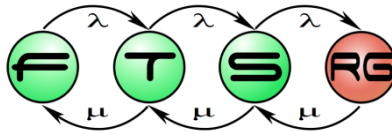


Graphical Editor Development

GEF, GMF, Graphiti



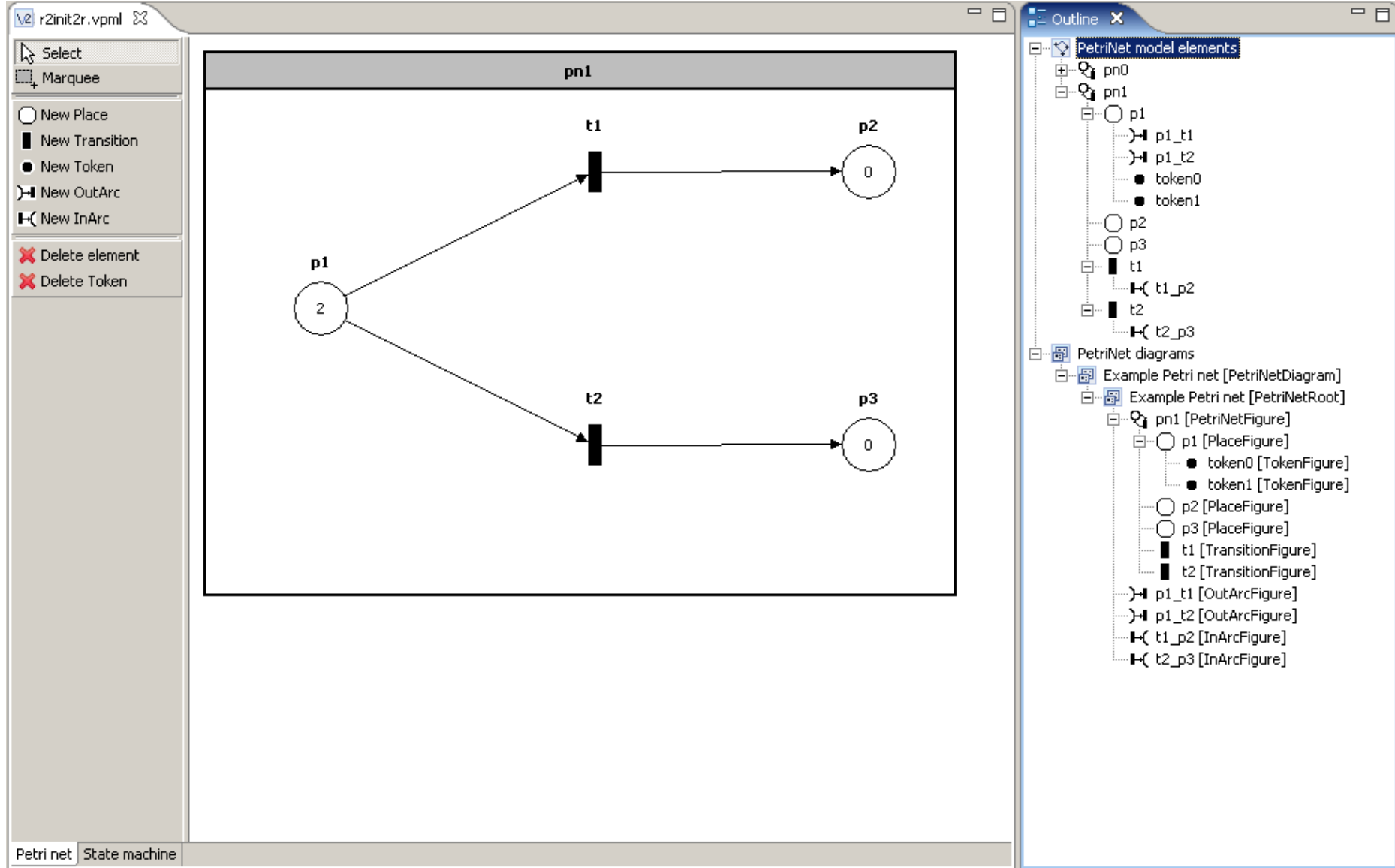
Designing modeling languages

- Metamodel: a model of models
 - Abstract syntax
 - **Concrete syntax**
 - Well-formedness rules
 - Behavioral (dynamic) semantics
 - Translation to other languages

Goal of GEF

- Graphical editors
- Eclipse integration
- Model technology independent
 - Requires change notification support!
- Increased abstraction level
 - Wrt platform

Sample Graphical Editor: Petri net



Social Network example

Eclipse IDE interface showing a social network diagram and its properties.

Project Explorer:

- DSE
 - default.socialnetwork
 - Social Network
 - Acquaintance friendship
 - Acquaintance marriage
 - Community Foo Club
 - Community Qux Fellowship
 - Person J. Random
 - Person Jane Doe
 - Person John Doe

Diagram:

The diagram illustrates a social network structure. It features a hierarchy of nested boxes representing communities: "Foo Club" contains "Bar Society", which contains "Baz Community". "Baz Community" is connected to "Jane Doe". "Jane Doe" is connected to "J. Random" and "John Doe". "John Doe" is connected to "Qux Fellowship".

Properties:

Property	Value
Name	John Doe
Sex	male
X	677
Y	240

More complex example: Sheet music

The screenshot displays a music notation software interface with the following components:

- Top Bar:** Contains musical symbols (b, b, b, #, X, +, -).
- Left Panel:**
 - Szólamok (Parts):** Includes a checked **Staff**.
 - Properties:** A table with columns **Property** and **Value**.

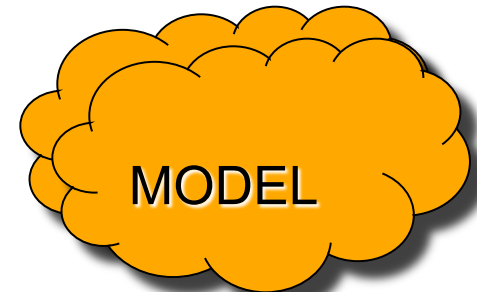
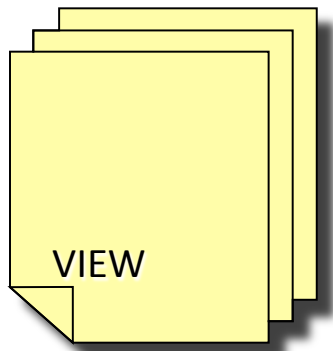
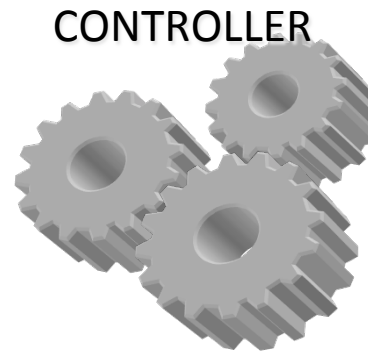
Property	Value
Ambit show	
Arranger	
Bracket	
Composer	
Connectivity	
Copyright	
Dedication	
Instrument	
Name	
Opus	
Poet	
Short name	

- Center Panel (Kottaszerkesztő):**
- Palette:** Contains sections for **Hang** (notes), **Szünet** (rests), **Szólam** (parts), and **Kulcs** (clefs).
- Score:** Displays a single staff with the title **Allegro (♩=120)**, a **mf** dynamic marking, and a musical phrase consisting of eighth and quarter notes.
- Bottom Panel (Áttekintés):** Provides a thumbnail overview of the score.

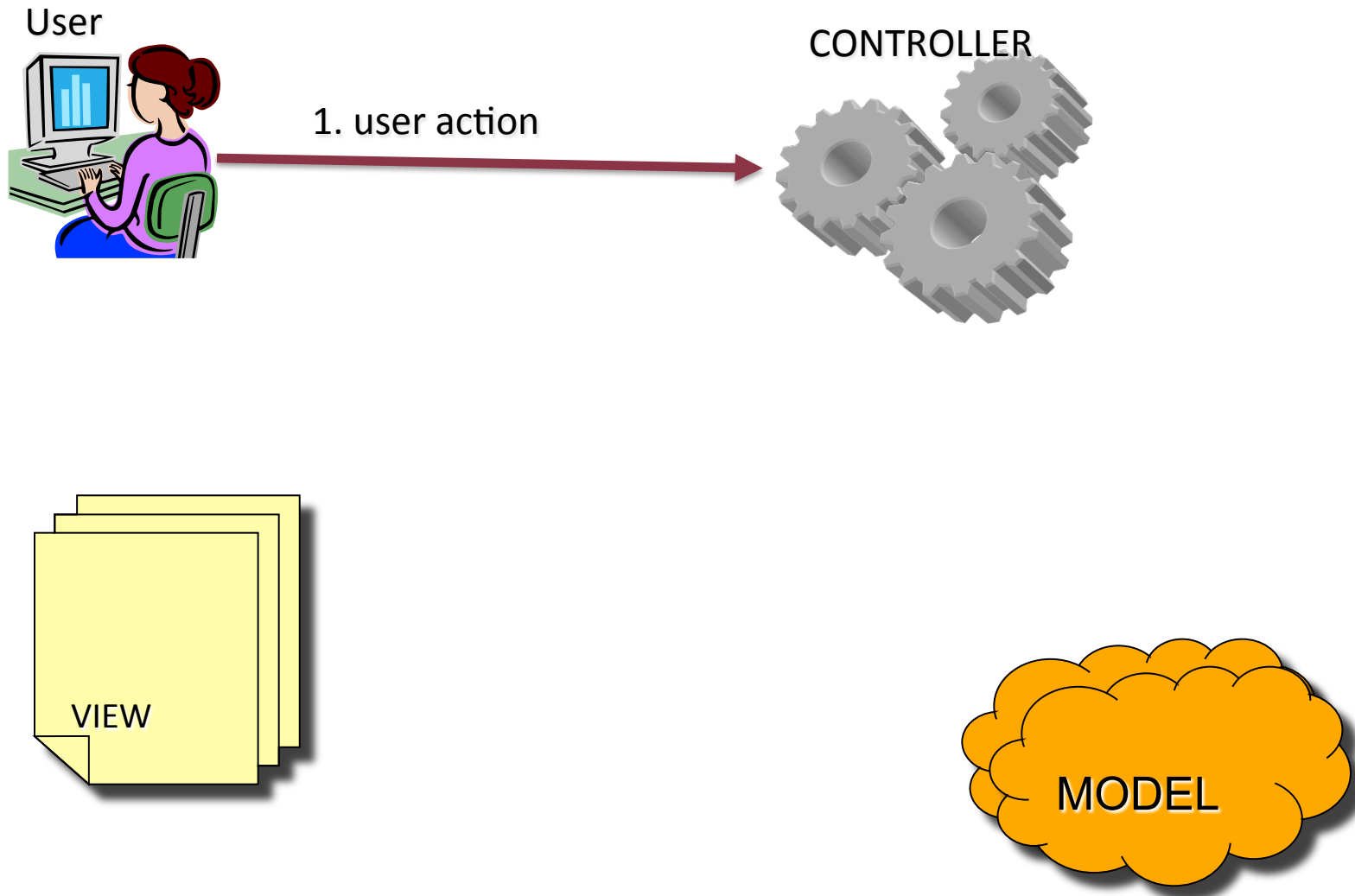
Model-View-Controller (MVC) architecture

- Separate data and presentation
 - Model: stores data
 - View: displays data
 - Controller: user interaction support

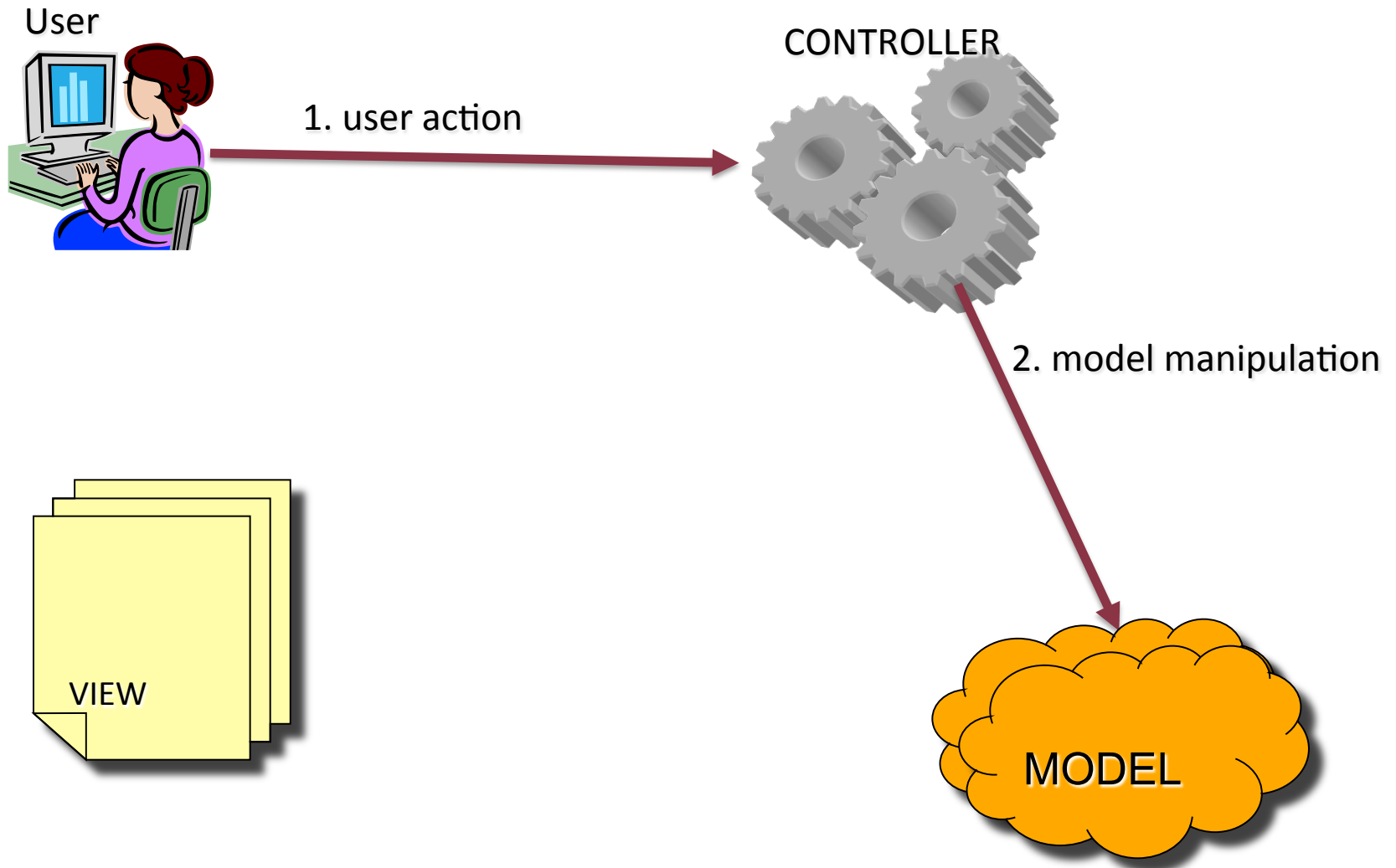
MVC in GEF



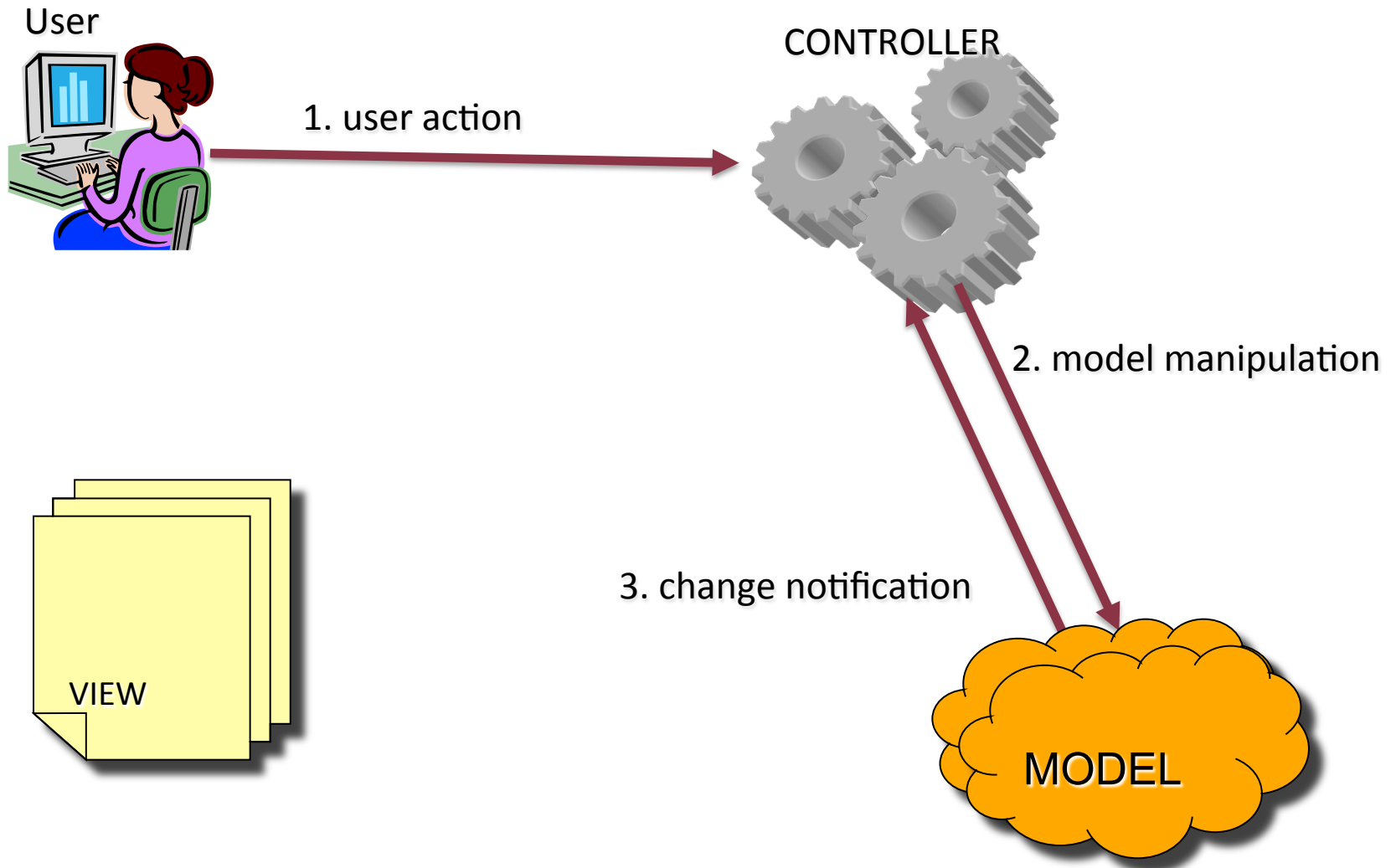
MVC in GEF



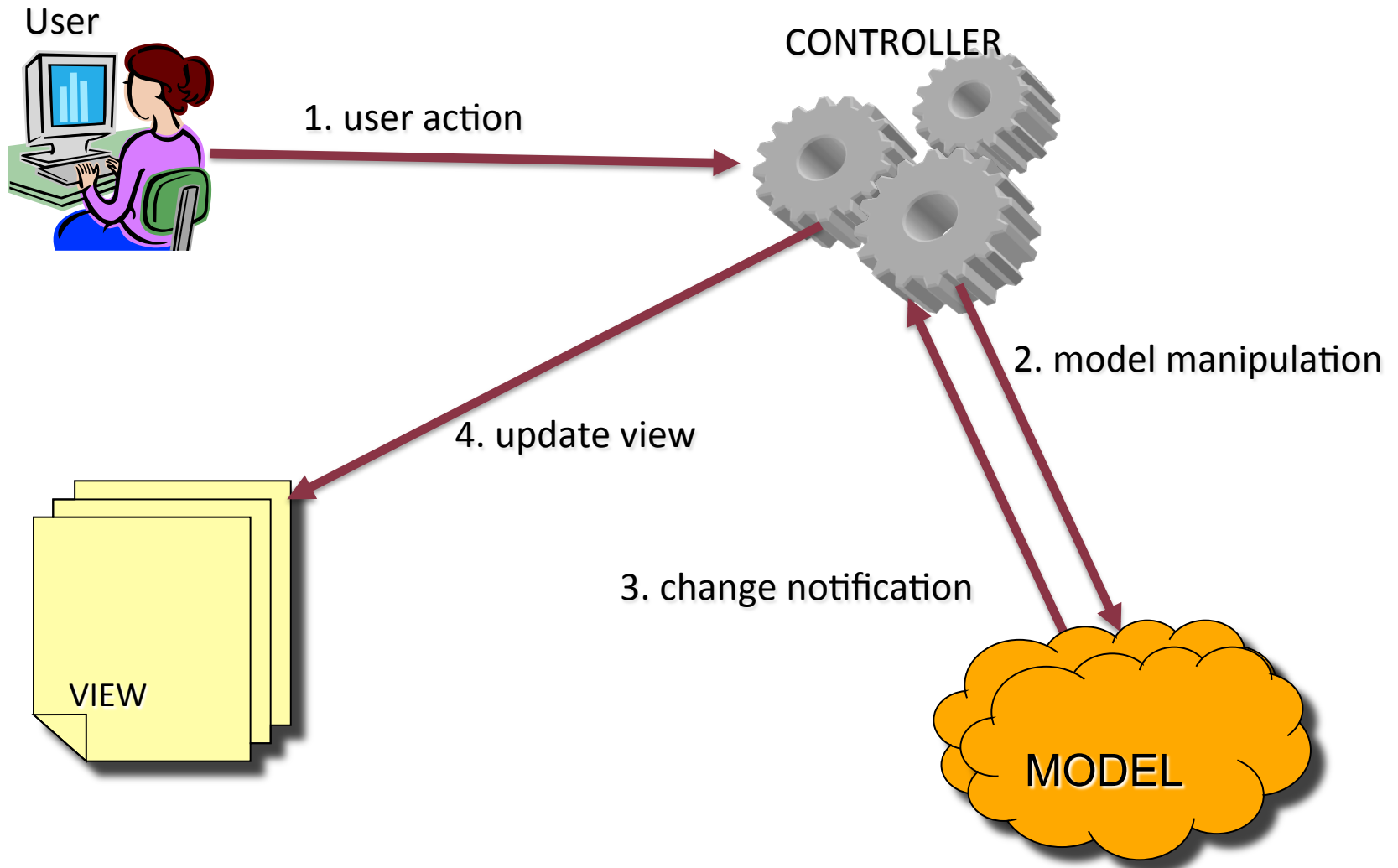
MVC in GEF



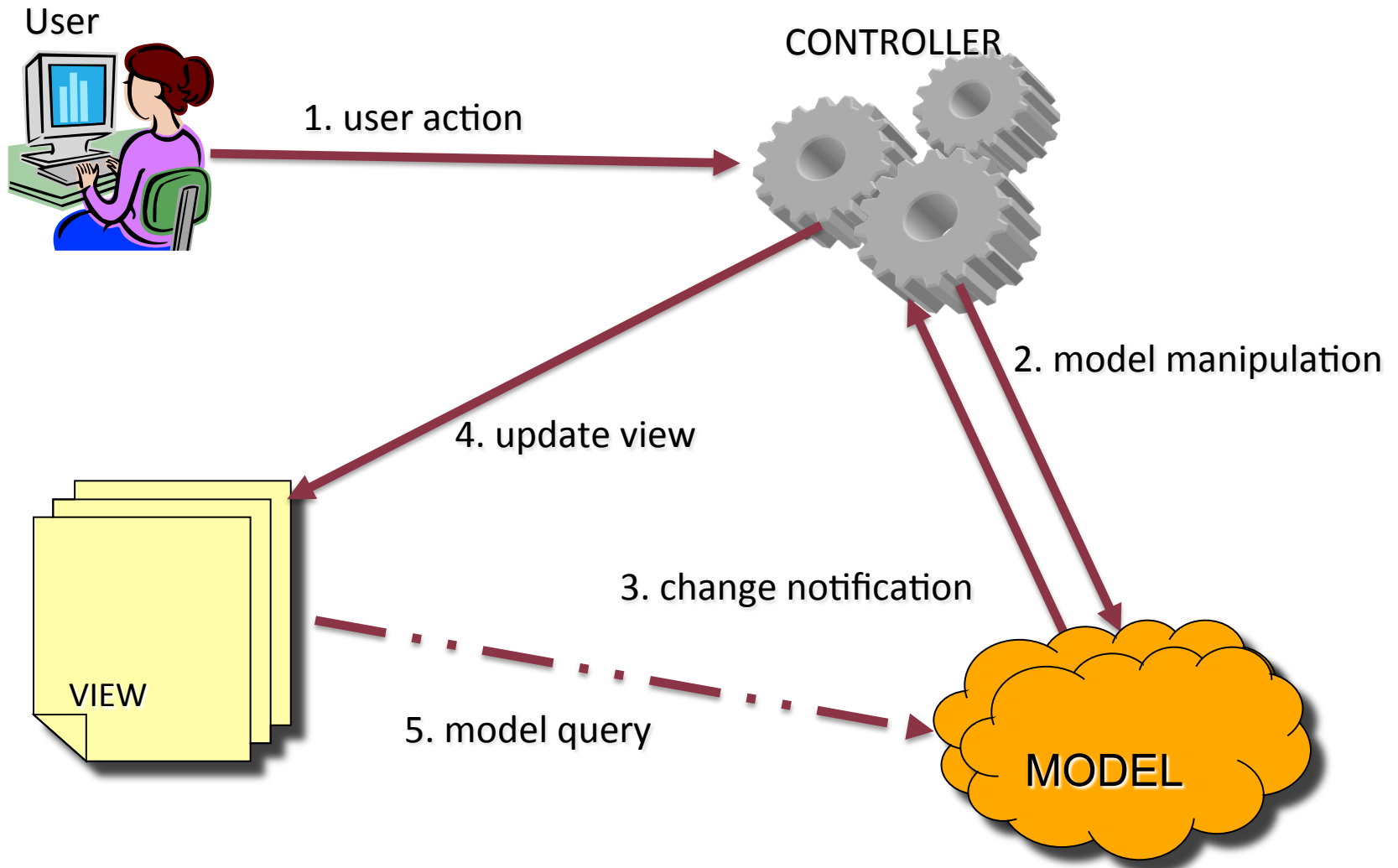
MVC in GEF



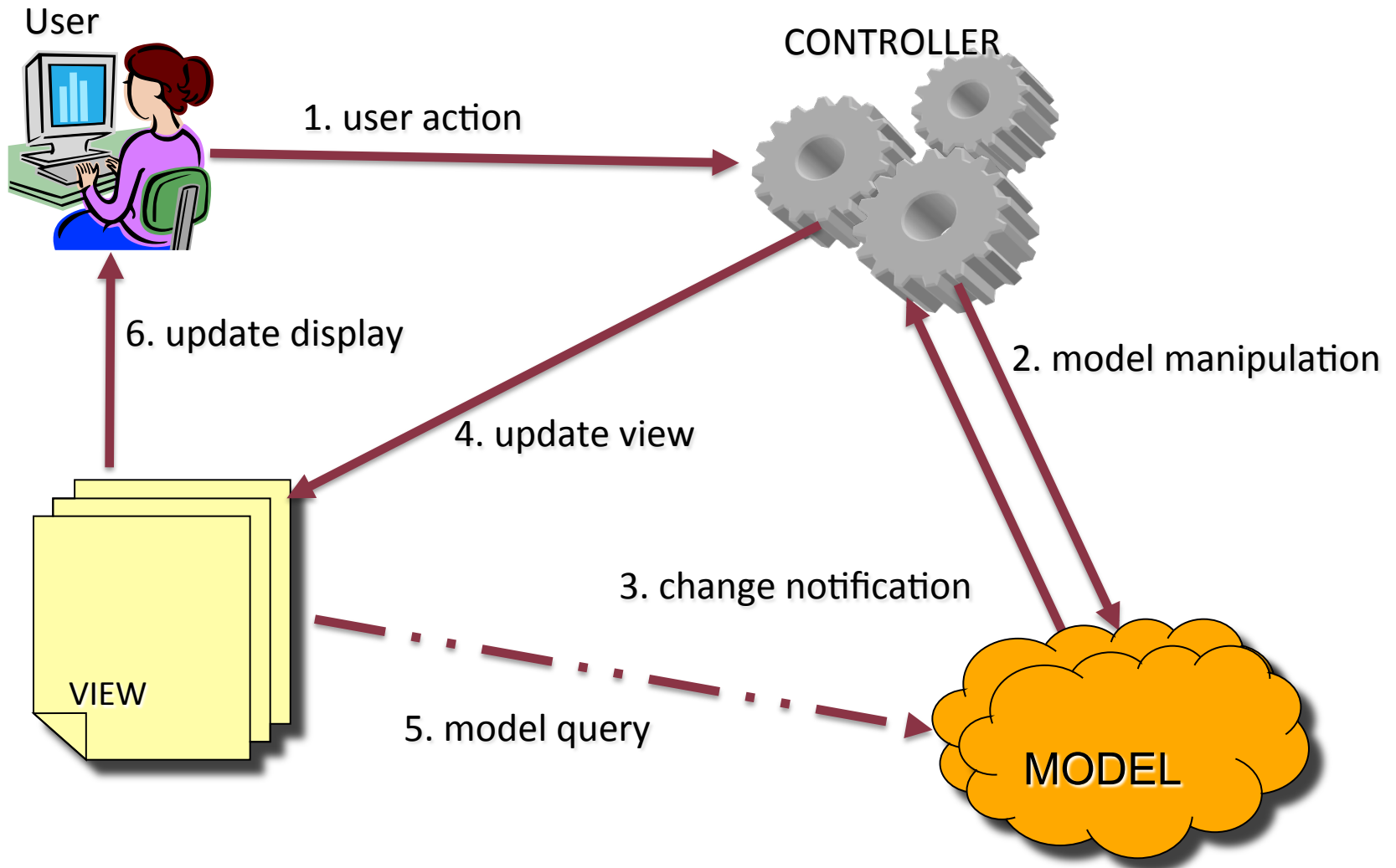
MVC in GEF



MVC in GEF



MVC in GEF



GEF workflow

Model

GEF workflow

Model

Notification

MVC in GEF: Model

- Arbitrary models supported
 - E.g., Java classes, EMF, database
 - Expect hierarchic storage (tree)
 - Notification support required
 - Notifies controller in case of changes
 - Critical if multiple views are present over the same model
 - EMF Notifications are useable
 - Business model
 - Structure, data
 - View model
 - Display information
 - E.g., position, size, color, ...

GEF workflow

Model

View

GEF workflow

Model

View

Drawing

GEF workflow

Model

View

Drawing

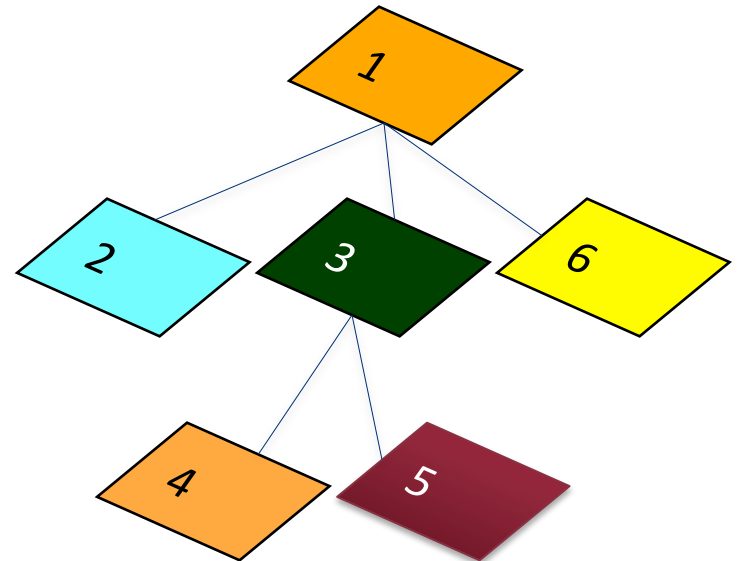
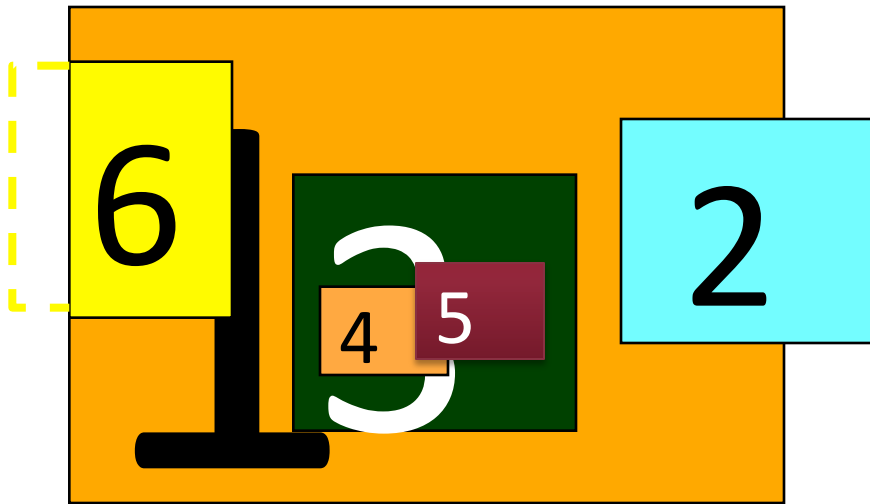
Layouting

MVC in GEF: View

- View: Draw2D Figures
 - Vector graphic library over SWT
 - Predefined simple elements
 - Label
 - Rectangle
 - Connector (arrow)
 - Hierarchic display
 - Basic component: Figure
- GEF views are implemented by
 - Draw2D Figure instances

Draw2D hierarchy

- Child position is relative to its parent
 - Negative coordinates (left, top) trimmed!
- Exactly one root element

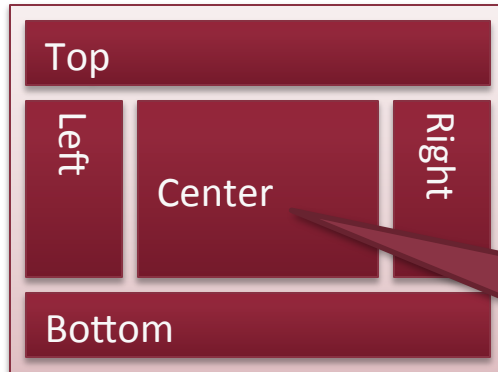


Draw2D LayoutManager

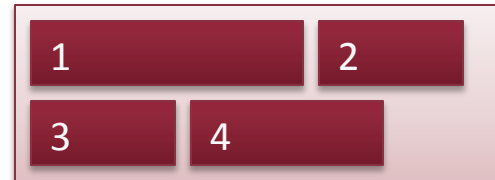
- Organizes children of a Figure
- Multiple implementations available, extensible
- Constraint: how to position children
 - Container position and size + LayoutManager + Constraint
 - Child position and size
 - Constraints are attached to children
 - Parent LayoutManager **may** consider constraints

Draw2D LayoutManagers

BorderLayout



FlowLayout

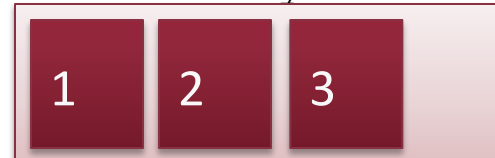


Constraints

XYLayout



ToolBarLayout



Draw2D elements

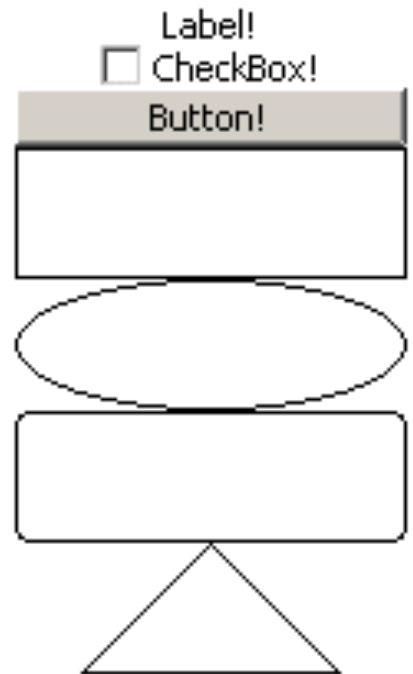
- Simple elements
 - Label, Button, CheckBox, Image
- Geometric shapes
 - RectangleFigure, Ellipse, Triangle
- Panel: generic container element
- ScrollPane: scrollable container element

Draw2D other elements

- Arrows
- Borders
- Custom figures
 - Sometimes combining existing elements is not enough
 - `paintFigure()` method can be redefined
 - SWT-based drawing code can be used

Base elements - example

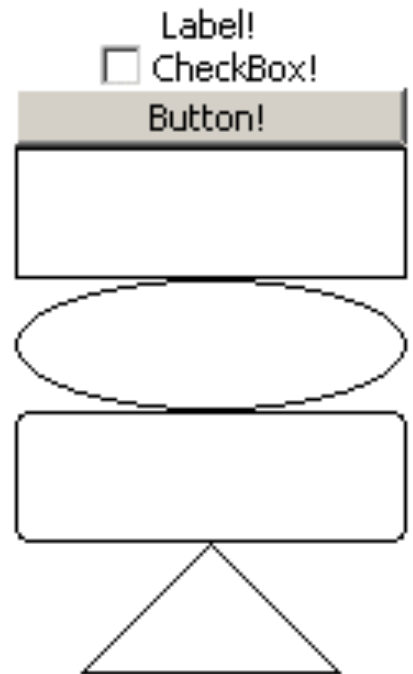
```
public class Pelda1 extends Figure {  
    public Pelda1() {  
        setOpaque(true);  
        setBackgroundColor(ColorConstants.white);  
        setLayoutManager(new ToolbarLayout());  
        add(new Label("Label!"));  
        add(new CheckBox("CheckBox!"));  
        add(new Button("Button!"));  
        add(new RectangleFigure());  
        add(new Ellipse());  
        add(new RoundedRectangle());  
        add(new Triangle());  
        for (int i = 3; i <= 6; i++)  
            ((Figure) getChildren().get  
                (i)).setPreferredSize(-1, 40);  
    }  
}
```



Base elements - example

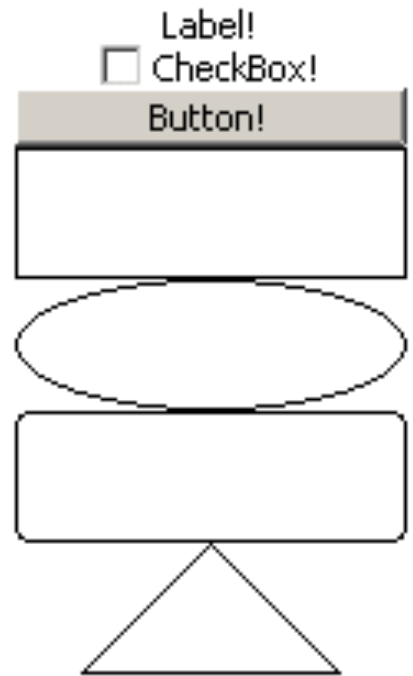
By default
everything is
transparent

```
public class Pelda1 extends JFrame {
    public Pelda1() {
        setOpaque(true);
        setBackgroundColor(ColorConstants.white);
        setLayoutManager(new ToolbarLayout());
        add(new Label("Label!"));
        add(new CheckBox("CheckBox!"));
        add(new Button("Button!"));
        add(new RectangleFigure());
        add(new Ellipse());
        add(new RoundedRectangle());
        add(new Triangle());
        for (int i = 3; i <= 6; i++)
            ((Figure) getChildren().get(i)).setPreferredSize(-1, 40);
    }
}
```



Base elements - example

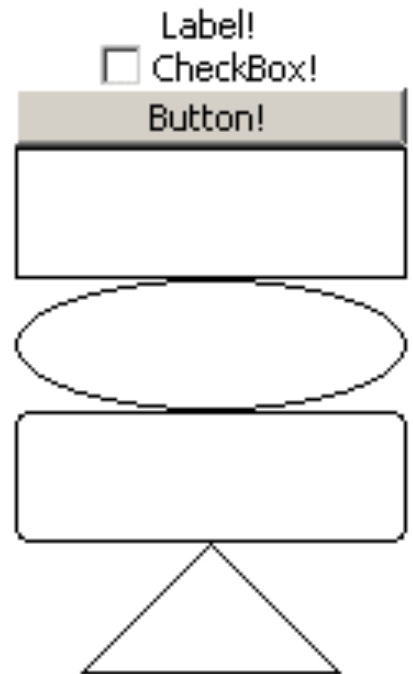
```
public class Pelda1 extends Figure {  
    public Pelda1() {  
        setOpaque(true);  
        setBackgroundColor(ColorConstants.white);  
        setLayoutManager(new ToolbarLayout());  
        add(new Label("Label!"));  
        add(new CheckBox("CheckBox!"));  
        add(new Button("Button!"));  
        add(new RectangleFigure());  
        add(new Ellipse());  
        add(new RoundedRectangle());  
        add(new Triangle());  
        for (int i = 3; i <= 6; i++)  
            ((Figure) getChildren().get  
                (i)).setPreferredSize(-1, 40);  
    }  
}
```



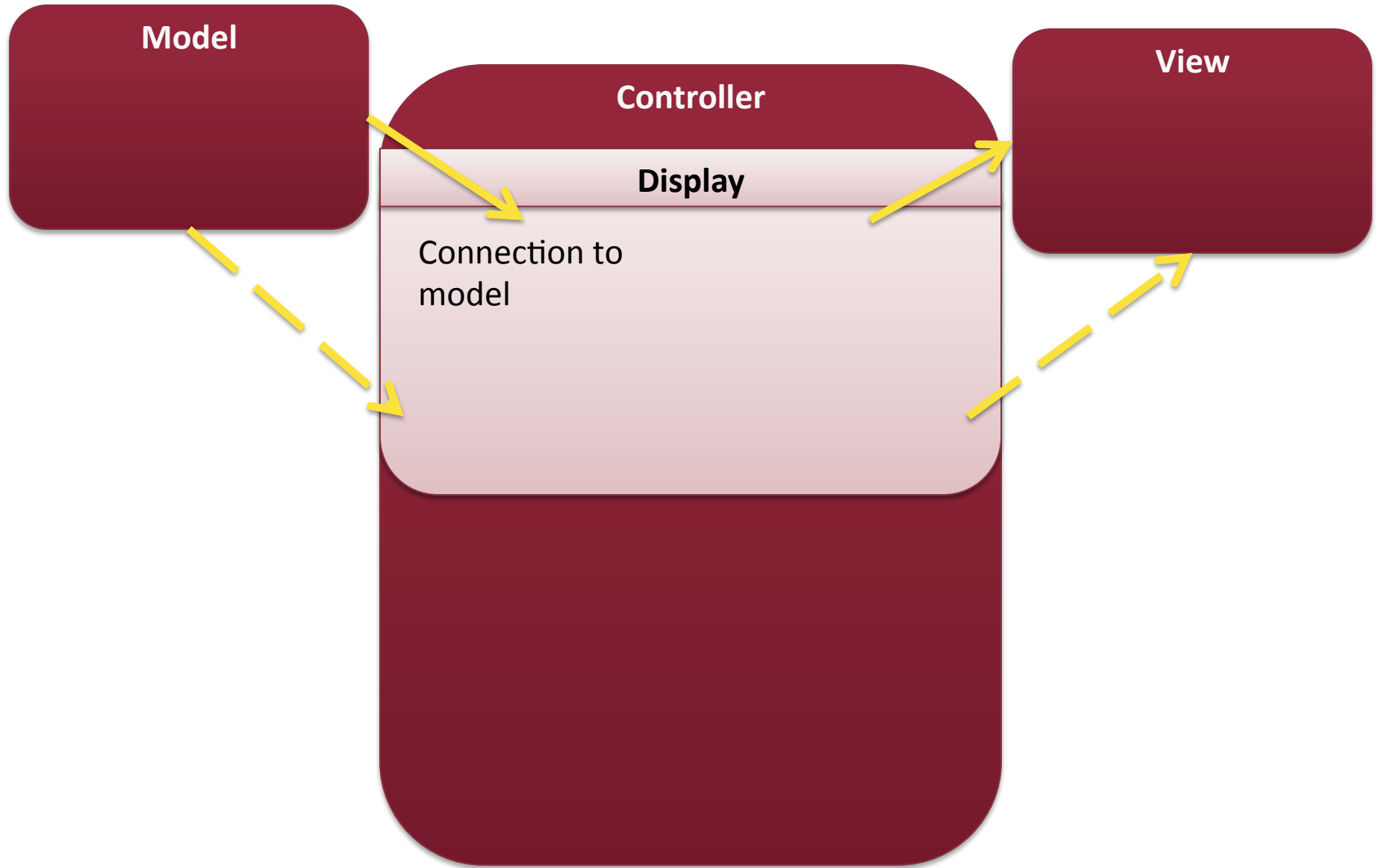
Base elements - example

```
public class Pelda1 extends Figure {  
    public Pelda1() {  
        setOpaque(true);  
        setBackgroundColor(ColorConstants.white);  
        setLayoutManager(new ToolbarLayout());  
        add(new Label("Label!"));  
        add(new CheckBox("CheckBox!"));  
        add(new Button("Button!"));  
        add(new RectangleFigure());  
        add(new Ellipse());  
        add(new RoundedRectangle());  
        add(new Triangle());  
        for (int i = 3; i <= 6; i++)  
            ((Figure) getChildren().get  
                (i)).setPreferredSize(-1, 40);  
    }  
}
```

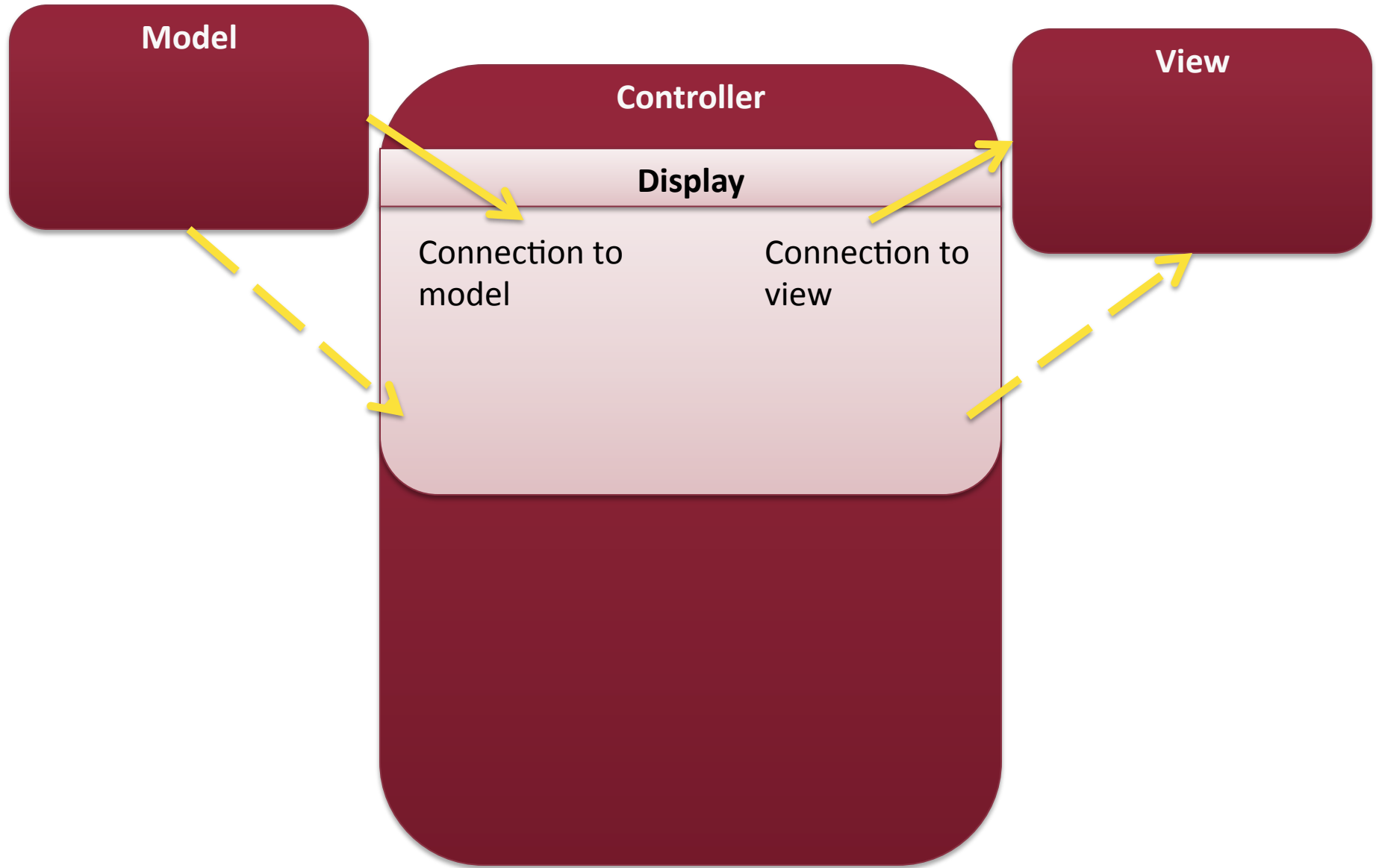
PreferredSize constraint



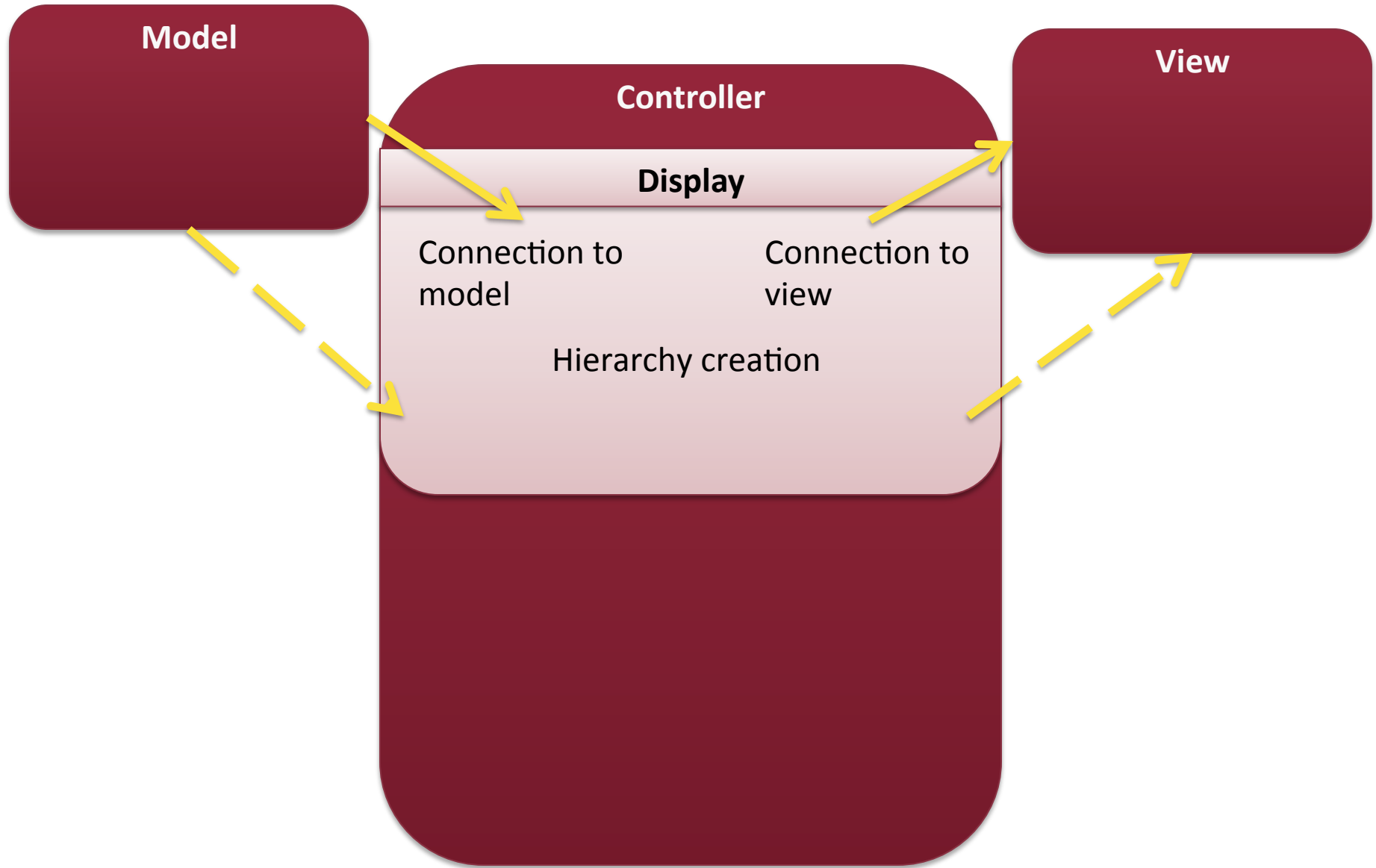
GEF workflow



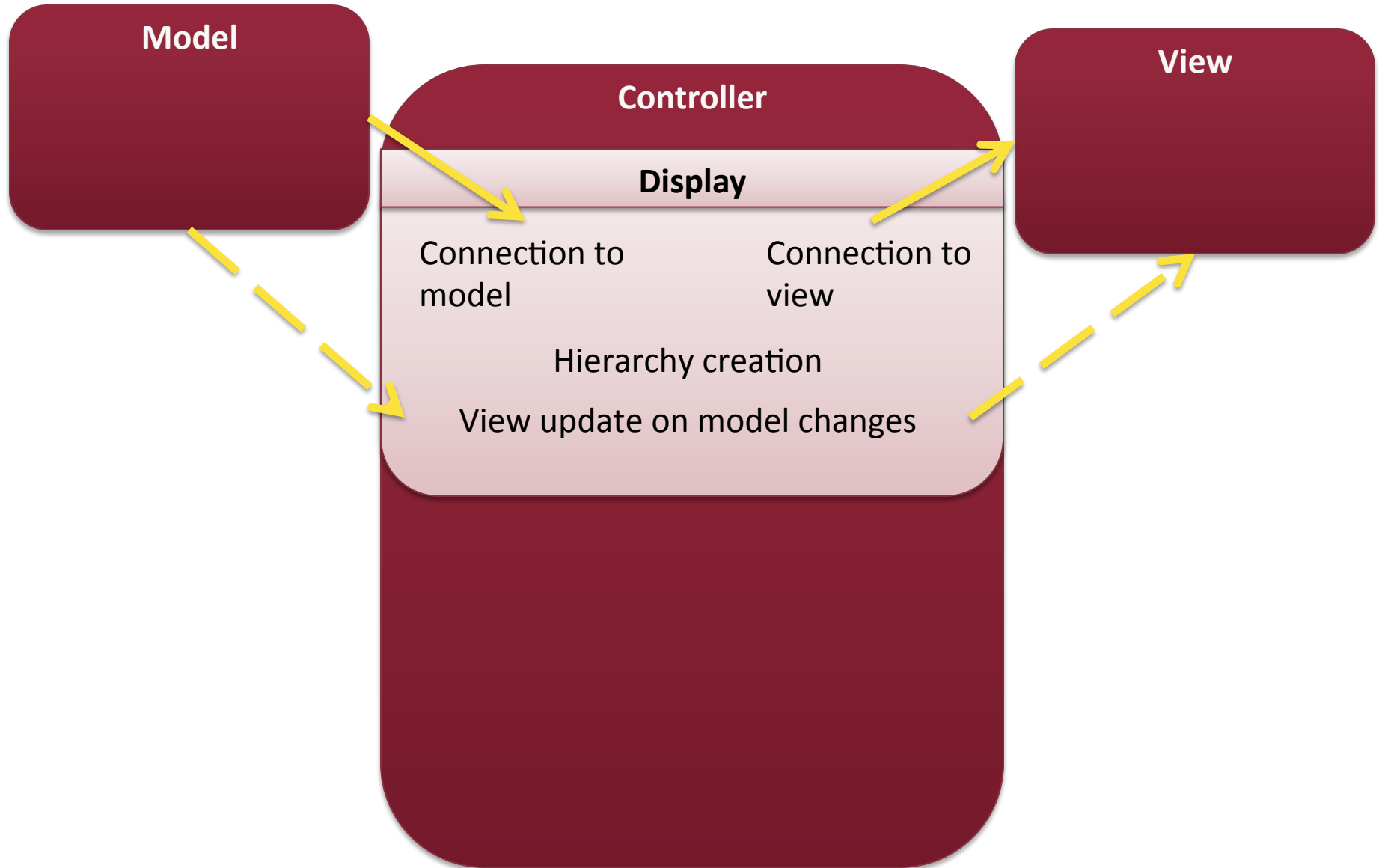
GEF workflow



GEF workflow



GEF workflow

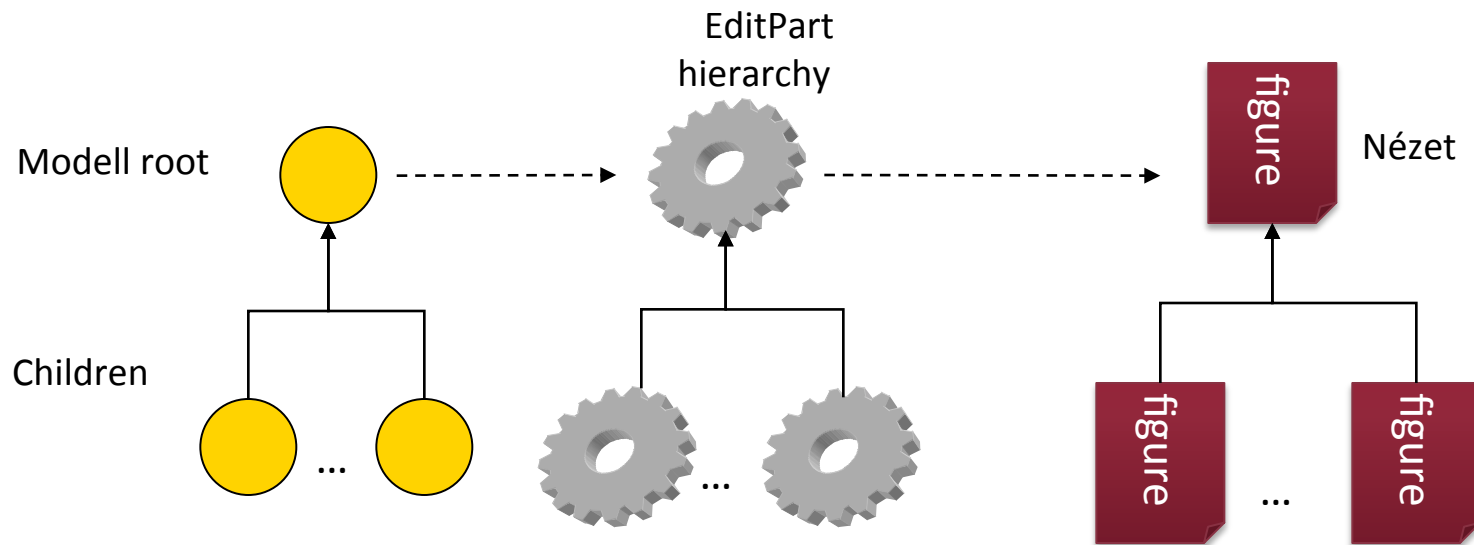


MVC in GEF: Controller

- Controller: EditPart classes
 - Connection between model and view
 - 1 Figure <-> 1 EditPart
 - 1 model element-> multiple EditPart
 - Thus multiple Figures
 - Updating Figures on changes
 - Handling user actions
 - Executes model manipulation

View initialization

- EditPartFactory
 - Creates EditPart instances for selected model element
- Figure instantiation
 - GraphicalEditPart.createFigure()



EditPartFactory

```
public class TestGEFEditPartFactory
    implements EditPartFactory {
    public EditPart createEditPart(EditPart
    context,    Object model) {
        EditPart ep = null;

        if (model instanceof ElementModel)
            ep = new ElementEditPart();
        else if (model instanceof ParentModel)
            ep = new ParentEditPart();

        if (ep != null)
            ep.setModel(model);
        return ep;
    }
}
```

EditPartFactory

```
public class TestGEFEditPartFactory
    implements EditPartFactory {
    public EditPart createEditPart(EditPart
    context,    Object model) {
        EditPart ep = null;

        if (model instanceof Element)
            ep = new ElementEditPart();
        else if (model instanceof ParentModel)
            ep = new ParentEditPart();

        if (ep != null)
            ep.setModel(model);
        return ep;
    }
}
```

Parent EditPart

EditPartFactory

```
public class TestGEFEditPartFactory
    implements EditPartFactory {
    public EditPart createEditPart(EditPart
    context,    Object model) {
        EditPart ep = null;

        if (model instanceof ElementModel)
            ep = new ElementEditPart();
        else if (model instanceof ParentModel)
            ep = new ParentEditPart();

        if (ep != null)
            ep.setModel(model);
        return ep;
    }
}
```

EditPartFactory

```
public class TestGEFEditPartFactory
    implements EditPartFactory {
    public EditPart createEditPart(EditPart
    context,    Object model) {
        EditPart ep = null;

        if (model instanceof ElementModel)
            ep = new ElementEditPart();
        else if (model instanceof ParentModel)
            ep = new ParentEditPart();

        if (ep != null)
            ep.setModel(model);
        return ep;
    }
}
```

EditPart stores a
model reference

View generation

- In EditPart class

```
public class ElementEditPart extends
    AbstractGraphicalEditPart {
    ...
    @Override
    protected IFigure createFigure() {
        // Create custom figure
        ElementFigure fig = new
ElementFigure();
        return fig;
    }
    ...
}
```

Model traversal

- Initial GEF model
 - EditPartFactory
 - Model root
- How to traverse to other elements?
 - EditPart instances know their children
 - Recursive traversal
 - Containment circle is a bad idea 😊
 - Containment hierarchy should be connected
 - EMF containment hierarchy **may** be the same

Model traversal

- `EditPart.getModelChildren()`
 - Returns the children of the connected model element
 - Ordering important -> determines covering

```
public class TestParentEditPart extends
    AbstractGraphicalEditPart {
    ...
    @Override
    protected List getModelChildren() {
        // Query own model
        ParentModel pm = ((ParentModel)
            getModel());
        return pm.getChildren();
    }
    ...
}
```

Model dependent,
not GEF specific

Model traversal

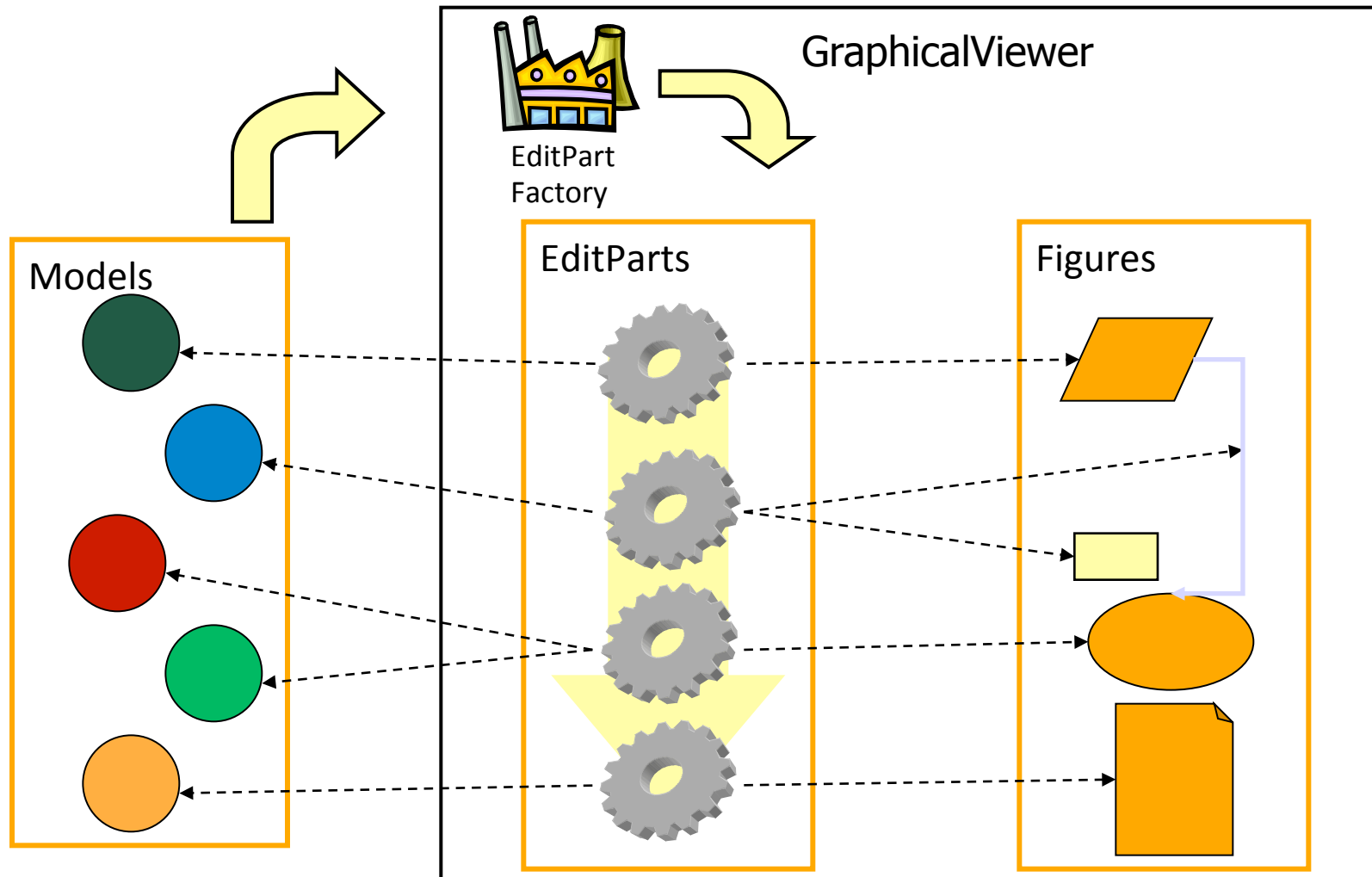
- `EditPart.getModelChildren()`
 - Returns the children of the connected model element
 - Ordering important -> determines covering

```
public class TestParentEditPart extends
    AbstractGraphicalEditPart {
    ...
    @Override
    protected List getModelChildren() {
        // Query own model
        ParentModel pm = ((ParentModel)
            getModel());
        return pm.getChildren();
    }
    ...
}
```

Model query

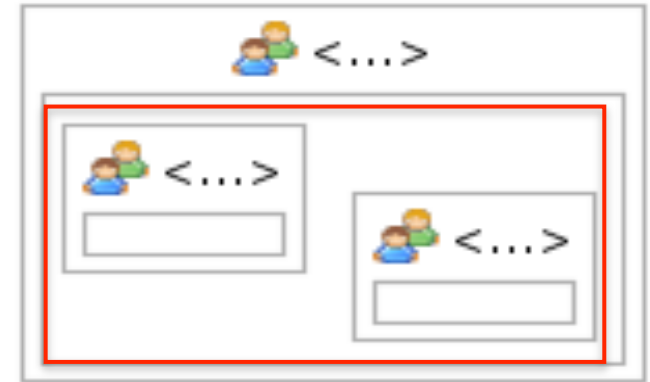
Model dependent,
not GEF specific

Building view - Summary



ContentPane

- ContentPane: place of Figure to present children
- By default: entire Figure
- EditPart overrides
 - If only a child of Figure needed



...

```
@Override
public IFigure getContentPane() {
    return ((MyFigure)
getFigure()).getPlaceOfChildren();
}
```

...

View update on changes

- EditPart listens to model changes
 - activate(), deactivate()
- Calls refresh methods
 - refreshVisuals(): non-structural change
 - refreshChildren(): child list changed

EditPart example

```
public class ParentEditPart extends AbstractGraphicalEditPart
    implements MyModelListener {
    protected void refreshVisuals() {
        ((ParentView) getFigure()).setLabel(
            ((ParentModel) getModel()).getName());
    }
    public void activate() {
        super.activate();
        ((ParentModel) getModel()).addListener(this);
    }
    public void deactivate() {
        ((ParentModel) getModel()).removeListener(this);
        super.deactivate();
    }
    public void modelChanged() {
        refreshVisuals();
        refreshChildren();
    }
}
```

EditPart example

```
public class ParentEditPart extends AbstractGraphicalEditPart
    implements MyModelListener {
    protected void refreshVisuals() {
        ((ParentView) getFigure()).setLabel(
            ((ParentModel) getModel()).getName());
    }
    public void activate() {
        super.activate();
        ((ParentModel) getModel()).addListener(this);
    }
    public void deactivate() {
        ((ParentModel) getModel()).removeListener(this);
        super.deactivate();
    }
    public void modelChanged() {
        refreshVisuals();
        refreshChildren();
    }
}
```

View refresh

EditPart example

```
public class ParentEditPart extends AbstractGraphicalEditPart
    implements MyModelListener {
    protected void refreshVisuals() {
        ((ParentView) getFigure()).setLabel(
            ((ParentModel) getModel()).getName());
    }
    public void activate() {
        super.activate();
        ((ParentModel) getModel()).addListener(this);
    }
    public void deactivate() {
        ((ParentModel) getModel()).removeListener(this);
        super.deactivate();
    }
    public void modelChanged() {
        refreshVisuals();
        refreshChildren();
    }
}
```

Register
change listener

EditPart example

```
public class ParentEditPart extends AbstractGraphicalEditPart
    implements MyModelListener {
    protected void refreshVisuals() {
        ((ParentView) getFigure()).setLabel(
            ((ParentModel) getModel()).getName());
    }
    public void activate() {
        super.activate();
        ((ParentModel) getModel()).addListener(this);
    }
    public void deactivate() {
        ((ParentModel) getModel()).removeListener(this);
        super.deactivate();
    }
    public void modelChanged() {
        refreshVisuals();
        refreshChildren();
    }
}
```

Remove
change listener

EditPart example

```
public class ParentEditPart extends AbstractGraphicalEditPart
    implements MyModelListener {
    protected void refreshVisuals() {
        ((ParentView) getFigure()).setLabel(
            ((ParentModel) getModel()).getName());
    }
    public void activate() {
        super.activate();
        ((ParentModel) getModel()).addListener(this);
    }
    public void deactivate() {
        ((ParentModel) getModel()).removeListener(this);
        super.deactivate();
    }
    public void modelChanged() {
        refreshVisuals();
        refreshChildren();
    }
}
```

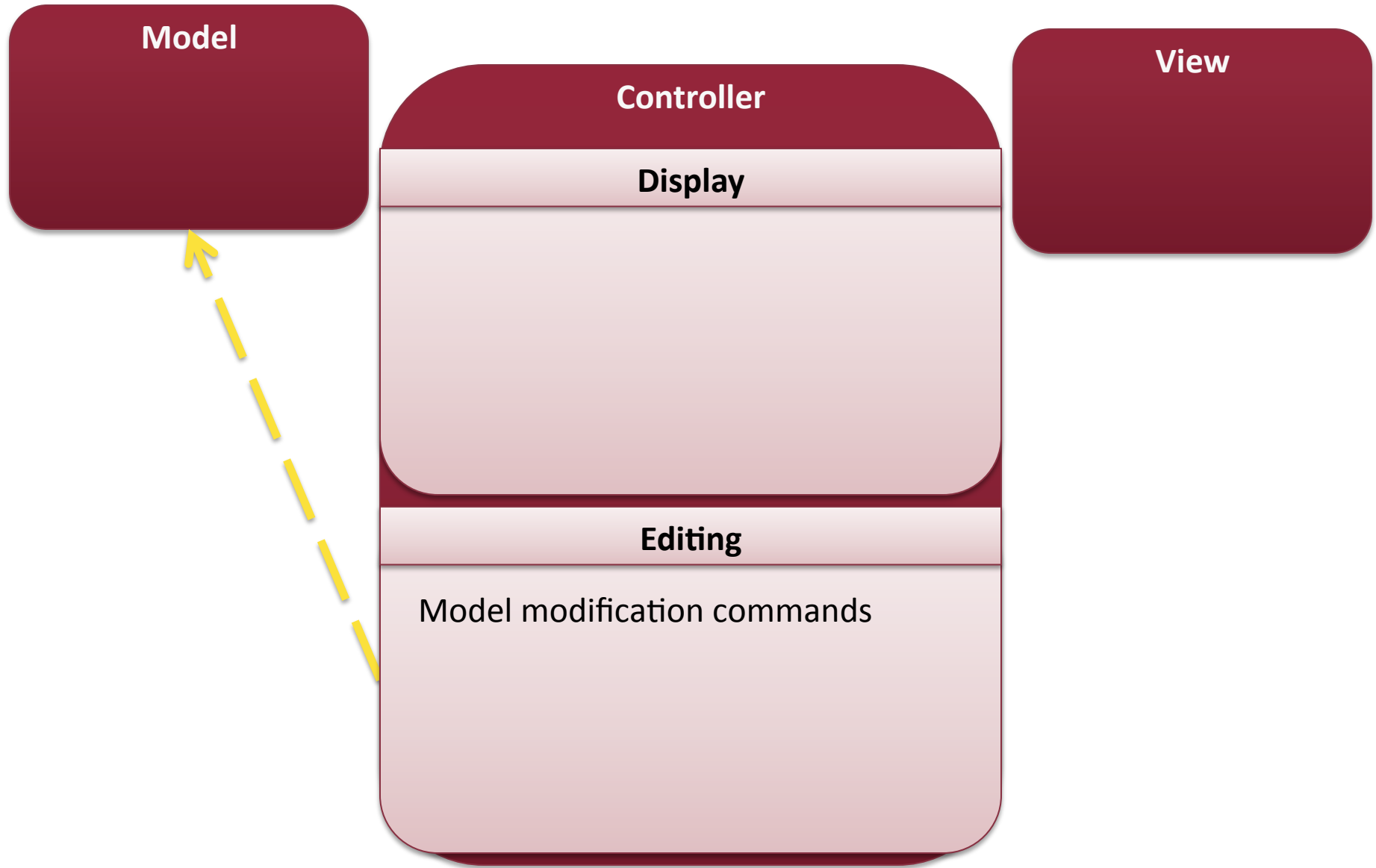
This is called
by model

EditPart example

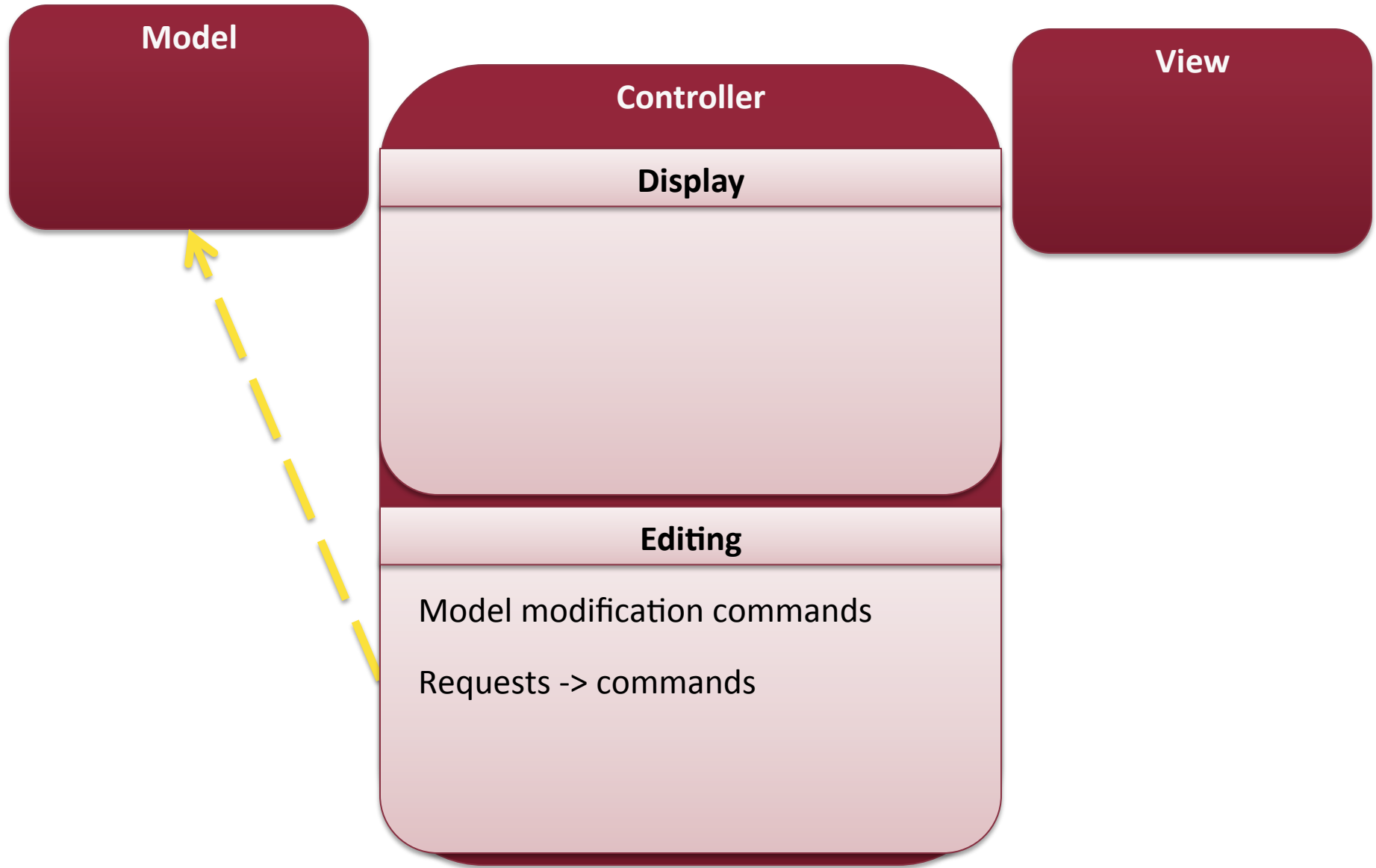
```
public class ParentEditPart extends AbstractGraphicalEditPart
    implements MyModelListener {
    protected void refreshVisuals() {
        ((ParentView) getFigure()).setLabel(
            ((ParentModel) getModel()).getName());
    }
    public void activate() {
        super.activate();
        ((ParentModel) getModel()).addListener(this);
    }
    public void deactivate() {
        ((ParentModel) getModel()).removeListener(this);
        super.deactivate();
    }
    public void modelChanged() {
        refreshVisuals();
        refreshChildren();
    }
}
```

Refreshing
children

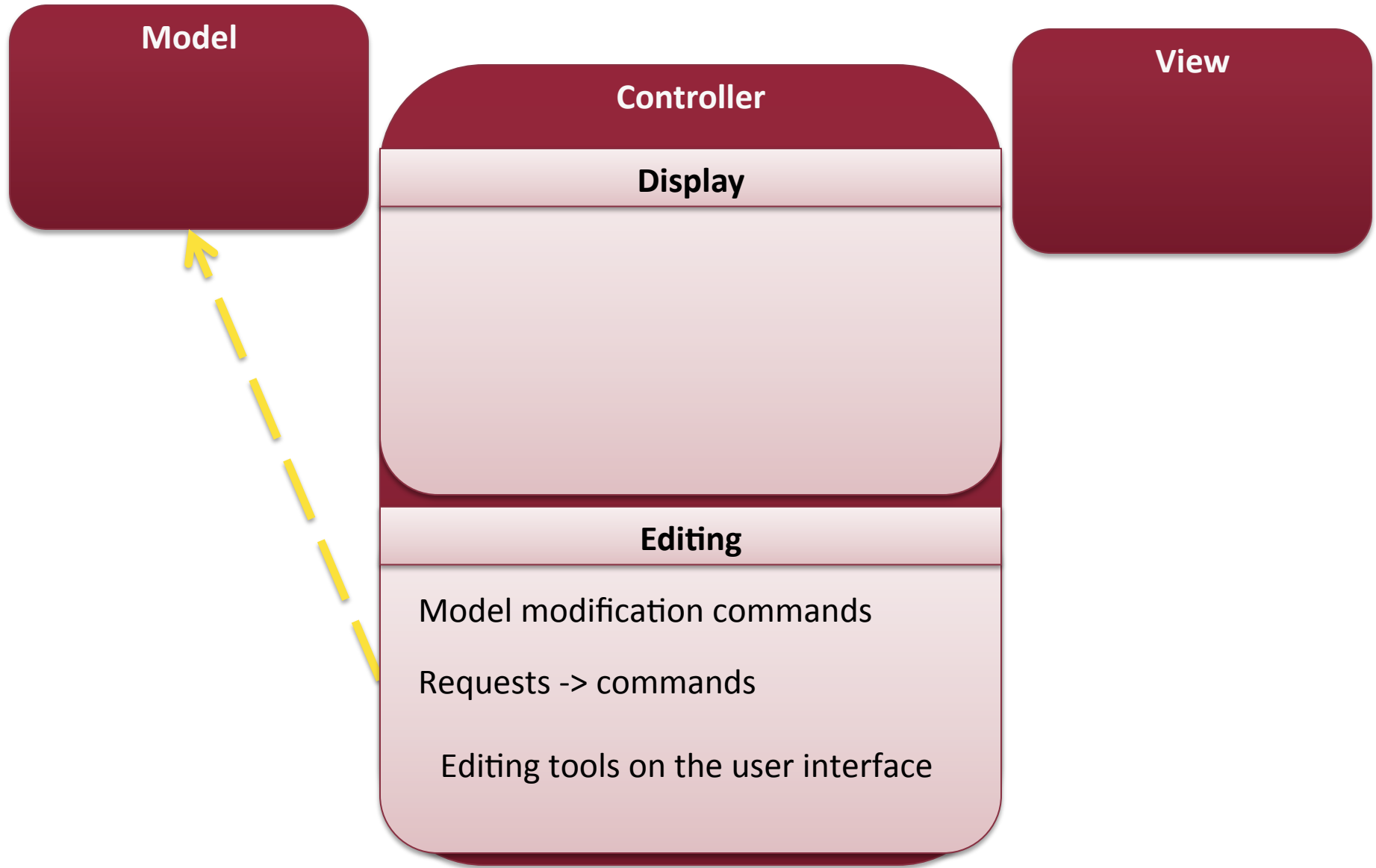
GEF workflow



GEF workflow



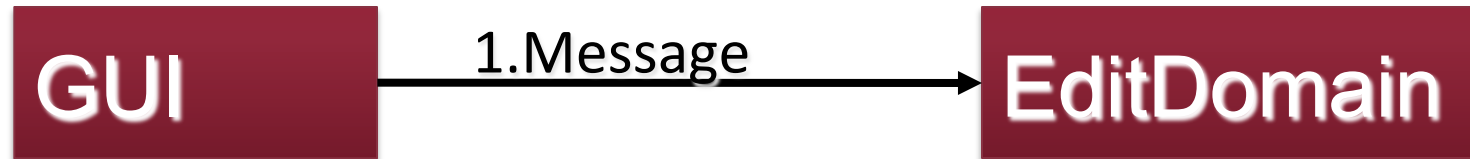
GEF workflow



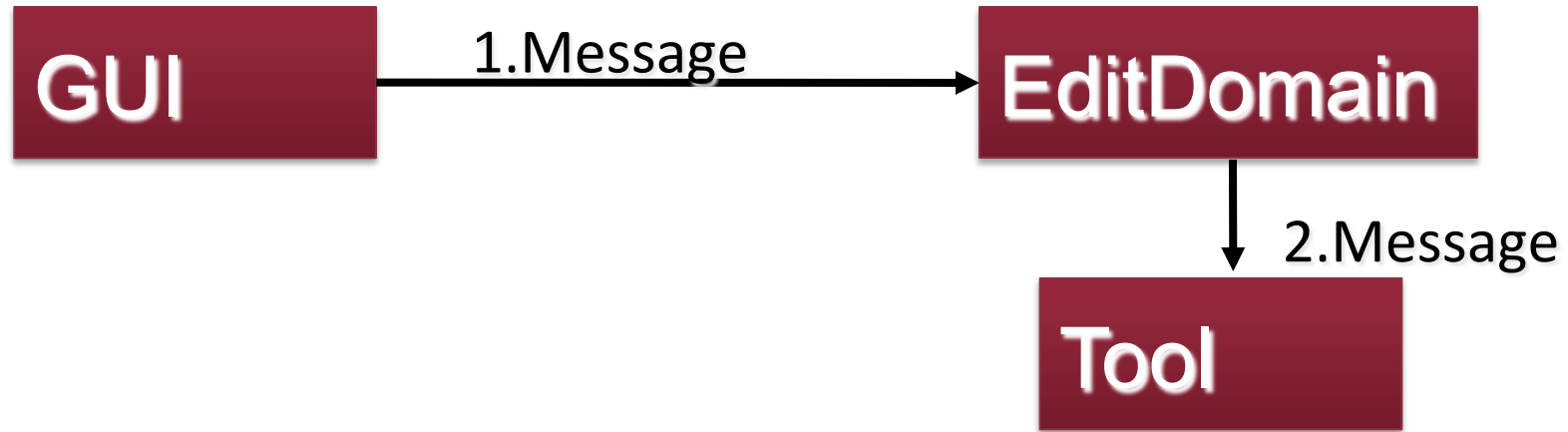
Editing Workfow

GUI

Editing Workflow



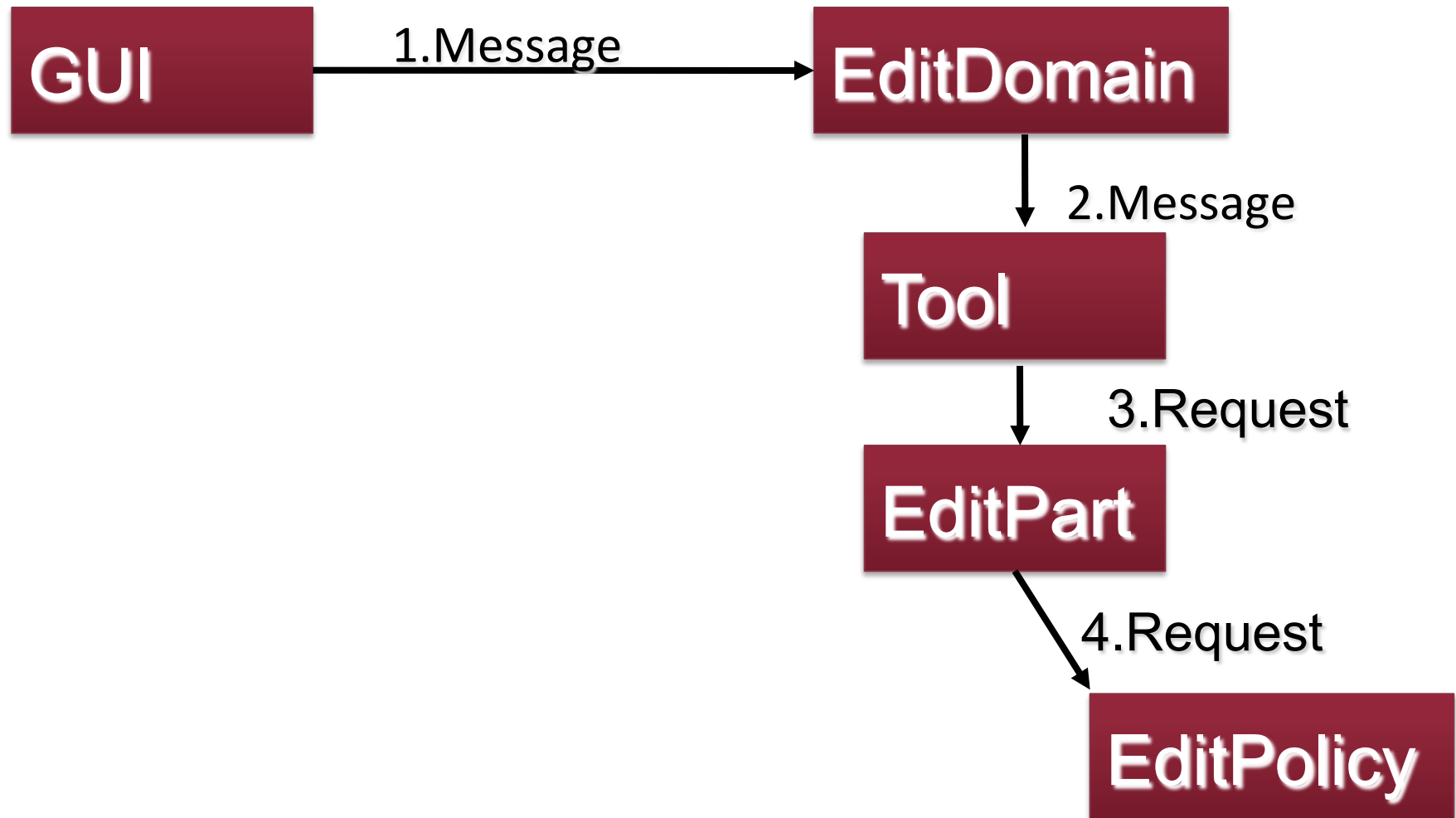
Editing Workflow



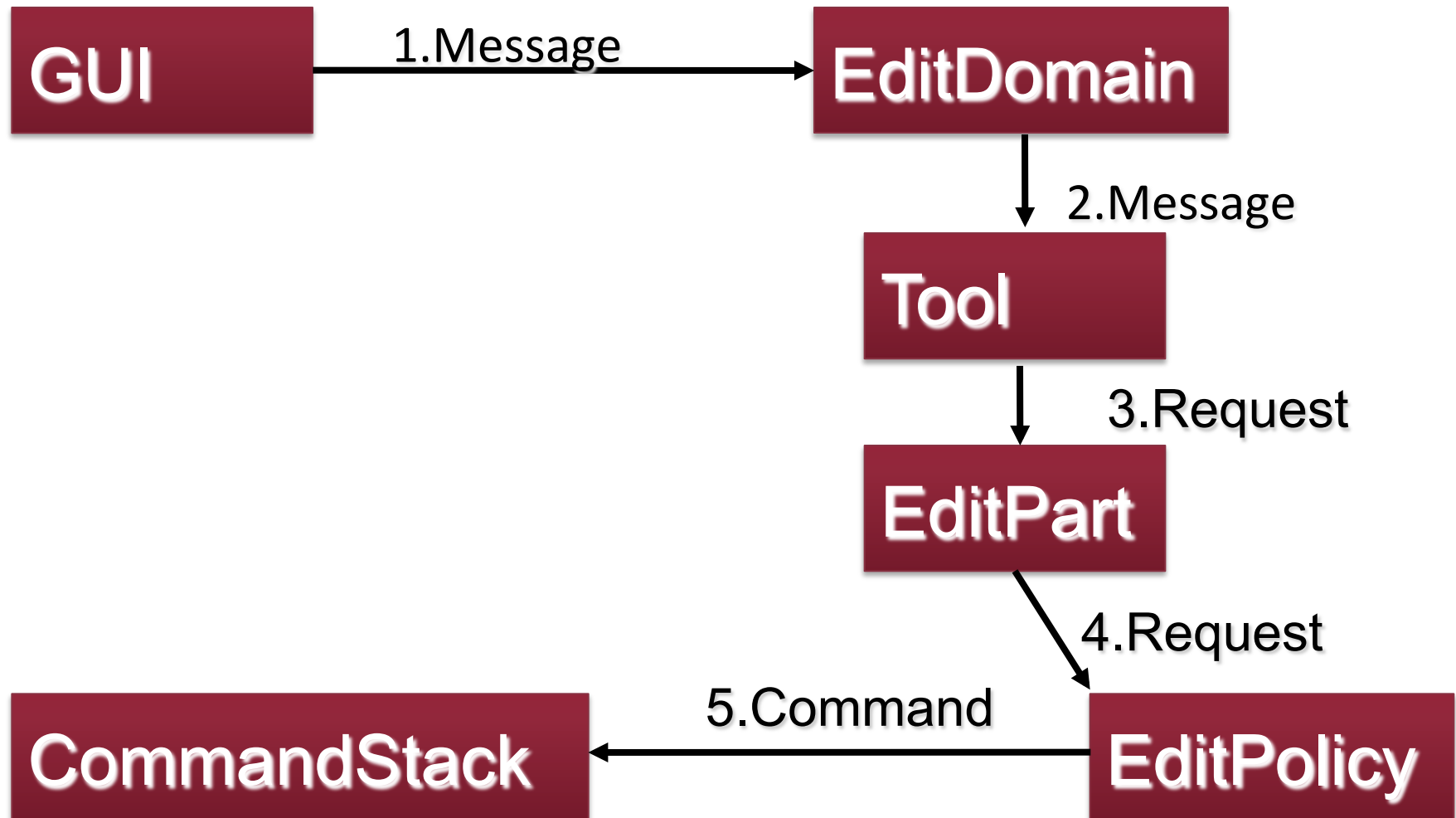
Editing Workflow



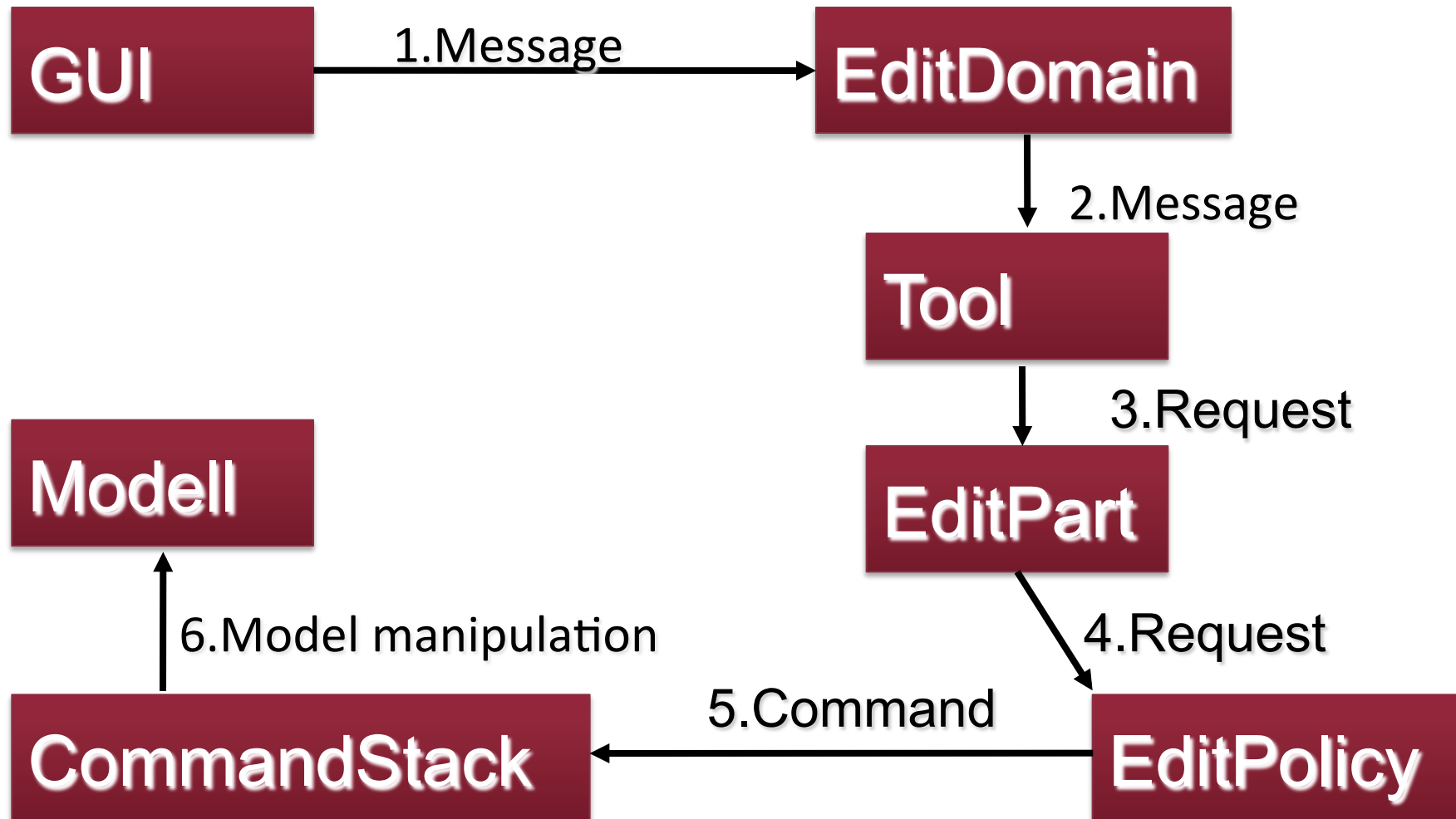
Editing Workflow



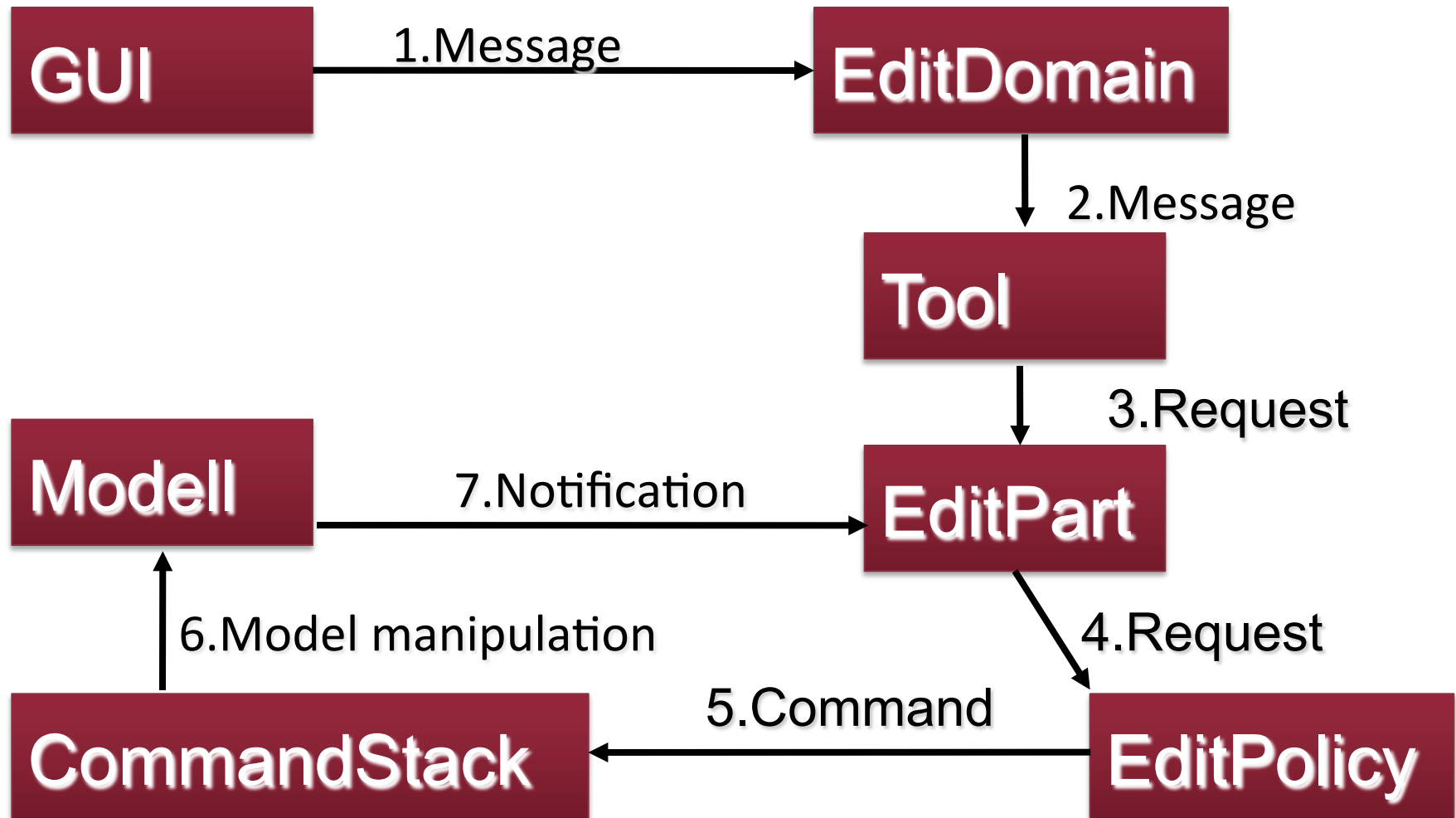
Editing Workflow



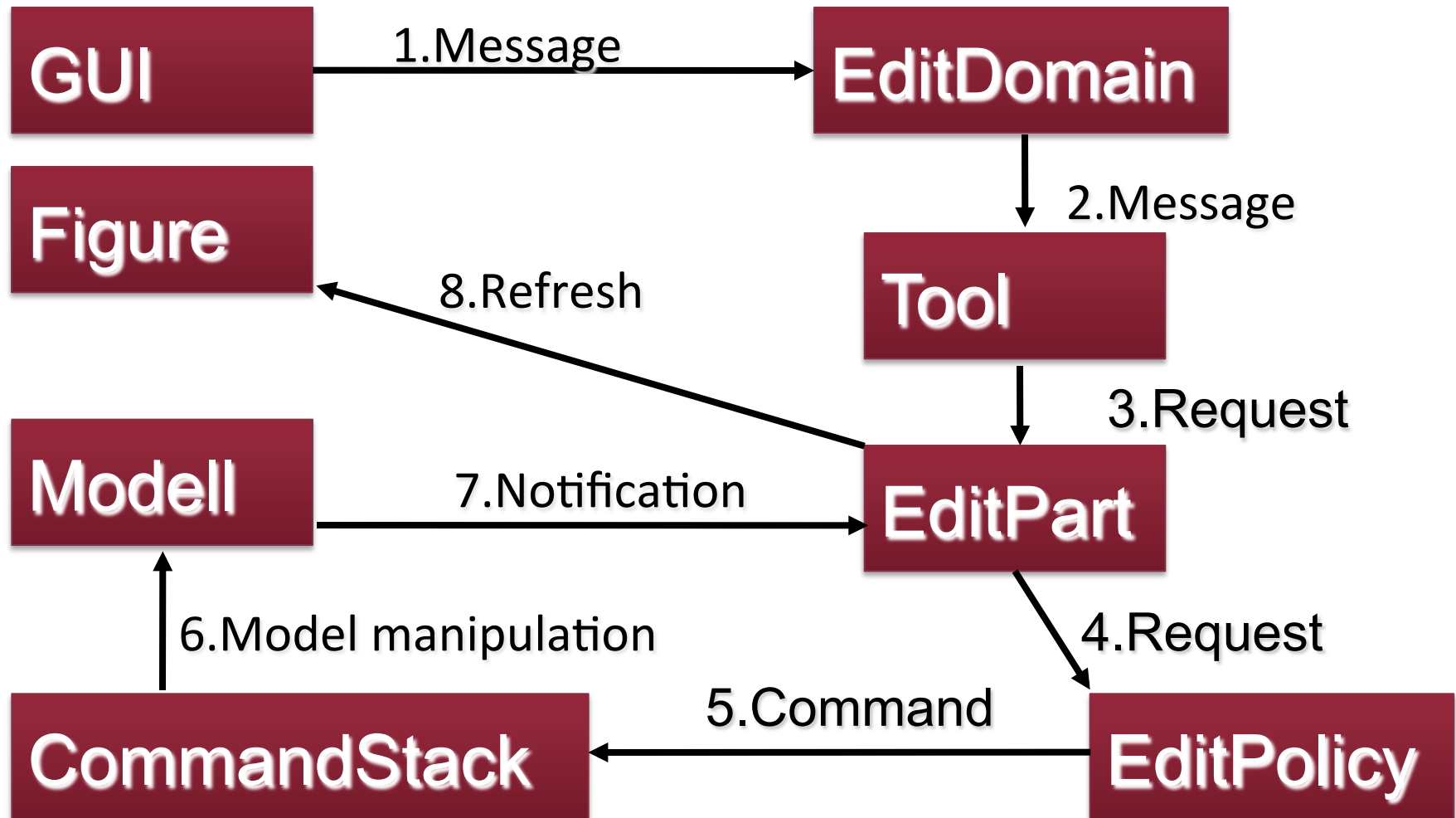
Editing Workflow



Editing Workflow



Editing Workflow



Actors of editing I.

- EditDomain
 - Dispatches GUI events to selected tool
 - Does not process
- Tool:
 - Represents an editing function
 - Processes GUI messages
 - Creates (one or more) Requests
 - E.g., SelectionTool, CreationTool, MarqueeTool
 - Custom tools can be used

Actors of Editing II.

■ Request

- GEF-level event
- Pl. CreateRequest, DeleteRequest, ChangeBoundsRequest
- Forwarded to target EditPart

■ EditPolicy

- „Editing rule” of EditPart
- Request -> Command mapping
- 1 EditPart -> multiple EditPolicies (often)

Actors of Editing III.

- EditPart
 - Based on EditPolicies creates Commands
 - Receives model change notification
 - Refreshes structure (and may create child EditPart)

Actors of Editing IV.

■ Command

- Executes model manipulation
- Undoable (if implemented 😊)
- *See EMF.Edit commands (same idea, different impl.)*

■ CommandStack

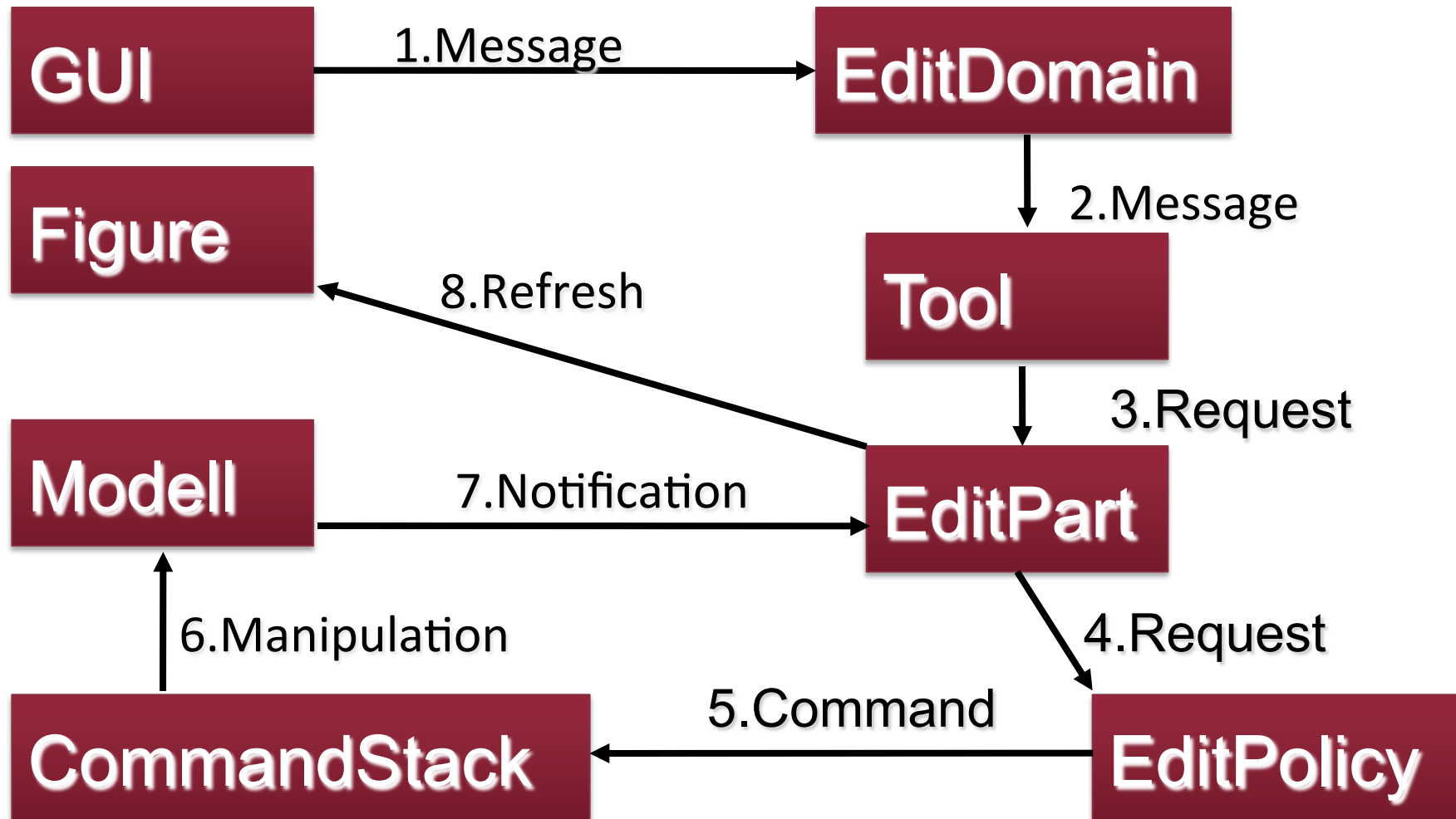
- A stack of executed commands
- Undo/redo executed over this
- One for each EditDomainen
- Always execute commands through stack
- *See EMF.Edit EditingDomain*

Actors of Editing V.

■ Action

- Not GEF-specific (JFace)
- Custom commands
 - Menu items
- GEF provides wrappers to connect to CommandStack
- ActionRegistry: list of actions

Editing workflow



What to implement? - 1.

- Model code with notification support
 - Generatable with EMF
- View classes
- EditPart classes 1.
 - Model display
 - createFigure(), refreshVisuals()
 - Model change listening
 - activate(), deactivate()
- EditPartFactory (model -> EditPart)

What to implement? – 2.

- Model modification Commands
- Custom EditPolicies
 - Defines available operations
- EditPart classes 2.
 - EditPolicy assignment
- Editor and related parts
 - EditPartViewer, Palette

GEF workflow

Model

Notification

Controller

Display

Connection to
model

Connection to
view

Hierarchy creation

View refresh on model update

Editing

Model modification commands

Edit requests -> commands

Editing tools on user interface

View

Drawing

Layout

GEF workflow

Model

Notification

EditPartFactory

Controller

Display

Connection to
model

Connection to
view

Hierarchy creation

View refresh on model update

Editing

Model modification commands

Edit requests -> commands

Editing tools on user interface

View

Drawing

Layout

Model

Notification

EditPartFactory

Controller

Display

Connection to
model

Connection to
view

Hierarchy creation

View refresh on model update

Editing

Model modification commands

Edit requests -> commands

Editing tools on user interface

View

Drawing

Layout

Model

Notification

Controller

Display

Connection to
model

Connection to
view

Hierarchy creation

View refresh on model update

Editing

Model modification commands

Edit requests -> commands

Editing tools on user interface

View

Drawing

Layout

EditPartFactory

LayoutManager

Model

Notification

Controller

Display

Connection to
model

Connection to
view

Hierarchy creation

View refresh on model update

Editing

Model modification commands

Edit requests -> commands

Editing tools on user interface

View

Drawing

Layout

EditPartFactory

LayoutManager

EditPart

Model

Notification

Controller

Display

Connection to
model

Connection to
view

Hierarchy creation

View refresh on model update

Editing

Model modification commands

Edit requests -> commands

Editing tools on user interface

View

Drawing

Layout

EditPartFactory

LayoutManager

Command

EditPart

GEF w

Figure

Model

Notification

Controller

Display

Connection to
model

Connection to
view

Hierarchy creation

View refresh on model update

Editing

Model modification commands

Edit requests -> commands

Editing tools on user interface

View

Drawing

Layout

EditPartFactory

LayoutManager

Command

EditPart

EditPolicy

GEF w

Figure

Model

Notification

Controller

Display

Connection to
model

Connection to
view

Hierarchy creation

View refresh on model update

Editing

Model modification commands

Edit requests -> commands

Editing tools on user interface

View

Drawing

Layout

EditPartFactory

LayoutManager

Command

EditPart

Tool

EditPolicy

Arrows (connectors)

- Similar to object
 - With important distinctions
- Display in higher layer
- Connector Edit Parts
 - AbstractConnectionEditPart descendants
 - EditPolicy, Request, etc.
- Directed connection (on model level)
- *Not detailed here*

Further Capabilities

- Eclipse Properties view support
- Direct editing of labels
- Zoom support
- Arrangement
- Separate tree and graphical outline
- *Not detailed here*

GEF – Further materials

- Create an Eclipse-based application using the Graphical Editing Framework
 - IBM Developerworks:
<http://www.ibm.com/developerworks/library/os-eclipse-gef11/>
- Vainolo tutorials:
 - <http://www.vainolo.com/tutorials/>
- GEF wiki:
 - http://wiki.eclipse.org/GEF/Articles%2C_Tutorials%2C_Slides

Graphiti, GMF

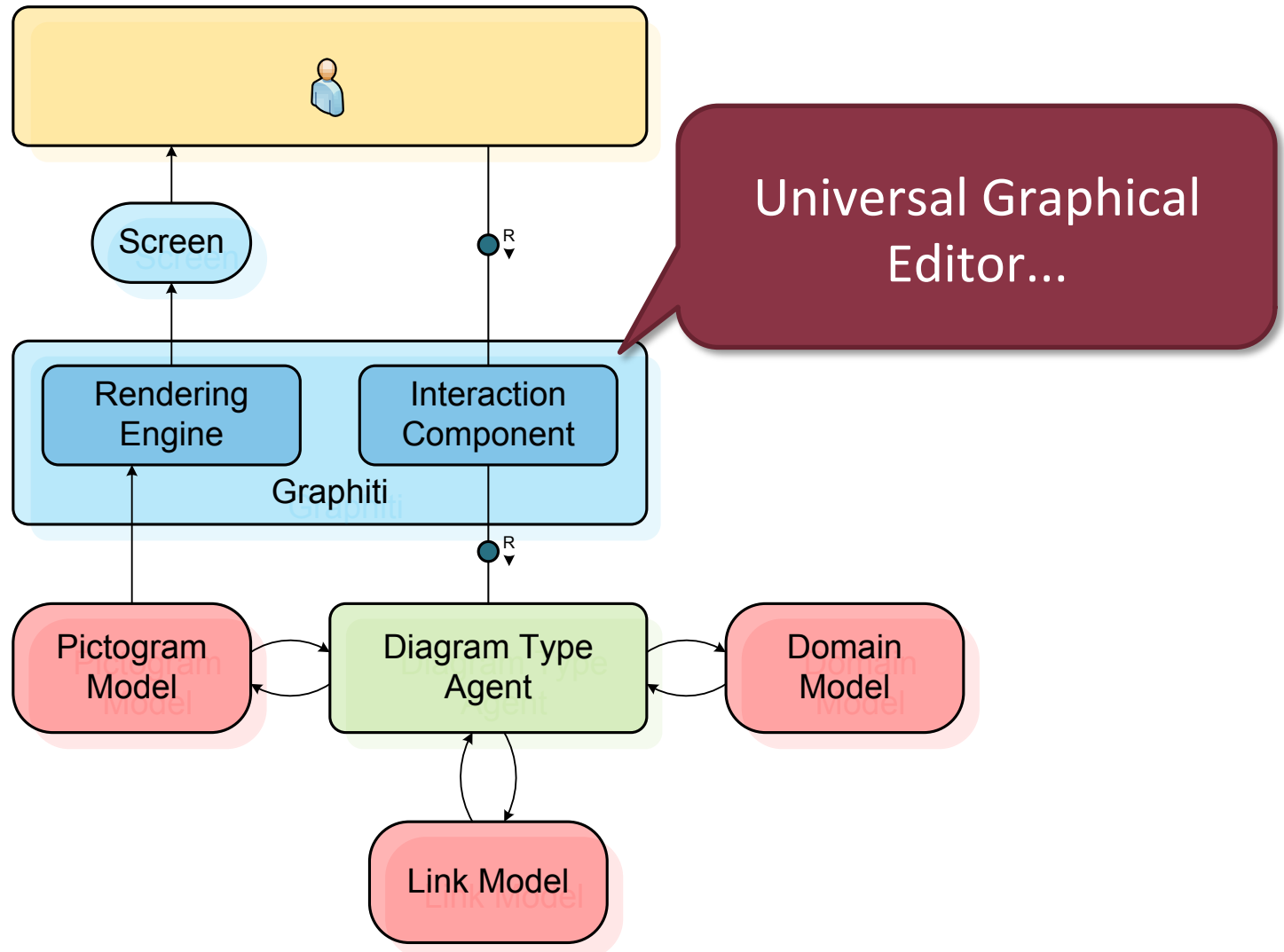
Problems using GEF

- Very complex
- Lot of redundant, hand-written code
- Example: simple GEF editor
 - Two node types with edges between them
 - 3400 lines of code (w/o model)
 - 16 classes, 150 methods...
- Something more simple is needed

Simplification

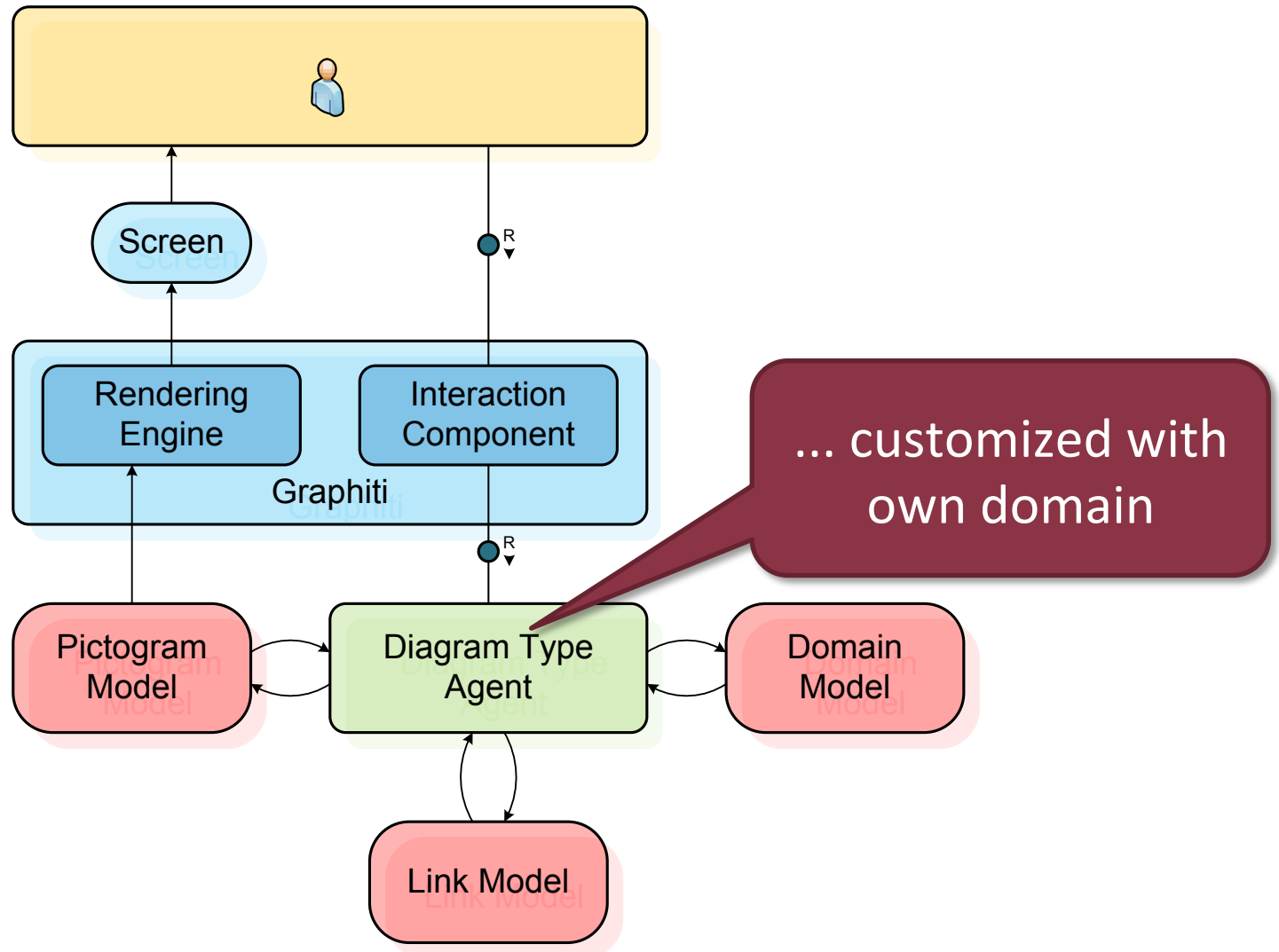
- EMF models
 - Uniform handling
 - Load/save simple
 - Interfile dependencies manageable
- Simpler interaction
 - Only high-level functions directly supported

Architecture of Graphiti



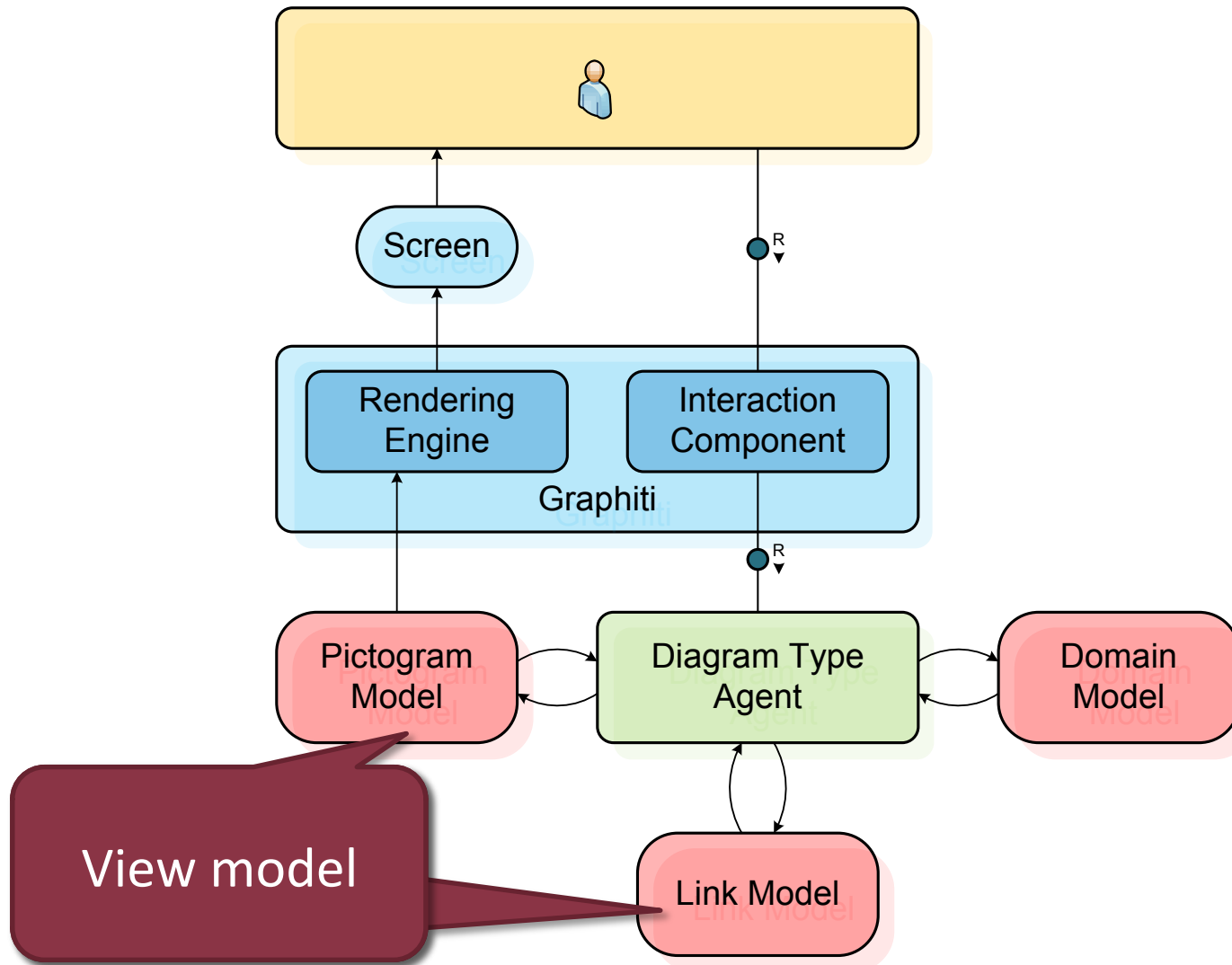
Source: <http://www.slideshare.net/michaelwenz/short-talk-on-graphiti-at-eclipsecon-2010>

Architecture of Graphiti



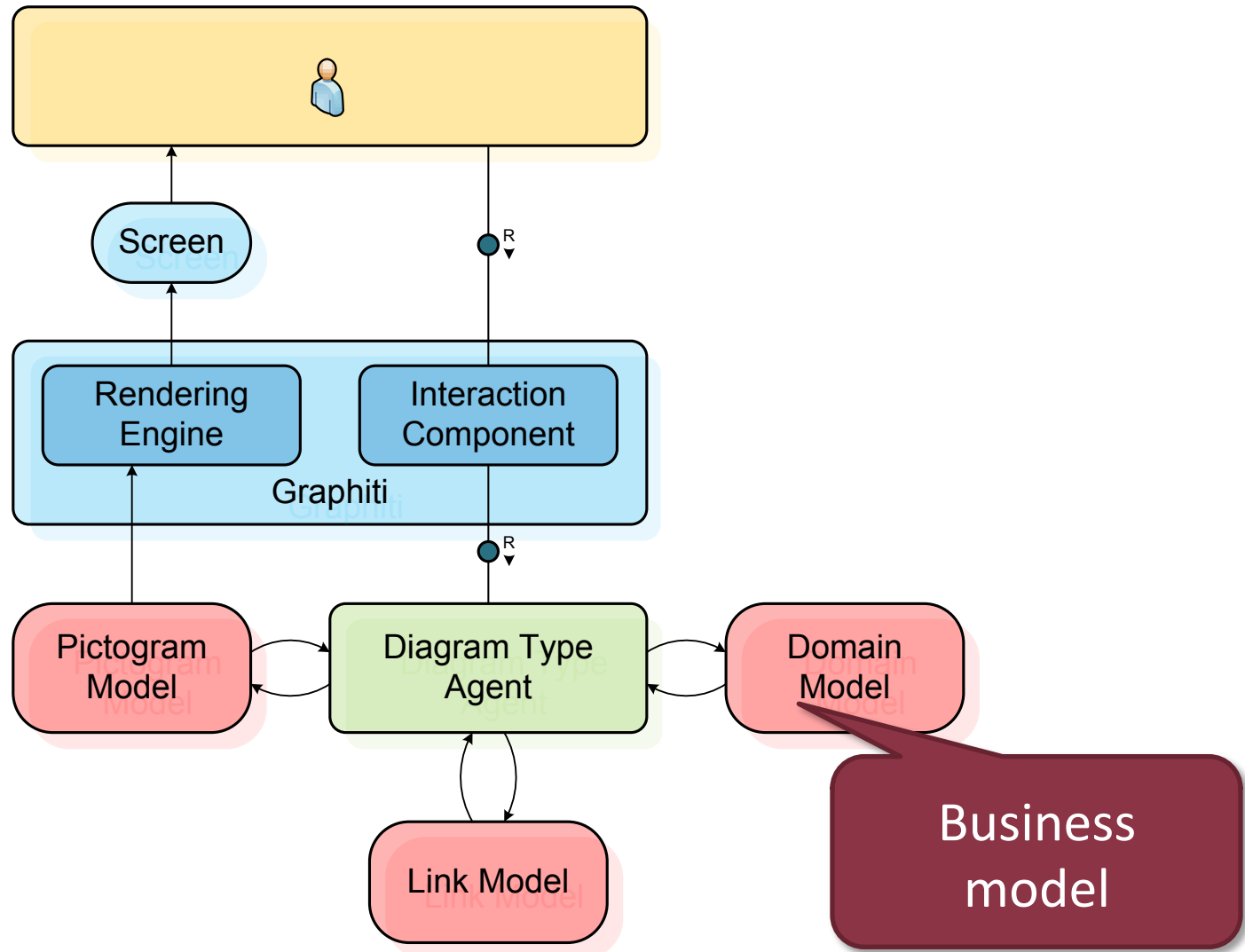
Source: <http://www.slideshare.net/michaelwenz/short-talk-on-graphiti-at-eclipsecon-2010>

Architecture of Graphiti



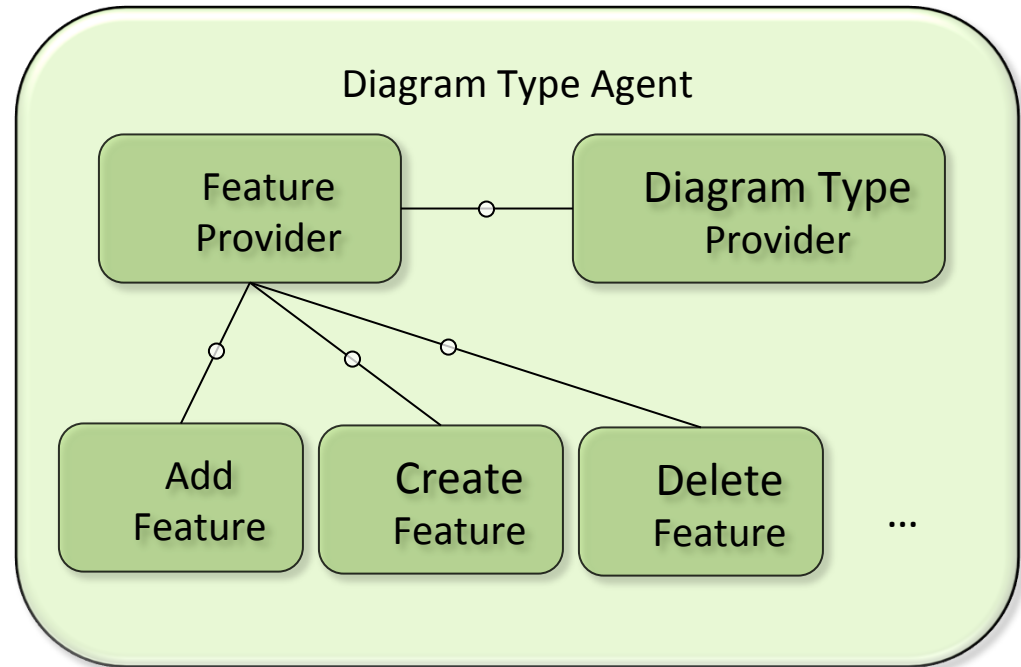
Source: <http://www.slideshare.net/michaelwenz/short-talk-on-graphiti-at-eclipsecon-2010>

Architecture of Graphiti



Source: <http://www.slideshare.net/michaelwenz/short-talk-on-graphiti-at-eclipsecon-2010>

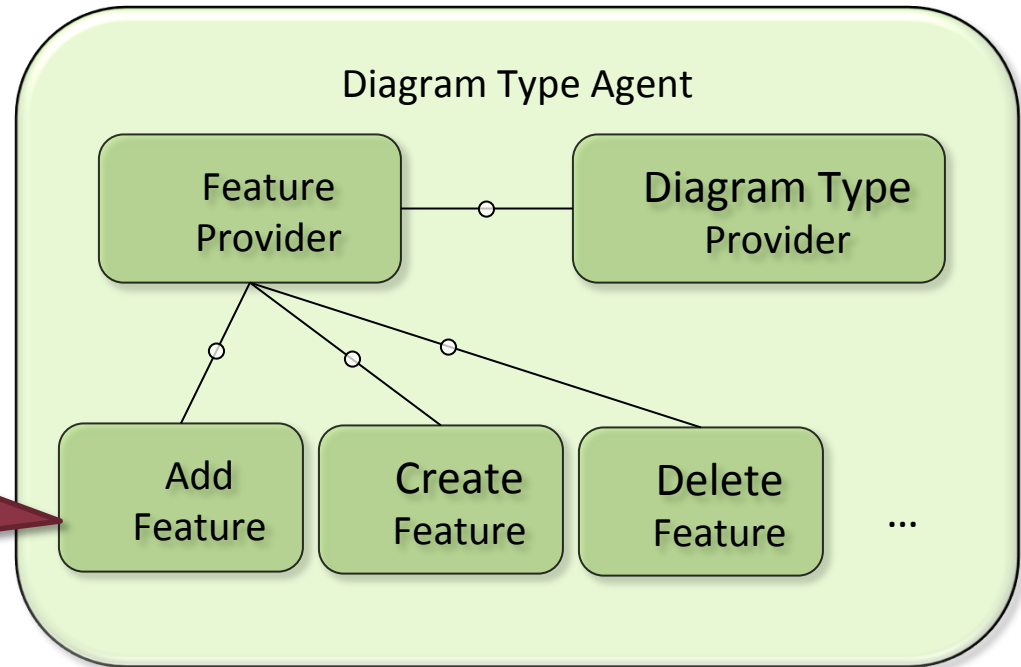
Graphiti architecture 2. – Diagram Type Agent



Source: <http://www.slideshare.net/michaelwenz/short-talk-on-graphiti-at-eclipsecon-2010>

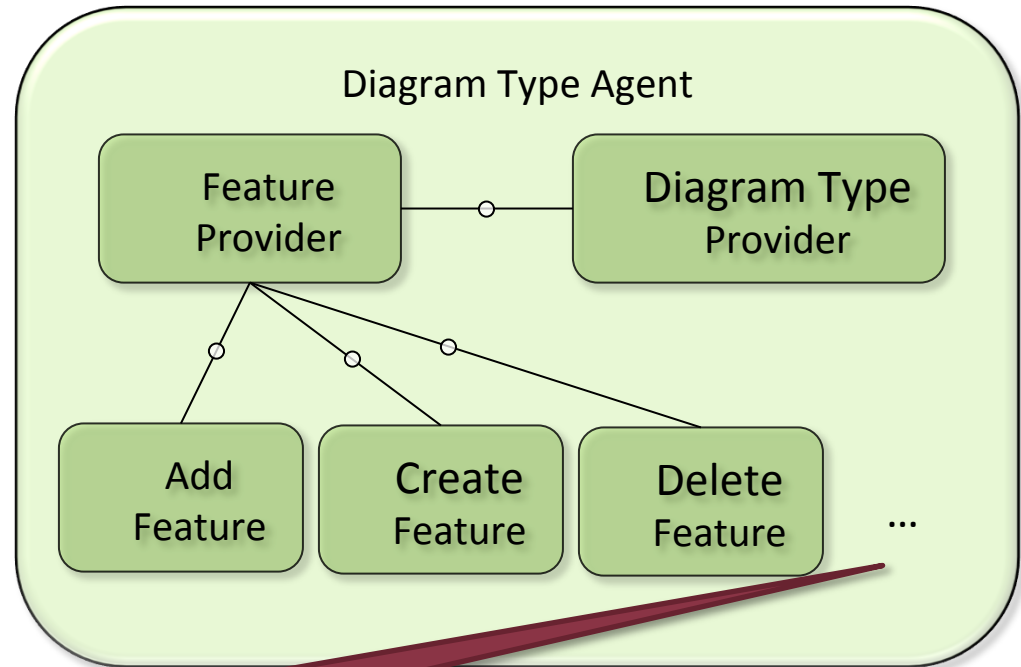
Graphiti architecture 2. – Diagram Type Agent

Feature: editing operation on business model



Source: <http://www.slideshare.net/michaelwenz/short-talk-on-graphiti-at-eclipsecon-2010>

Graphiti architecture 2. – Diagram Type Agent



Most common user
commands available

Source: <http://www.slideshare.net/michaelwenz/short-talk-on-graphiti-at-eclipsecon-2010>

Basic terms

Domain

Links

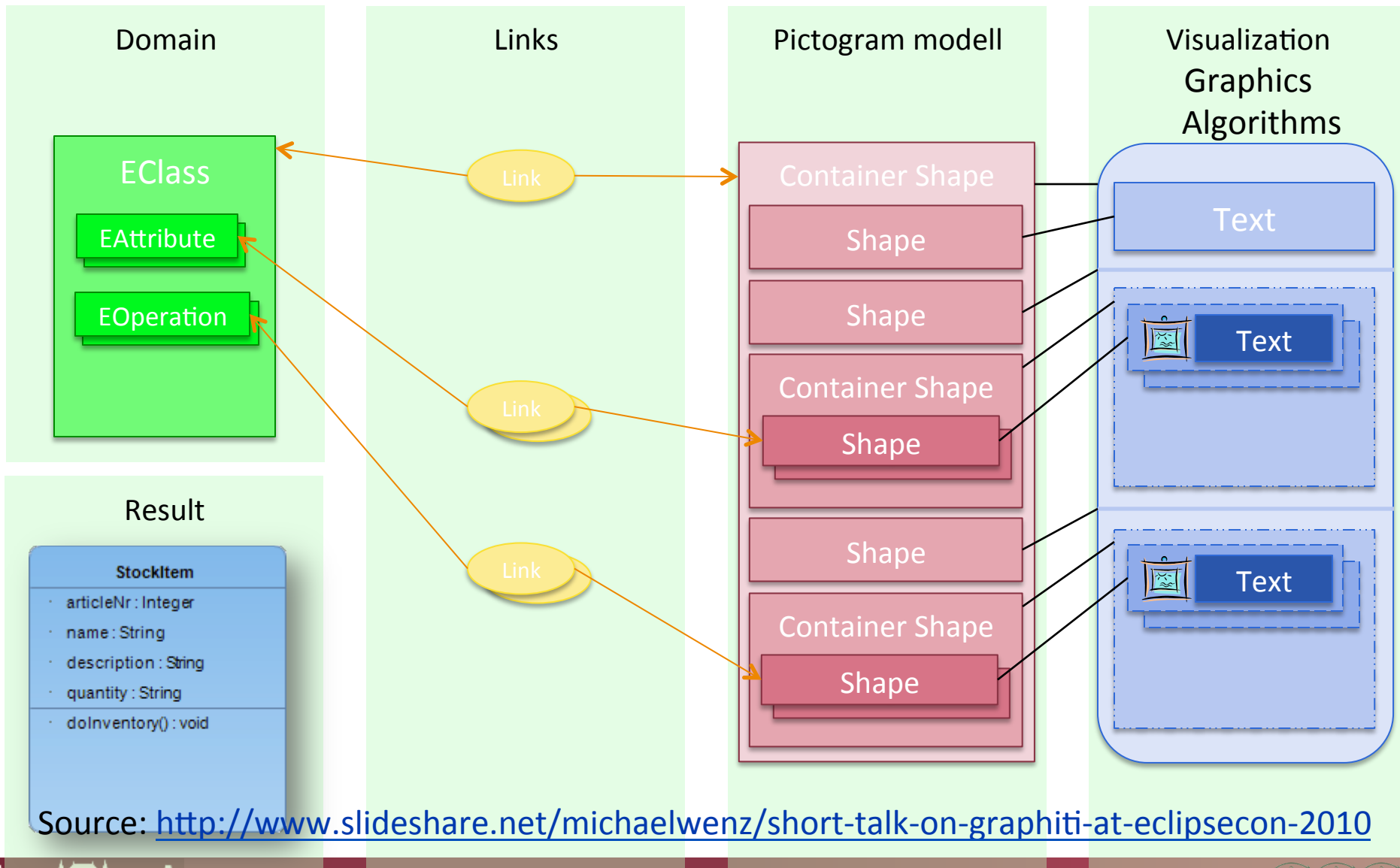
Pictogram modell

Visualization
Graphics
Algorithms

Result

Source: <http://www.slideshare.net/michaelwenz/short-talk-on-graphiti-at-eclipsecon-2010>

Basic terms



Pictogram and link model

■ Pictogram metamodel

- Displayable object
- EMF model
- Metamodel available: <http://eclipse.org/graphiti/images/pictograms.pdf>

■ Link

- Traceability connections between pictogram and business model elements
- Generic (not business model specific)

Tool building steps

1. Diagram Type Provider implementation
2. Diagram Type Provider registration
 - Extension point
3. Feature Provider implementation
4. Feature implementation

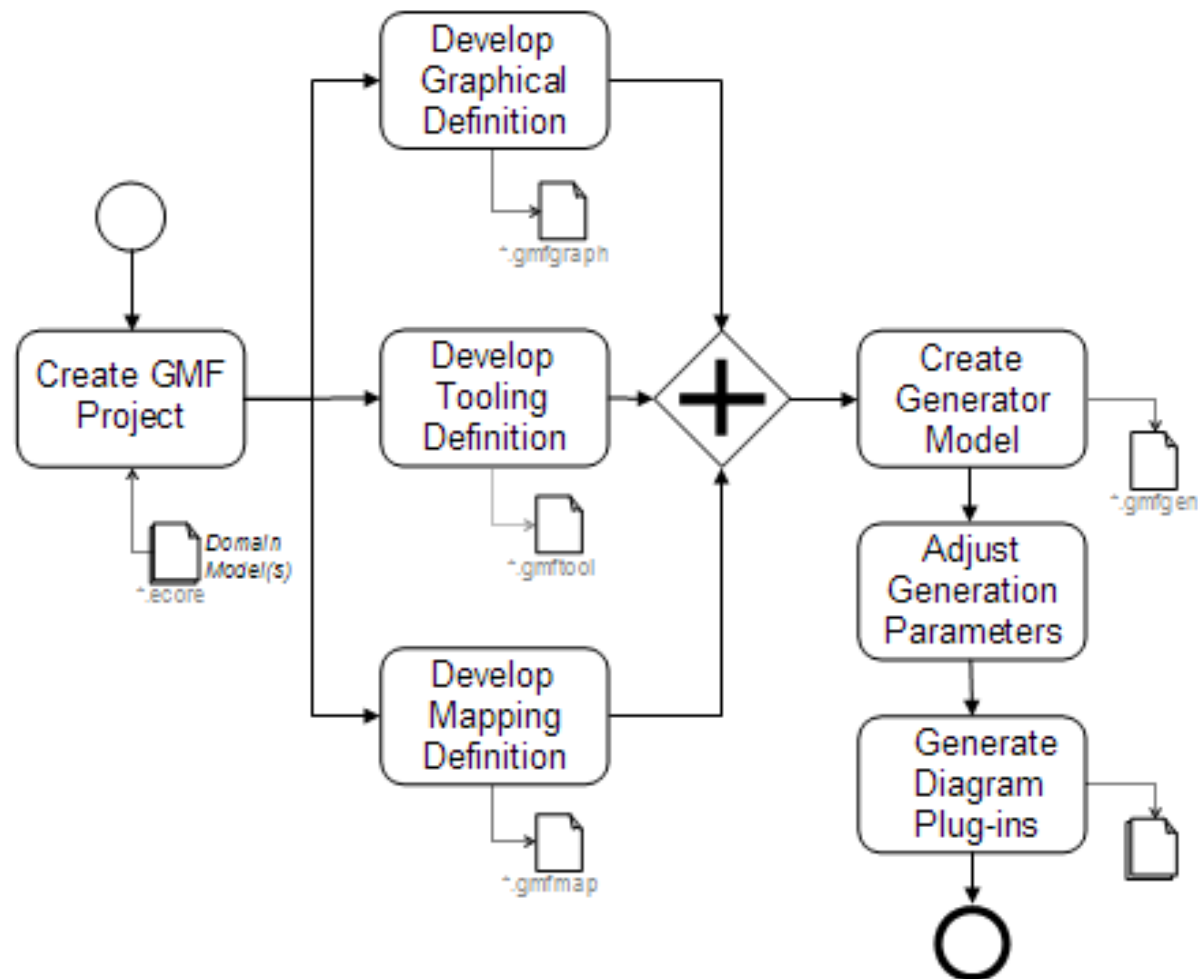
Graphiti - Summary

- Higher level library over GEF
 - Uses EMF models
 - Universal editor
- Less coding
- Uniform presentation
- But missing functions

Graphical Modeling Framework

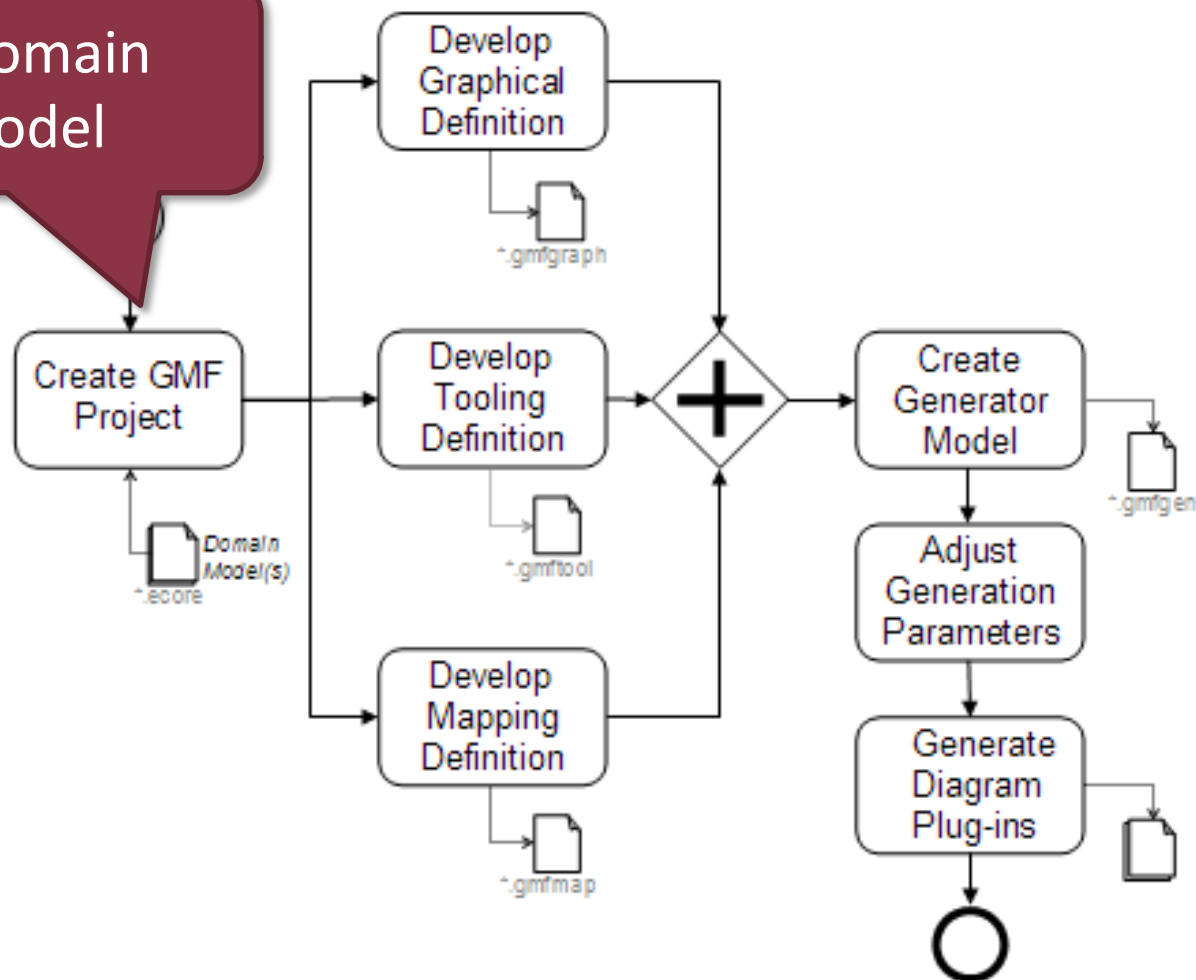
- Graphical Editors
 - Over GEF and EMF
 - Similar logic to GEF
- Code generation based tooling

Generating GMF Editors



Generating GMF Editors

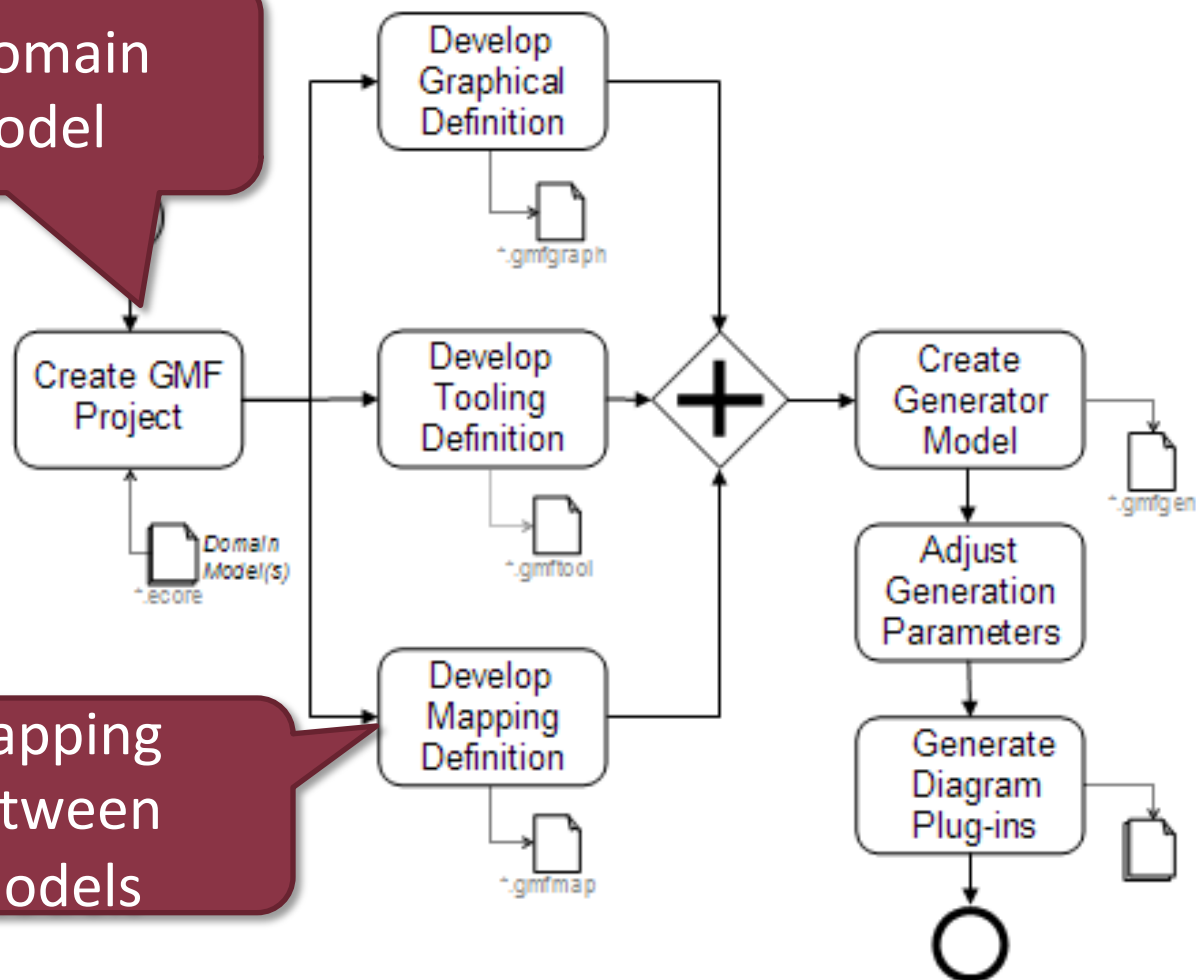
Import domain
metamodel



Generating GMF Editors

Import domain
metamodel

Mapping
between
models

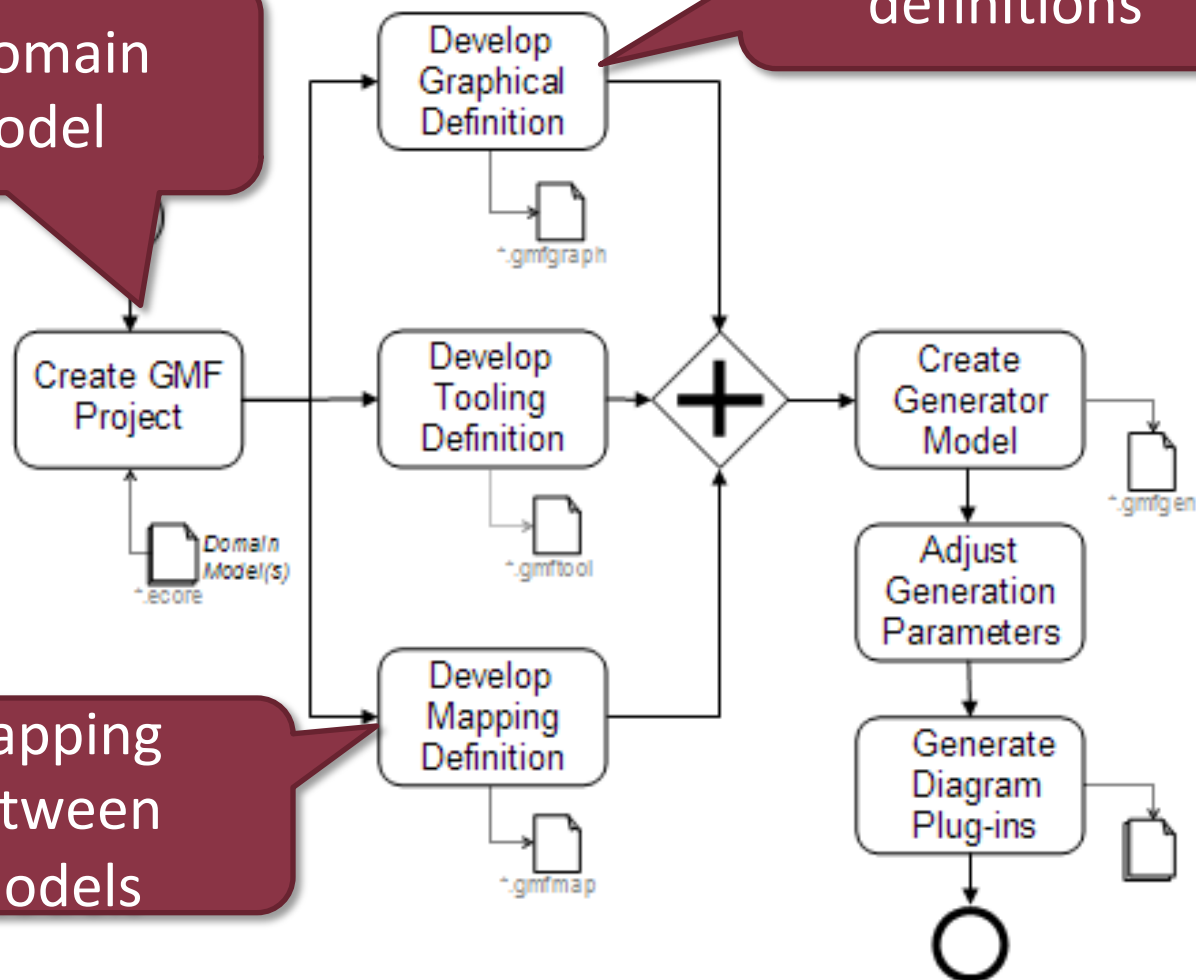


Generating GMF Editors

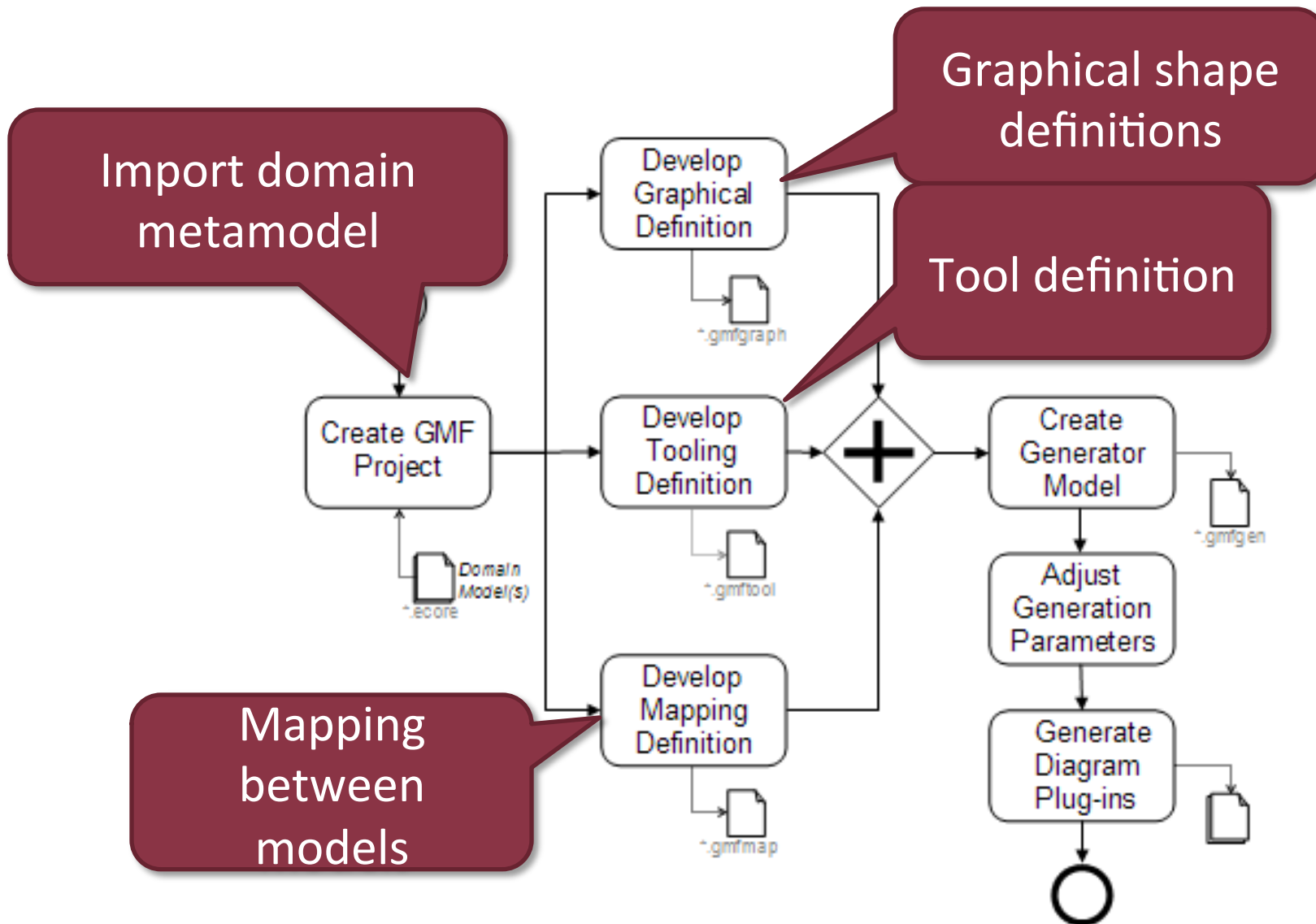
Import domain
metamodel

Graphical shape
definitions

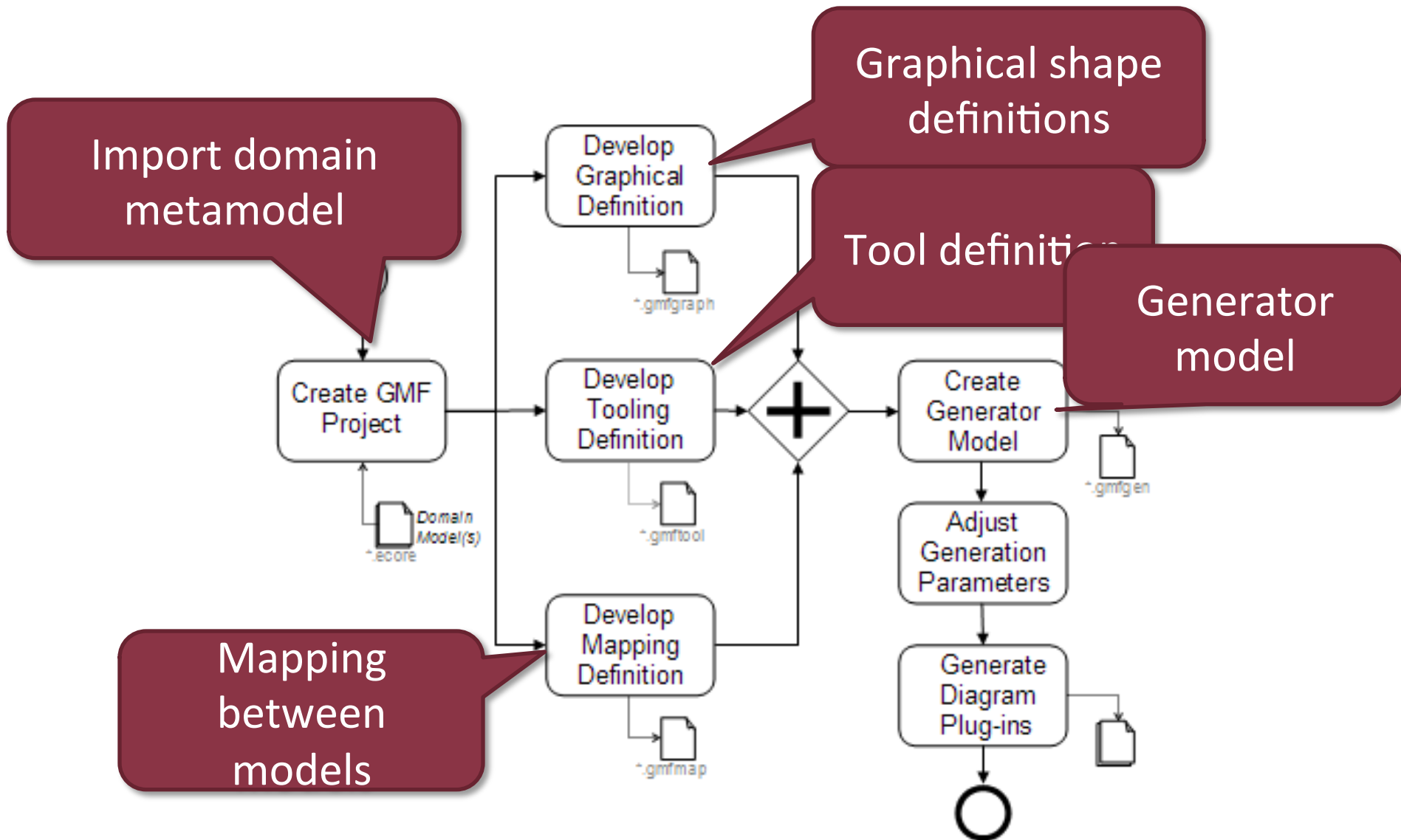
Mapping
between
models



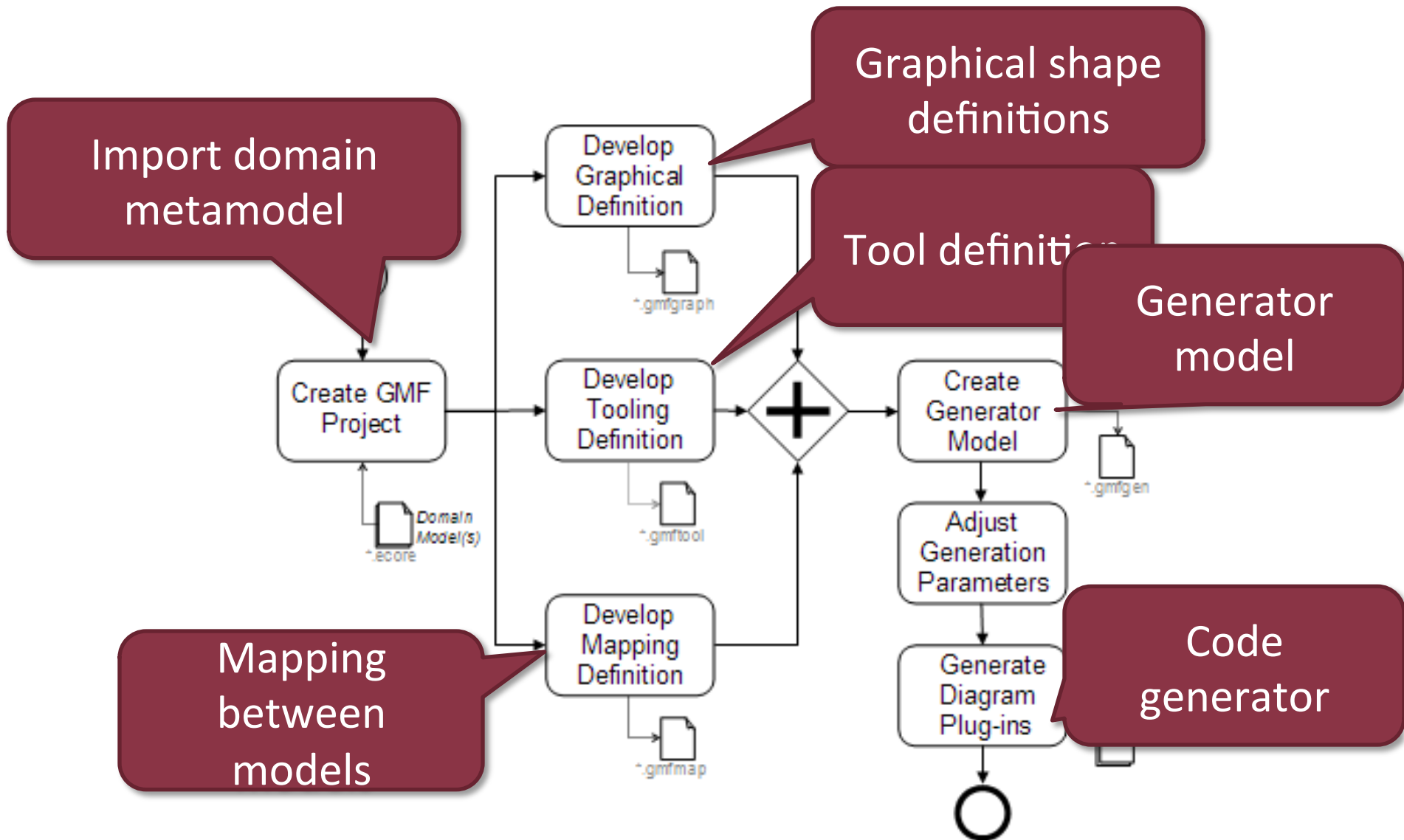
Generating GMF Editors



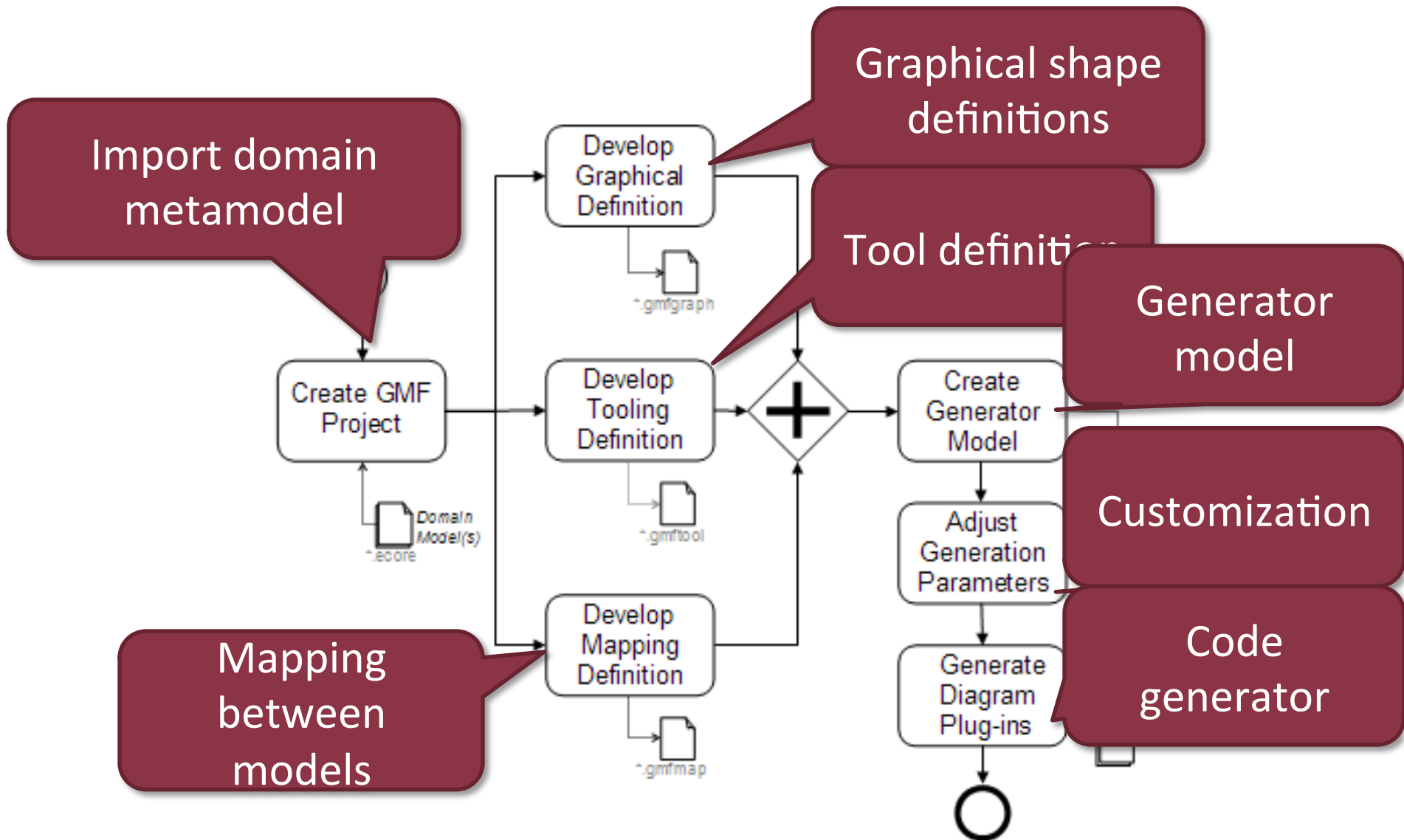
Generating GMF Editors



Generating GMF Editors

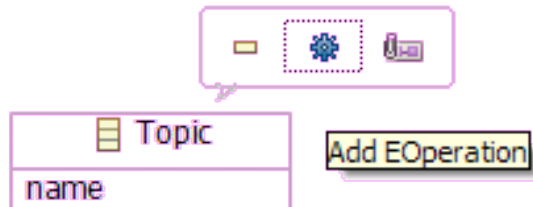


Generating GMF Editors

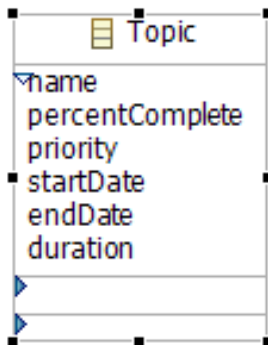


Standard components

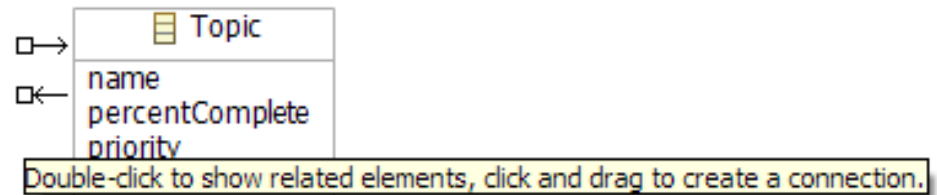
Popup Action Bar:



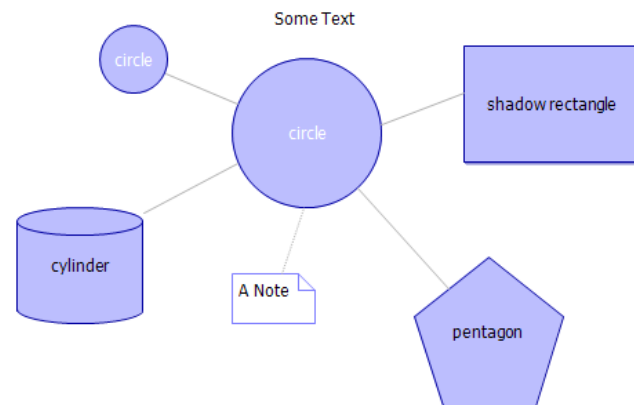
Compartment (collapsible):



Connection Handle:



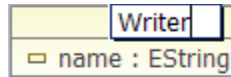
Geometrical Shape:



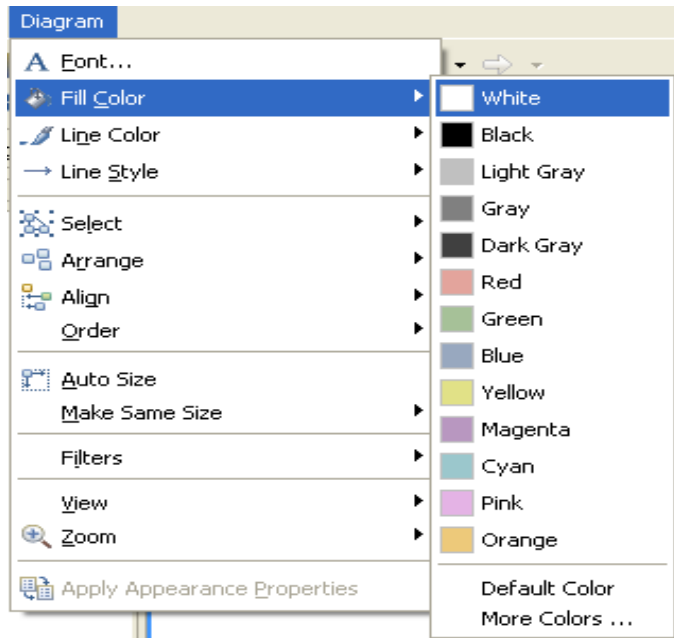
Standard components

Action:

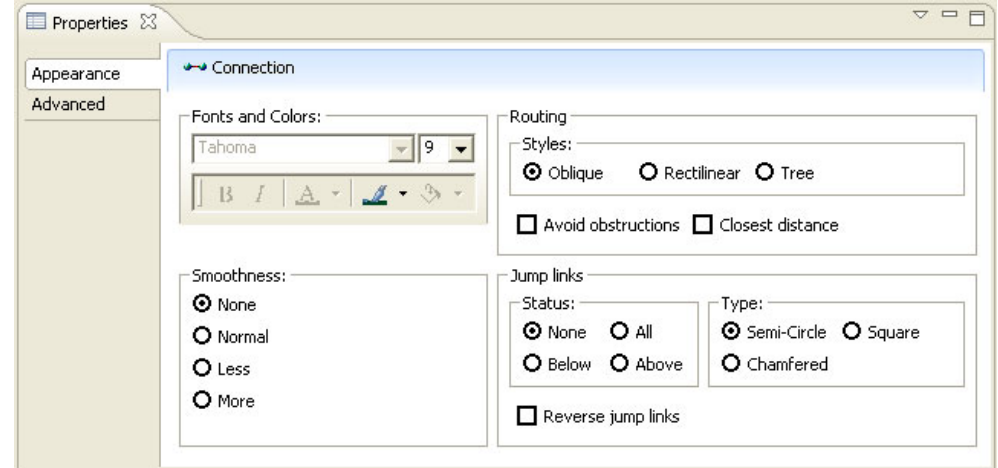
Direct Edit:



Toolbar:

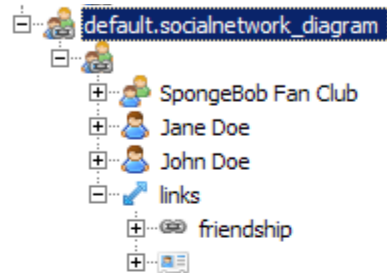


Properties View:

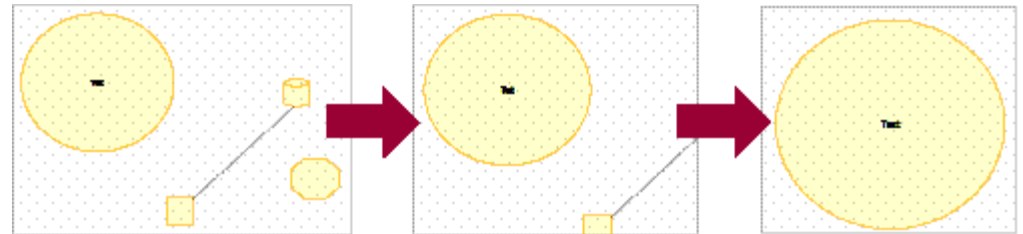


Standard components

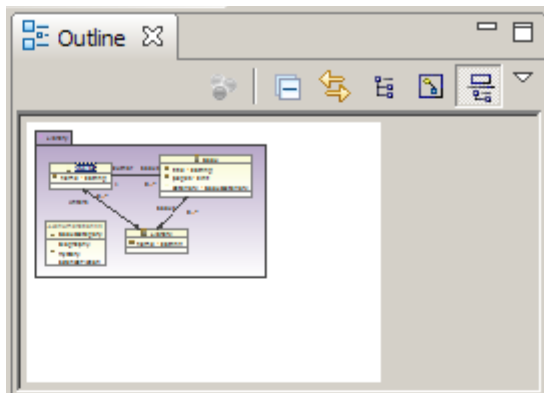
Model navigator



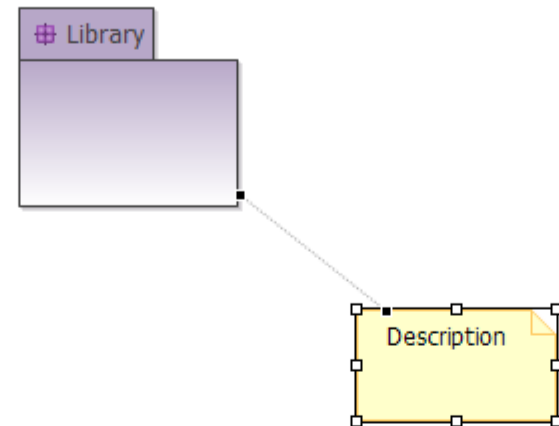
Animated zoom:



Outline view support

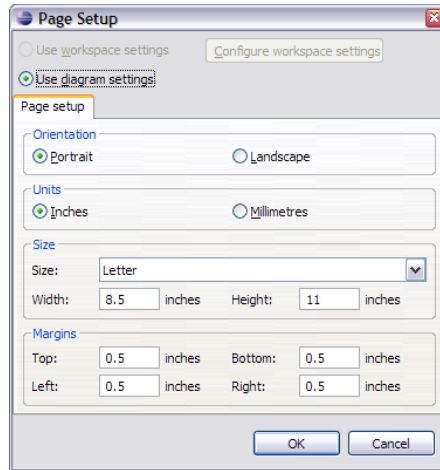


Note Attachment:



Standard components

Print setup:



Print
preview:

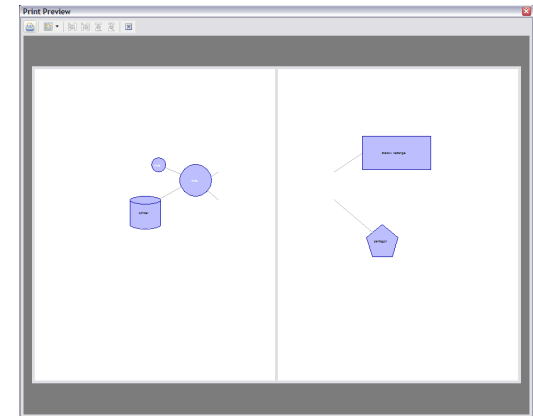


Diagram export to image

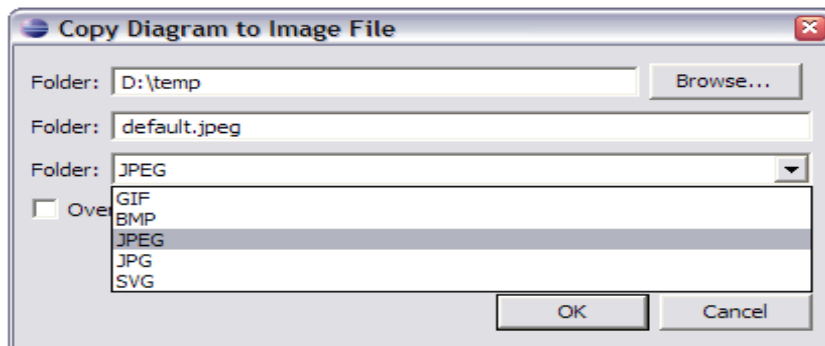
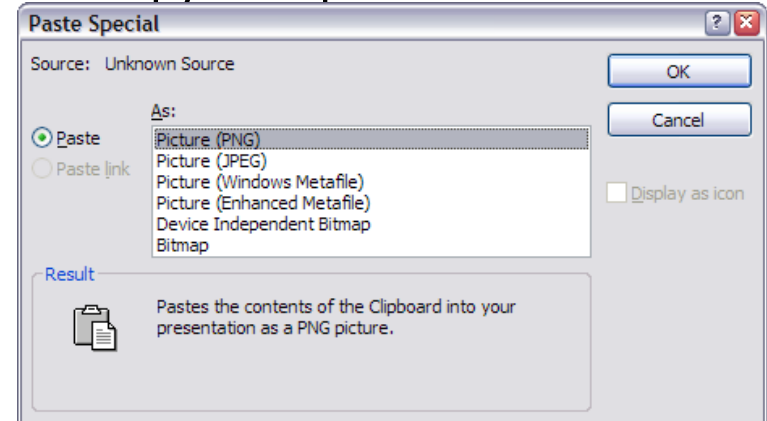


Diagram copy to clipboard

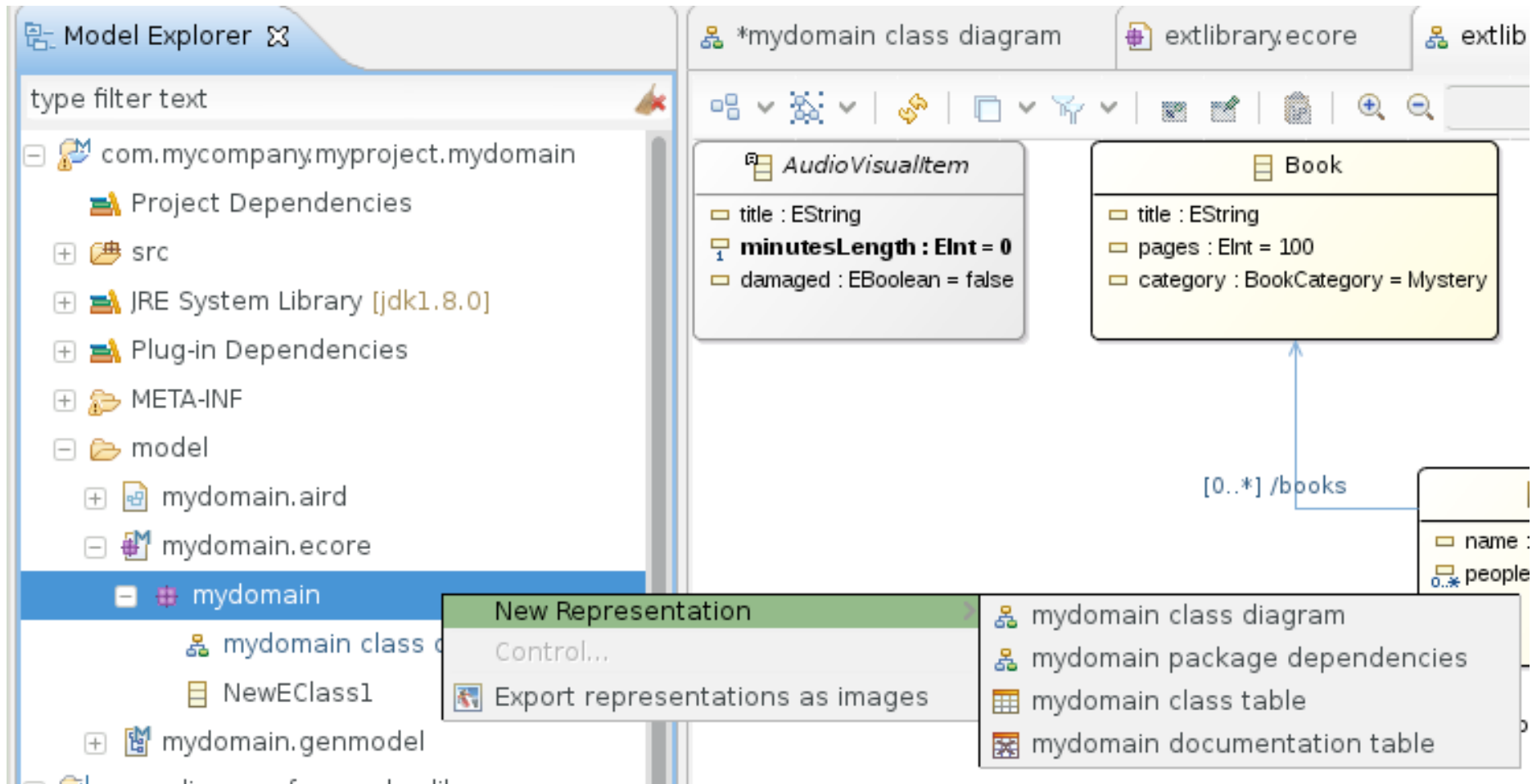


Sirius

Sirius

- New modeling project
 - Since 2013 on eclipse.org
 - Previously Obeo Designer – commercial tool
- How stable is it?
 - Old projects are to be migrated
 - Version history
 - 0.9: 2013-12-10
 - 1.0: 2014-06-25 (Kepler release train)
 - 2.0: 2014-10-24, planned
 - 3.0: 2015-06-25, planned (Mars release train)

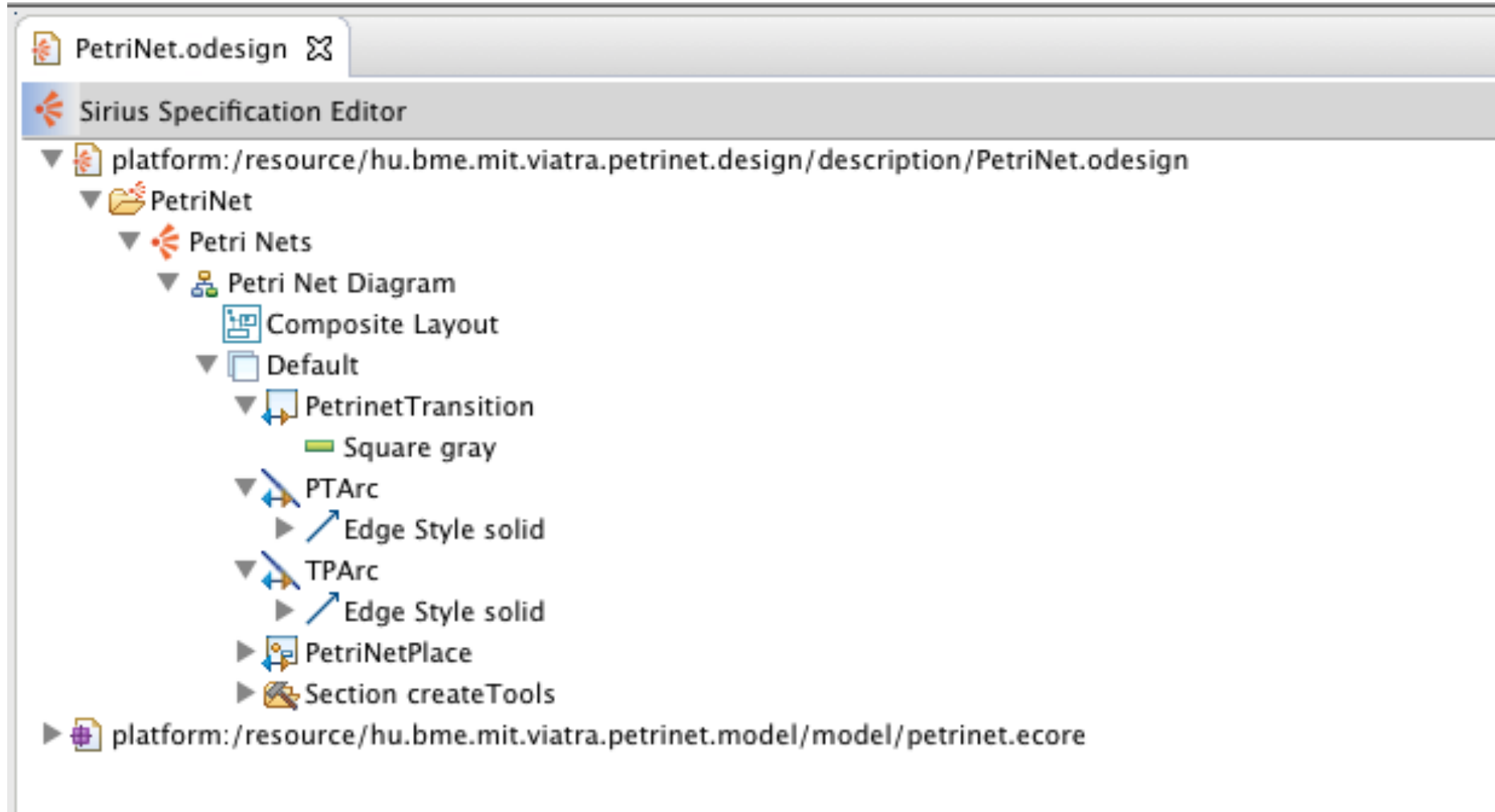
Sirius in Action – Ecore Tools 2.0



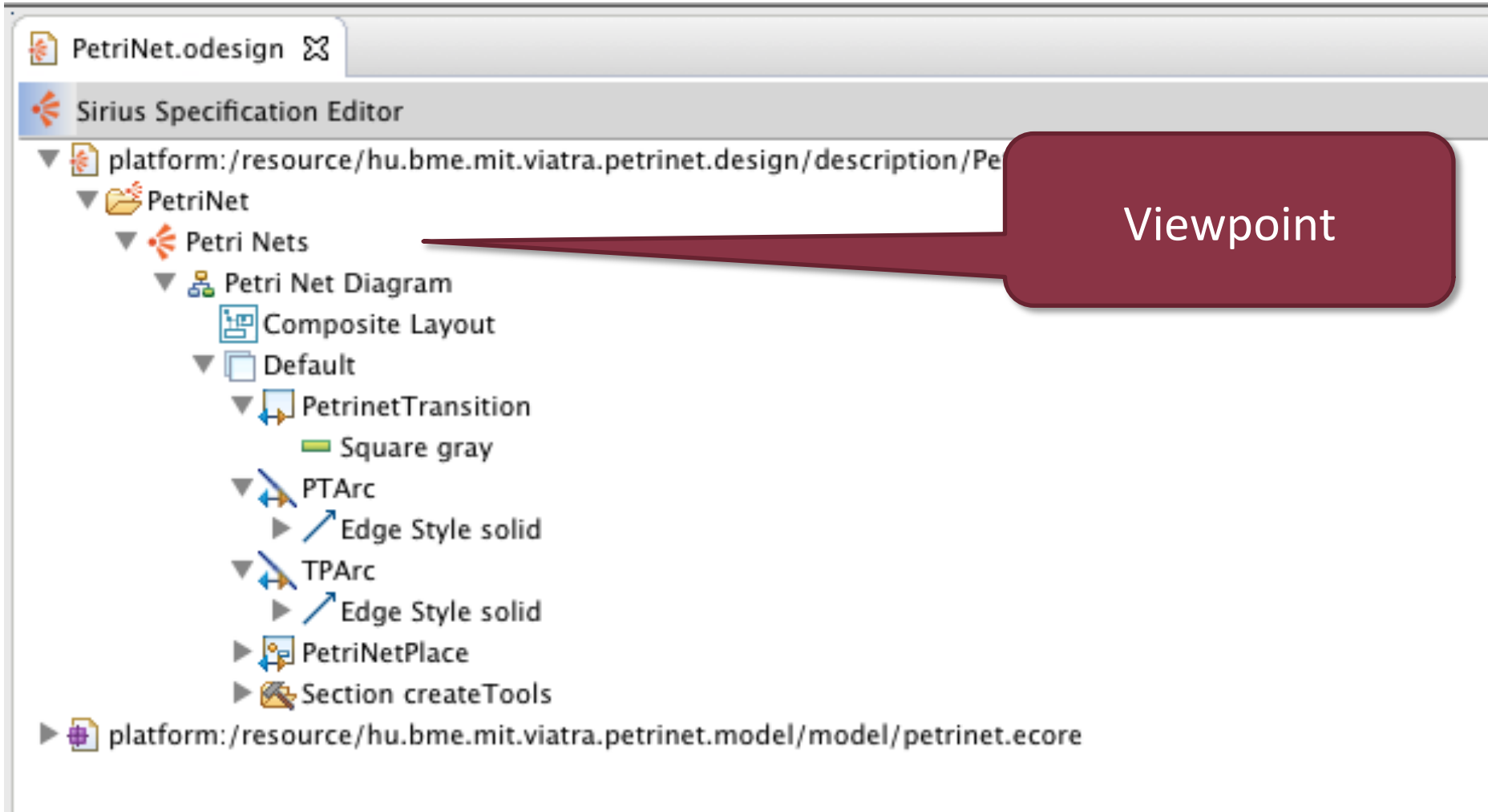
Sirius Viewpoints

- Base concept:
 - Every diagram is a view of the model
 - With a defined syntax
 - **Graphical**
 - Table/Tree syntax
 - Xtext-based textual syntax
- Viewpoint definition
 - Viewpoint specification model

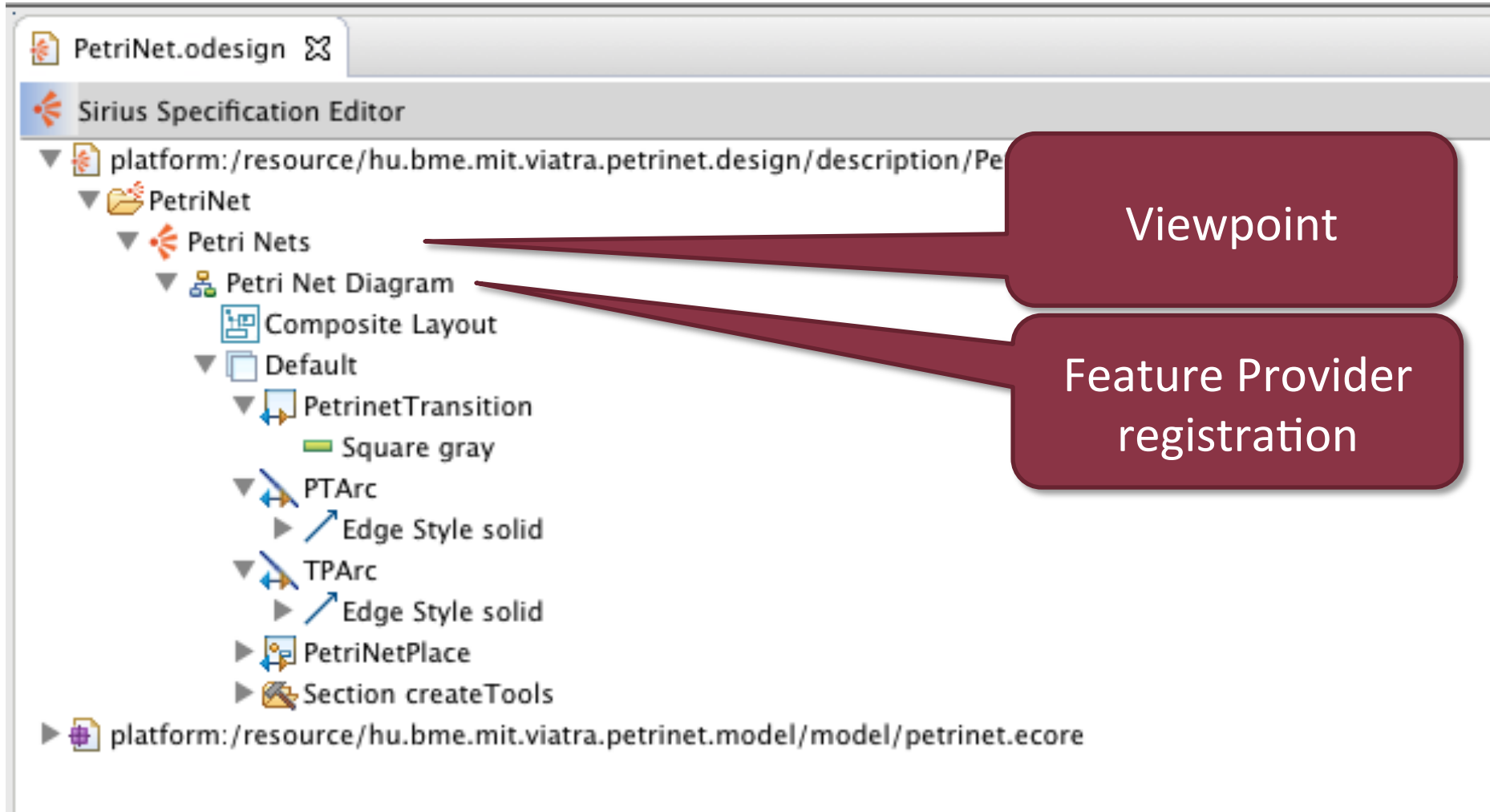
Viewpoint Specification Model



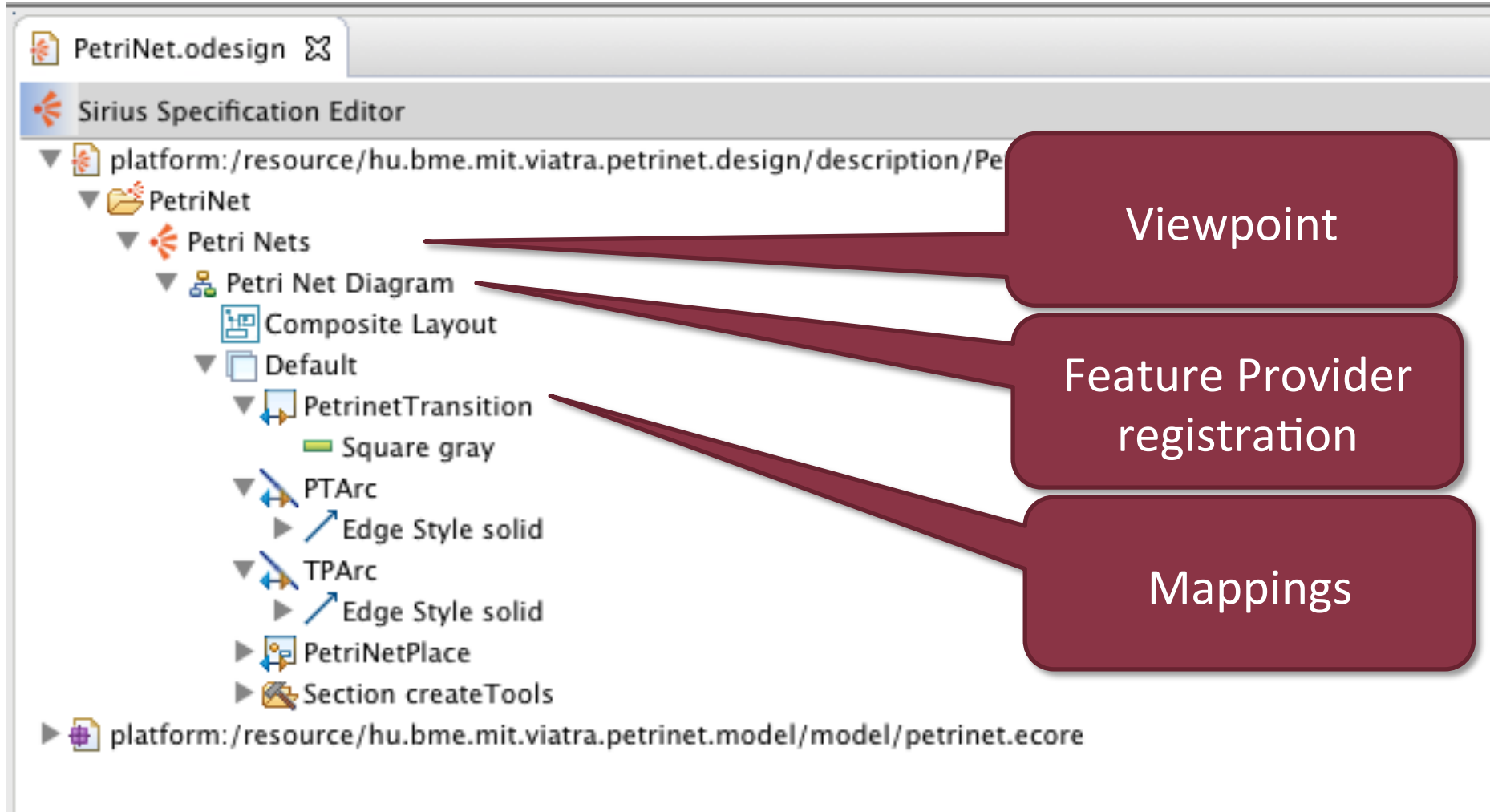
Viewpoint Specification Model



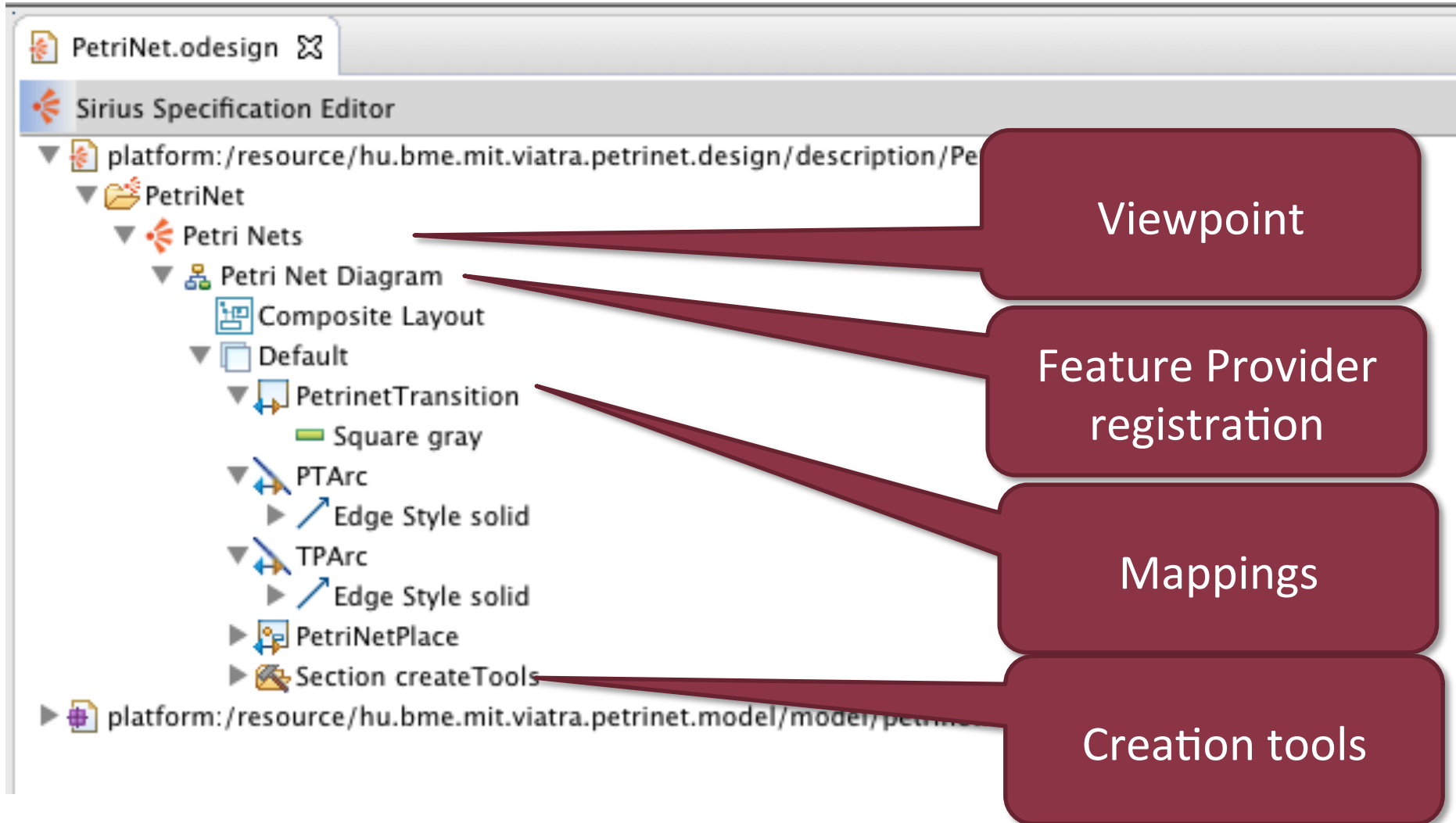
Viewpoint Specification Model



Viewpoint Specification Model



Viewpoint Specification Model



Node Mapping

Properties  Model requests interpreter  Problems 

 **PetrinetTransition**

General

Import

Documentation

Behavior

Advanced

Id:  **PetrinetTransition** Label:  PetrinetTransition

Domain Class*:  **petrinet.Transition**

Semantic Candidates Expression:  **feature:transitions**

Node Mapping

Domain class

Properties Model requests interpreter Problems

PetrinetTransition

General

Import

Documentation

Behavior

Advanced

ID: **PetrinetTransition** Label: **PetrinetTransition**

Domain Class*: **petrinet.Transition**

Semantic Candidates Expression: **feature:transitions**

Node Mapping

Domain class

The screenshot shows a software interface for node mapping. At the top, there are tabs for 'Properties', 'Model requests interpreter', and 'Problems'. Below this, the 'PetrinetTransition' node is selected. On the left, a sidebar contains a list of tabs: 'General', 'Import', 'Documentation', 'Behavior', and 'Advanced'. The 'General' tab is active, showing the 'id:' field with the value 'PetrinetTransition' and the 'Domain Class:' field with the value 'petrinet.Transition'. The 'Semantic Candidates Expression:' field contains the text 'feature:transitions'. On the right, there are three rows of candidate expressions, each with a question mark icon and a text field. The first row is 'PetrinetTransition' with a label 'PetrinetTransition'. The second row is 'petrinet.Transition' with a label 'PetrinetTransition'. The third row is 'feature:transitions' with a label 'PetrinetTransition'. A red callout bubble points to the 'petrinet.Transition' row, and another red callout bubble points to the 'feature:transitions' row.

Properties ⓘ Model requests interpreter ⓘ Problems

PetrinetTransition

General

Import

Documentation

Behavior

Advanced

id: PetrinetTransition

Domain Class: petrinet.Transition

Semantic Candidates Expression: feature:transitions

Label: PetrinetTransition

Label: PetrinetTransition

Label: PetrinetTransition

Filter settings

Feature Selection

- Interpreted expressions
 - Special interpreters
 - **var**: accessing specification model variables
 - **feature**: accessing EMF model features
 - **service**: accessing service methods
 - Aceleo
 - Aceleo expressions
 - Basic operations
 - Comparison with single '=' symbols
 - Syntax: **[theExpression/]**
 - Raw OCL
 - Not recommended, Aceleo provides superset features
 - Custom interpreter

Edge Mapping

Properties ⓘ Model requests interpreter ⓘ Problems ⓘ

TPArc

General
Path
Documentation
Behavior
Advanced

Id*: Label:

Domain Class*:

Source Mapping*: ...

Source Finder Expression:

Target Mapping*: ...

Target Finder Expression:

Semantic Candidates Expression:

Edge Mapping

Edge class

Properties ⓘ Model requests interpreter ⓘ Problems ⓘ

TPArc

General
Path
Documentation
Behavior
Advanced

Id*: Label:

Domain Class*:

Source Mapping*: ...

Source Finder Expression:

Target Mapping*: ...

Target Finder Expression:

Semantic Candidates Expression:

Edge Mapping

The screenshot shows the 'TPAArc' properties dialog with the following configuration:

Property	Value
Id:	TPArc
Domain Class:	petrinet.TPArc
Source Mapping:	PetrinetTransition
Source Finder Expression:	feature:source
Target Mapping:	PetriNetPlace
Target Finder Expression:	feature:target
Semantic Candidates Expression:	

Two callout boxes are present:

- Edge class**: Points to the 'petrinet.TPArc' value in the 'Domain Class' field.
- Source features**: Points to the 'feature:source' value in the 'Source Finder Expression' field.

Edge Mapping

The screenshot shows the TPArc configuration window with the following fields and values:

Field	Value
Id:	TPArc
Domain Class:	petrinet.TPArc
Source Mapping:	PetrinetTransition
Source Finder Expression:	feature:source
Target Mapping:	PetriNetPlace
Target Finder Expression:	feature:target
Semantic Candidates Expression:	

Callouts from the image:

- Edge class:** Points to the `petrinet.TPArc` value in the Domain Class field.
- Source features:** Points to the `feature:source` value in the Source Finder Expression field.
- Target features:** Points to the `feature:target` value in the Target Finder Expression field.

Node Creation Tool

- ▼  Section createTools
 - ▼  Container Creation createPlace
 -  Node Creation Variable container
 -  Container View Variable containerView
 - ▼  Begin
 - ▼  Change Context var:container
 - ▼  Create Instance petrinet.Place

(x)= Set name

Node Creation Tool

- ▼  Section createTools
- ▼  Container Creation createPlace
-  Node Creation Variable container
-  Container View Variable containerView
- ▼  Begin
- ▼  Change Context var:container
- ▼  Create Instance petrinet.Place

Tool parameter
variables

(x)= Set name

Node Creation Tool

- ▼  Section createTools
 - ▼  Container Creation createPlace
 -  Node Creation Variable container
 -  Container View Variable containerView
 - ▼  Begin
 - ▼  Change Context var:container
 - ▼  Create Instance petrinet.Place
 -  (=) Set name

Tool parameter
variables

Model creation
steps

Edge Creation Tool

▼ Edge Creation createArc

-  Source Edge Creation Variable source
-  Target Edge Creation Variable target
-  Source Edge View Creation Variable sourceView
-  Target Edge View Creation Variable targetView

▼ Begin

▼ Switch

▼ Case [source.eClass().name = 'Transition' /]

▼ Create Instance TPArc

 = Set source

 = Set target

▶ Case [source.eClass().name = 'Place' /]

Edge Creation Tool

Edge Creation createArc

- Source Edge Creation Variable source
- Target Edge Creation Variable target
- Source Edge View Creation Variable sourceView
- Target Edge View Creation Variable targetView

Begin

Switch

Case [source.eClass().name = 'Transition' /]

Create Instance TPArc

(x)= Set source

(x)= Set target

Case [source.eClass().name = 'Place' /]

Different variables

Edge Creation Tool

Edge Creation createArc

- Source Edge Creation Variable source
- Target Edge Creation Variable target
- Source Edge View Creation Variable sourceView
- Target Edge View Creation Variable targetView

Begin

Switch

Case [source.eClass().name = 'Transition' /]

Create Instance TPArc

(x)= Set source

(x)= Set target

Case [source.eClass().name = 'Place' /]

Different variables

More complex
creation steps

Interpreted Modeler Development

The image displays the Sirius Specification Editor interface. On the left, the 'PetriNet.odesign' project is open, showing a hierarchical tree of elements. The 'Edge Creation createArc' element is selected. The right pane shows a Petri net diagram in the '*Hello world' view. The diagram consists of two places, p1 and p2, and a transition, t1. Place p1 contains a single token. Arcs connect p2 to t1, t1 to p1, and p2 to p1. A palette on the right lists the available elements: Place, Token, and Arc.

Sirius Specification Editor

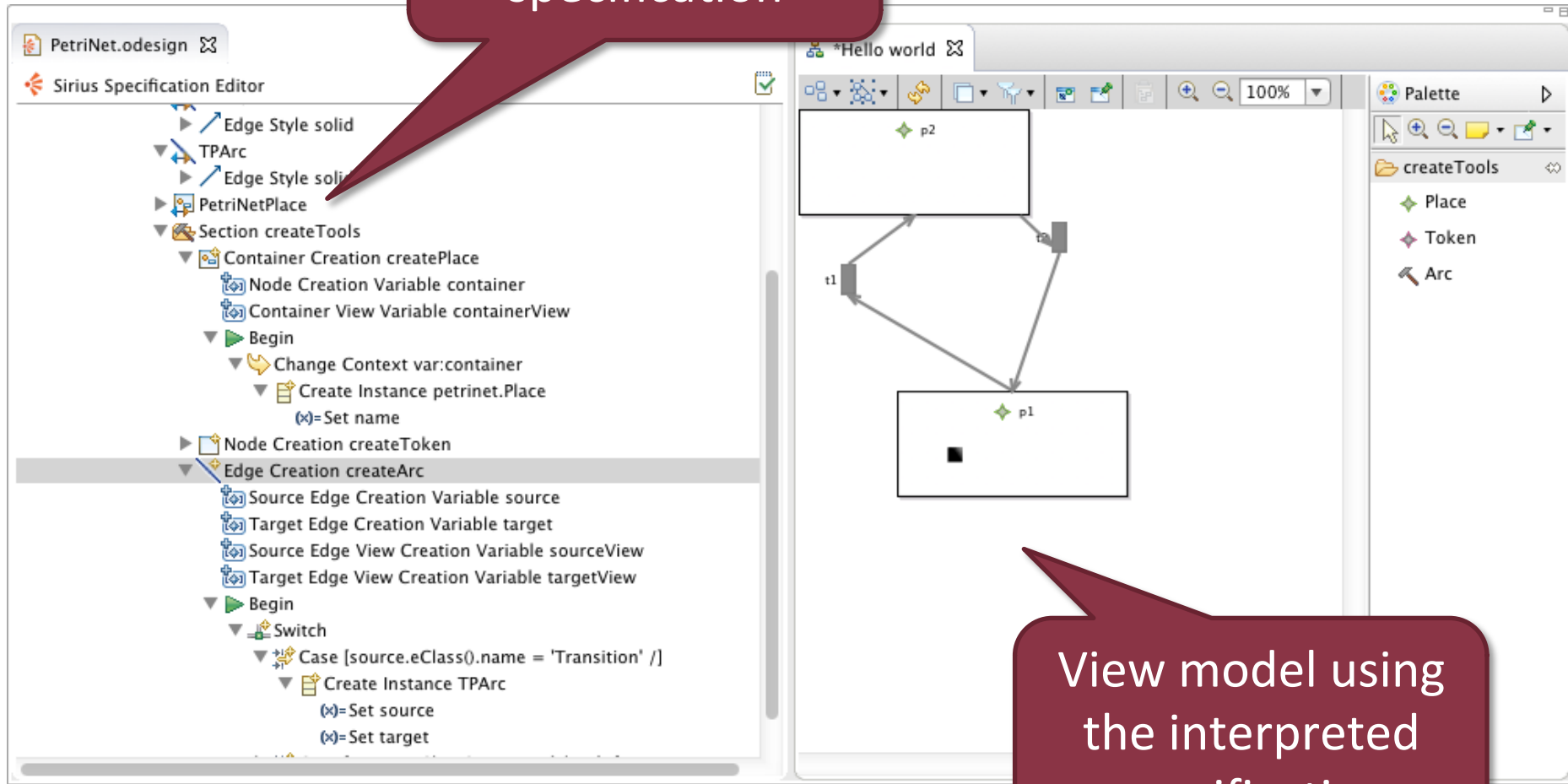
- PetriNet.odesign
 - Edge Style solid
 - TPArc
 - Edge Style solid
 - PetriNetPlace
 - Section createTools
 - Container Creation createPlace
 - Node Creation Variable container
 - Container View Variable containerView
 - Begin
 - Change Context var:container
 - Create Instance petrinet.Place
 - (x)=Set name
 - Node Creation createToken
 - Edge Creation createArc
 - Source Edge Creation Variable source
 - Target Edge Creation Variable target
 - Source Edge View Creation Variable sourceView
 - Target Edge View Creation Variable targetView
 - Begin
 - Switch
 - Case [source.eClass().name = 'Transition' /]
 - Create Instance TPArc
 - (x)=Set source
 - (x)=Set target

Viewpoint specification



Interpreted Modeler Development

Viewpoint
specification



View model using
the interpreted
specification

Sirius Editors

- Very opinionated
 - Custom project nature (Modeling project)
 - Custom perspective + Model Browser
 - Results in unusual workflows
 - E.g. how to open a diagram from the Java perspective?

Summary

Summary – Graphical Editors

	GEF	Graphiti	GMF	Sirius
Model	Any	EMF	EMF	EMF
Non-graph display	Possible	Not	Many, complex coding needed	Tree, table, textual views supported
Implementation amount	Many, repetitive code	Medium amount of code	Mostly modeling, some coding	High-level modeling
Workflow	Only coding	Only coding	Multi-step	Modeling