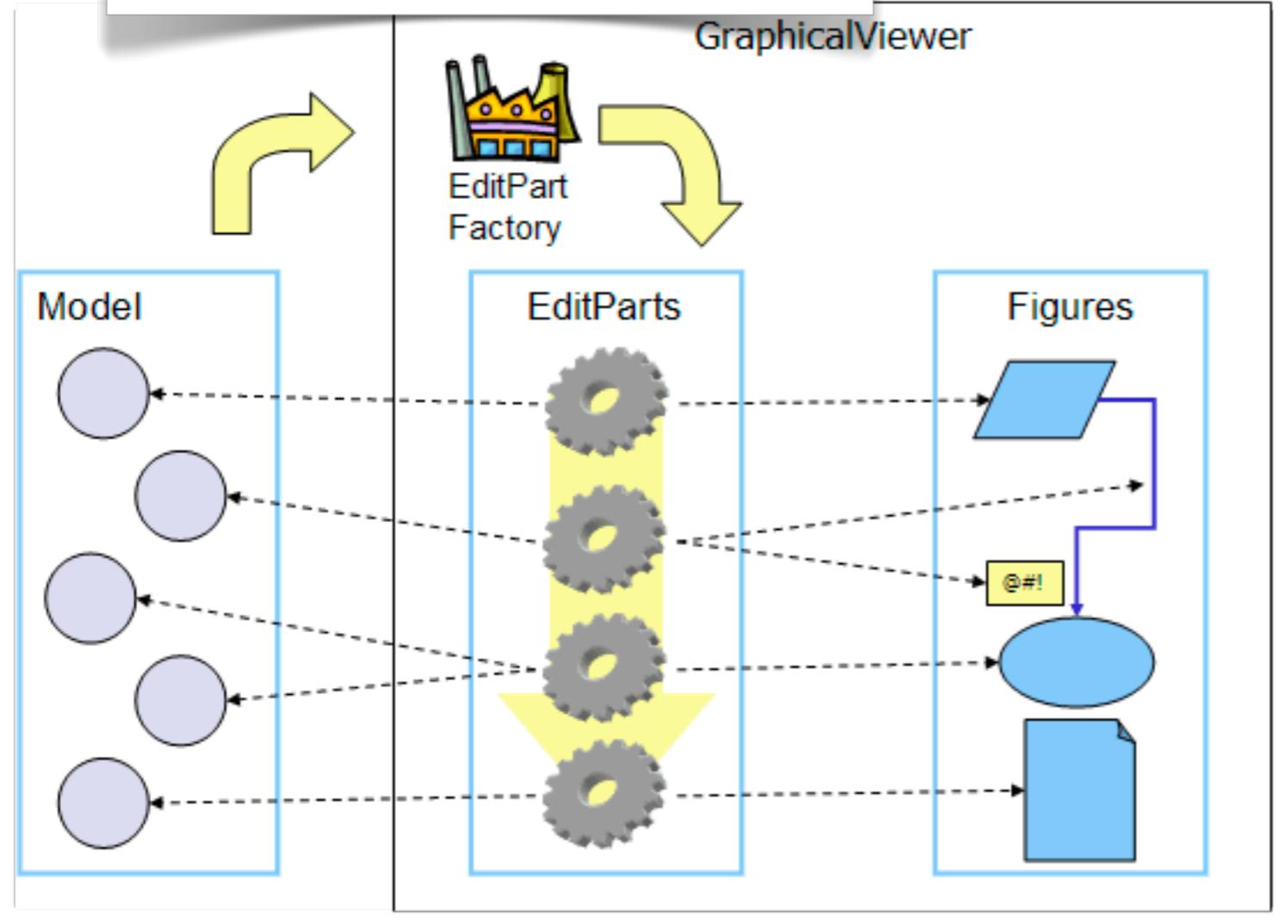
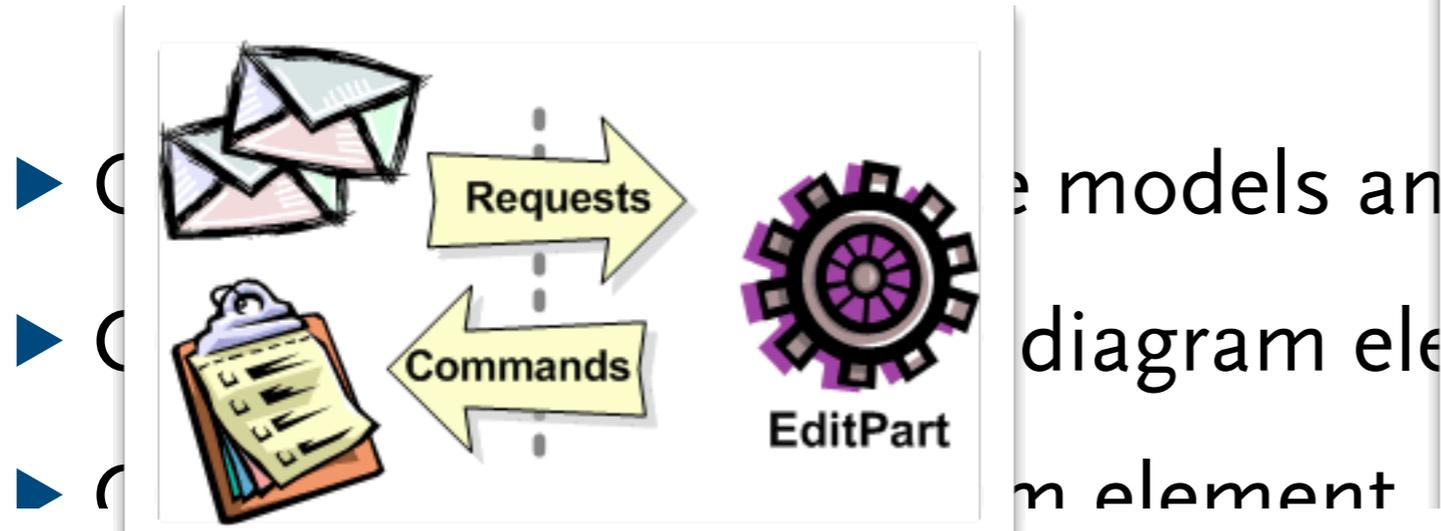
- ▶ Create models and views (EditPartViewer)
- ▶ Create diagram element type
- ▶ Create diagram element



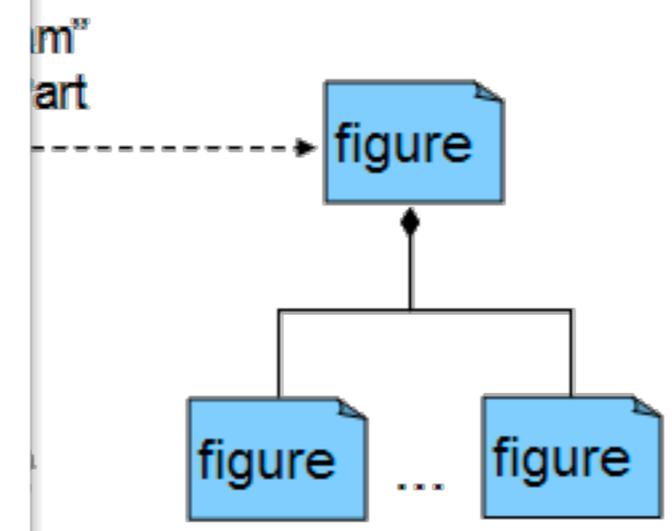
are managed by source



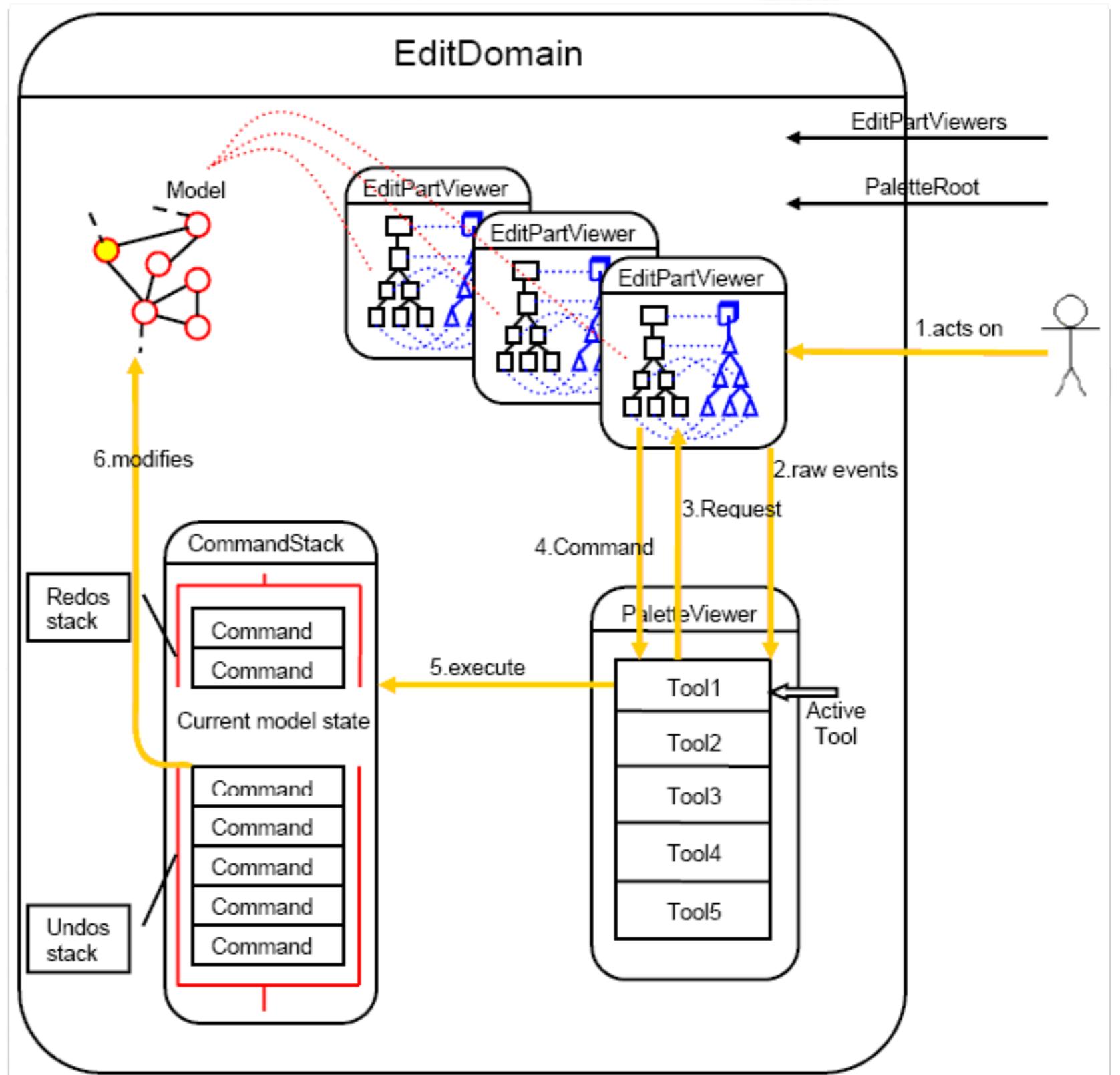
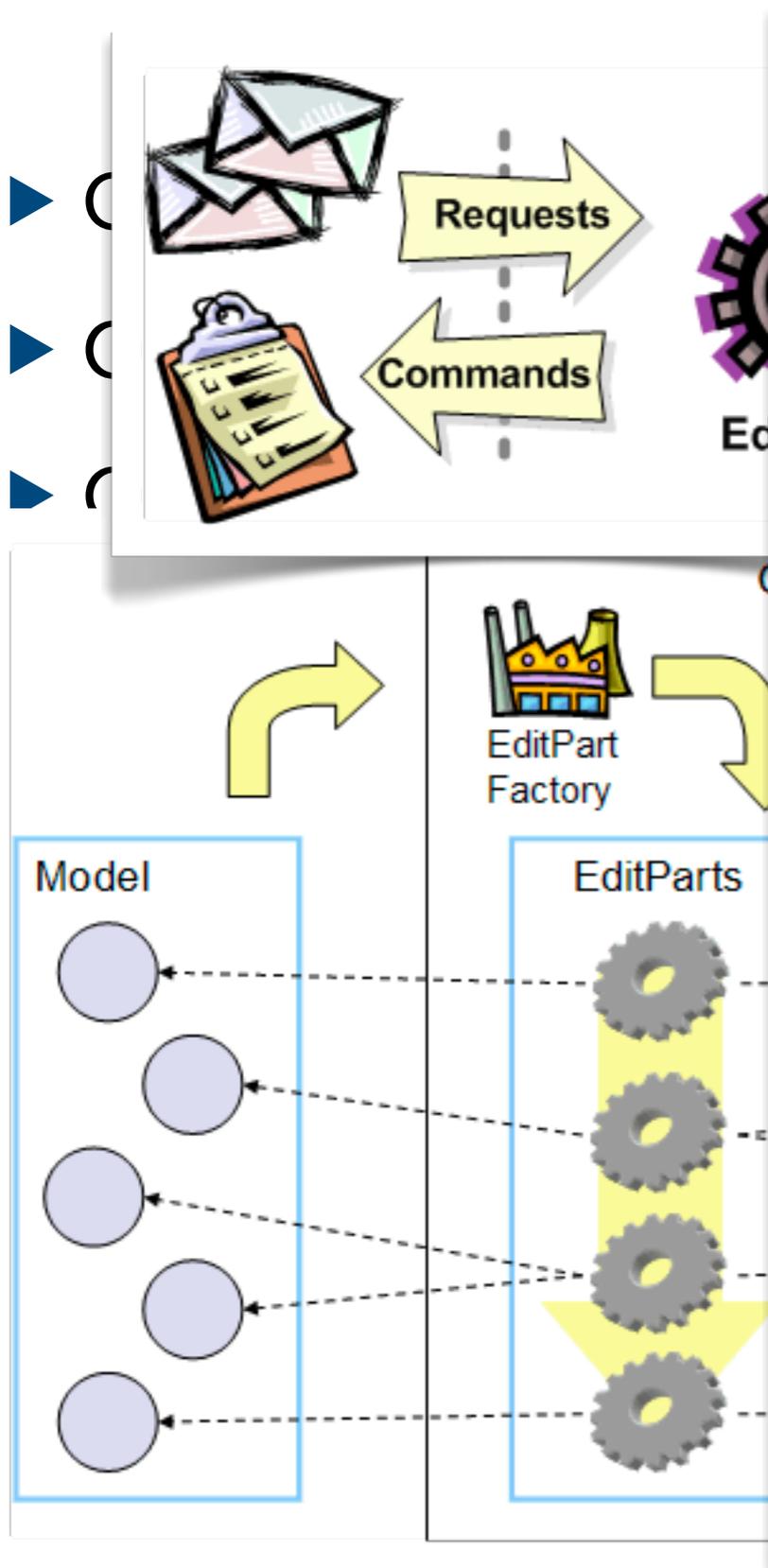
EditParts



are managed by source

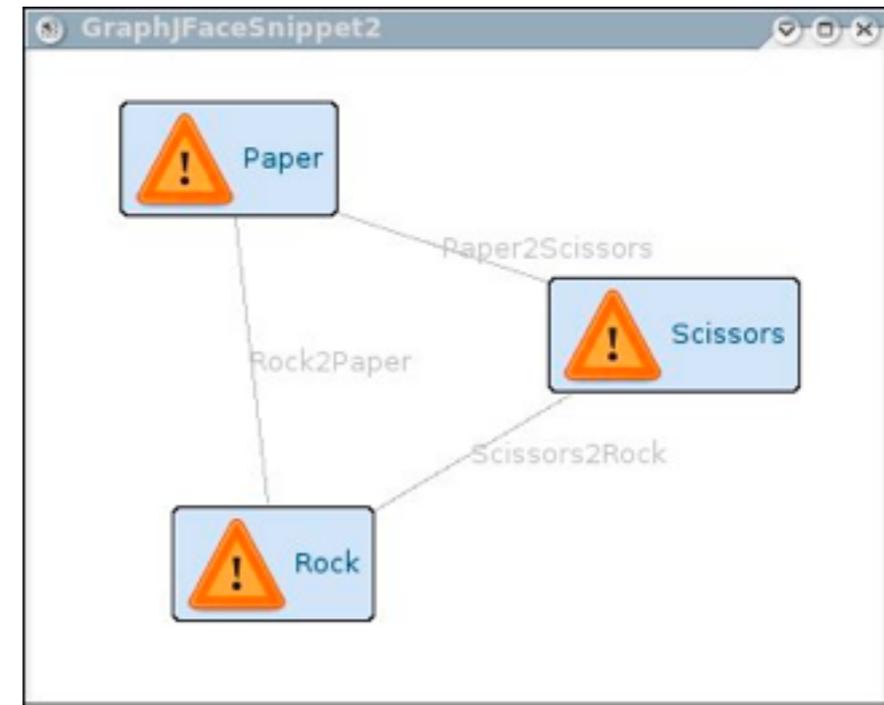
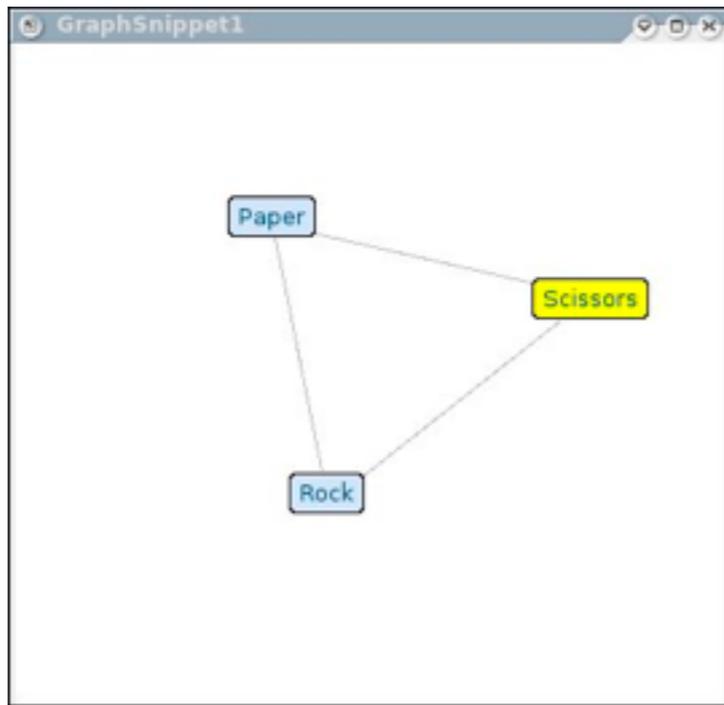


EditParts



Zest

- ▶ Zest is visualization toolkit for Eclipse. The primary goal of Zest is to make graph based programming easy. Using Zest, Graphs are considered SWT Components which have been wrapped using standard JFace viewers. This allows developers to use Zest the same way they use JFace Tables, Trees and Lists.



Graphical Modeling Framework (GMF)

Graphical Modeling Framework (GMF)

▶ GMF Runtime

- combines EMF and GEF, EMF as models in GEF's MVC implementation
- provides consistent representations of models
 - ◆ canvas composition
 - ◆ associations
 - ◆ composition in diagram elements (e.g. attributes in classes, multiplicities at associations)
- adds complexity and functionality, all GEF features fully retained

▶ GMF Tools

- allows to describe editor with several models (i.e. editor descriptions)
- generates GMF runtime code from these models
- hides GMF runtime's and GEF's complexity, but introduces heavy limitations

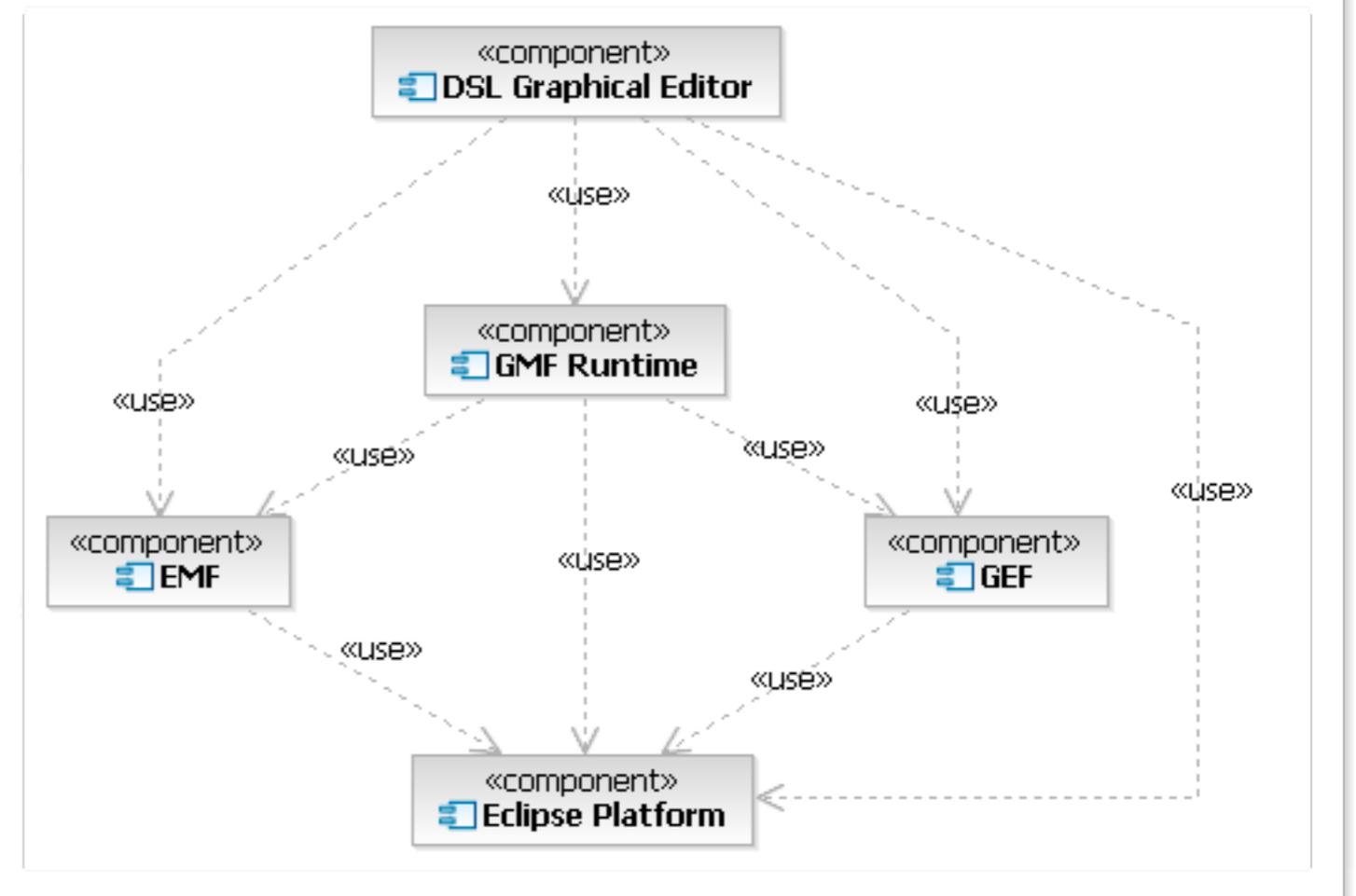
Graphical Modeling Framework (GMF)

▶ GMF Runtime

- combines EMF and GEF, EMF
- provides consistent represent
- ◆ canvas composition
- ◆ associations
- ◆ composition in diagram elem
associations)
- adds complexity and functiona

▶ GMF Tools

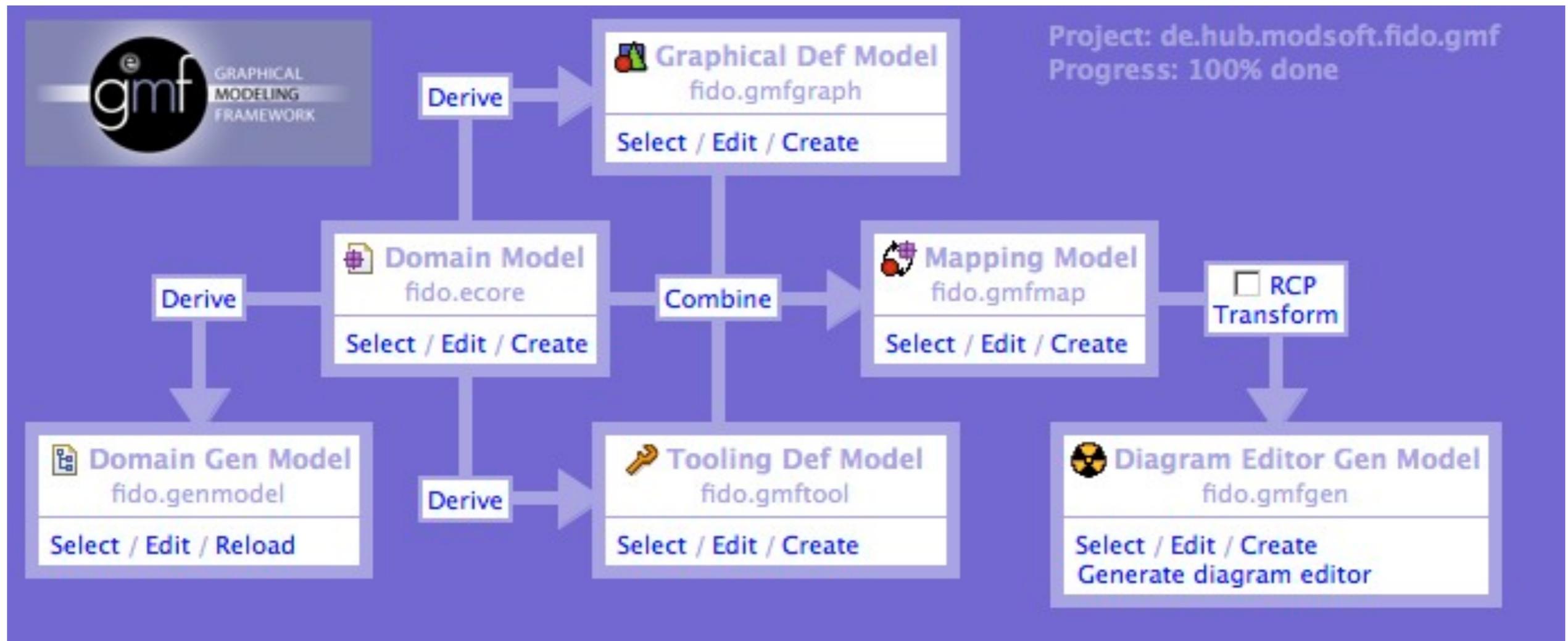
- allows to describe editor with several models (i.e. editor descriptions)
- generates GMF runtime code from these models
- hides GMF runtime's and GEF's complexity, but introduces heavy limitations



GMF Tools

- ▶ several editors (tree editors)
 - graph definition
 - tool definition
 - mapping definition
 - genmodel definition
- ▶ wizards for all editors
- ▶ dashboard

GMF Tools – Workflow



▶ demo time

Emphatic

- ▶ Generate GMF Tooling models from meta-model annotations

```

@namespace(uri="filesystem", prefix="filesystem")
package filesystem;

@gmf.diagram
class Filesystem {
    val Drive[*] drives;
    val Sync[*] syncs;
}

class Drive extends Folder {

}

class Folder extends File {
    @gmf.compartment
    val File[*] contents;
}

class Shortcut extends File {
    @gmf.link(target.decoration="arrow", style="dash")
    ref File target;
}

@gmf.link(source="source", target="target", style="dot", width="2")
class Sync {
    ref File source;
    ref File target;
}

@gmf.node(label = "name")
class File {
    attr String name;
}

```

ic

om meta-model

```

@namespace(uri="filesystem", prefix="filesystem")
package filesystem;

@gmf.diagram
class Filesystem {
    val Drive[*] drives;
    val Sync[*] syncs;
}

class Drive extends Folder {
}

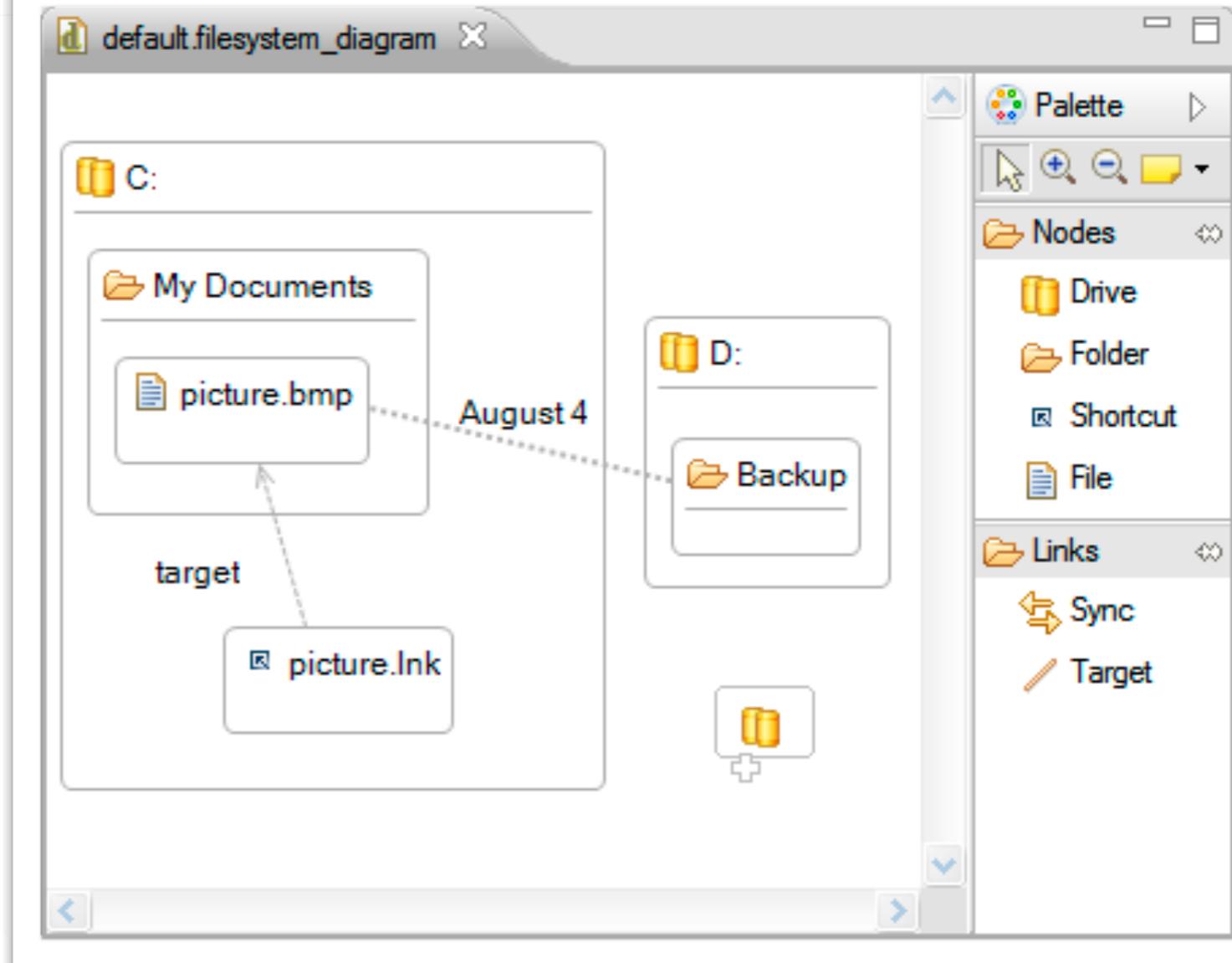
class Folder extends File {
    @gmf.compartment
    val File[*] contents;
}

class Shortcut extends File {
    @gmf.link(target.decoration="arrow", style="dash")
    ref File target;
}

@gmf.link(source="source", target="target", style="dot", width="2")
class Sync {
    ref File source;
    ref File target;
}

@gmf.node(label = "name")
class File {
    attr String name;
}

```

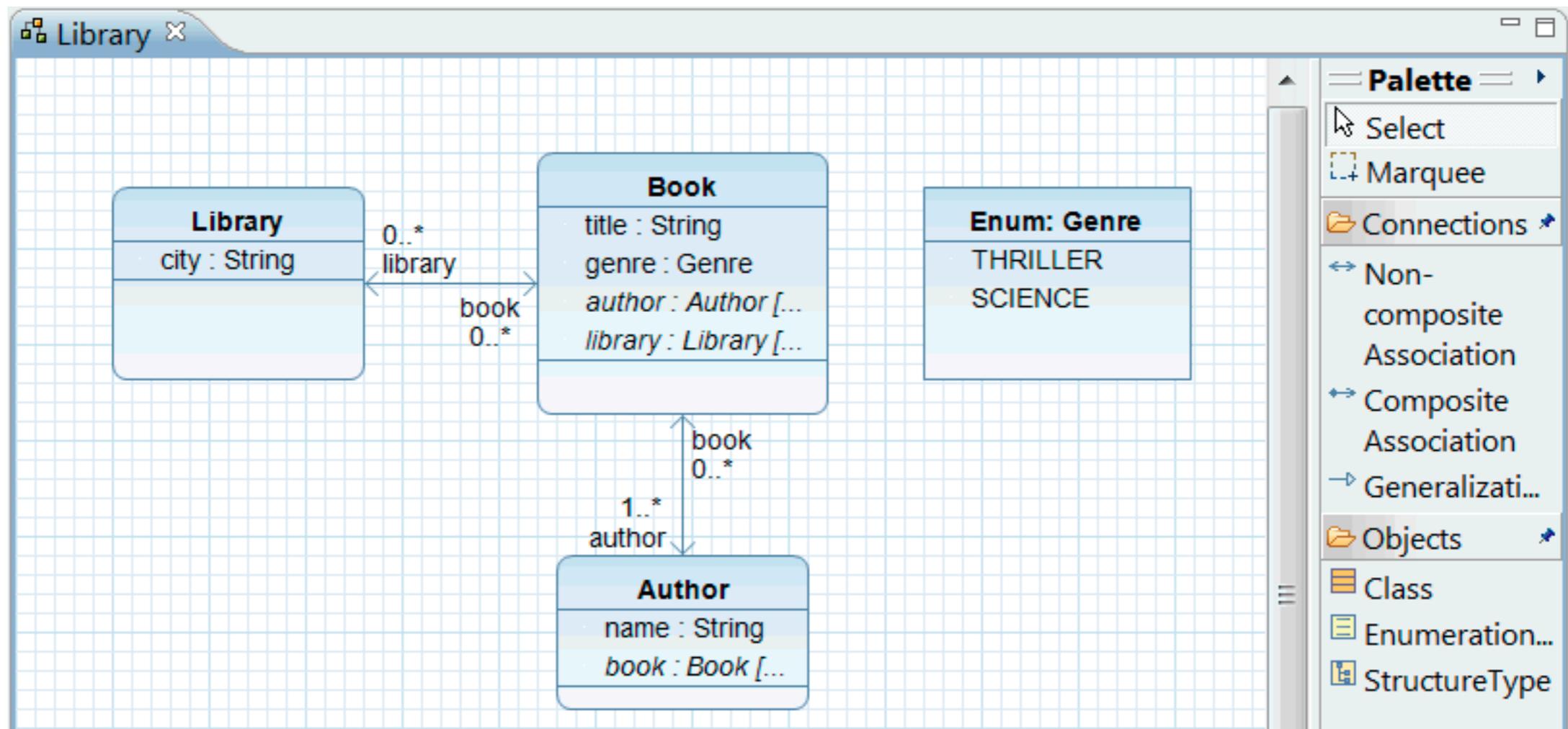


Graphiti

Graphiti

- ▶ Combines EMF and GEF
- ▶ Hides complexity and thereby introduces limitations

Typical Visuals



Architectural Overview

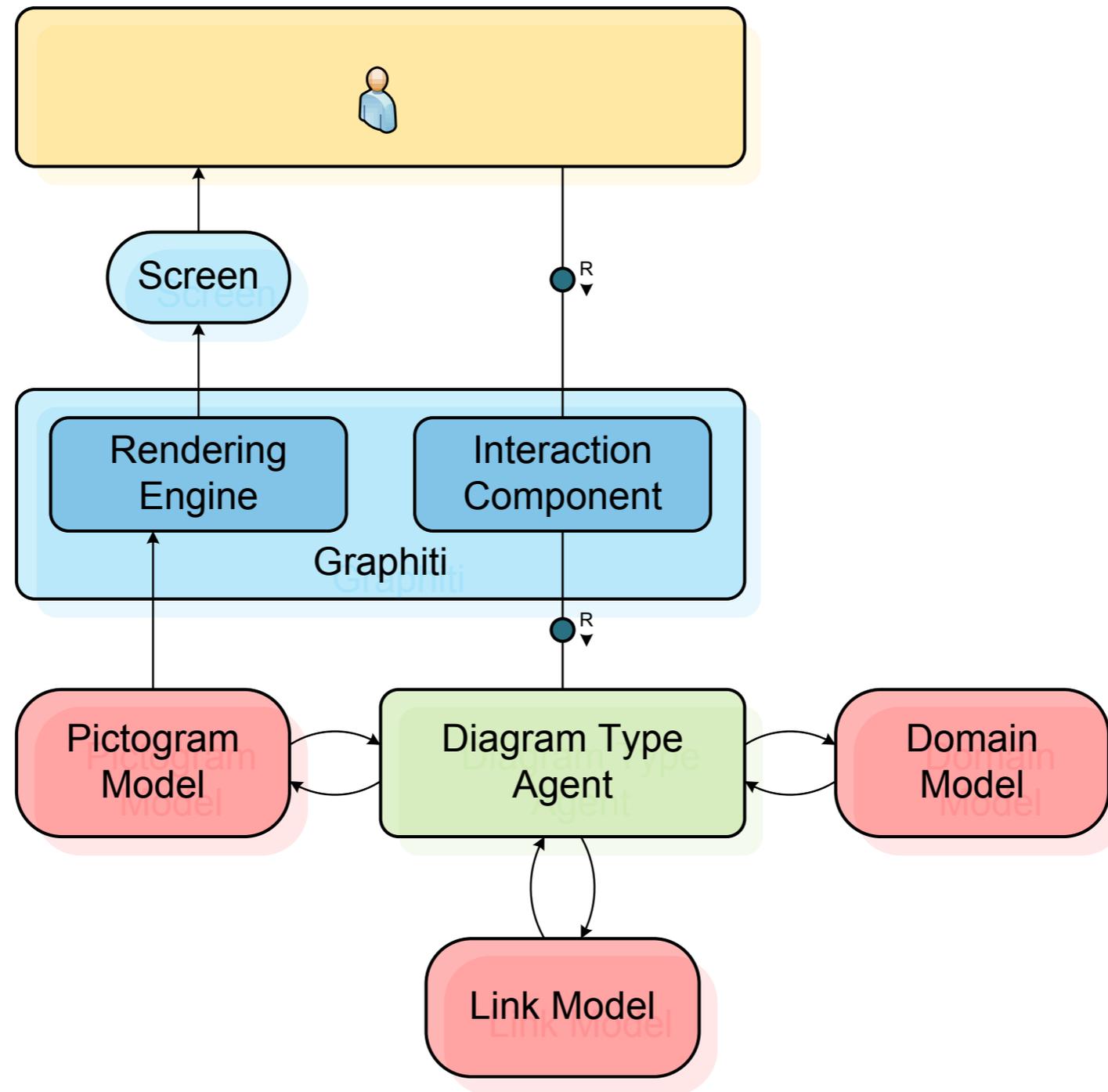
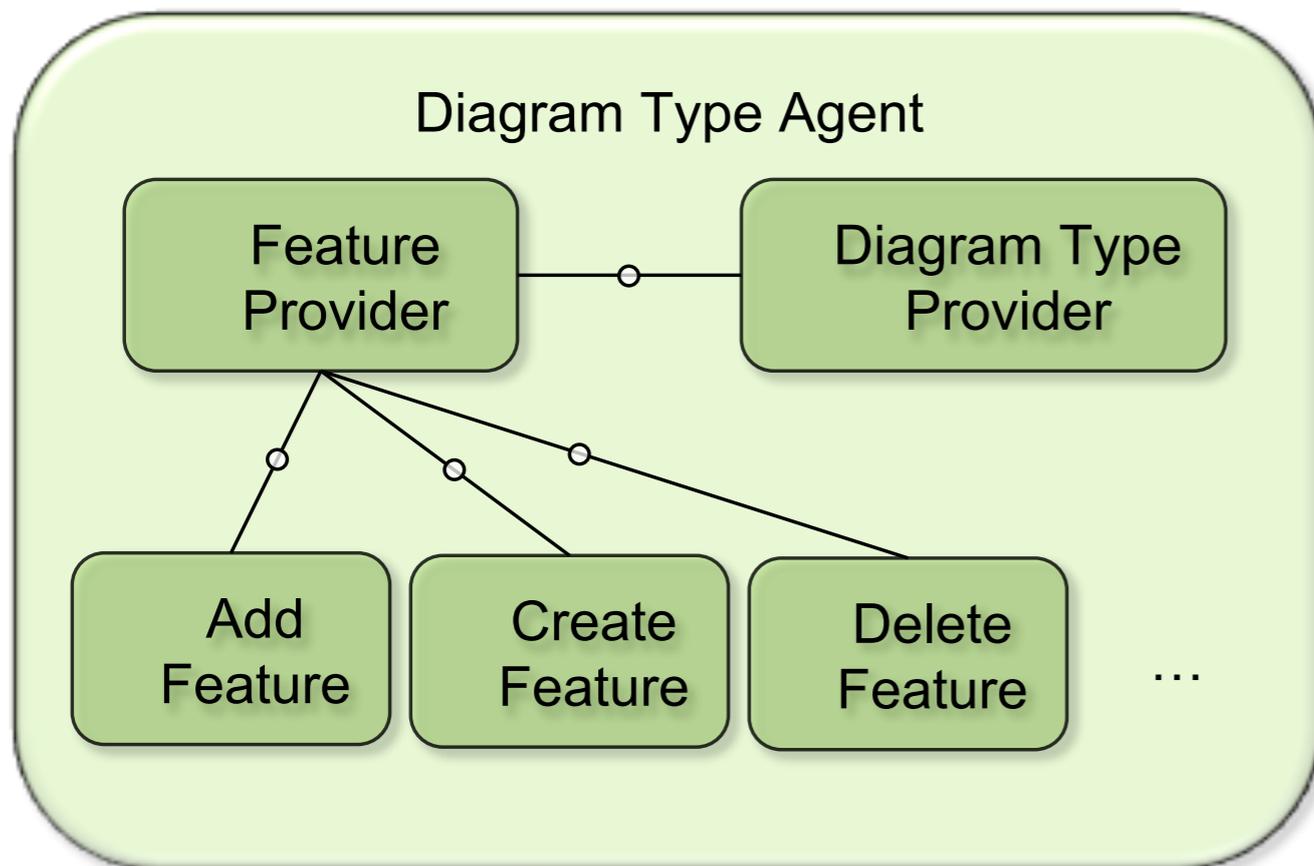
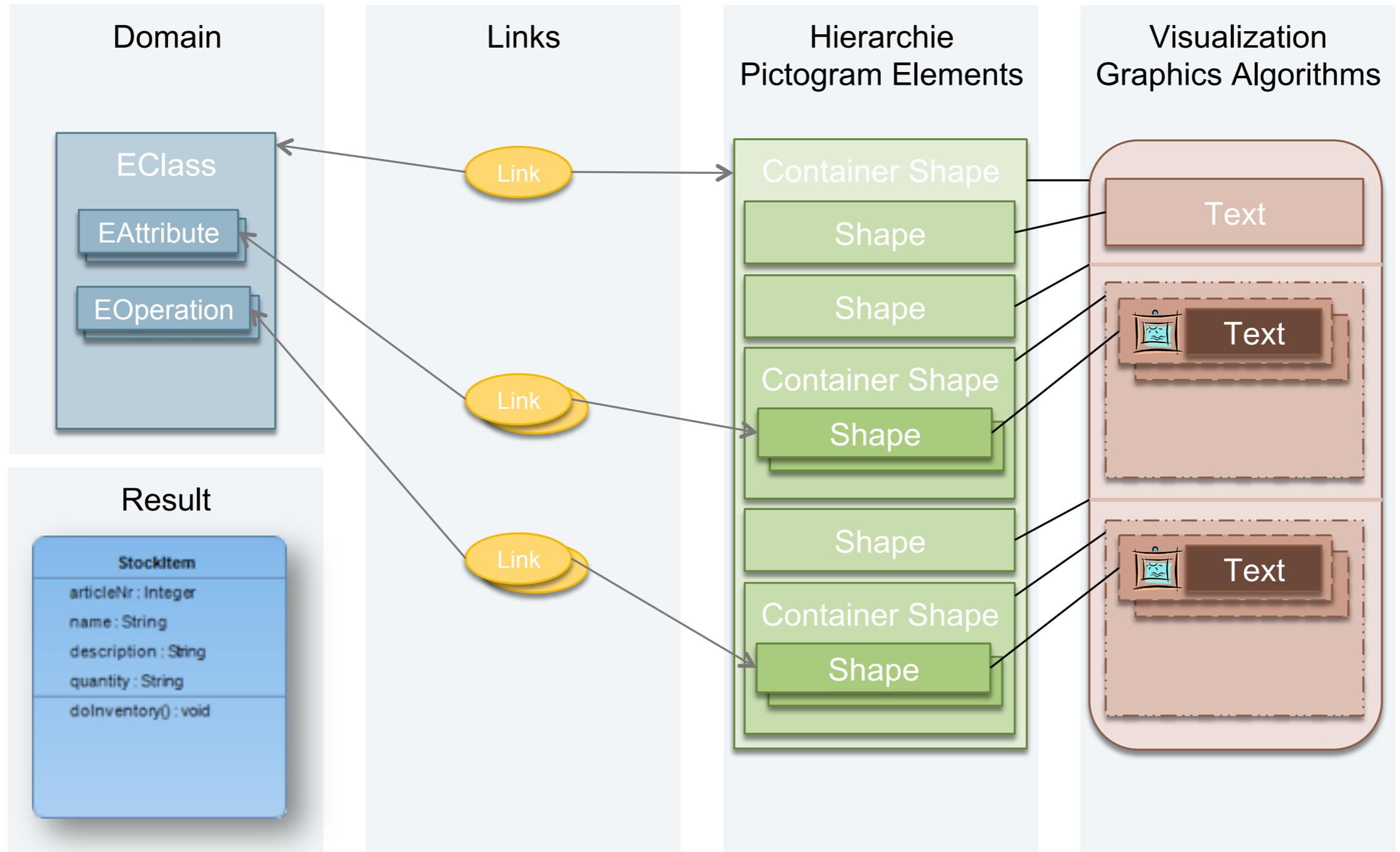


Diagram Type



```
org.eclipse.graphiti.examples.tutorial
├── JRE System Library [J2SE-1.5]
├── Plug-in Dependencies
├── src
│   ├── org.eclipse.graphiti.examples.tutorial
│   │   ├── org.eclipse.graphiti.examples.tutorial.diagram
│   │   │   ├── TutorialDiagramTypeProvider.java
│   │   │   ├── TutorialFeatureProvider.java
│   │   │   └── TutorialToolBehaviorProvider.java
│   │   └── org.eclipse.graphiti.examples.tutorial.features
│   │       ├── TutorialAddEClassFeature.java
│   │       ├── TutorialAddEReferenceFeature.java
│   │       └── TutorialAddEReferenceFeature
│   │           ├── TutorialAddEReferenceFeature(IFeatureProvider)
│   │           ├── add(IAddContext) : PictogramElement
│   │           ├── canAdd(IAddContext) : boolean
│   │           └── createArrow(GraphicsAlgorithmContainer) : Polyline
│   ├── org.eclipse.graphiti.examples.tutorial.handlers
│   ├── org.eclipse.graphiti.examples.tutorial.property
│   ├── icons
│   └── META-INF
│       ├── about.html
│       ├── build.properties
│       ├── plugin.properties
│       └── plugin.xml
```

How it Works



Graphiti vs. GMF Tools

Graphiti

GMF

Architecture

runtime-oriented

generative

Interface

self-contained

GEF-dependent

Client logic

centralized (feature
concept)

distributed
functionality

Look & feel

standardized, defined by
SAP usability experts

simple, adaptable
in generated code

Graphiti vs. GMF Runtime

▶ Graphiti :

- Contained API (using GEF/Draw2D)
- Hides complexity at expense of flexibility
- Decent documentation in form of tutorial
- Low cost of entry

▶ GMF Runtime:

- Extends GEF/Draw2D
- Adds complexity but also functionality
- Examples and documentation are available
 - But no simple tutorials
- High cost of entry

Sirius

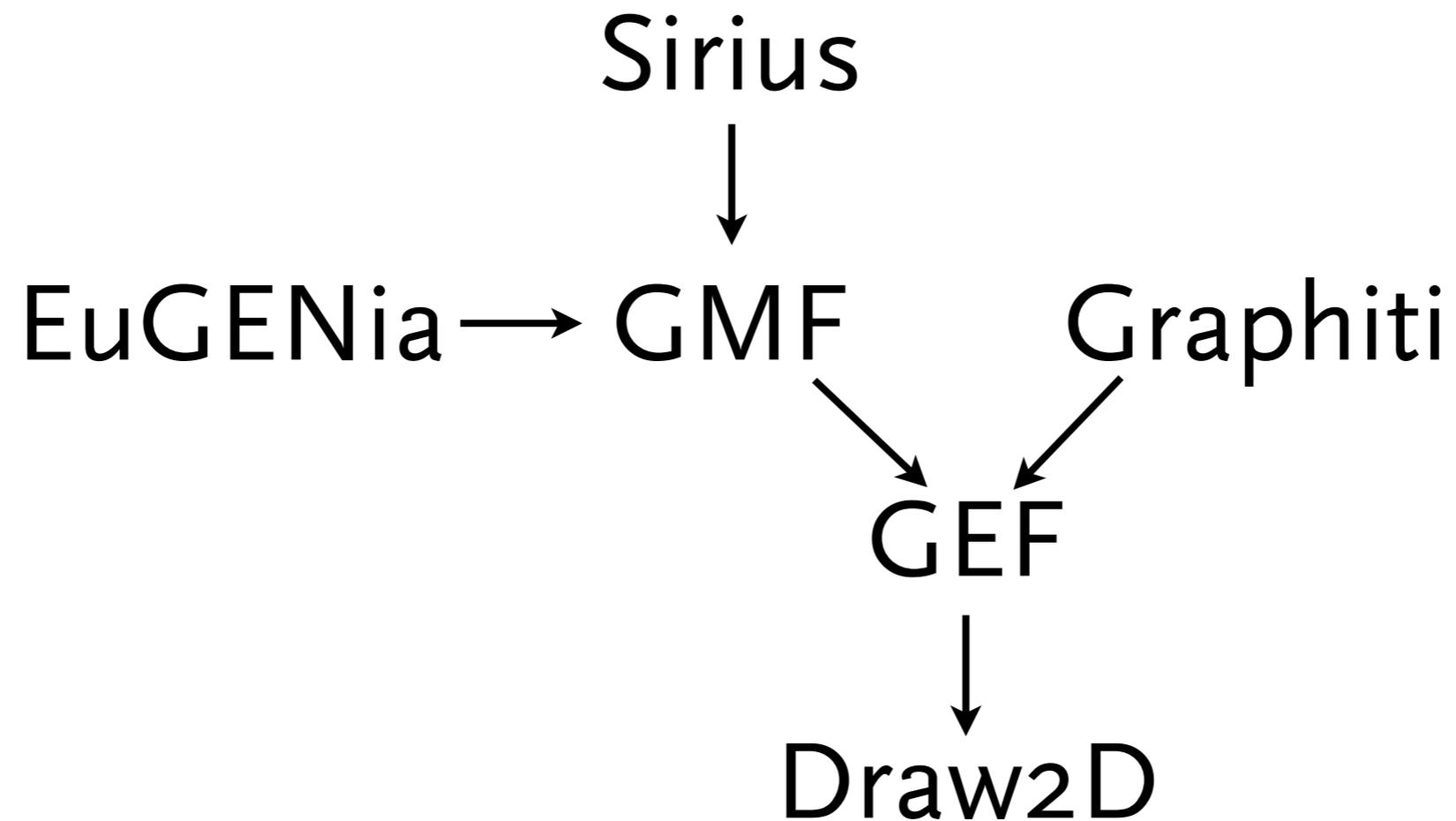
Sirius

- ▶ Multi-View modeling workbenches on top of eclipse modeling stack (EMF, EMFTools, GMF)
- ▶ Separation of modeling aspects and task, e.g. for multiple users
- ▶ Views
 - multiple representations for your model
 - diagrams (GMF)
 - trees
 - tables
 - (text)
- ▶ Layers
 - enable/disable elements in your notations

EMF Diagram Editor in Sirius

▶ show time

Summary



Combining Text and Diagrams

Combining Text and Diagrams

- ▶ Parsing vs. Model View Controller
 - a whole new model with each action vs. atomic commands on one persisting model
 - identification of model elements via identifier necessary to synchronize ever new textual notated model with the model in MVC
 - ◆ not always ambiguous
 - ◆ e.g. renaming vs. remove and add

Combining Text and Diagrams

- ▶ side by side, textual and graphical notation for the whole model
 - e.g. Ecore editors
 - ◆ Tree editor
 - ◆ Diagram editor
 - ◆ Emfatic
 - ◆ OclInEcore editor
 - ◆ ...
 - based on graphical and textual editors operating on Ecore models, i.e. serialized as XMI
 - synchronization limited by automated generation of secondary notations
 - ◆ white spaces in pretty printing
 - ◆ (partial) automatic diagram layout



Package Expl Hierarchy

test

Outline

An outline is not available.

Properties

Property	Value

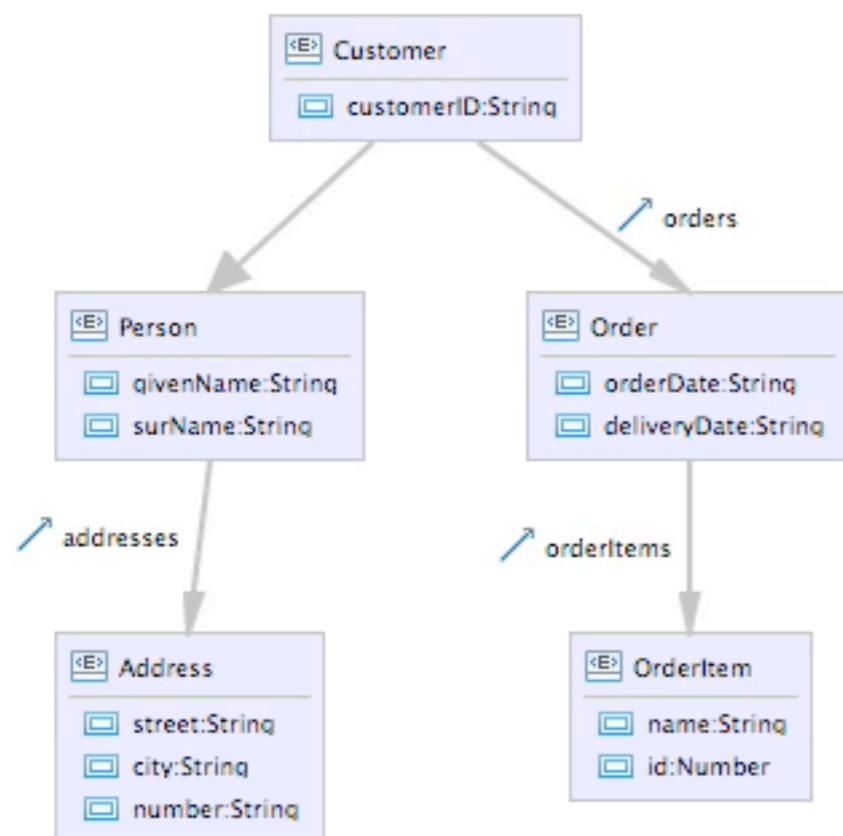
Combining Text and Diagrams

▶ integrated

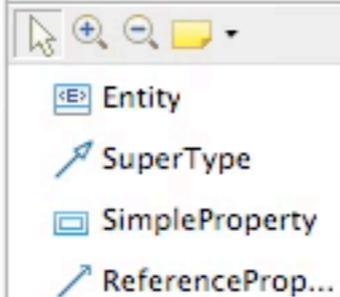
- textual fragments in graphical notations,
 - e.g.
 - ◆ OCL expressions in UML class diagrams
 - ◆ operation signatures in ECore models
 - ◆ expressions in state charts
- requires partial textual notations (i.e. different root nodes in grammar/AST)
- secondary notation is problematic
 - ◆ omit, purely pretty print -> loss of secondary notation
 - ◆ embed in model for graphical notation
- less but still synchronization, auto layout problems



test.entity_diagram



Palette



test.entity

```

entity Person {
  property givenName : String
  property surName : String
  addresses refs many Address
}

entity Address {
  property street : String
  property number : String
  property city : String
}

entity Order {
  orderItems refs many OrderItem
  property orderDate : String
  property deliveryDate : String
}

entity OrderItem {
  property name : String
  property id : Number
}

entity Customer extends Person {
  property customerID : String
  orders refs many Order
}
  
```

Summary

- ▶ Many different kinds of notations
 - trees, tables, texts, diagrams
- ▶ Multiple views
- ▶ Multiple notations
- ▶ Model-based development of notations and editors via frameworks
- ▶ Out of the box notations are almost free to generate
- ▶ Custom notations are more expensive
 - Trees are free, textual notations require some work, graphical notations require a lot of work
- ▶ Structural similarity between notation and meta-model is required in all cases. Composition is a major factor in notation and meta-model design.

Agenda

prolog
(1 VL)

Introduction: languages and their aspects, modeling vs. programming, meta-modeling and the 4 layer model

o.
(2 VL)

Eclipse/Plug-ins: eclipse, plug-in model and plug-in description, features, *p2*-repositories, *RCPs*

1.
(2 VL)

Structure: *Ecore*, *genmodel*, working with generated code, constraints with *Java* and *OCL*, *XML/XMI*

2.
(3 VL)

Notation: Customizing the tree-editor, textual with *XText*, graphical with *GEF* and *GMF*

➔ 3.
(4 VL)

Semantics: interpreters with *Java*, code-generation with *Java* and *XTend*, model-transformations with *Java* and *ATL*

epilog
(2 VL)

Tools: persisting large models, model versioning and comparison, model evolution and co-adaption, modular languages with *XBase*, *Meta Programming System (MPS)*