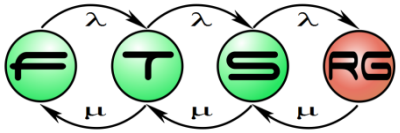
Graphical Editor Development 1.

GEF, Graphiti



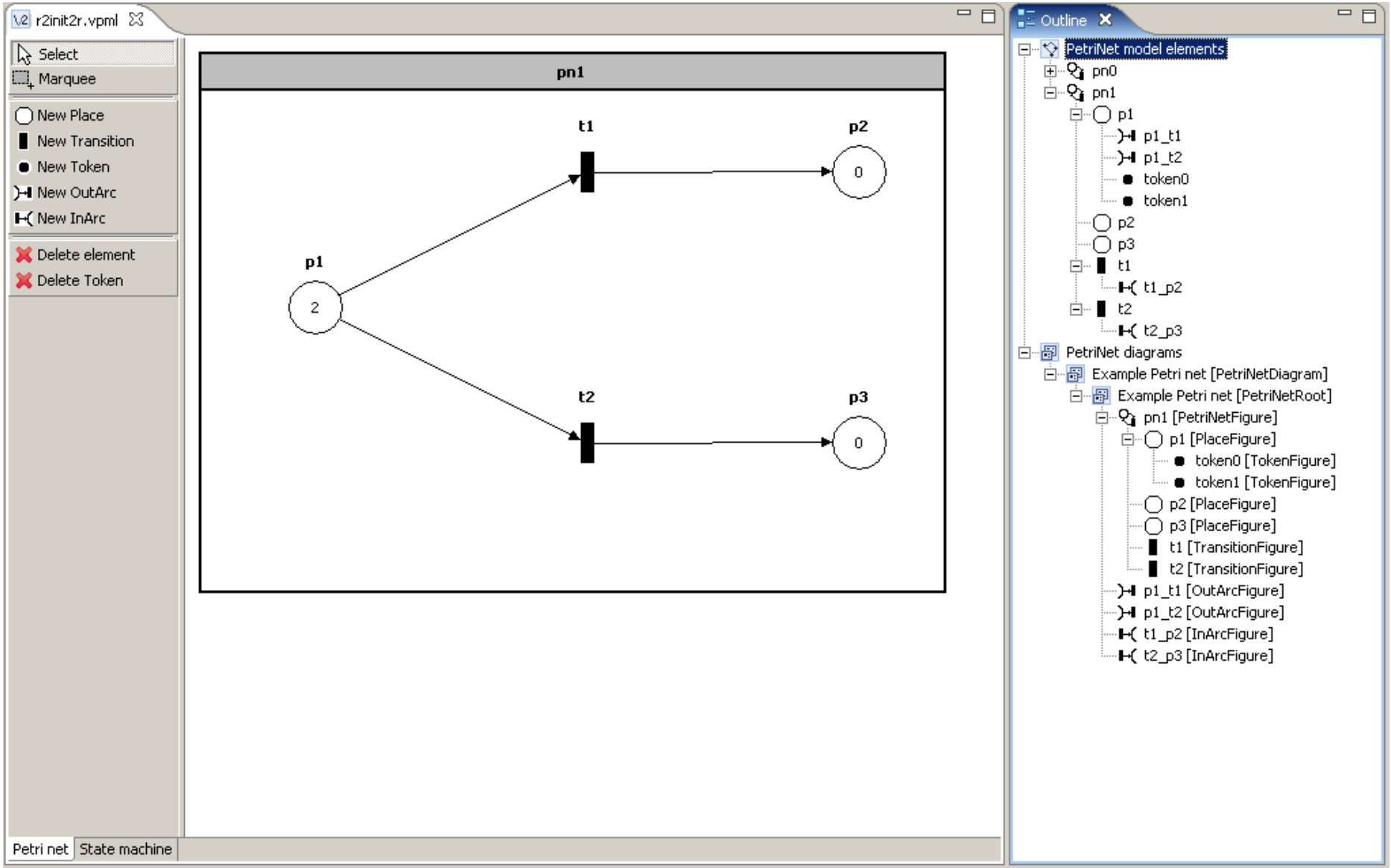
Designing modeling languages

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

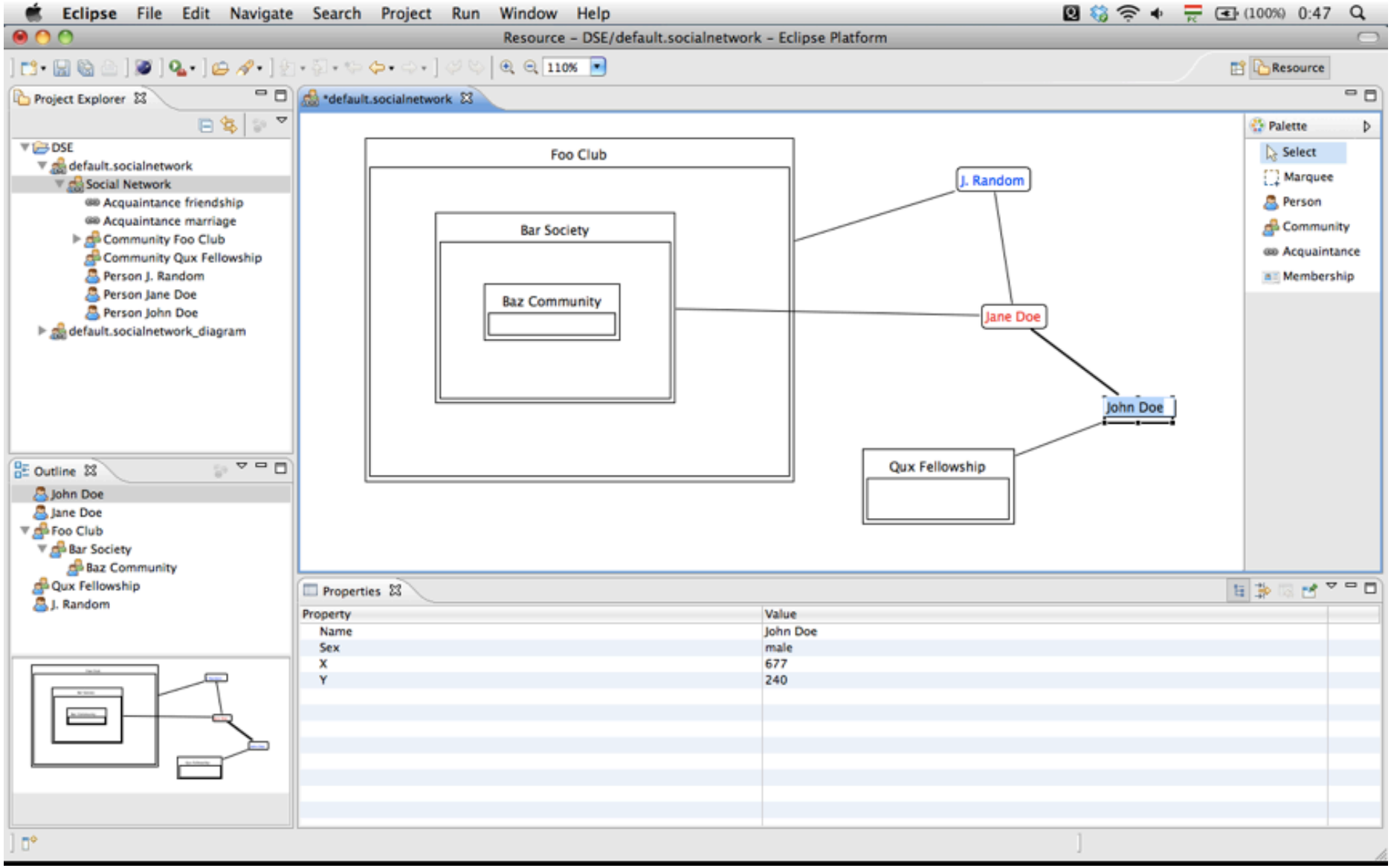
Goal of GEF

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

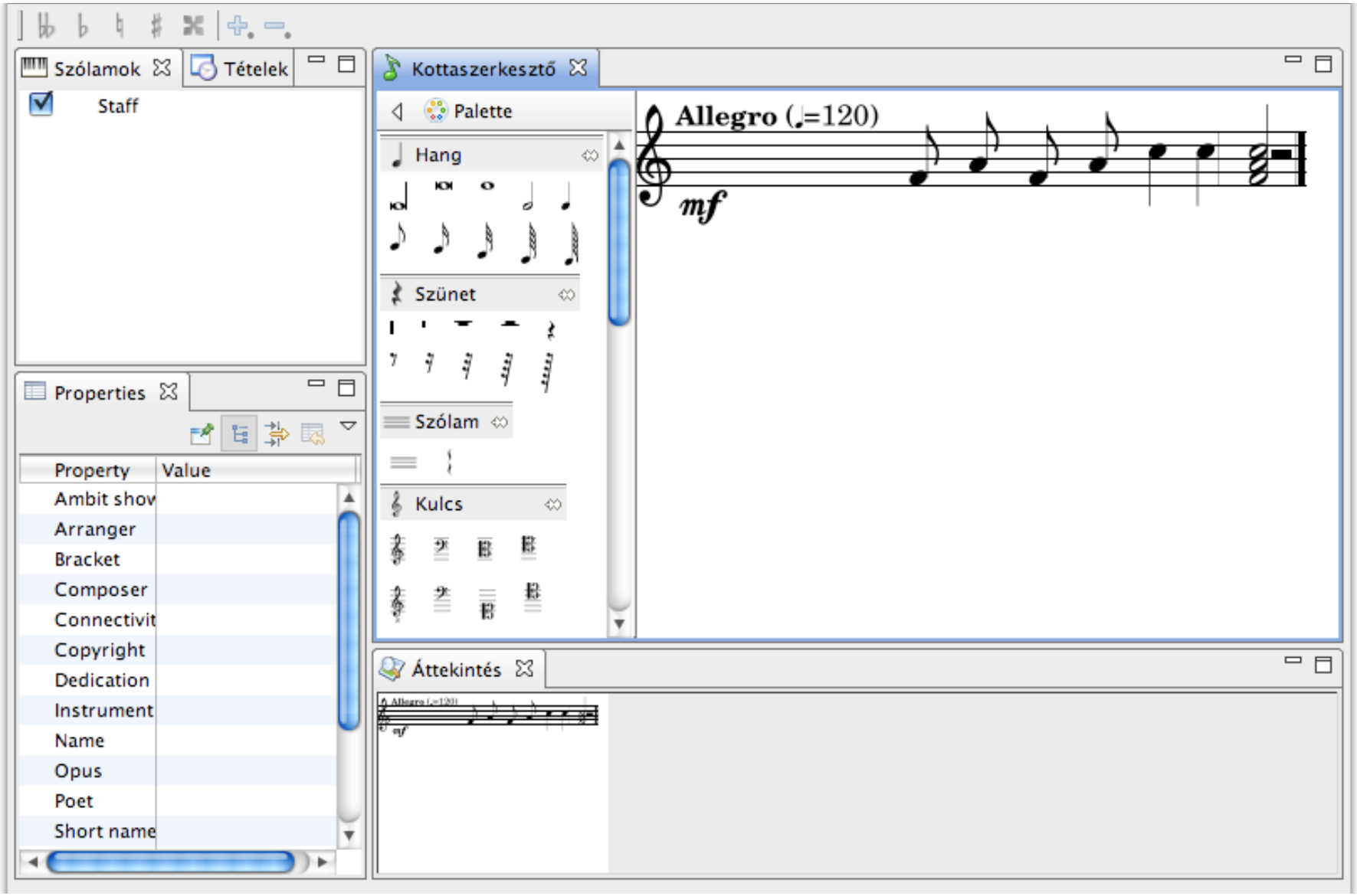
Sample Graphical Editor: Petri net



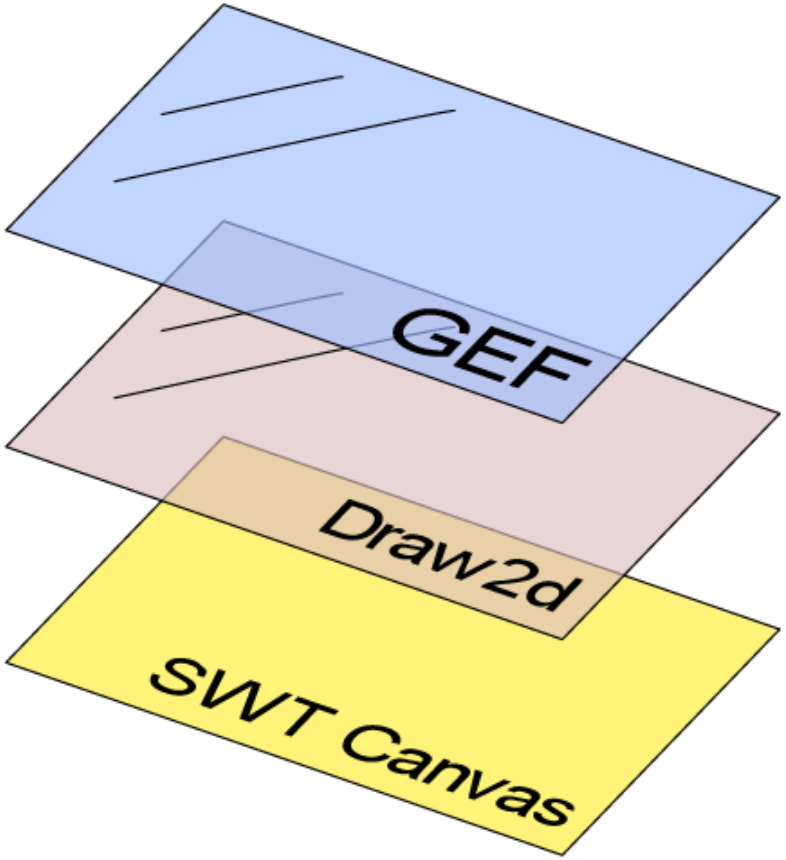
Social Network example



More complex example: Sheet music



Megjelenítési GEF környezetben



- Interaction (MVC)
- Model-view mapping
- Eclipse integration

- Presentation
  - Layouting
  - Zoom

- SWT layer

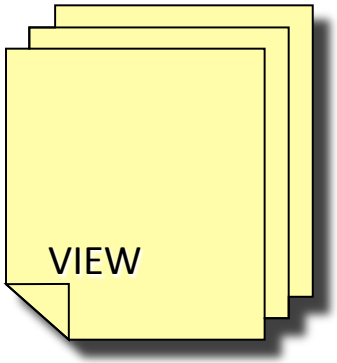
Model-View-Controller (MVC) architecture

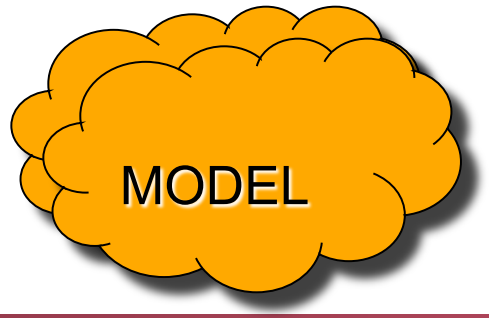
- Separate data and presentation
  - Model: stores data
  - View: displays data
  - Controller: user interaction support

MVC in GEF

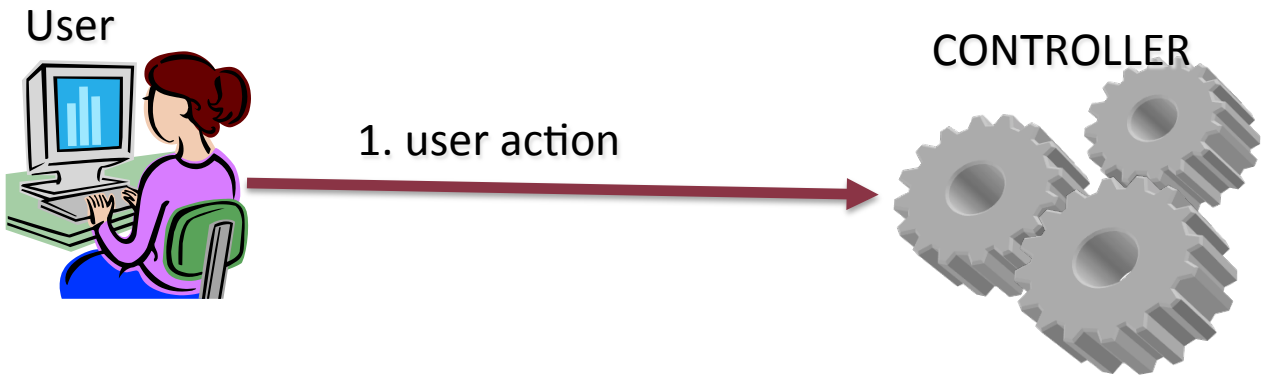
User

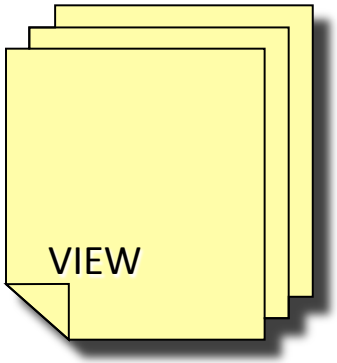
CONTROLLER

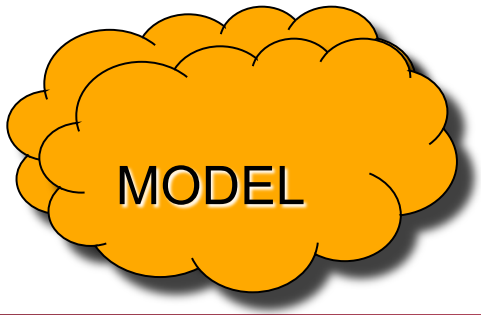




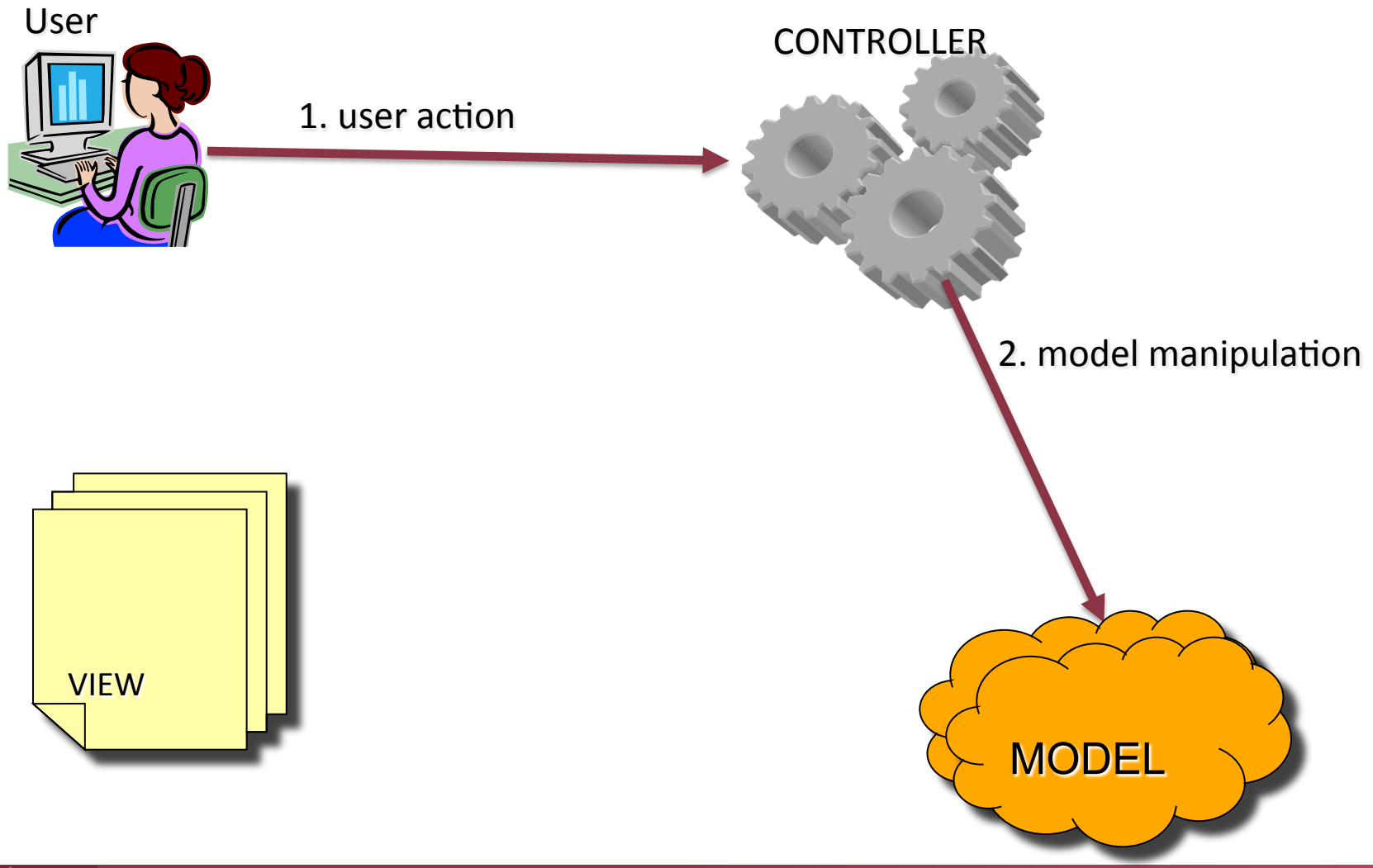
MVC in GEF



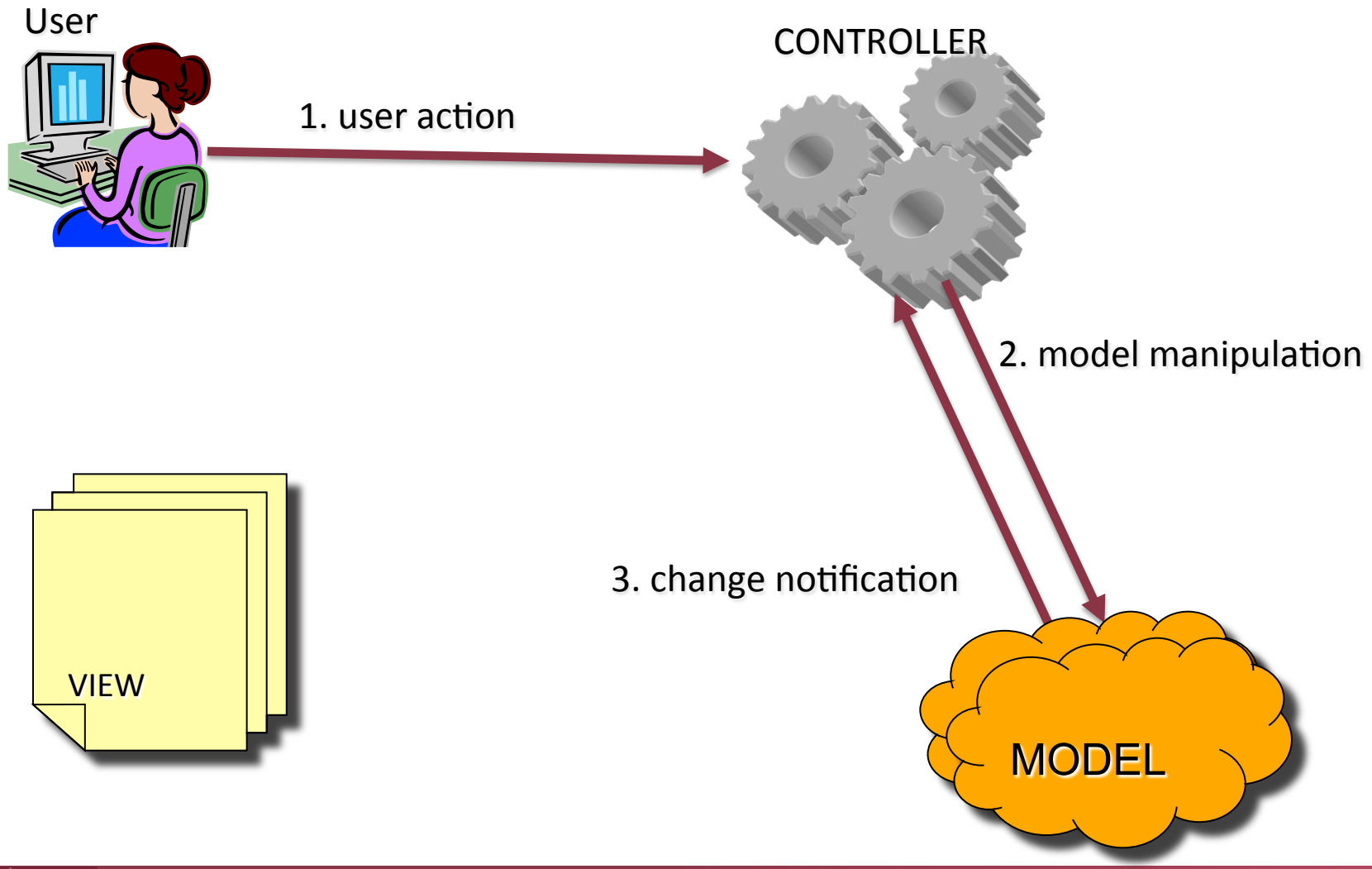




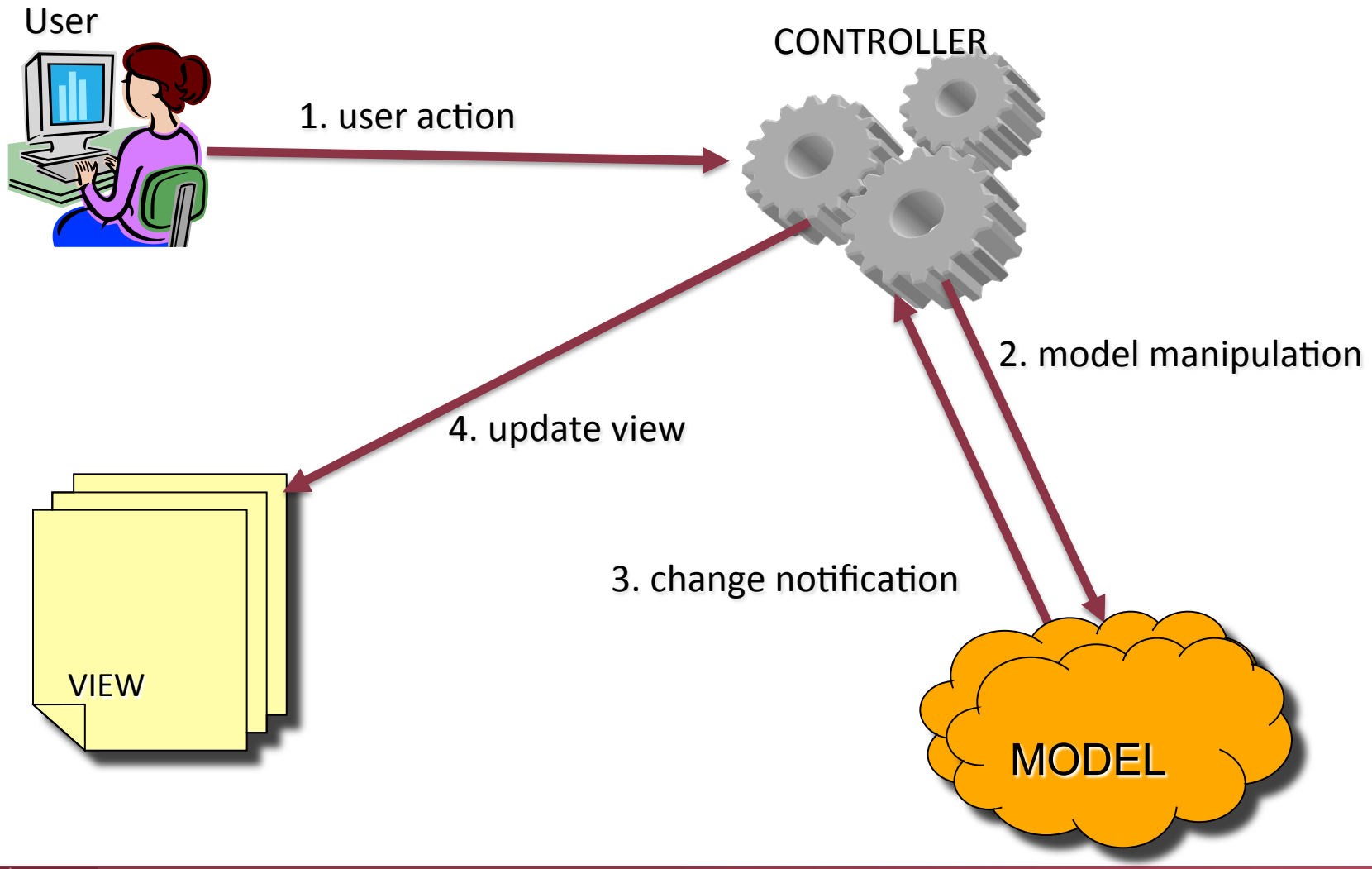
MVC in GEF


User
CONTROLLER
1. user action
2. model manipulation
VIEW
MODEL

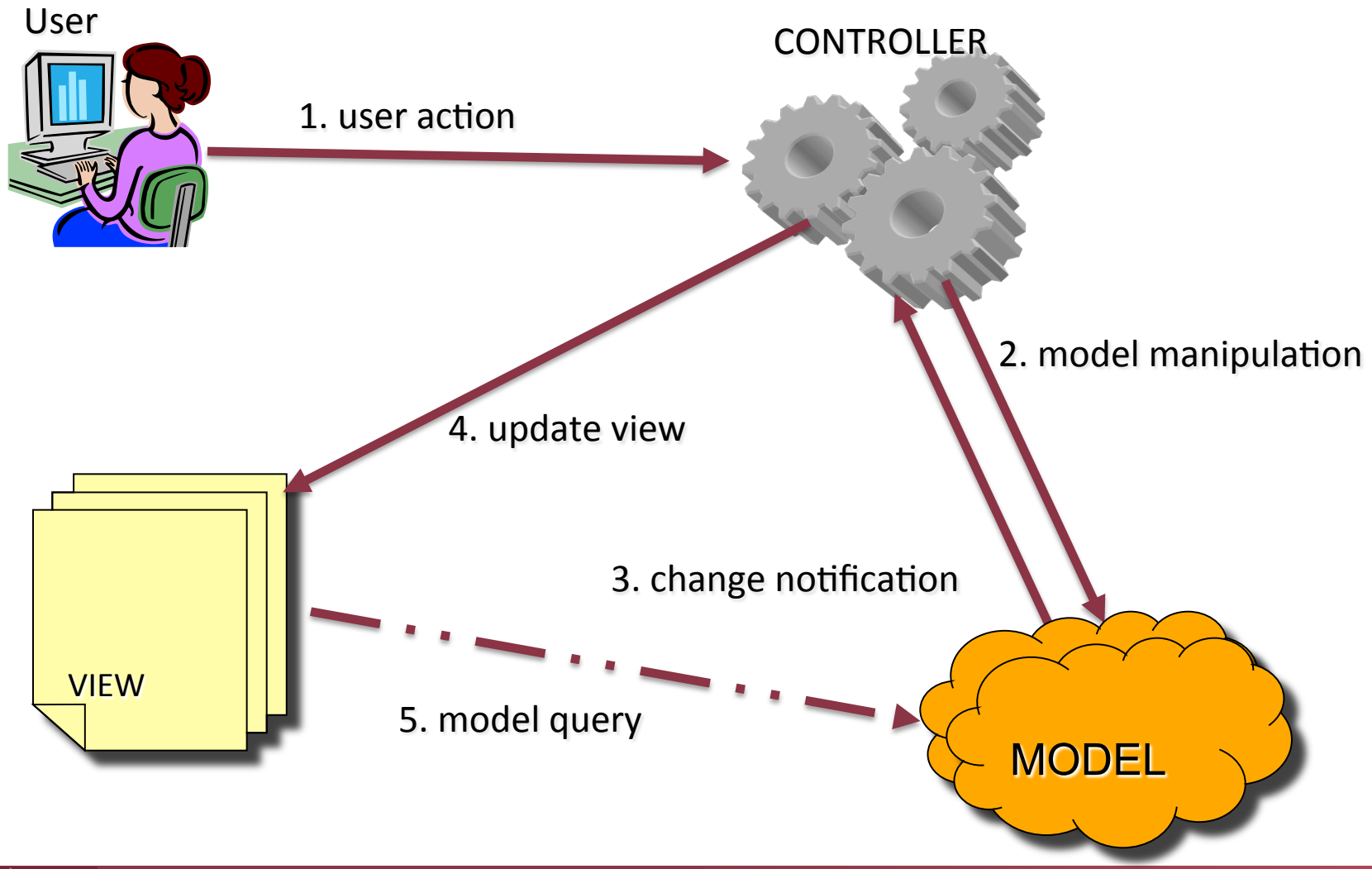
MVC in GEF


User
CONTROLLER
1. user action
2. model manipulation
3. change notification
VIEW
MODEL

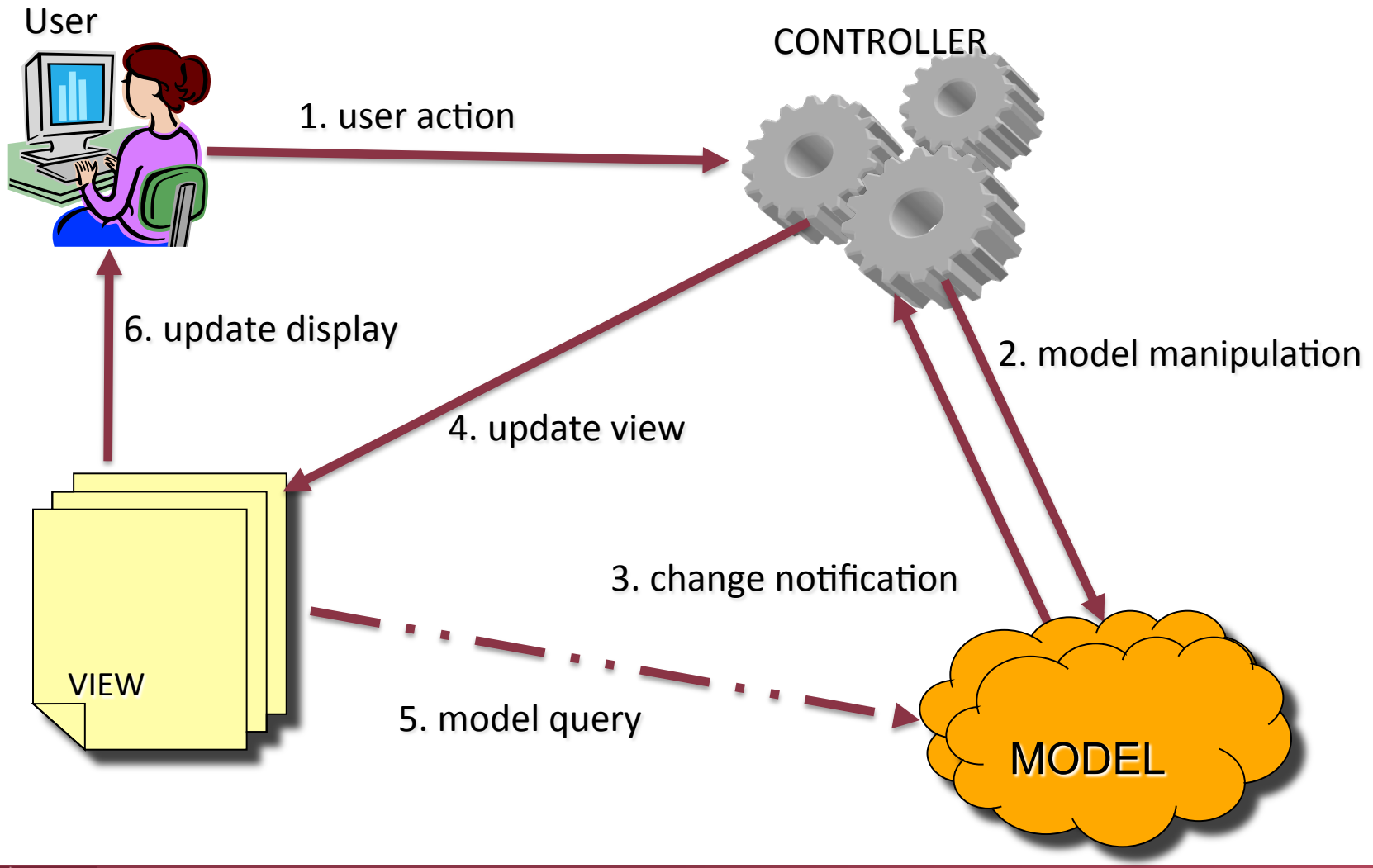
MVC in GEF


User
CONTROLLER
1. user action
2. model manipulation
4. update view
3. change notification
VIEW
MODEL

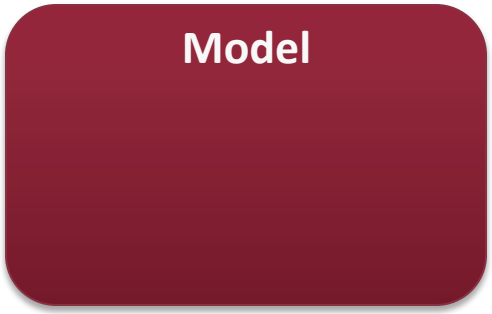
MVC in GEF


User
CONTROLLER
1. user action
2. model manipulation
4. update view
3. change notification
VIEW
5. model query
MODEL

MVC in GEF


User
CONTROLLER
1. user action
6. update display
2. model manipulation
4. update view
3. change notification
VIEW
5. model query
MODEL

GEF workflow



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

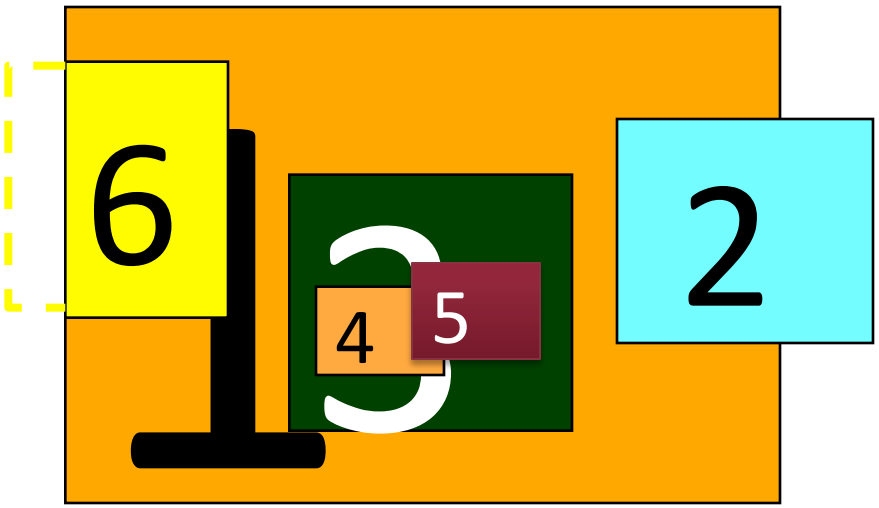
Layouting

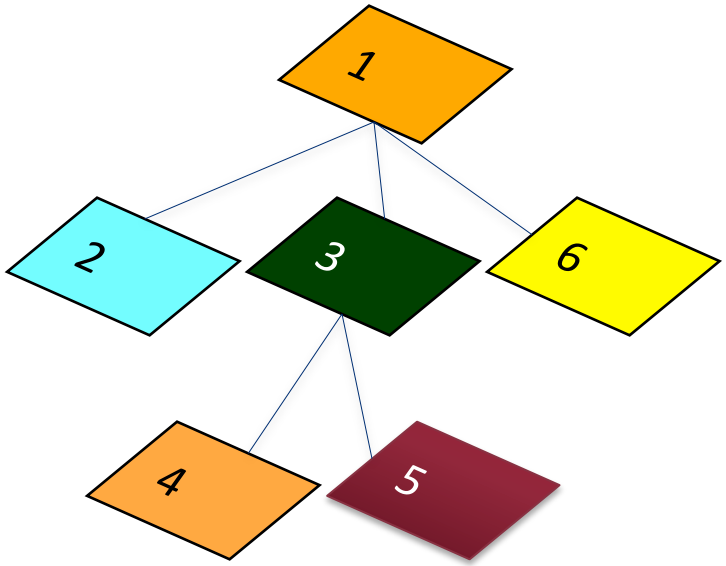
MVC in GEF: View

- 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

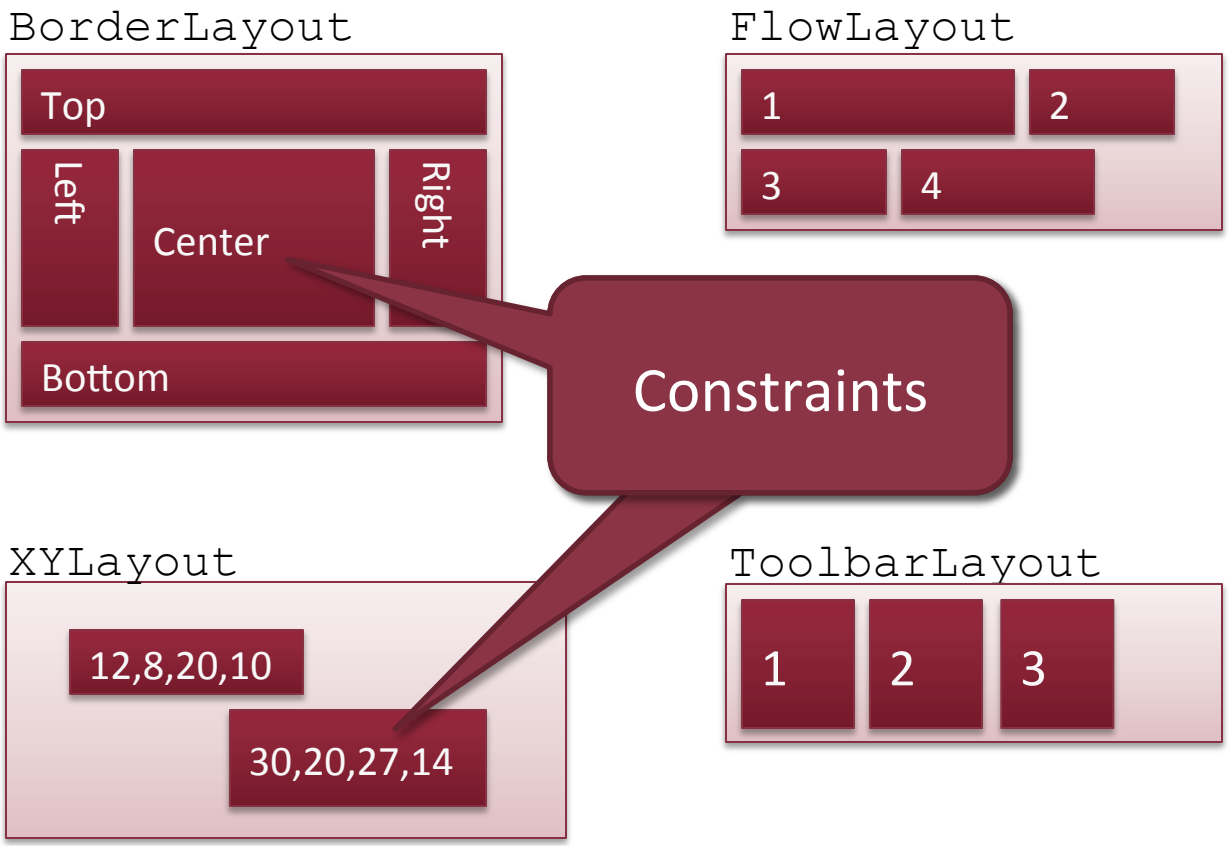




Draw2D LayoutManager

- 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 LayoutManagerek



Draw2D elements

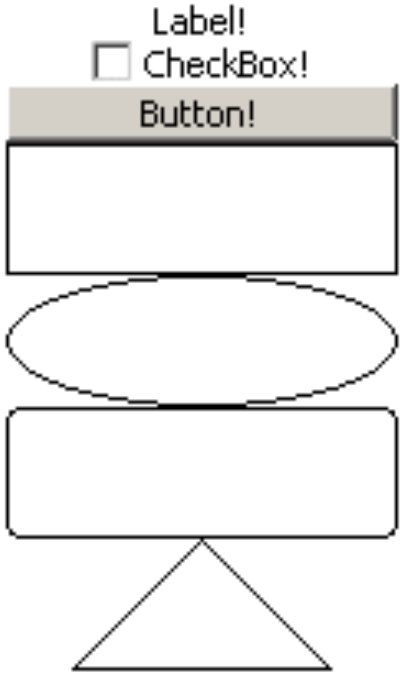
- Simple elements
  - Label, Button, CheckBox, Image
- Geometric shapes
  - RectangleFigure, Ellipse, Triangle
- Panel: generic container element
- ScrollPane: scrollable container element

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

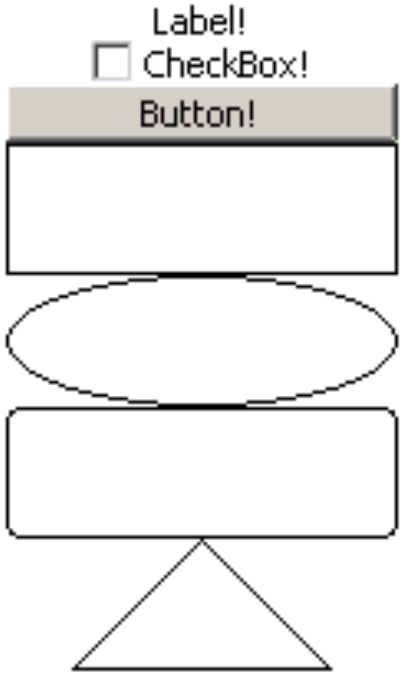
```
public class Pelda1 extends Figure {
 public Pelda1() {
    setOpaque(true);
    setBackgroundColor(ColorConstants.white);
    setLayoutManager(new ToolbarLayout());
    add(new Label("Label!"));
    add(new CheckBox("CheckBox!"));
    add(new Button("Button!"));
    add(new RectangleFigure());
    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

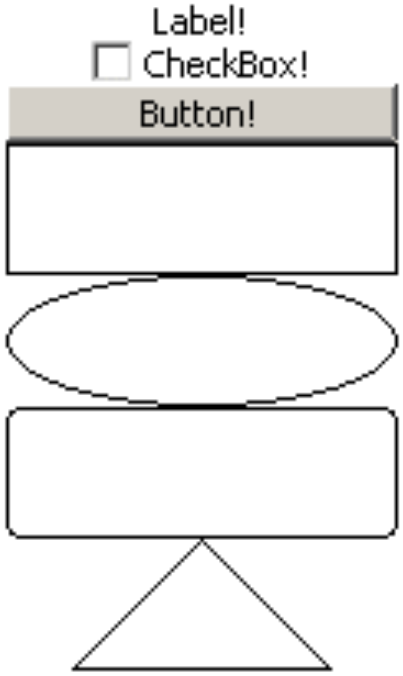
```
public class Pelda1 ext
 public Pelda1() {
    setOpaque(true);
    setBackgroundColor(ColorConstants.white);
    setLayoutManager(new ToolbarLayout());
    add(new Label("Label!"));
    add(new CheckBox("CheckBox!"));
    add(new Button("Button!"));
    add(new RectangleFigure());
    add(new Ellipse());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

By default everything is transparent



Base elements - example

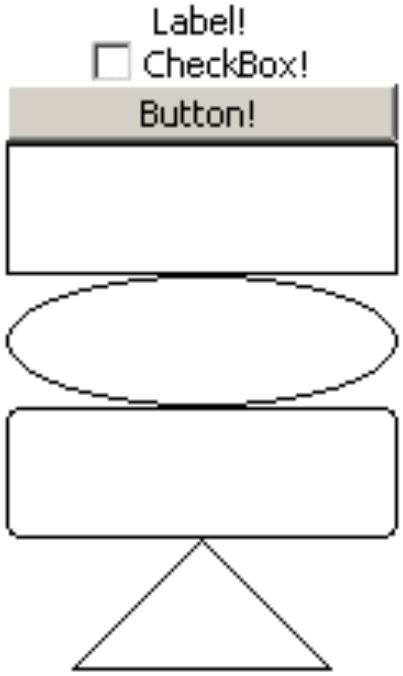
```
public class Pelda1 extends Figure {
 public Pelda1() {
    setOpaque(true);
    setBackgroundColor(ColorConstants.white);
    setLayoutManager(new ToolbarLayout());
    add(new Label("Label!"));
    add(new CheckBox("CheckBox!"));
    add(new Button("Button!"));
    add(new RectangleFigure());
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



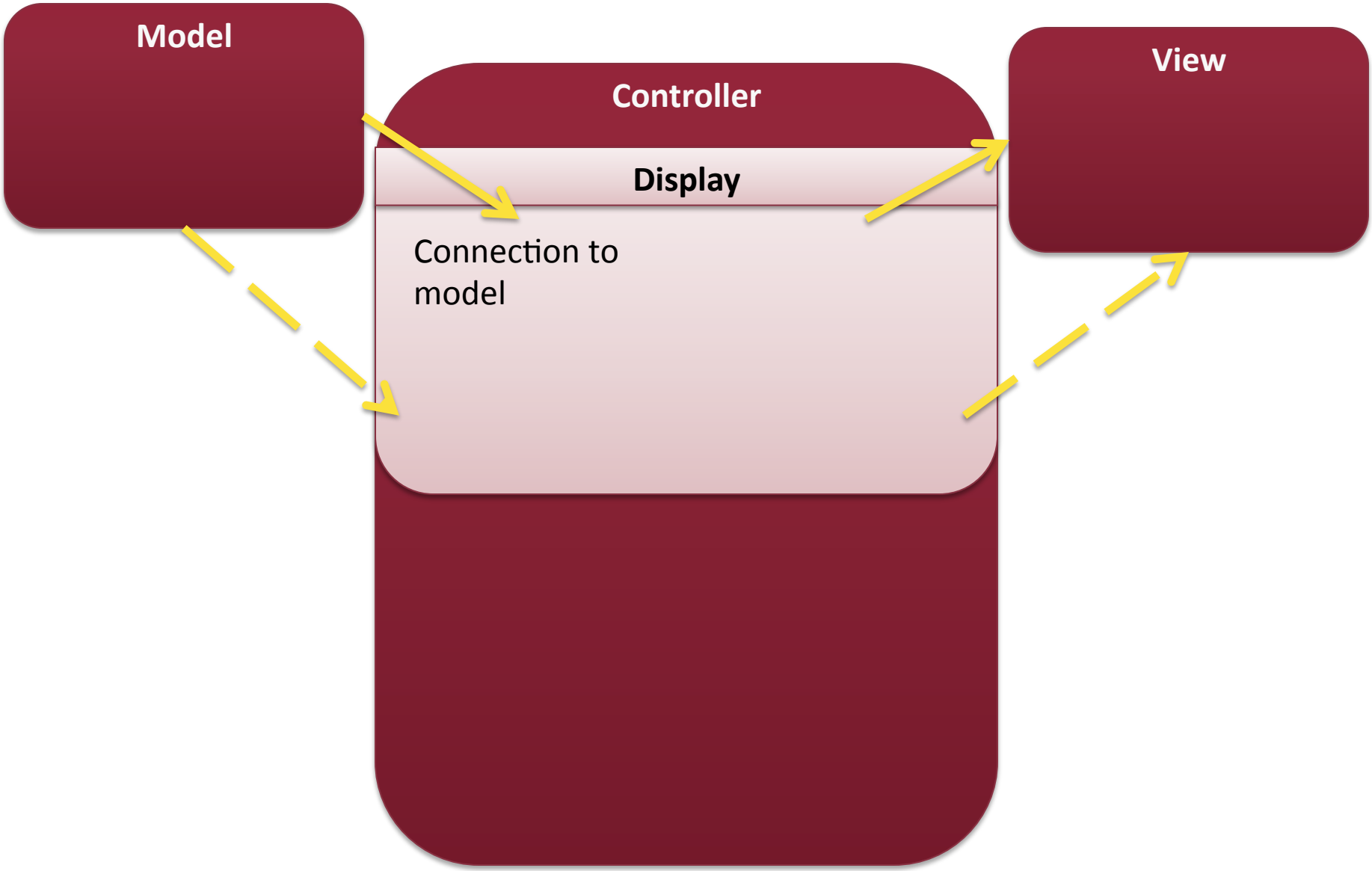
Base elements - example

```
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);
   }
 }
}
```

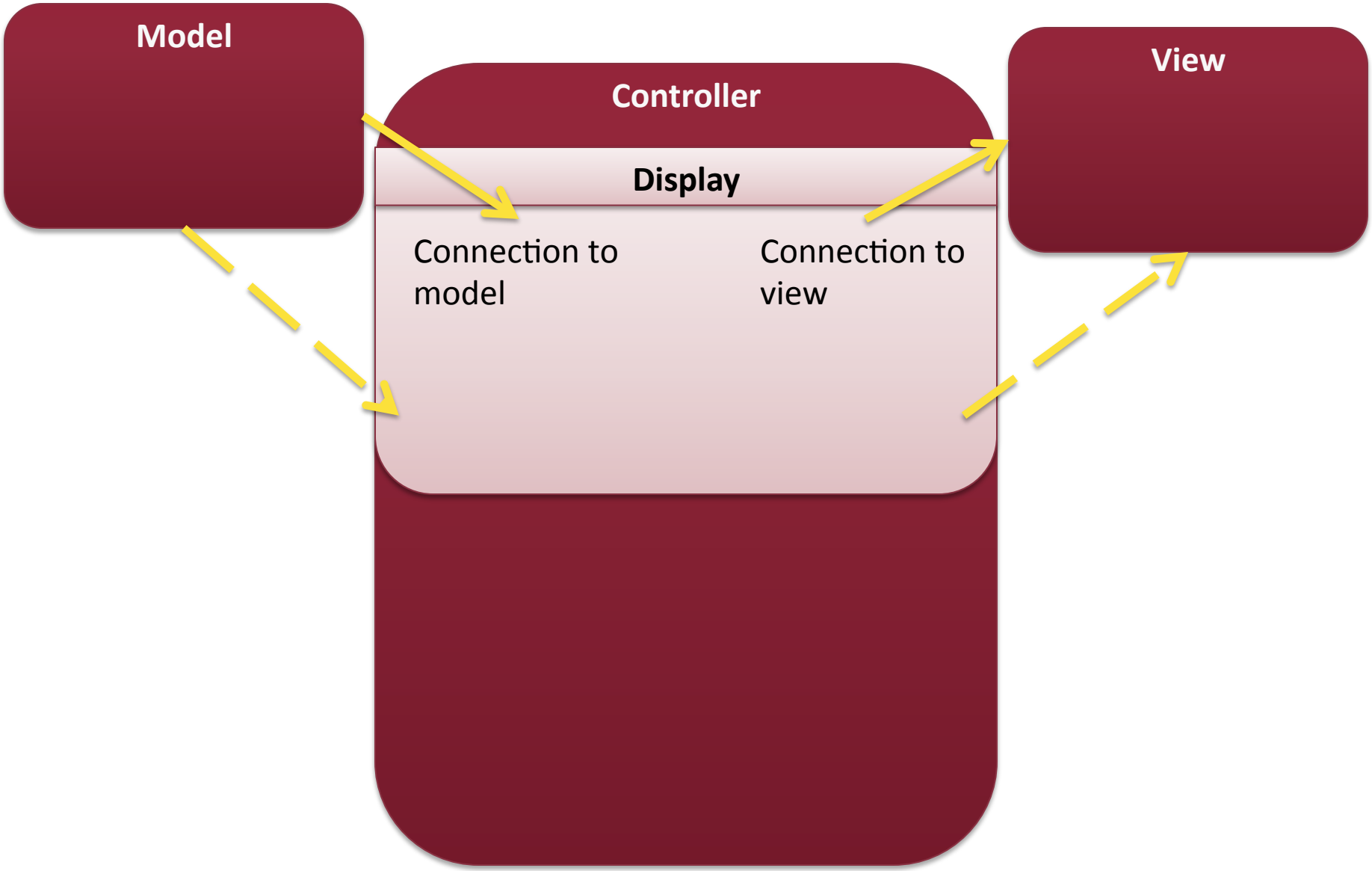
Preferred size constraint



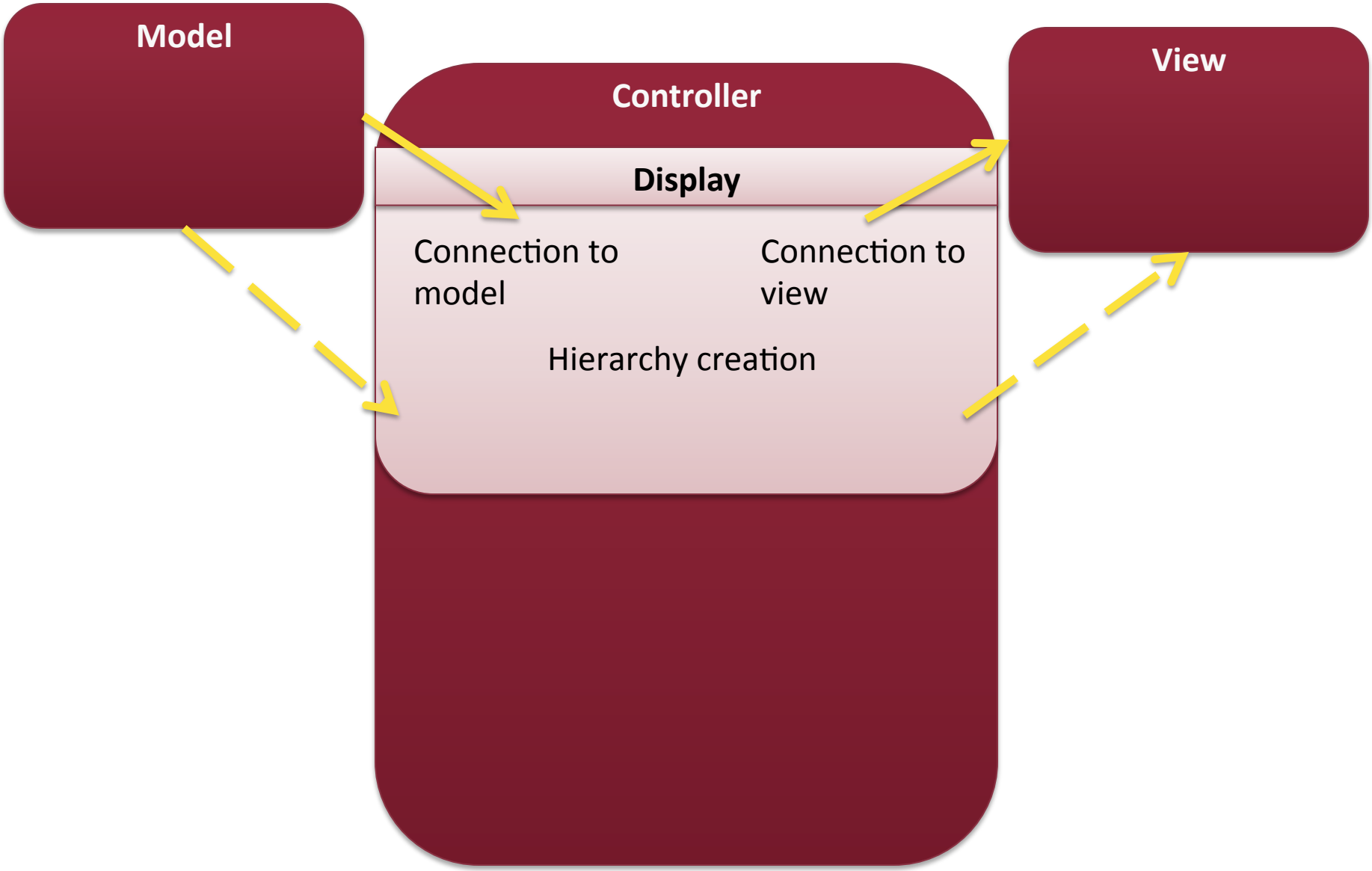
GEF workflow



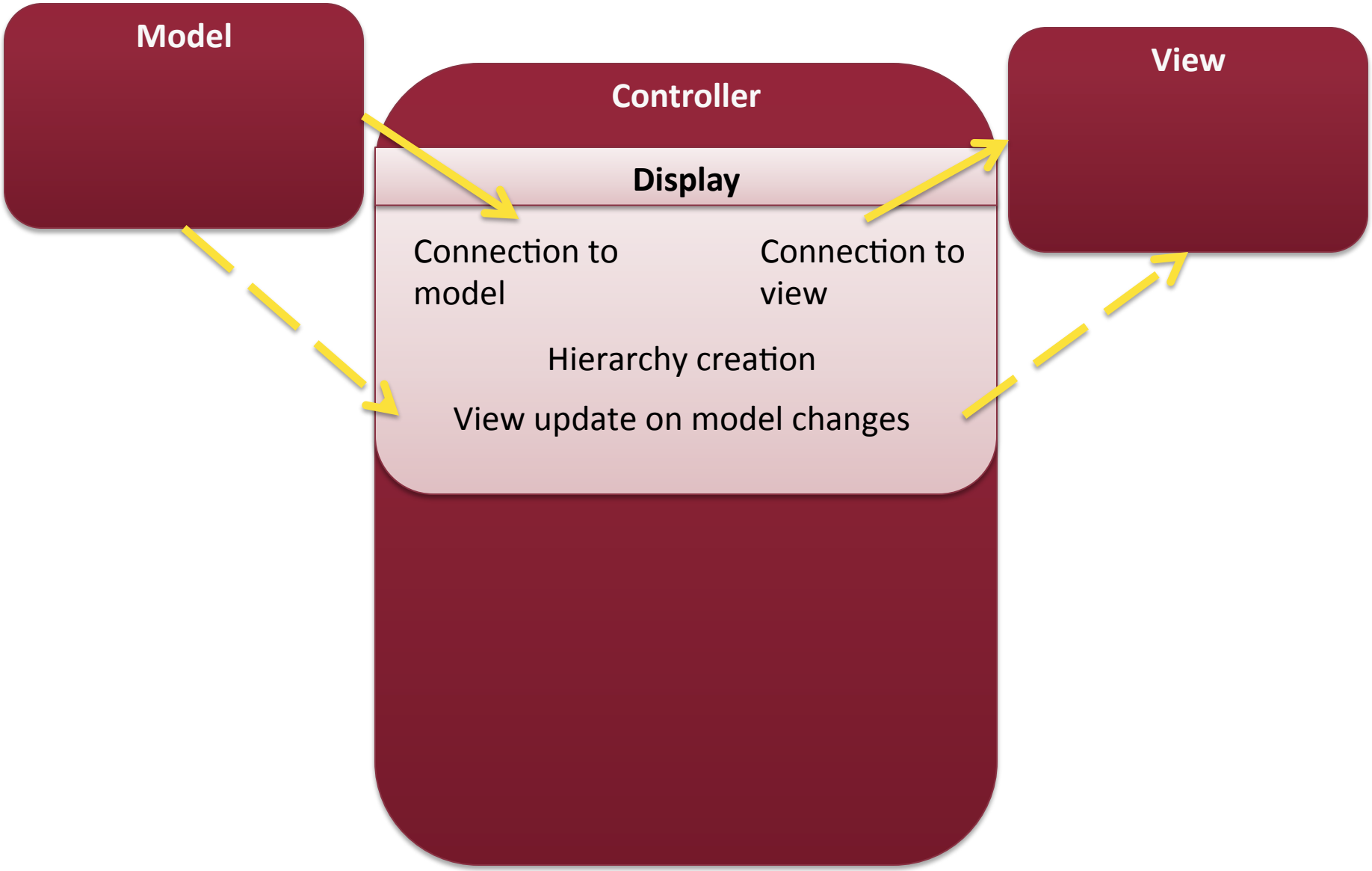
GEF workflow



GEF workflow



GEF workflow

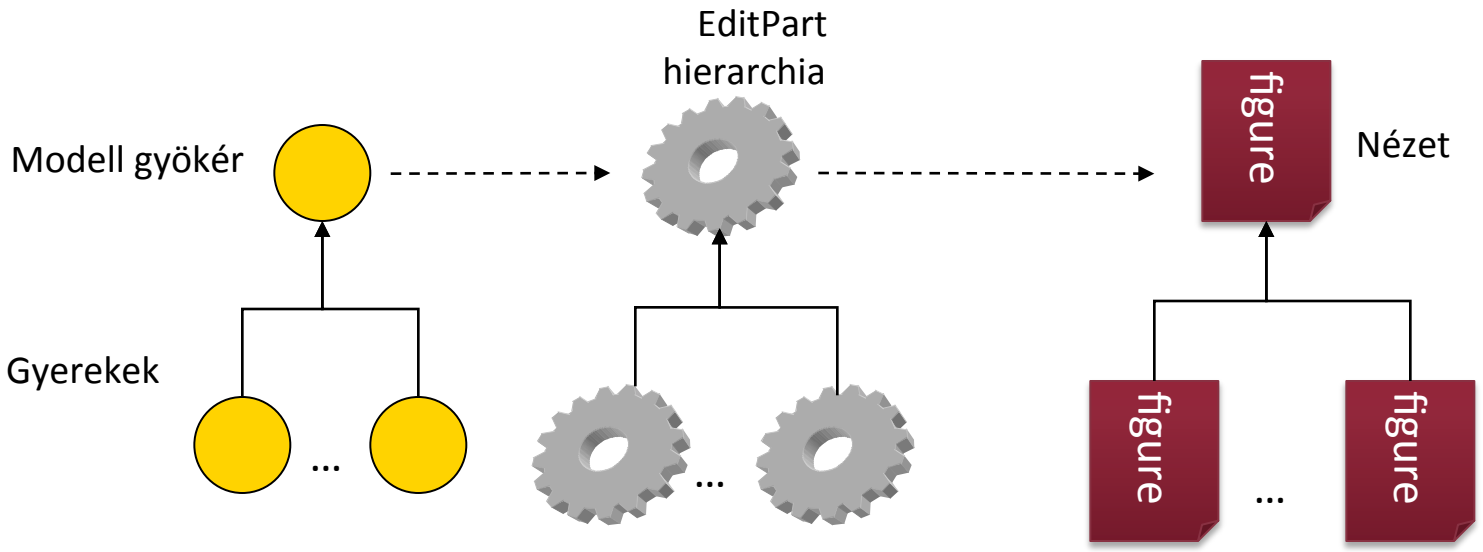


MVC in GEF: Controller

- 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() {
      // Saját Figure létrehozása
      ElementFigure fig = new
  ElementFigure();
      return fig;
    }
    ...
  }
```

Model traversal

- Initial GEF model
  - EditPartFactory
  - Model root
- How to traverse to other elements?
  - EditPart instaces 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() {
      // Saját modell lekérdezése
      ParentModel pm = ((ParentModel)
  getModel());
      return pm.getChildren();
    }
    ...
  }
```

Model dependent, not GEF specific

Model traversal

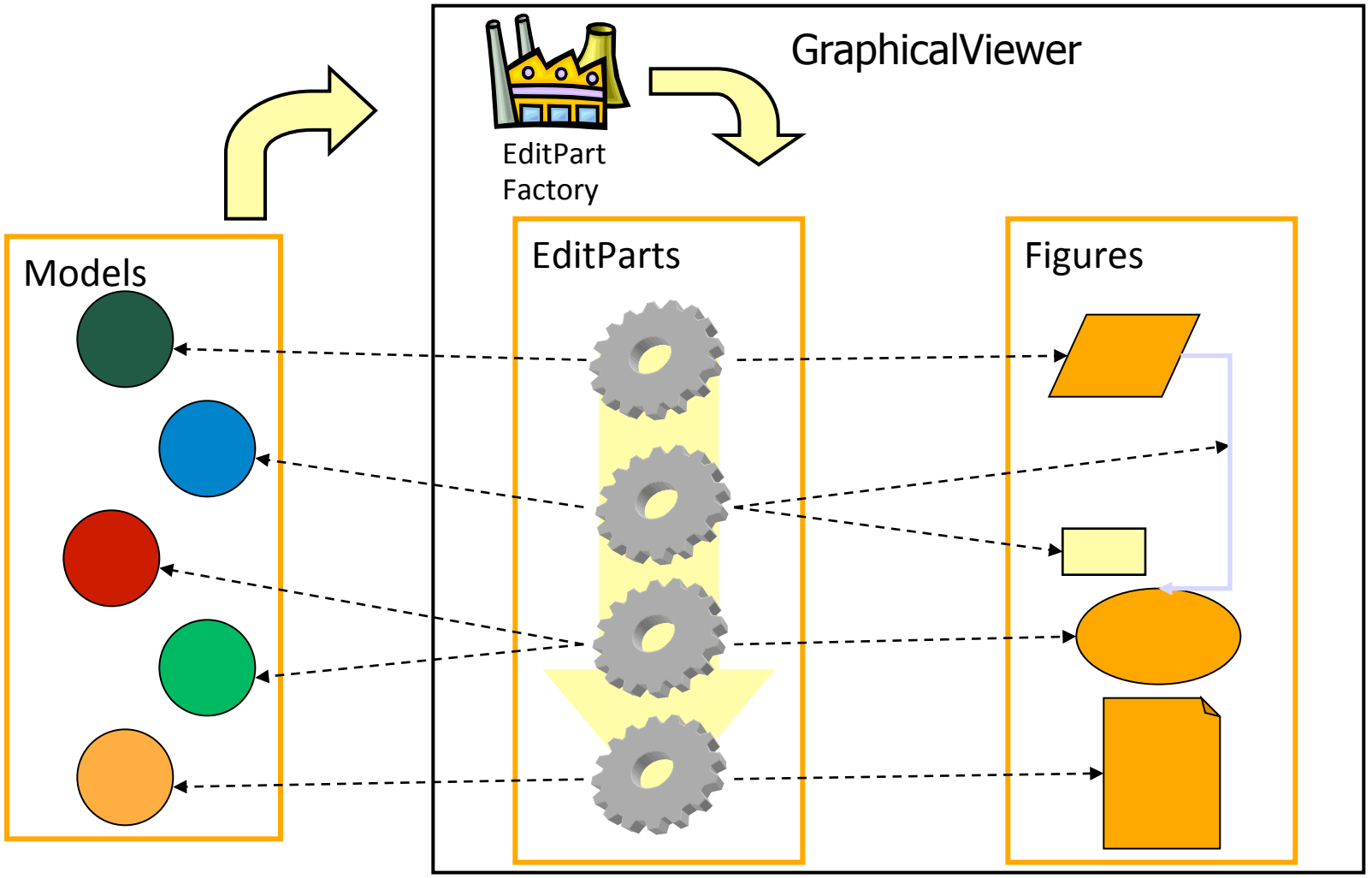
- EditPart.getModelChildren()
  - Returns the children of the connected model element
    - Ordering important -> determines covering

```
public class TestParentEditPart extends
  AbstractGraphicalEditPart {
    ...
    @Override
    protected List getMod                ) {
      // Saját modell leke       ebe
      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();

  }

}
```

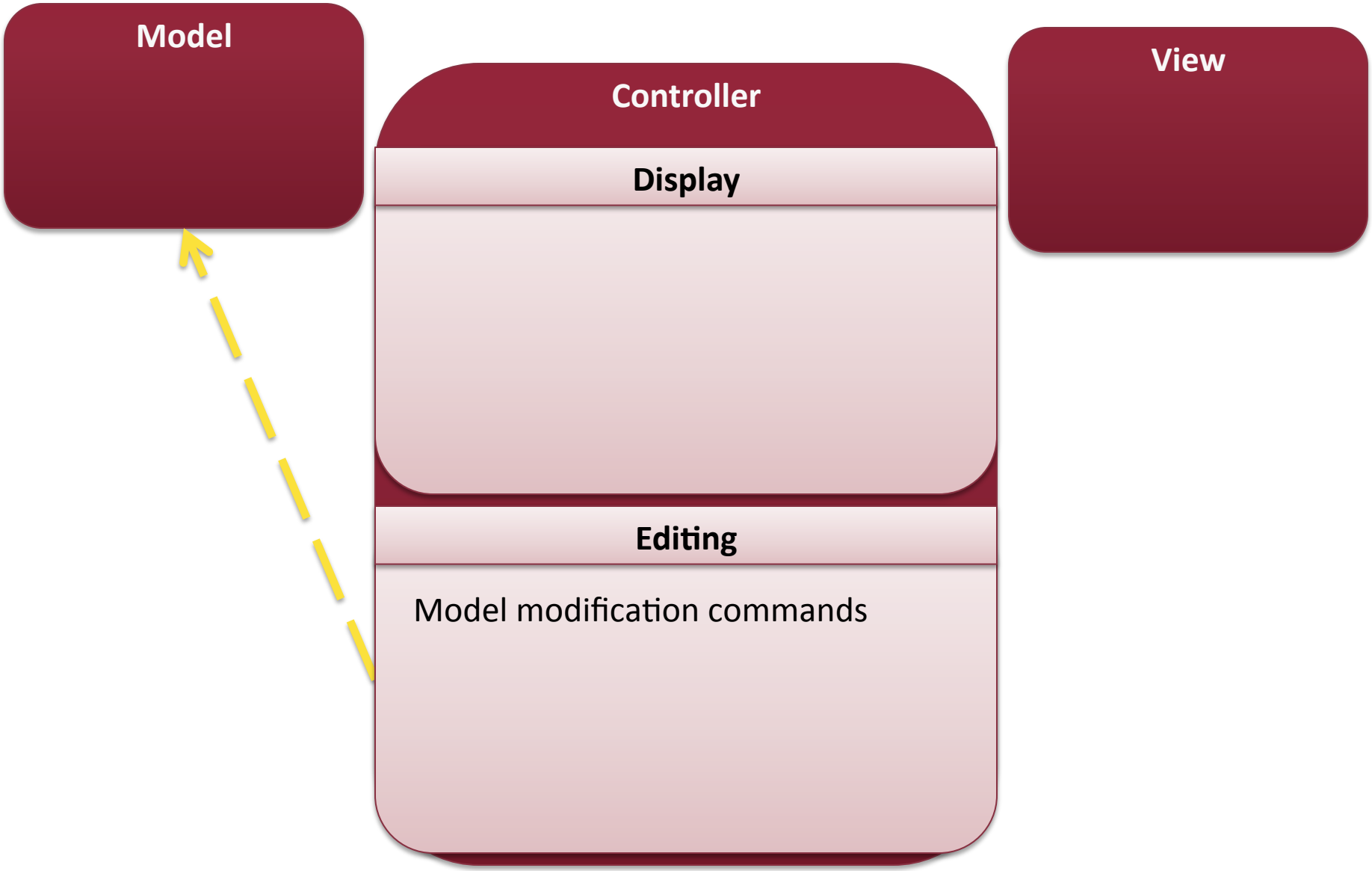
This is called by model

EditPart example

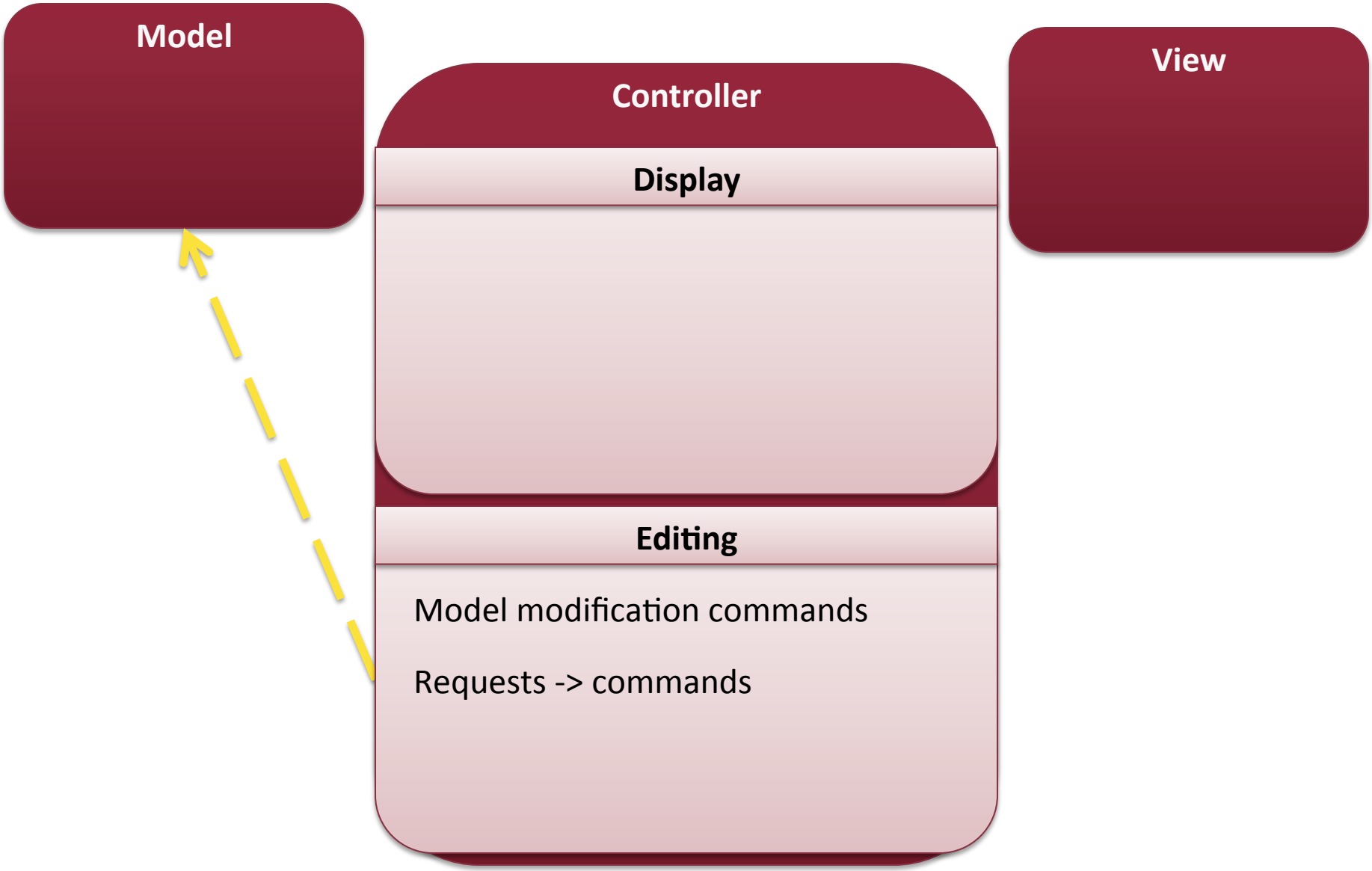
```
public class ParentEditPart extends AbstractGraphicalEditPart
      implements MyModelListener {

 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

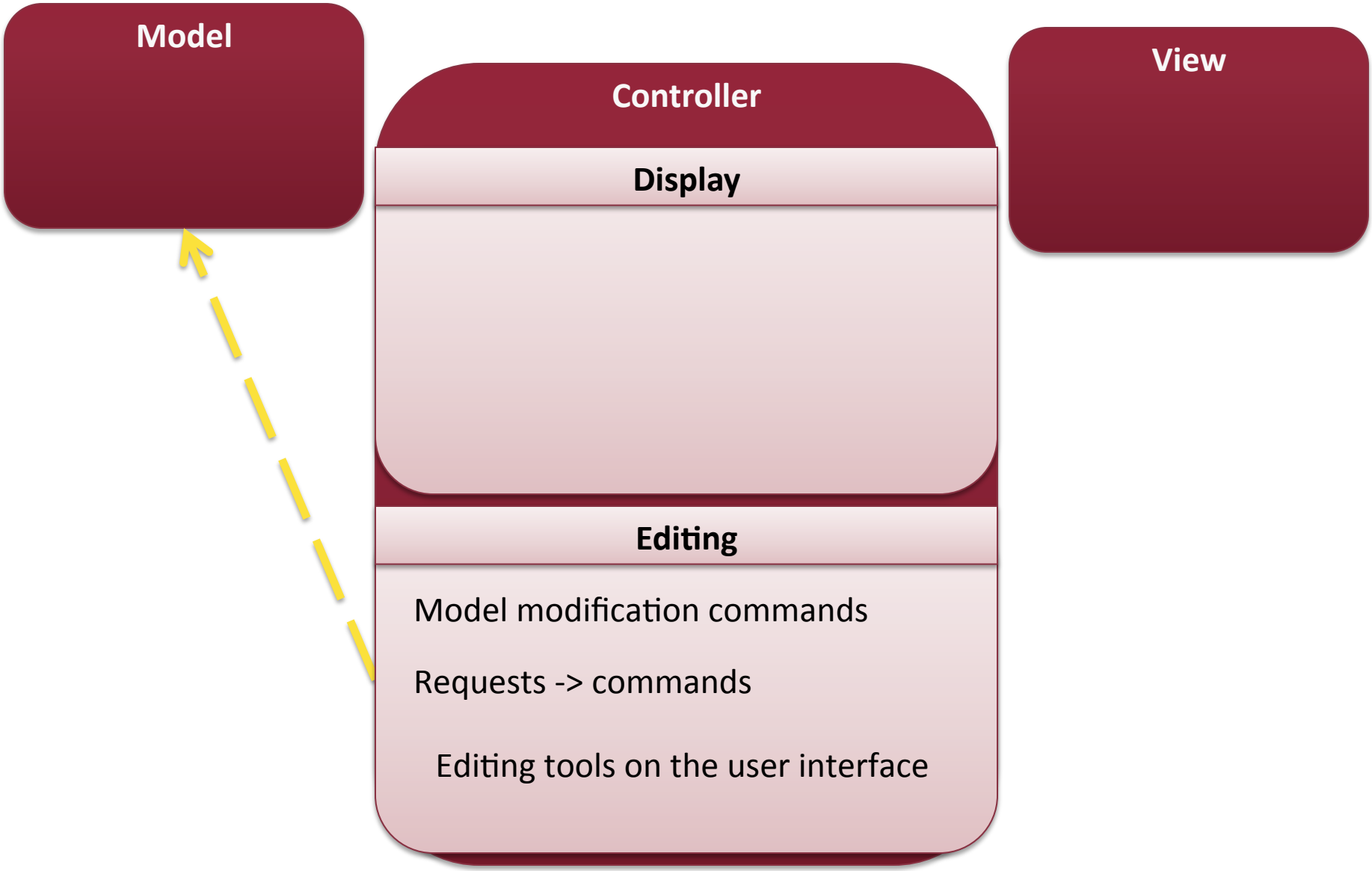
GEF workflow



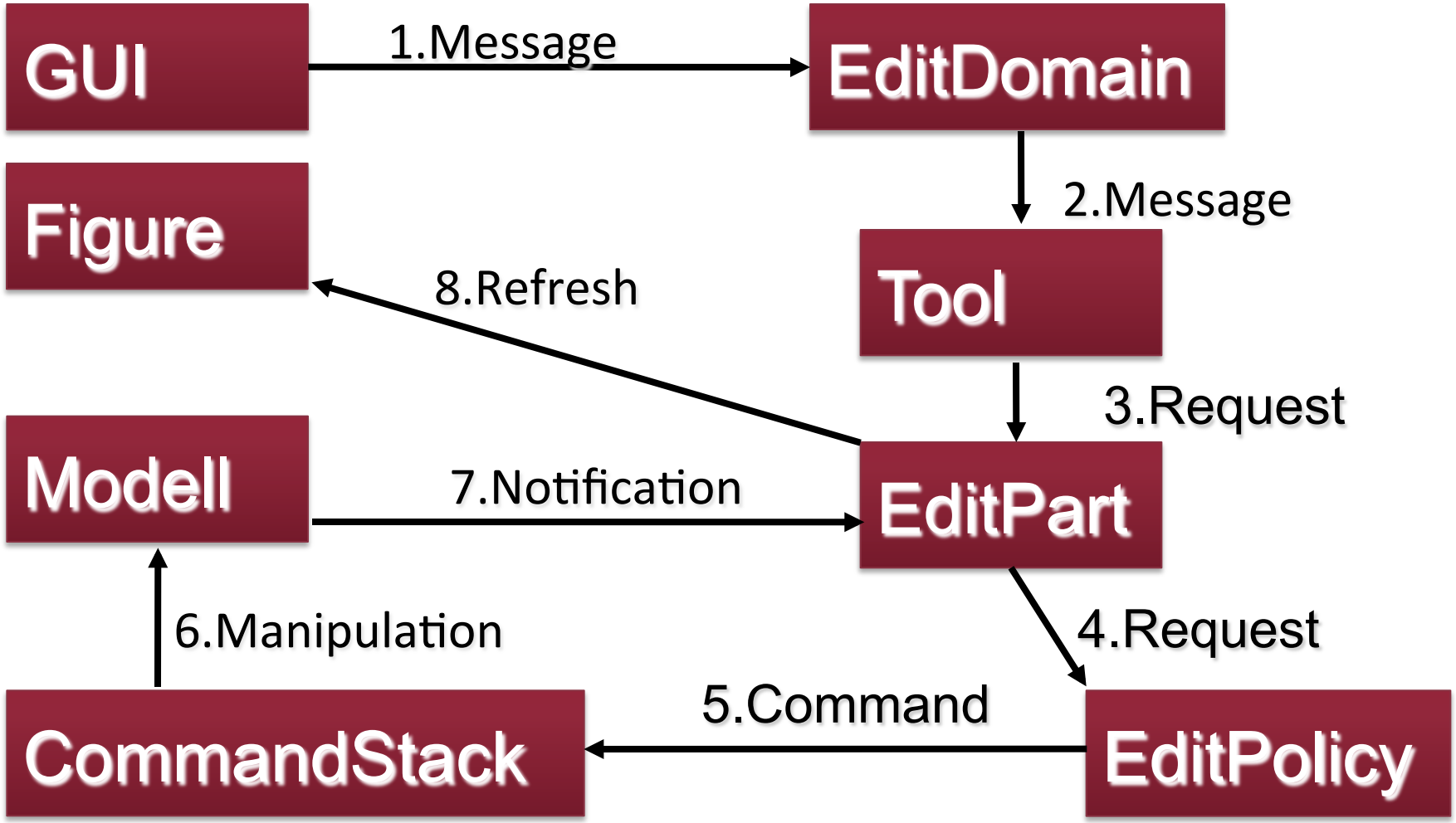
GEF workflow



GEF workflow



Editing 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.

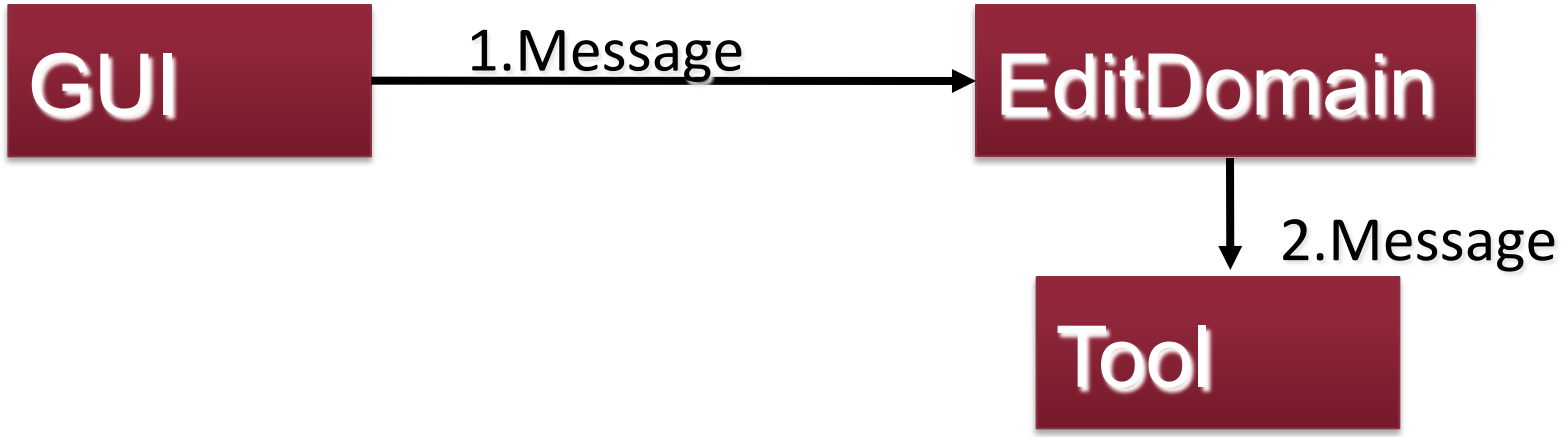
- Action
  - Not GEF-specific (JFace)
  - Custom commands
    - Menu items
  - GEF provides wrappers to connect to CommandStack
  - ActionRegistry: list of actions

Editing Workfow

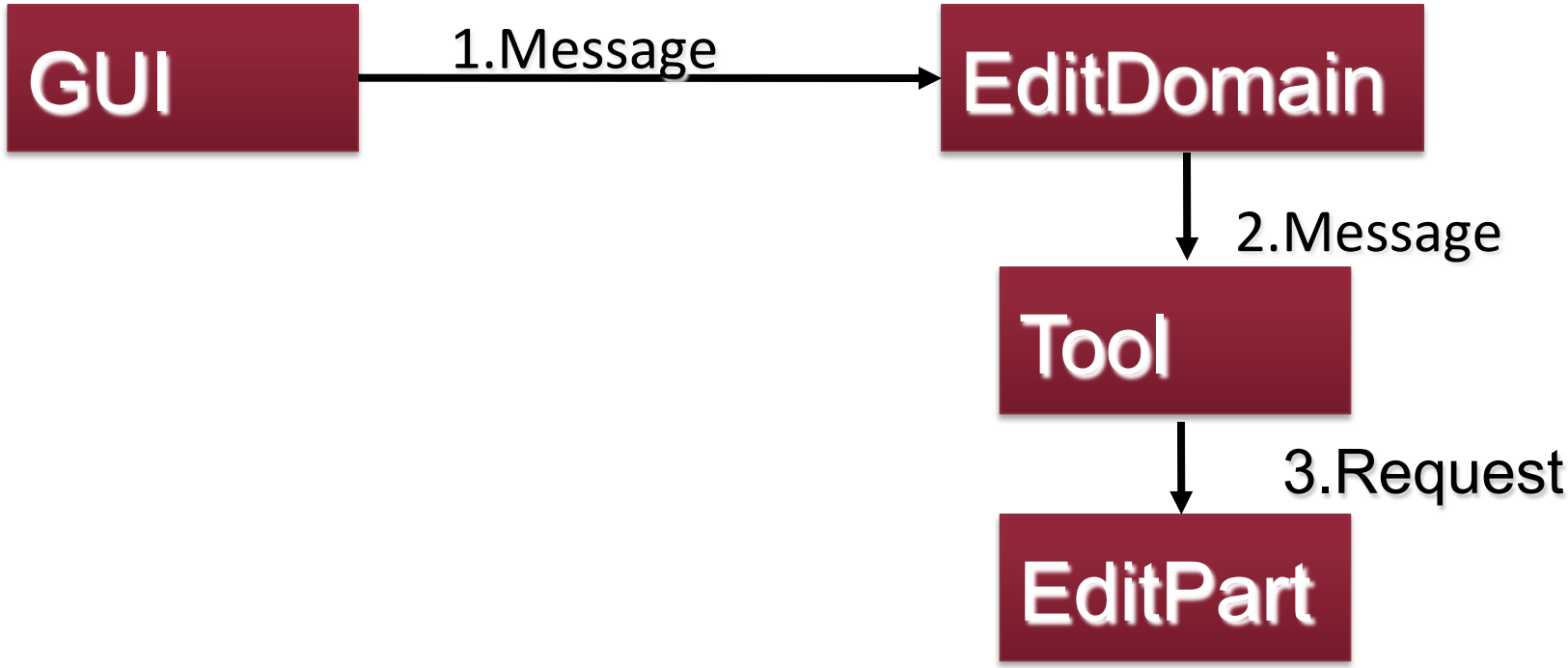
GUI

Editing Workfow

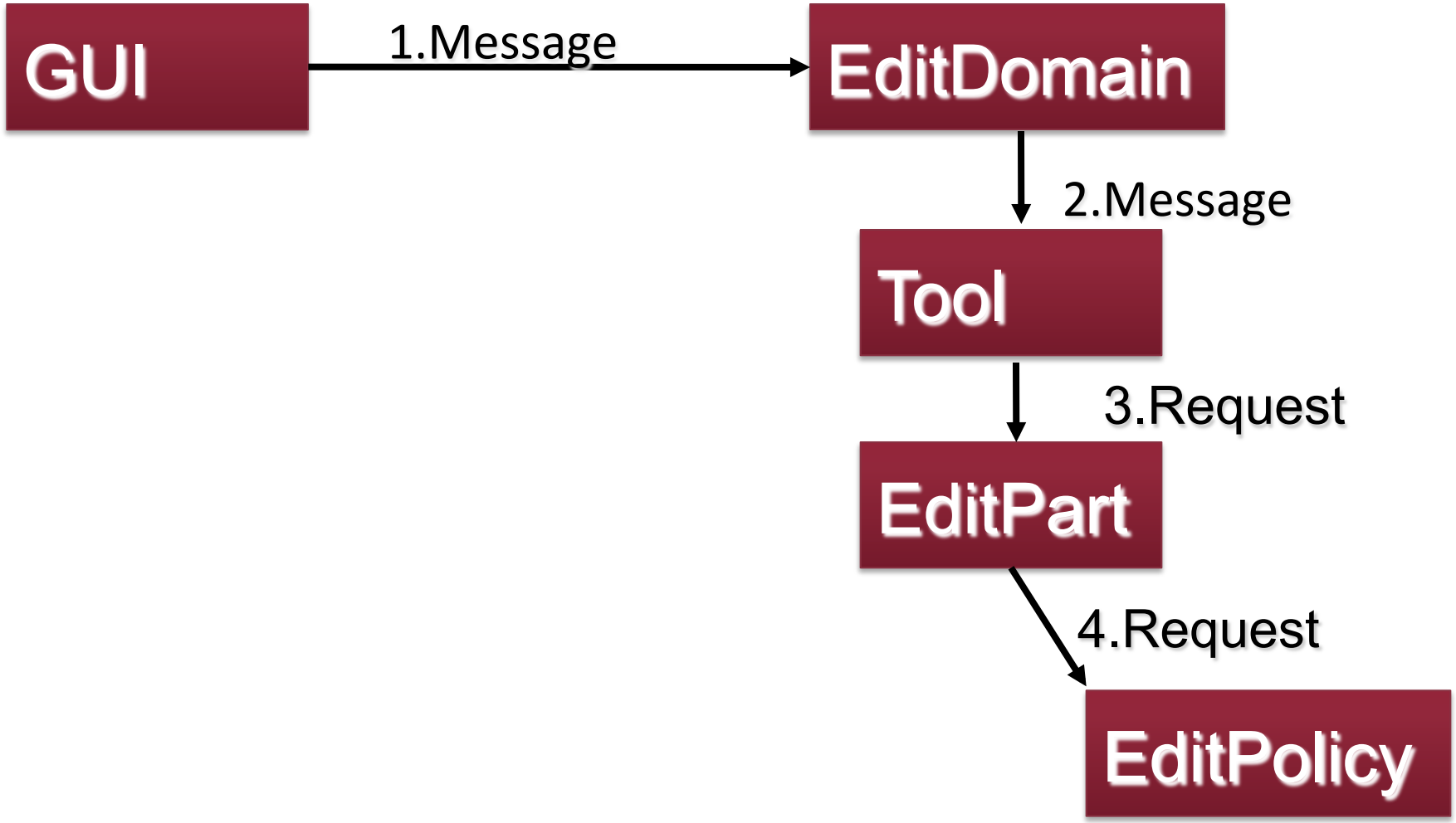
Editing Workfow



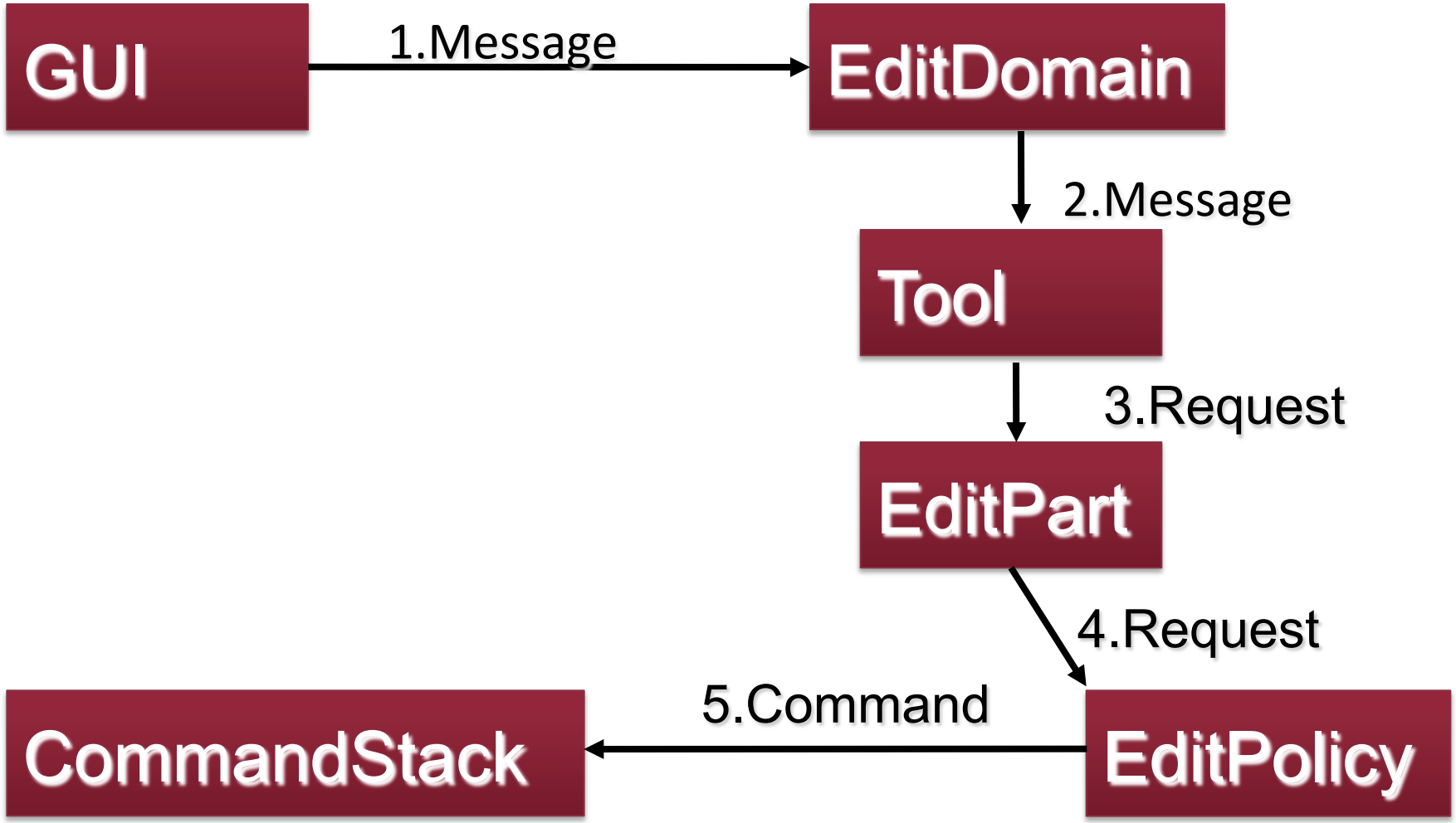
Editing Workfow



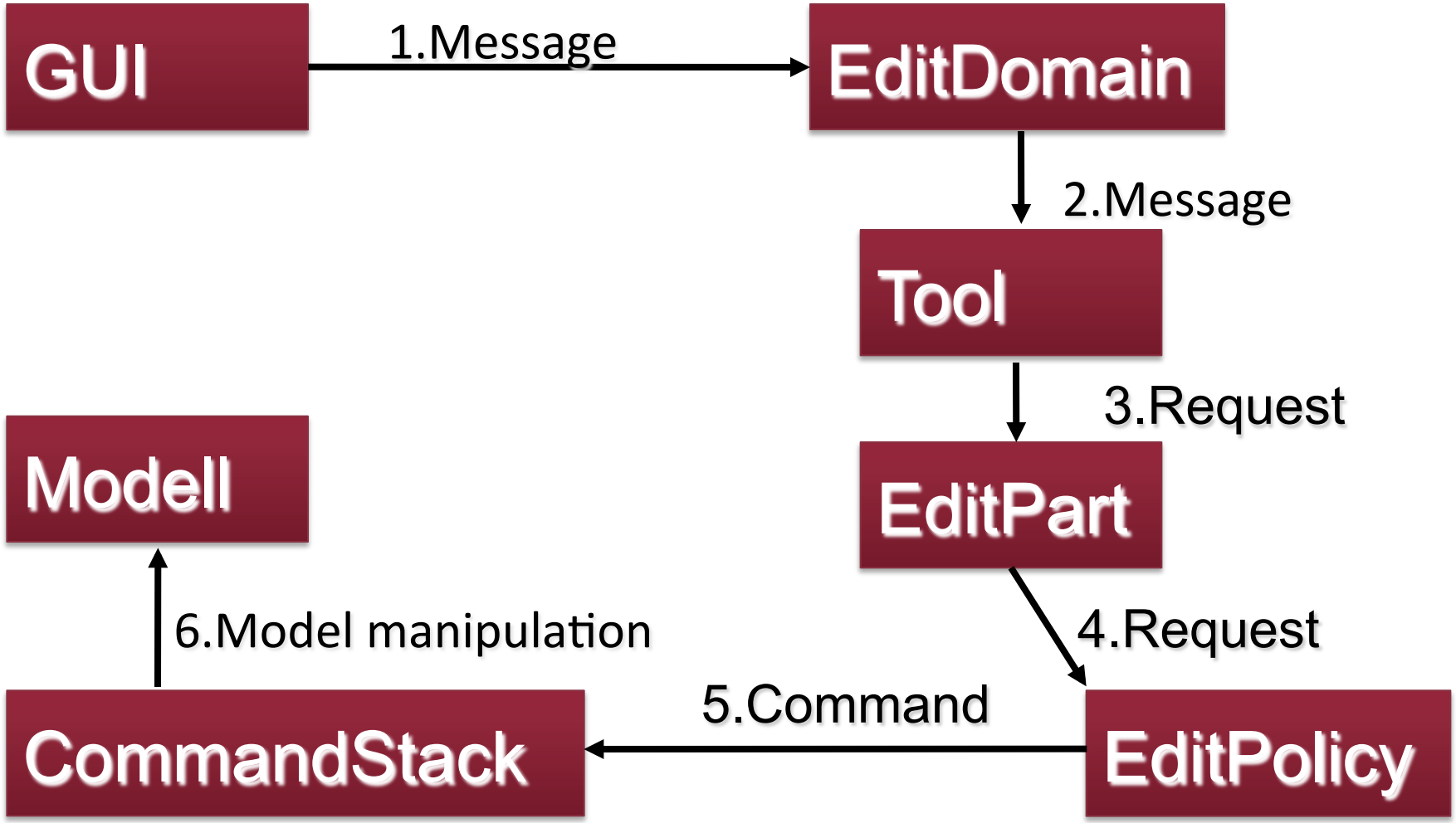
Editing Workfow



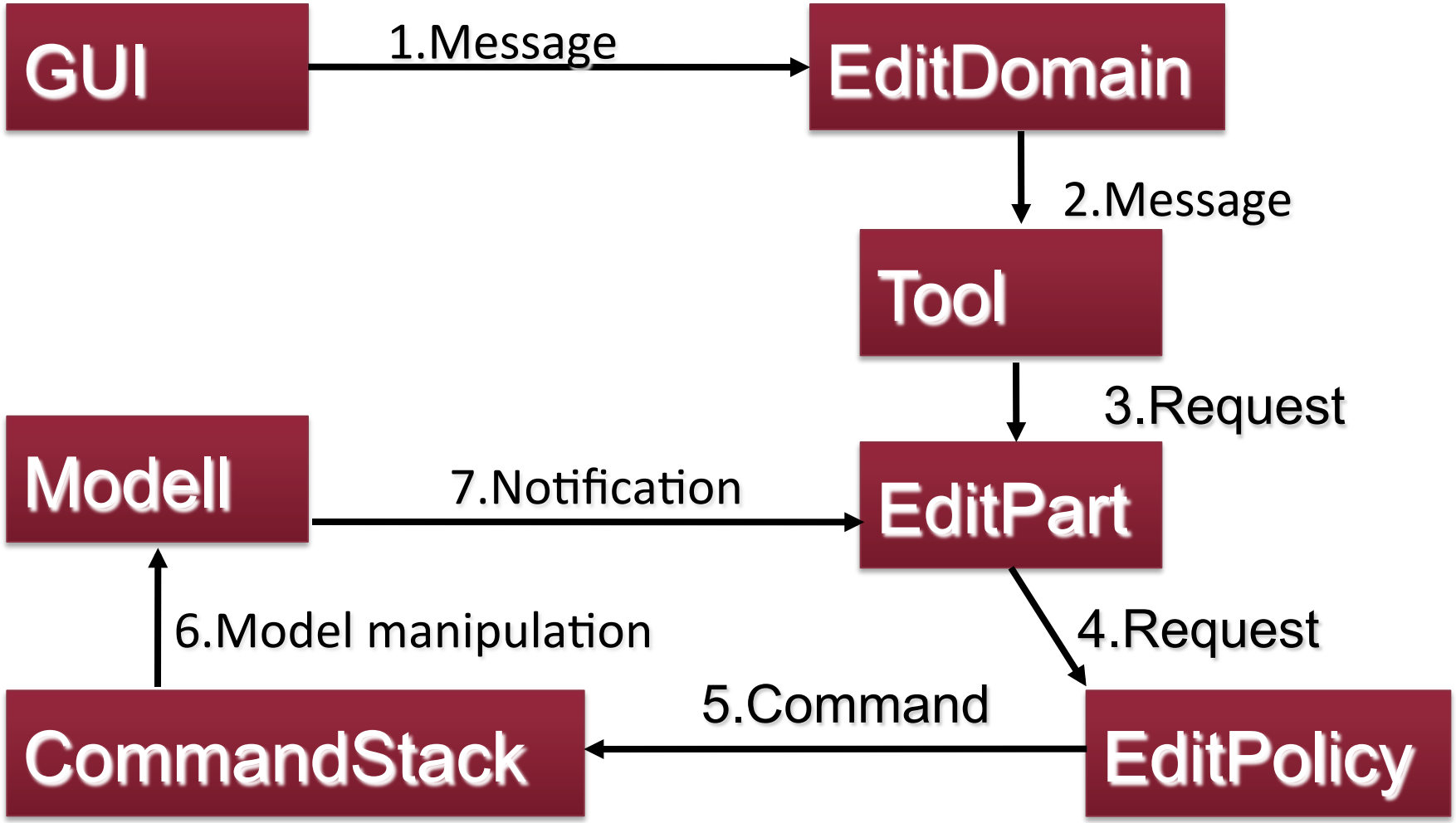
Editing Workfow



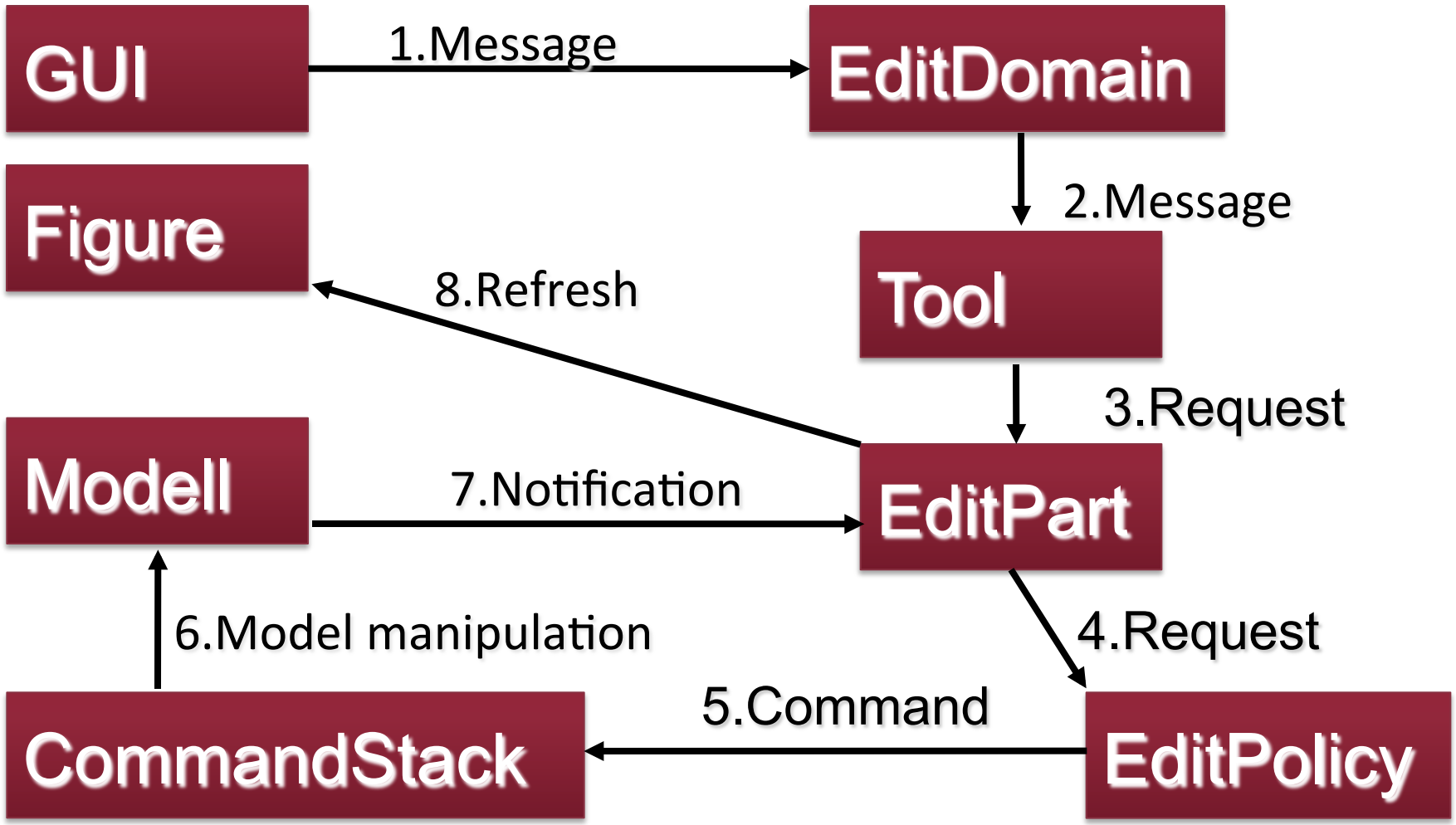
Editing Workfow



Editing Workfow



Editing Workfow



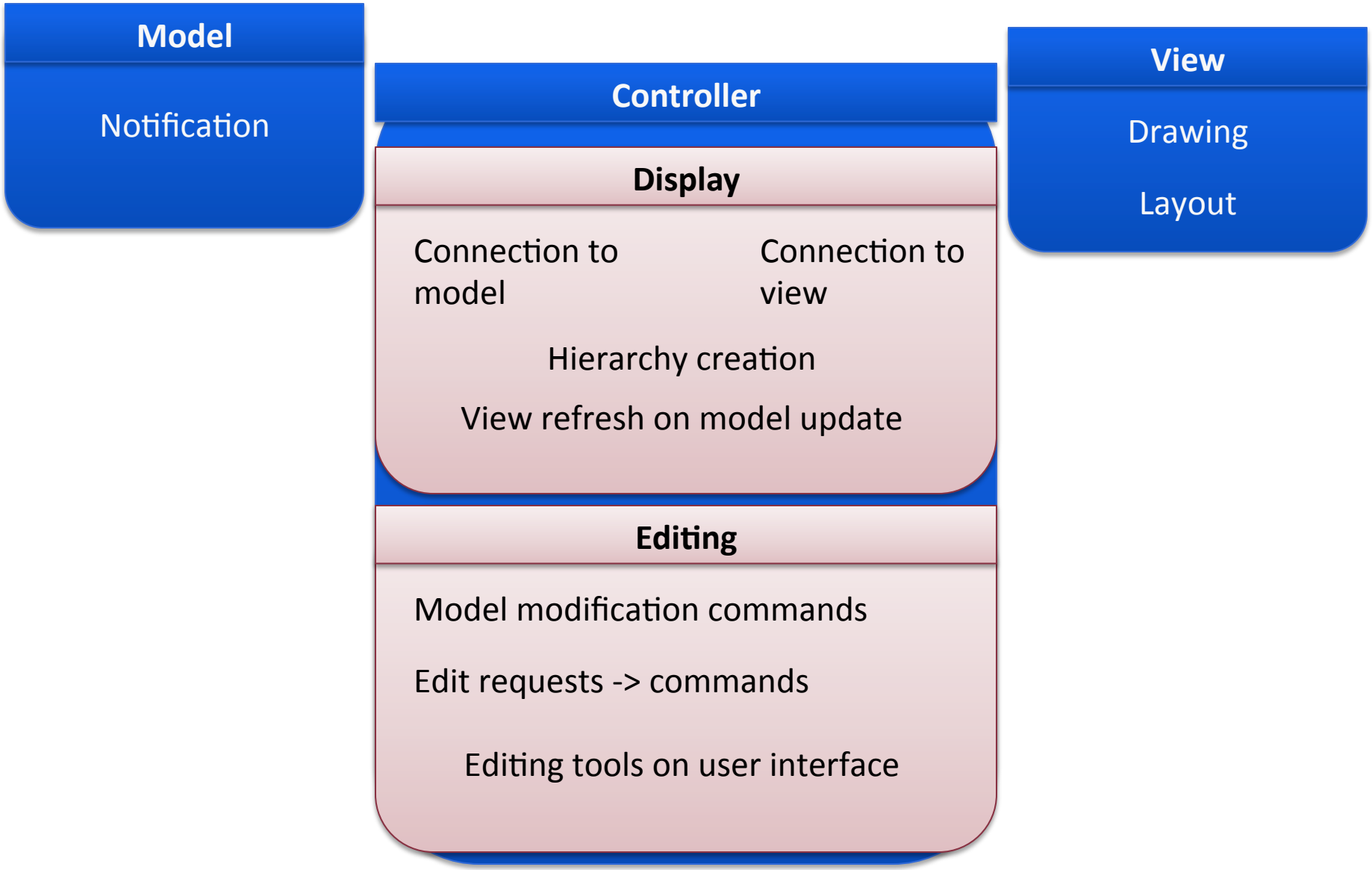
What to implement? - 1.

- 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)

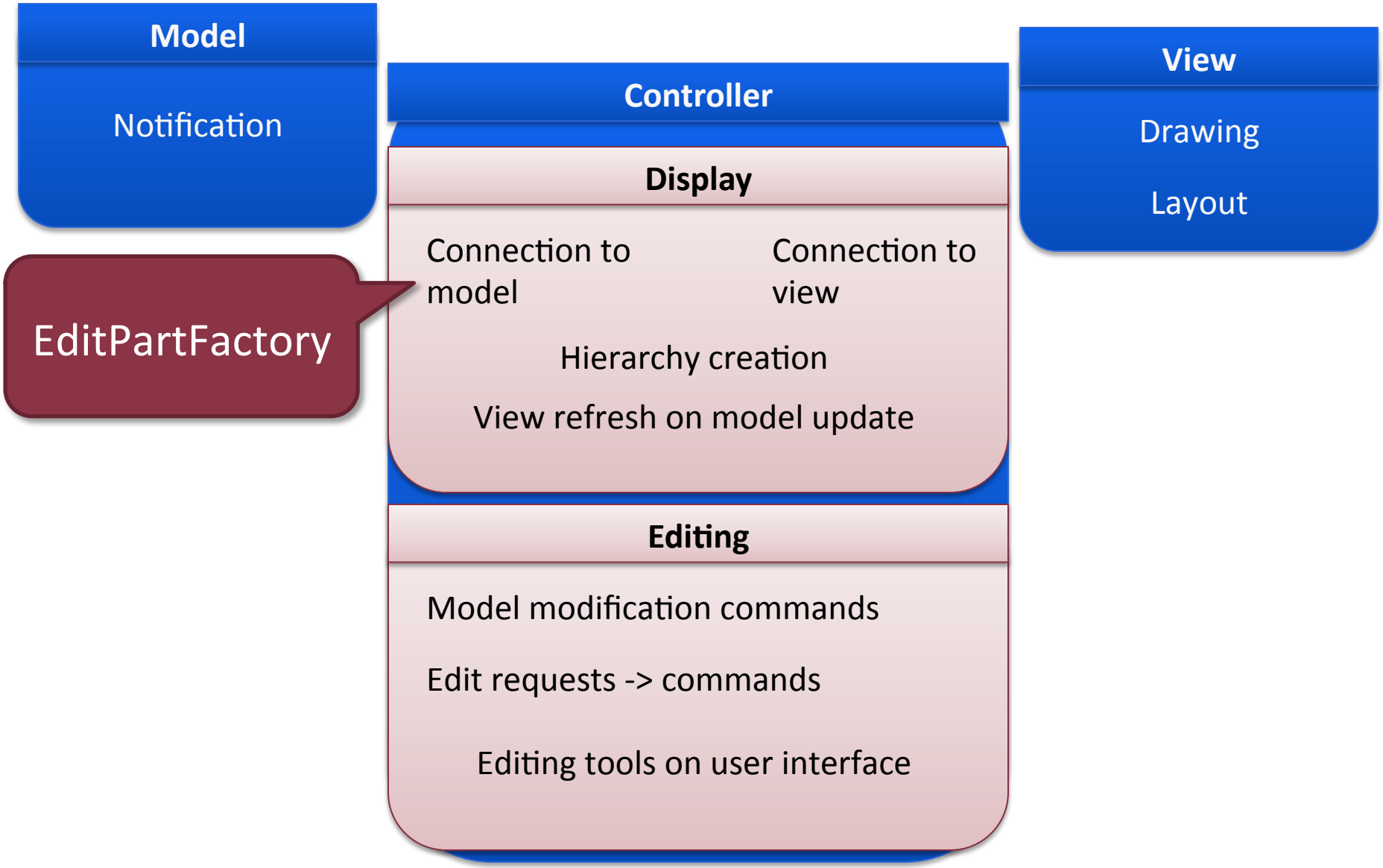
What to implement? – 2.

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

GEF workflow



GEF workflow



GEF w

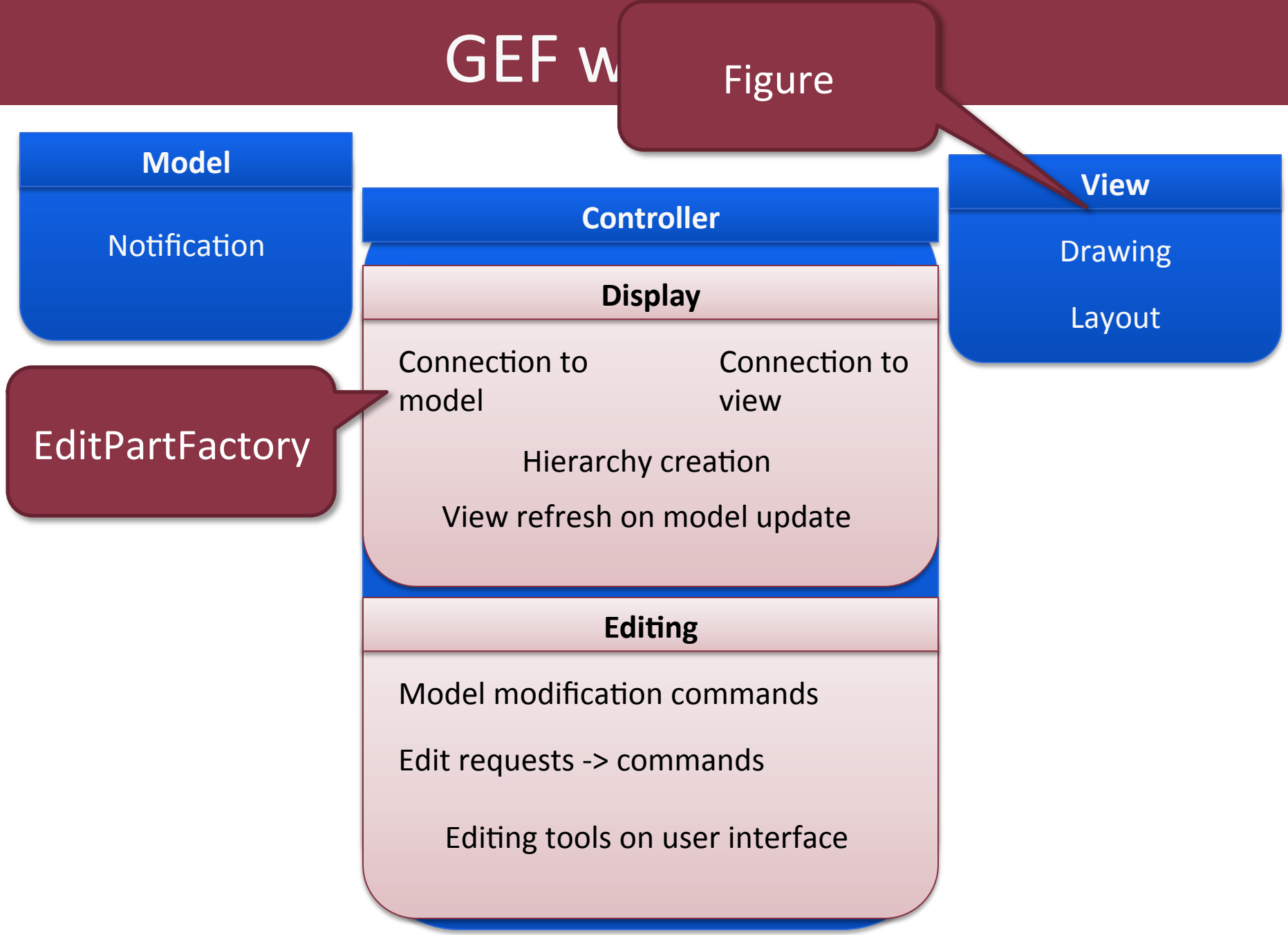


Figure
Model
Notification
Controller
Display
Connection to model
Connection to view
Hierarchy creation
View refresh on model update
Editing
Model modification commands
Edit requests -> commands
Editing tools on user interface
View
Drawing
Layout
EditPartFactory

GEF w

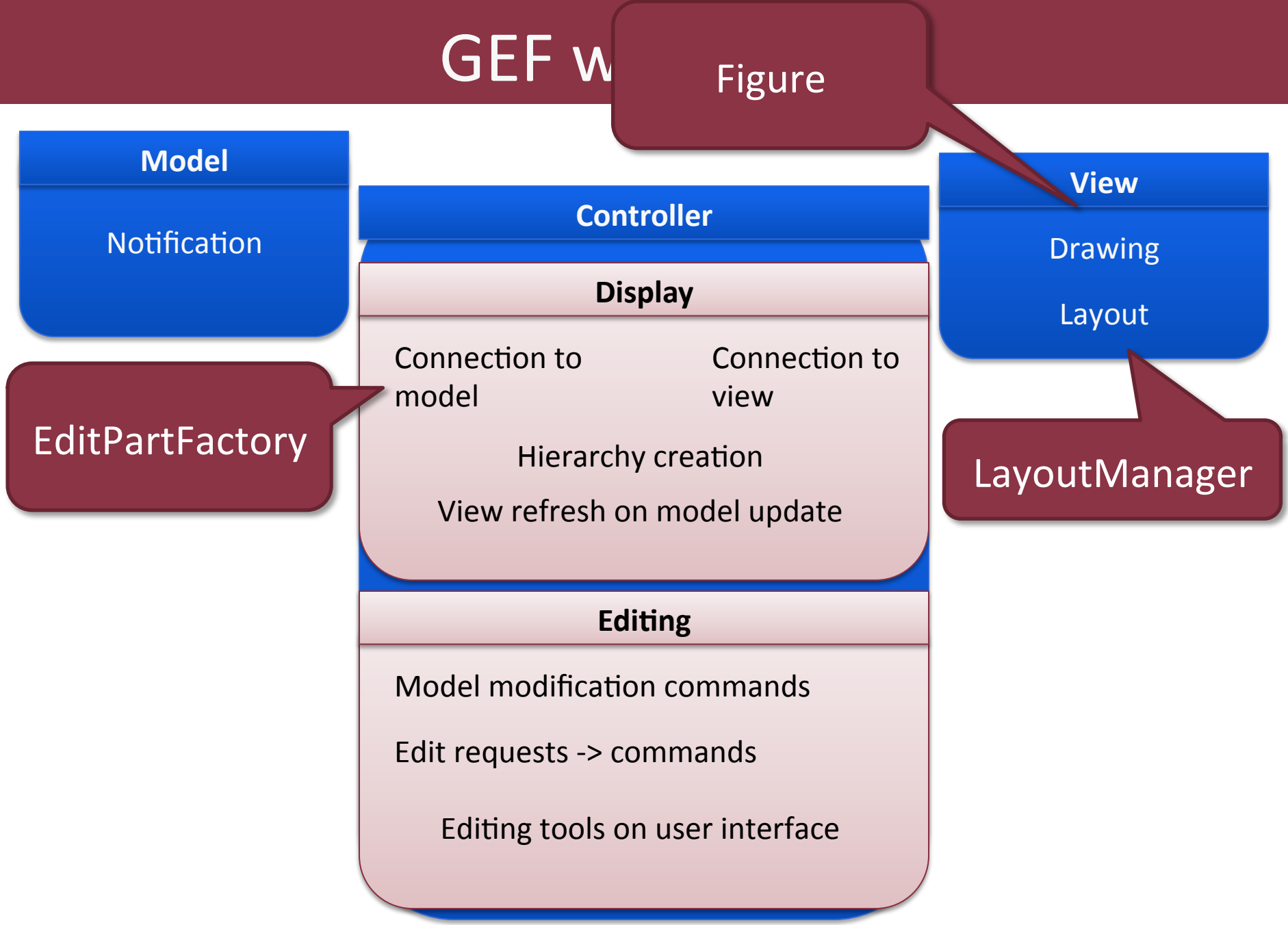


Figure
Model
Notification
Controller
Display
Connection to model
Connection to view
Hierarchy creation
View refresh on model update
View
Drawing
Layout
EditPartFactory
LayoutManager
Editing
Model modification commands
Edit requests -> commands
Editing tools on user interface

GEF w

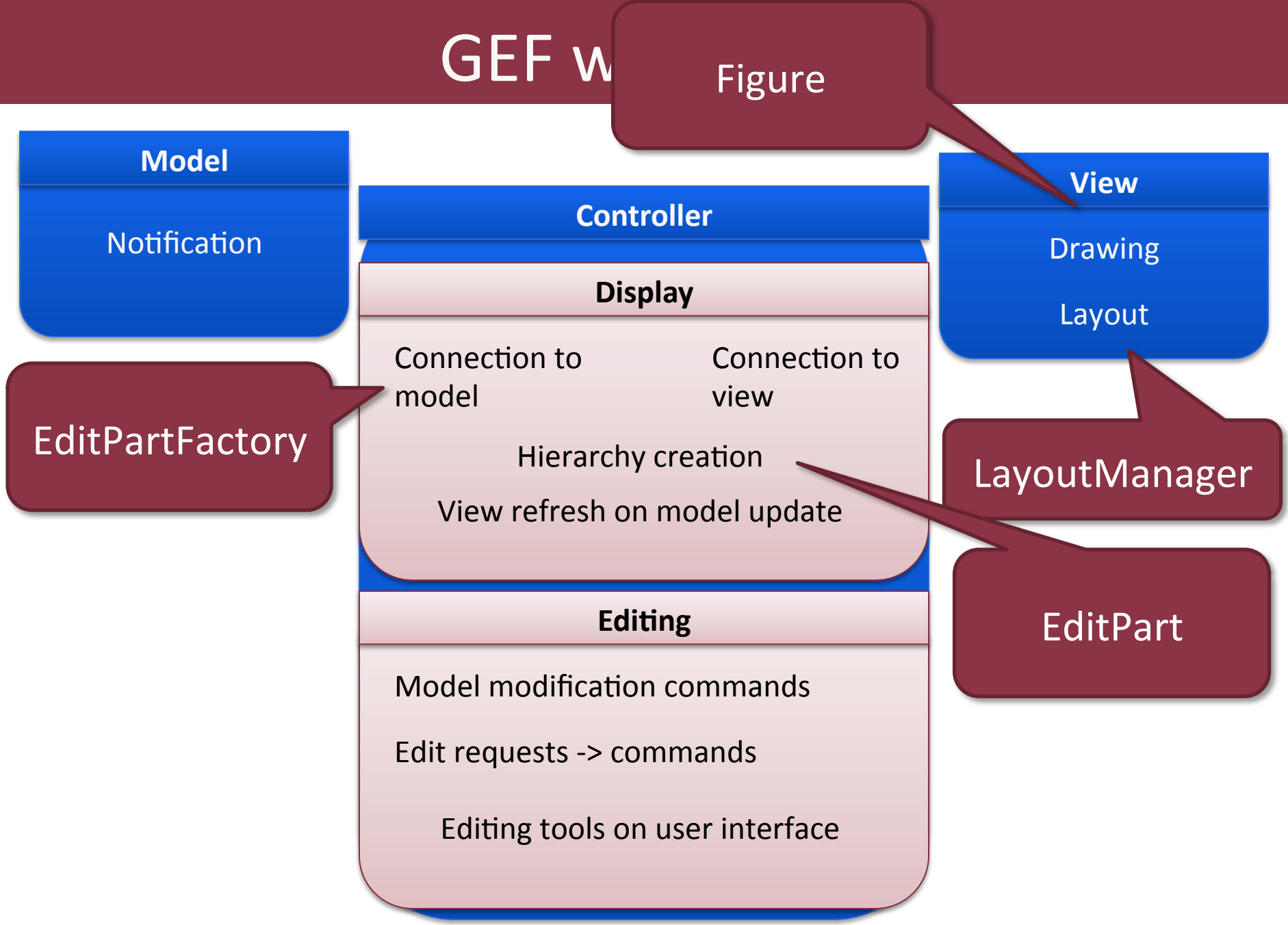


Figure
Model
Notification
Controller
Display
Connection to model
Connection to view
Hierarchy creation
View refresh on model update
Editing
Model modification commands
Edit requests -> commands
Editing tools on user interface
View
Drawing
Layout
EditPartFactory
LayoutManager
EditPart

GEF w

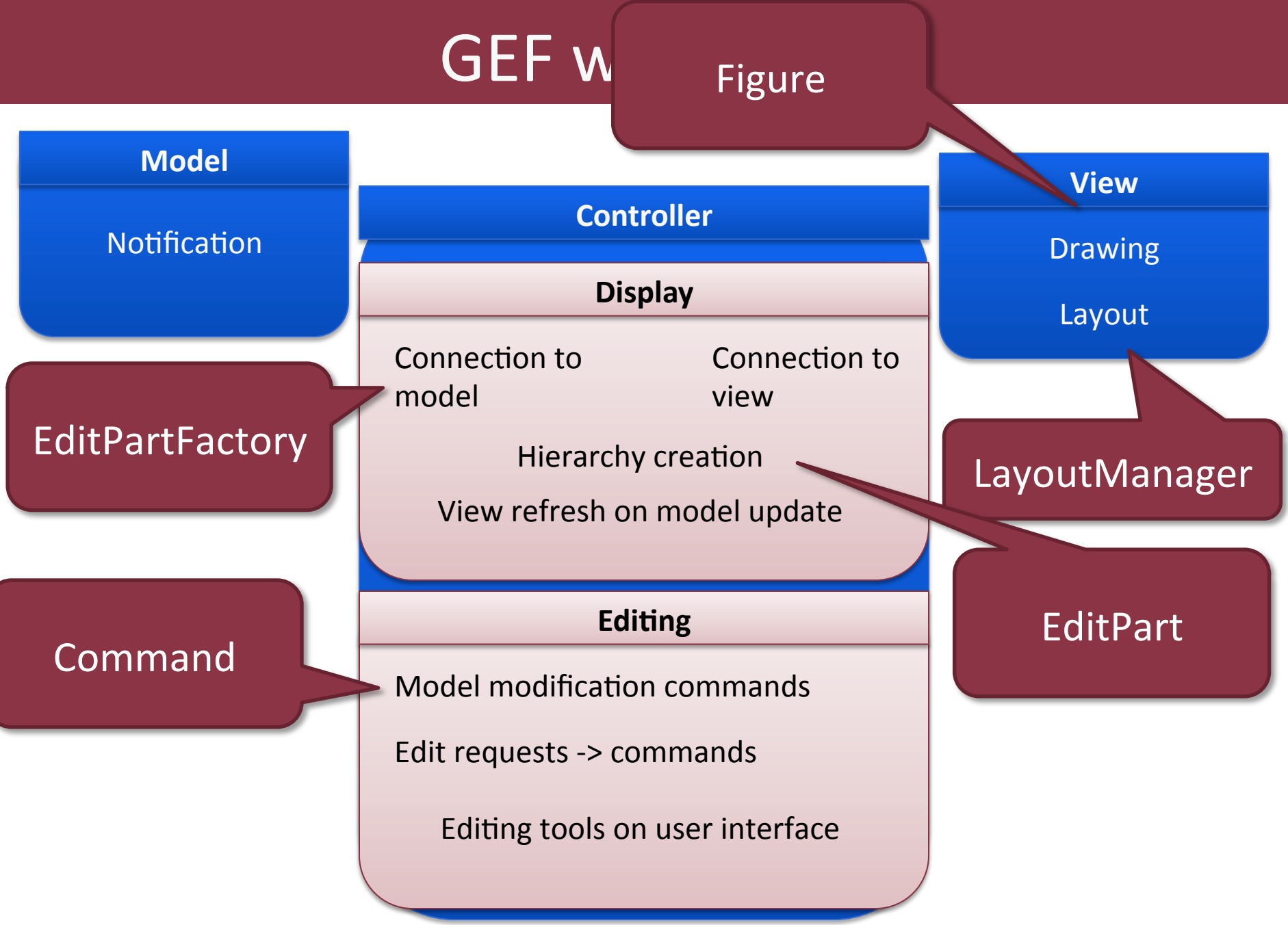


Figure
Model
Notification
Controller
Display
Connection to model
Connection to view
Hierarchy creation
View refresh on model update
View
Drawing
Layout
EditPartFactory
LayoutManager
EditPart
Command
Editing
Model modification commands
Edit requests -> commands
Editing tools on user interface

GEF w

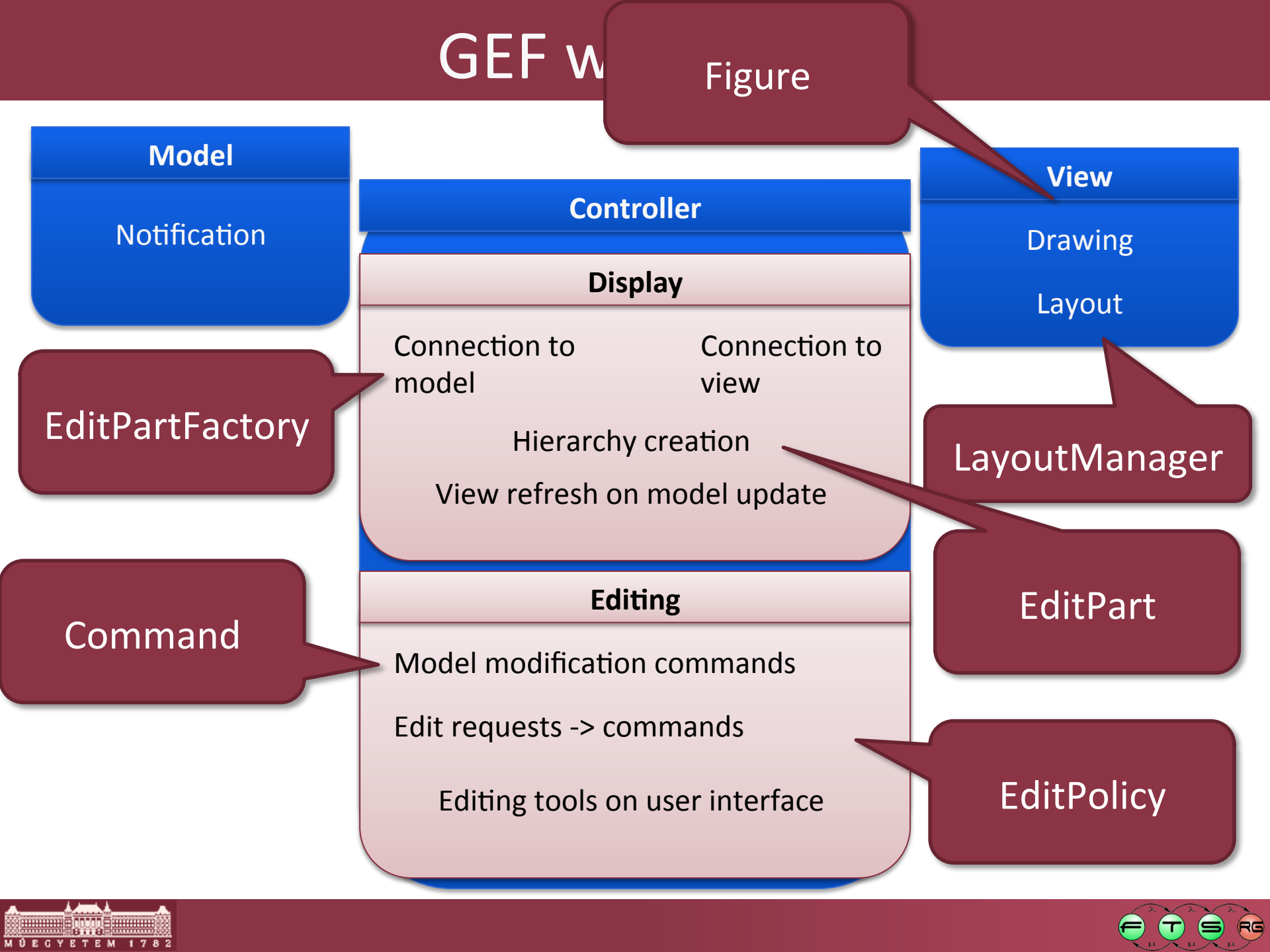


Figure
Model
Notification
Controller
Display
Connection to model
Connection to view
Hierarchy creation
View refresh on model update
Editing
Model modification commands
Edit requests -> commands
Editing tools on user interface
View
Drawing
Layout
EditPartFactory
LayoutManager
EditPart
Command
EditPolicy

GEF w

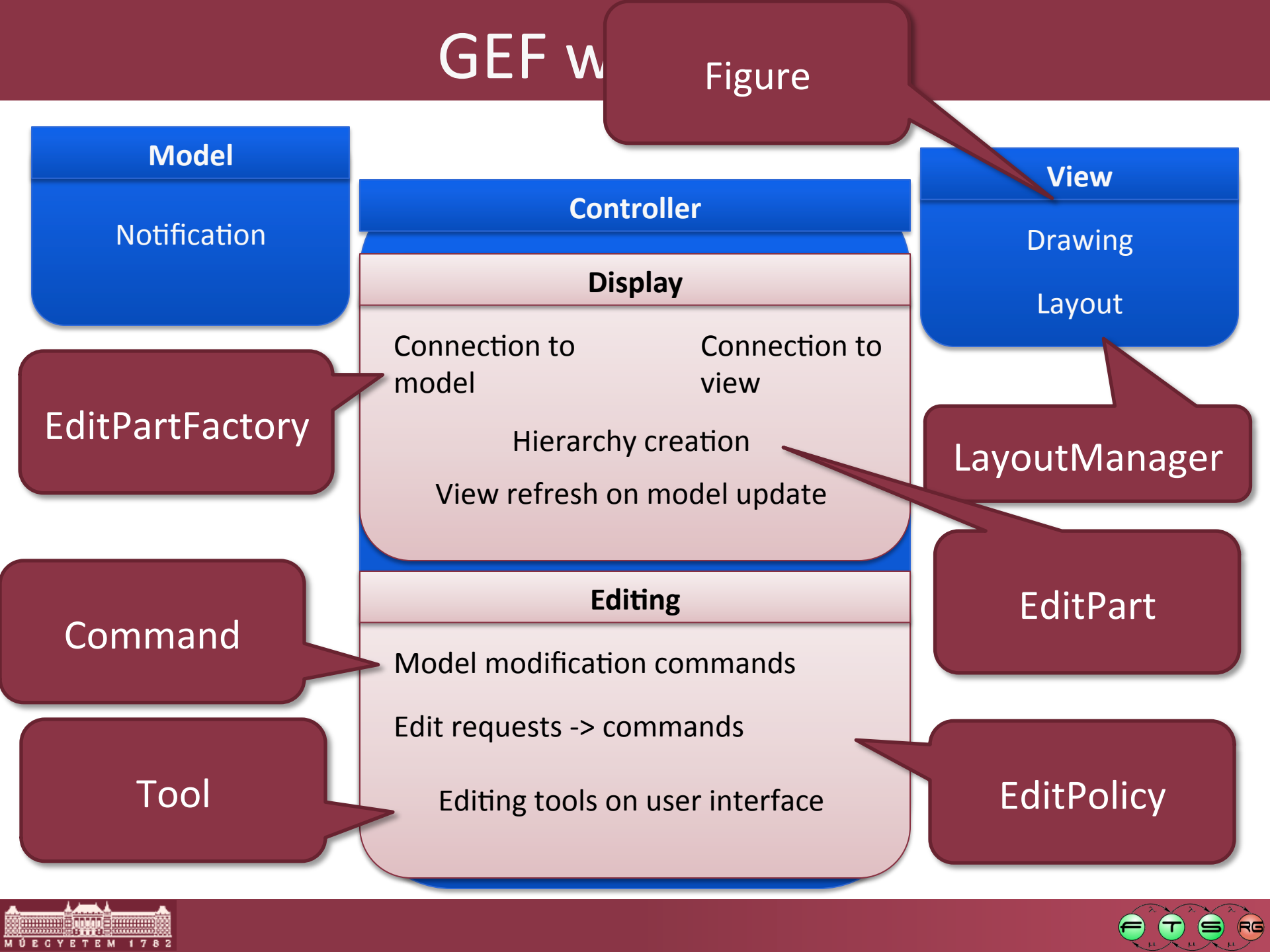


Figure
Model
Notification
Controller
Display
Connection to model
Connection to view
Hierarchy creation
View refresh on model update
View
Drawing
Layout
EditPartFactory
LayoutManager
EditPart
Editing
Model modification commands
Edit requests -> commands
Editing tools on user interface
Command
Tool
EditPolicy

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

Problems using GEF

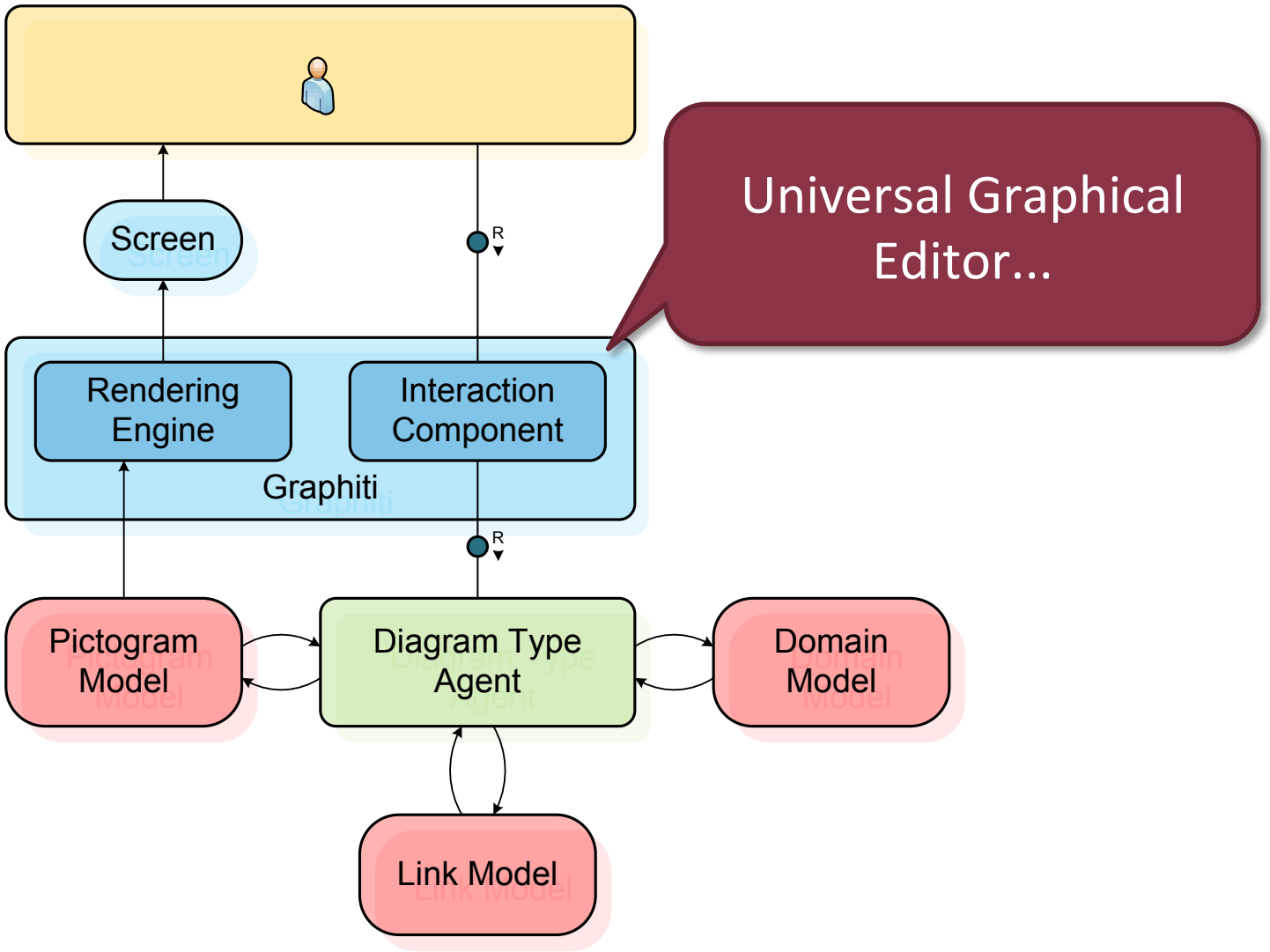
- Very complex
- Lot of redundant, hand-written code

- 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

Simplfication

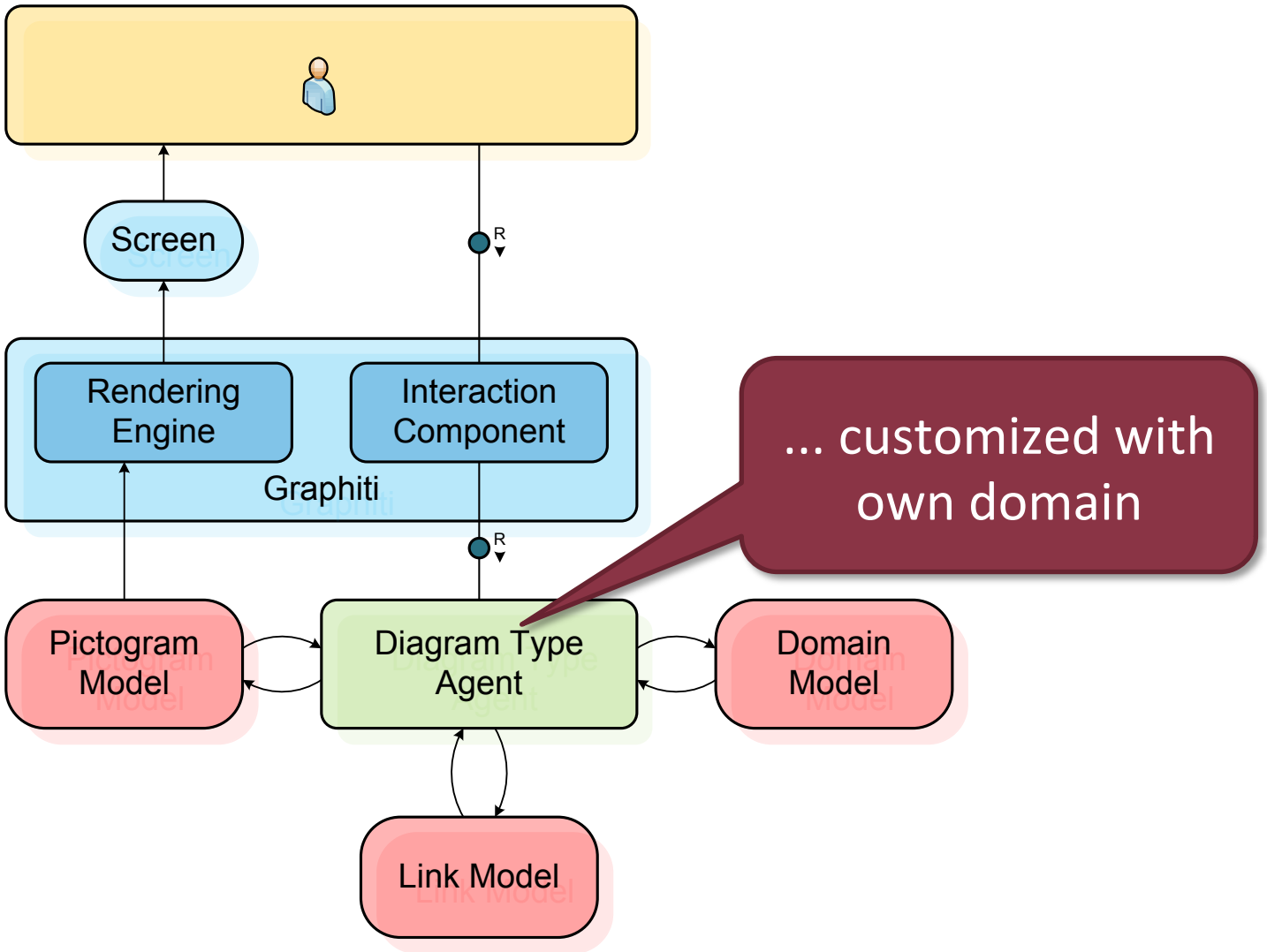
- EMF models
  - Uniform handling
  - Load/save simple
  - Interfile dependencies manageable
- Simpler interaction
  - Only high-level functions directly supported

Architecture of Graphiti



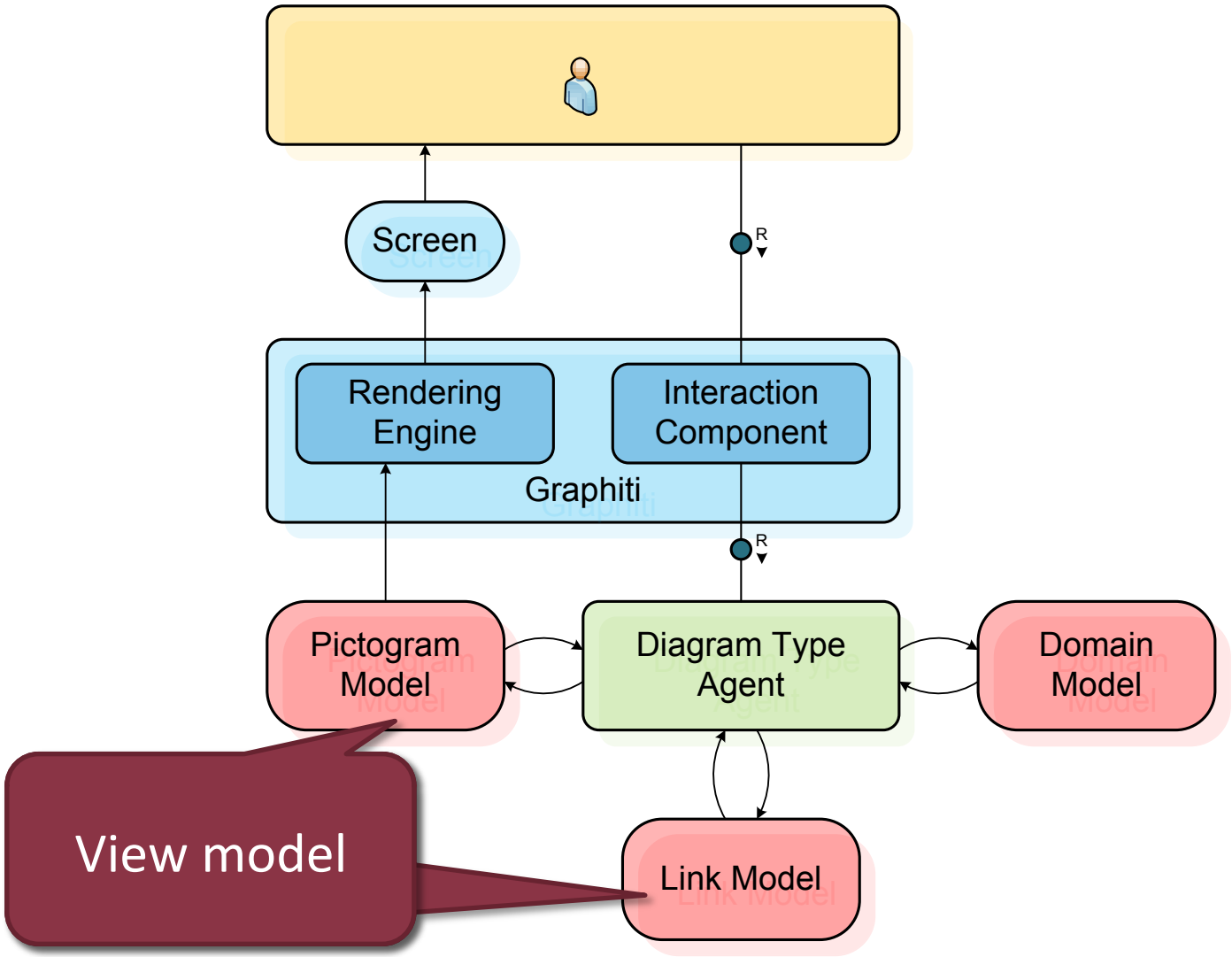
Forrás: http://www.slideshare.net/michaelwenz/short-talk-on-graphiti-at-eclipsecon-2010

Architecture of Graphiti



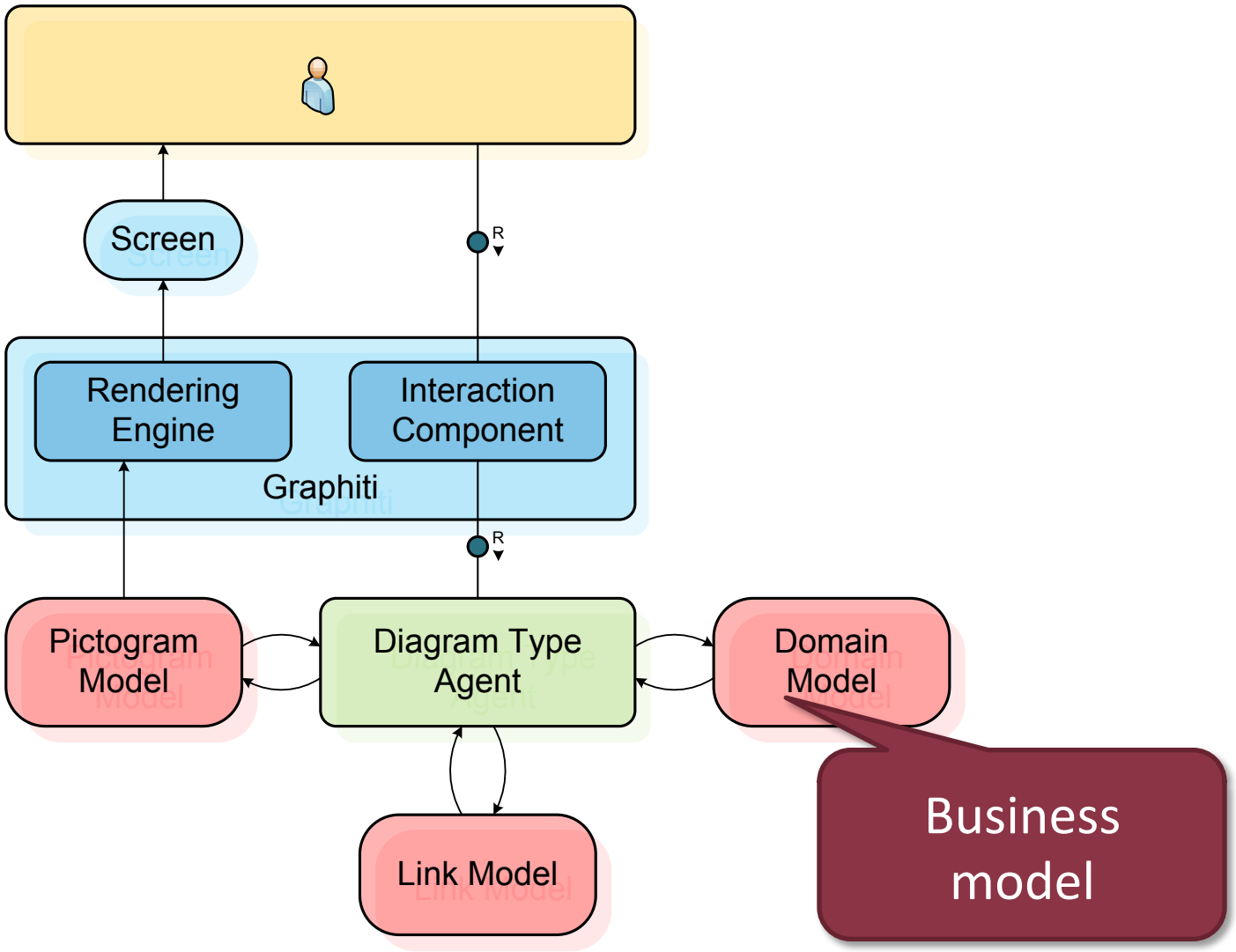
Forrás: http://www.slideshare.net/michaelwenz/short-talk-on-graphiti-at-eclipsecon-2010

Architecture of Graphiti



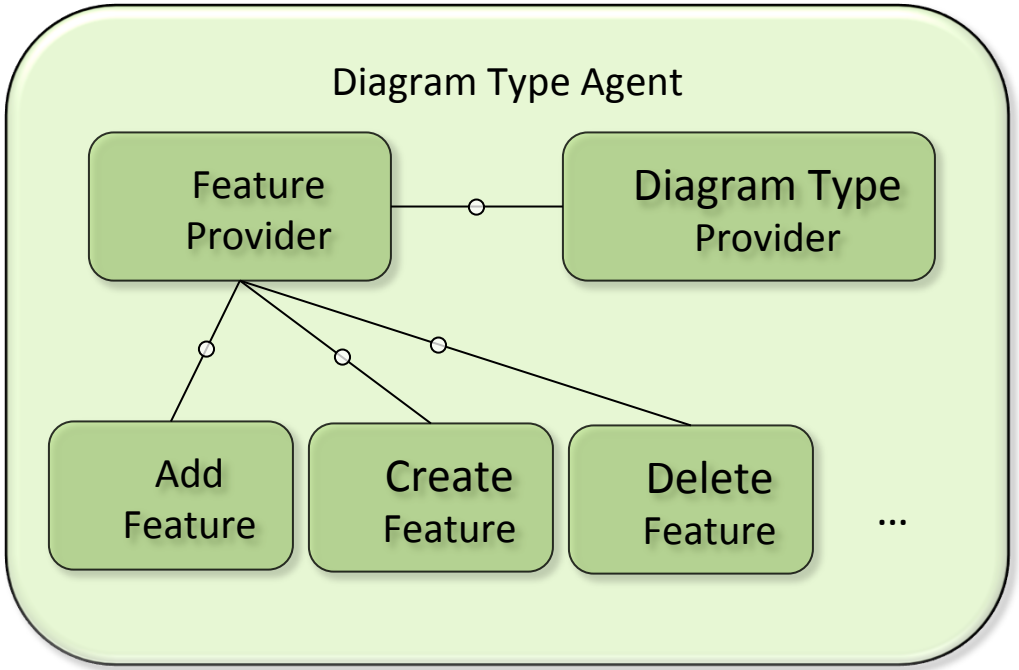
Forrás: http://www.slideshare.net/michaelwenz/short-talk-on-graphiti-at-eclipsecon-2010

Architecture of Graphiti



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

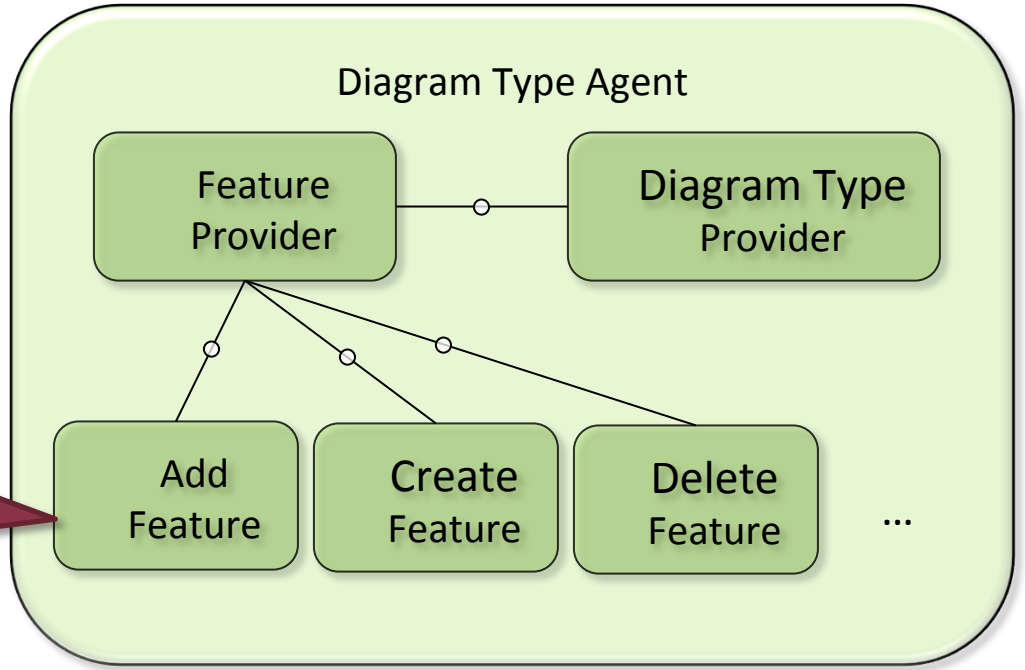
Graphiti architekture 2. – Diagram Type Agent



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

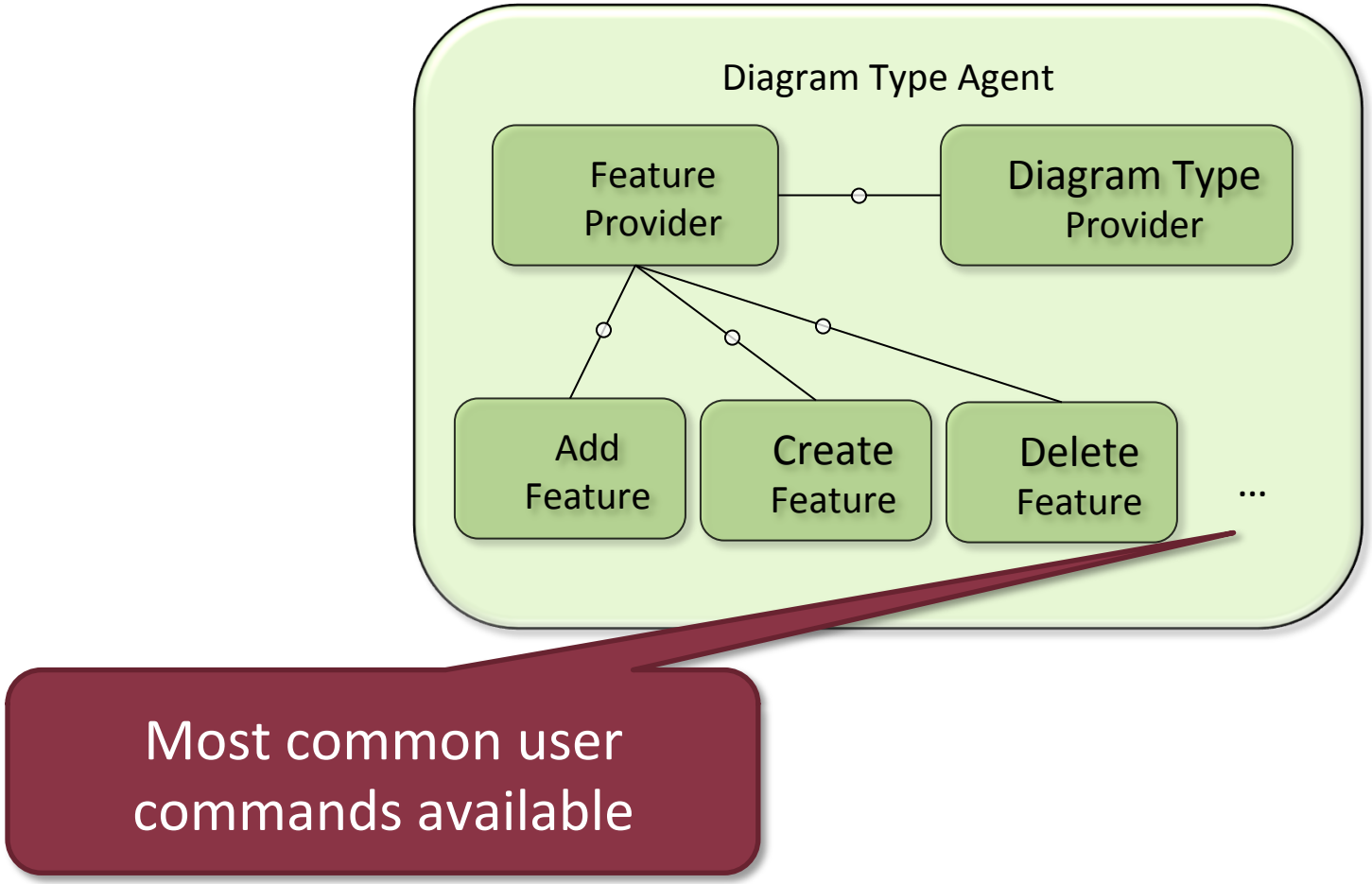
Graphiti architekture 2. – Diagram Type Agent

Feature: editing operation on business model



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

Graphiti architekture 2. – Diagram Type Agent



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

Basic terms

Domain

Links

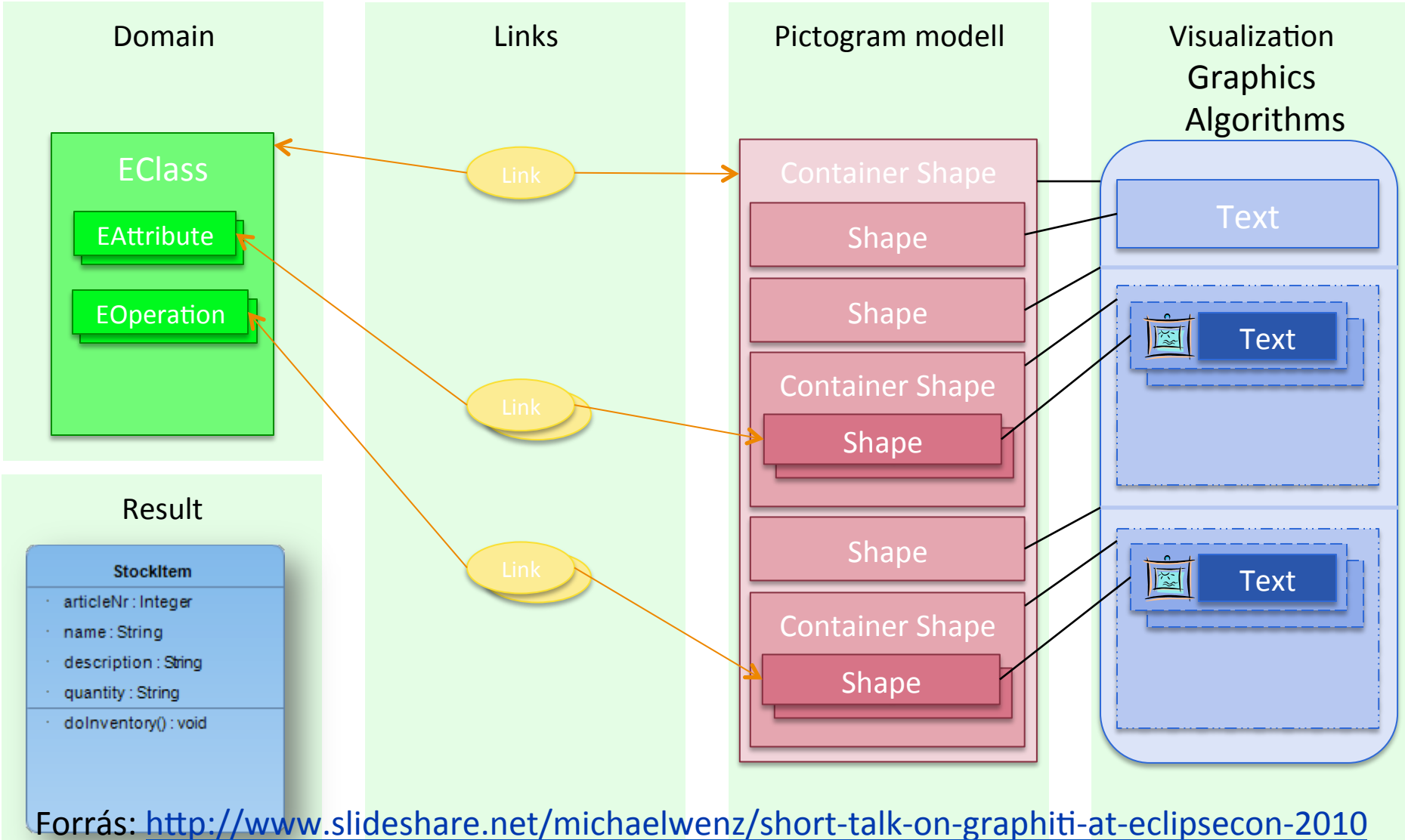
Pictogram modell

Visualization
Graphics
Algorithms

Result

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

Basic terms



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

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

Tool building steps

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

DiagramTypeProvider

```
public class SocialDiagramTypeProvider
      extends AbstractDiagramTypeProvider
      implements IDiagramTypeProvider {

      public SocialDiagramTypeProvider() {
      setFeatureProvider(new
SocialNetworkFeatureProvider(this));
      }
}
```

DiagramTypeProvider

```
public class SocialDiagramTypeProvider
      extends AbstractDiagramTypeProvider
      implements IDiagramTypeProvider {

      public SocialDiagramTypeProvider() {
      setFeatureProvider(new
SocialNetworkFeatureProvider(this));
      }
}
```

Feature Provider registration

Tool building steps

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

Feature Provider registration

- Two extension points
  - Type definition
    - Simple description
    - org.eclipse.graphiti.ui.diagramTypes
  - Implementation
    - Mapping type definition and feature provider
    - org.eclipse.graphiti.ui.diagramTypeProviders

Tool building steps

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

Feature Provider

- Re-use *AbstractFeatureProvider* class
- Feature registration into corresponding method
  - Return null if not applicable
- Multiple features
  - Create: creating model elements
  - Add: add existing elements to diagram
  - Copy, Paste
  - Update
  - DirectEditing

Tool building steps

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

Implementing features

- Abstract implementations available
  - E.g., AbstractAddShapeFeature
- Implementation depends on type
  - E.g., canAdd and add methods

Graphiti - Summary

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