AOT LAB

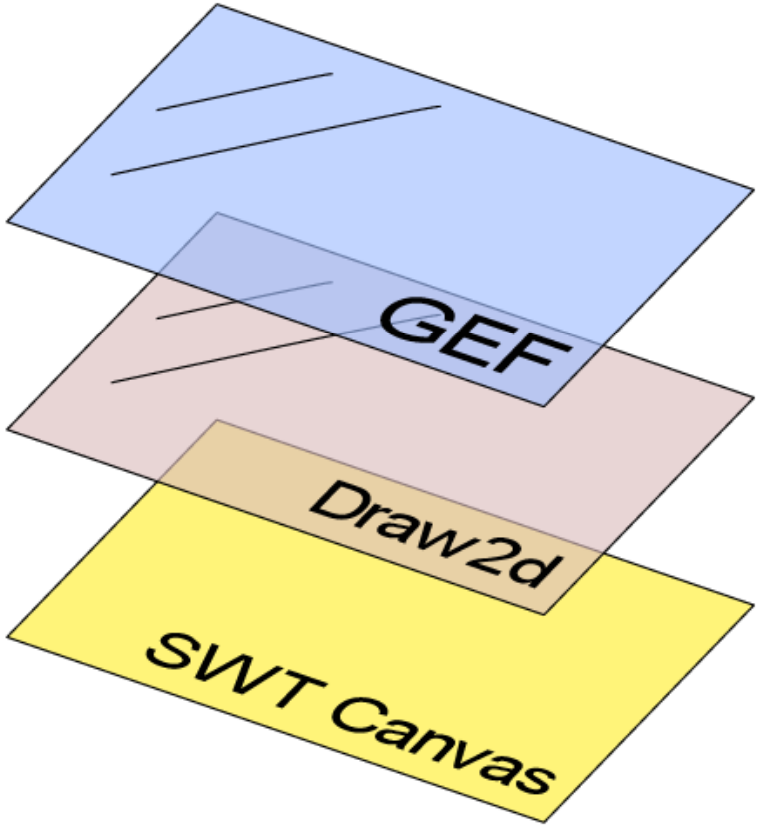
Agent and **O**bject **T**echnology **Lab**
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma

Graphical Editing Framework (GEF)

Alessandro Negri
negri@ce.unipr.it
http://www.ce.unipr.it/people/negri/

AOT LAB

Overview



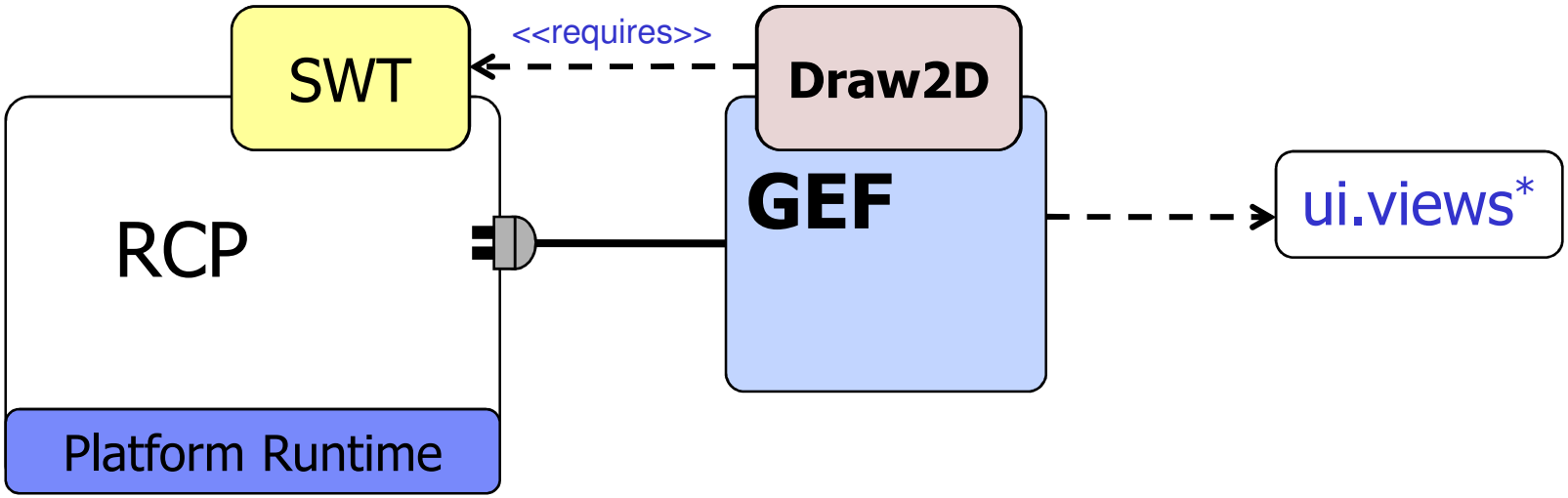
- Interaction Layer
- Model-to-View mapping
- Workbench Integration

- Rendering
- Layout
- Scaling

- Native (SWT) Layer

AOT LAB

GEF Components & Dependencies



*Optional

GEF Overview

- Consists of 2 Eclipse Plug-ins
  - Draw2D: Layout and rendering toolkit for displaying graphics
  - GEF: Framework using the old Smalltalk MVC Principles (Model, Figure and EditPart)
- General Working Mechanism:
  - Input events are translated into Requests
  - EditParts hand over Requests to EditPolicies
  - EditPolicies translate Requests into GEF Commands (when they know of the particular Request)
  - Commands get executed and change the model
  - Model is observed by EditPart: upon a change the EditPart refreshes the Figure

GEF Features

- Move, resize, create, bend, connect
- Delete, undo, redo, direct-edit
- Overview & zoom
- Palette and workbench view
- Palette customization
- Rulers & guides
- Snap-To-Grid or “geometry”
- Accessibility
- Shortest-path routing algorithm
- Directed graph layout algorithm

AOT LAB

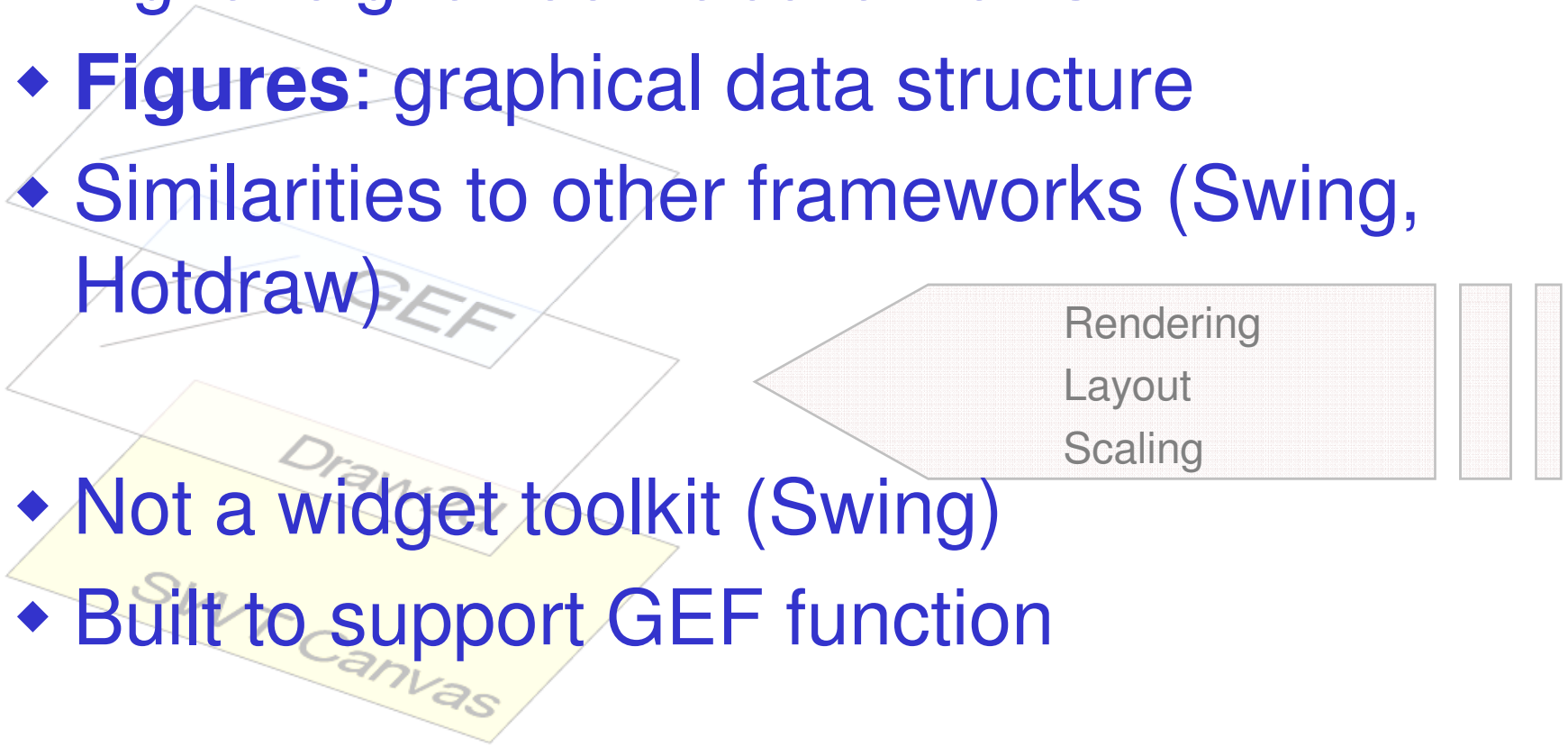
A Minimal GEF Editor

- Creation of Eclipse plug-in with Editor
- Add the GraphicalViewer: ScrollingGraphicalViewer
  - Special kind of EditPartViewer
  - EditPartViewers are adapters for SWT controls that manage the EditParts
  - They are populated by setting their contents
- Add the RootEditPart: ScalableFreeFormRootEditPart
  - Brigdes gap between EditPartViewer and its contents
  - Can provide servies such as zooming and freeform figures
- Define the model to be used: Graph
  - Initially a simplistic subclass of Object
  - Will be a container for Nodes later
- Create the view and the controller: GraphEditPart
- Define the GraphEditPartFactory
  - Only Graph objects will be considered initially

AOT LAB

Draw2d Overview

- Lightweight Toolkit built with SWT
- **Figures**: graphical data structure
- Similarities to other frameworks (Swing, Hotdraw)



- Not a widget toolkit (Swing)
- Built to support GEF function

AOT LAB

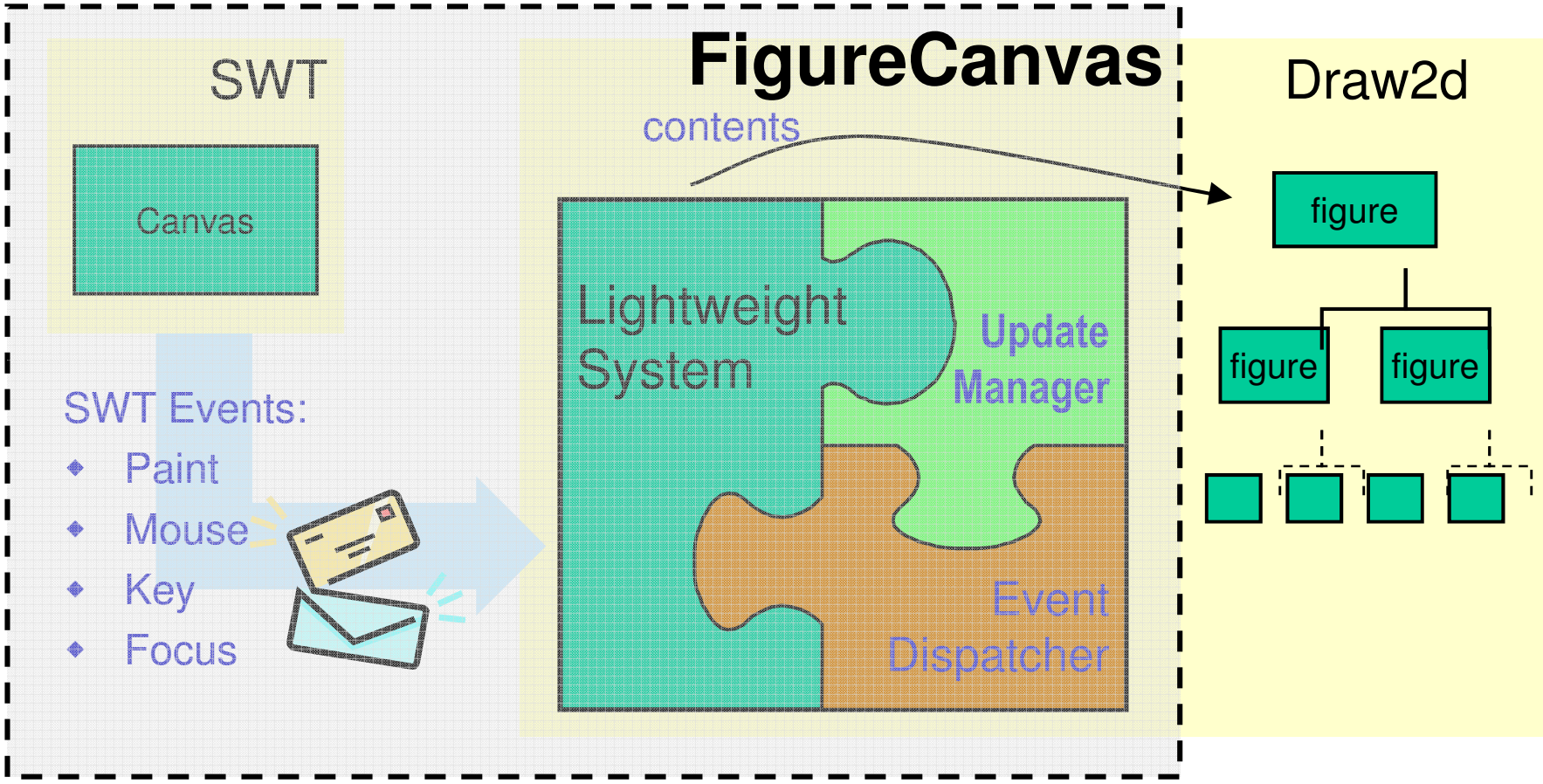
Meet the Players

- **Canvas** – the native SWT control that hosts a “drawing”
- **Figure** – the fundamental building block
  - It’s what you see
  - can contain other children figures
- **LayoutManager** – encapsulates the placement of children

- **Connection** – a special type of figure based on a polyline
- **ConnectionRouter** – determines a connection’s line, or route
- **ConnectionAnchor** – provides the endpoints to a connection

- **Border** – a decoration that can be set on a figure

Meet the Coach: Lightweight System



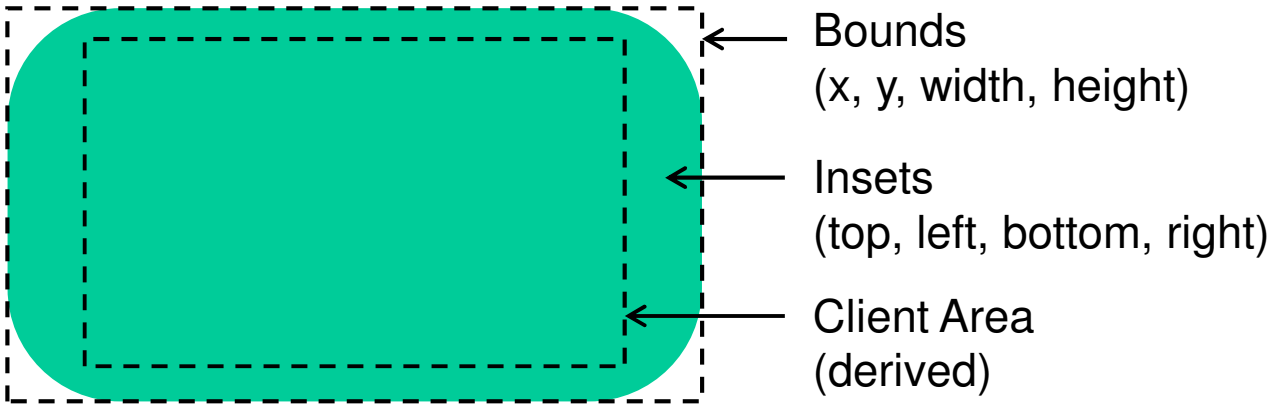
AOT LAB

Hello World


```
 1  Display d = new Display();
 2  Shell shell = new Shell(d);
 3  shell.setLayout(new FillLayout());
 4
 5  FigureCanvas canvas = new FigureCanvas(shell);
 6  canvas.setContents(new Label("Hello World"));
 7
 8  shell.setText(“draw2d”);
 9  shell.open();
10  while (!shell.isDisposed())
11      while (!d.readAndDispatch())
12          d.sleep();
```

Figures

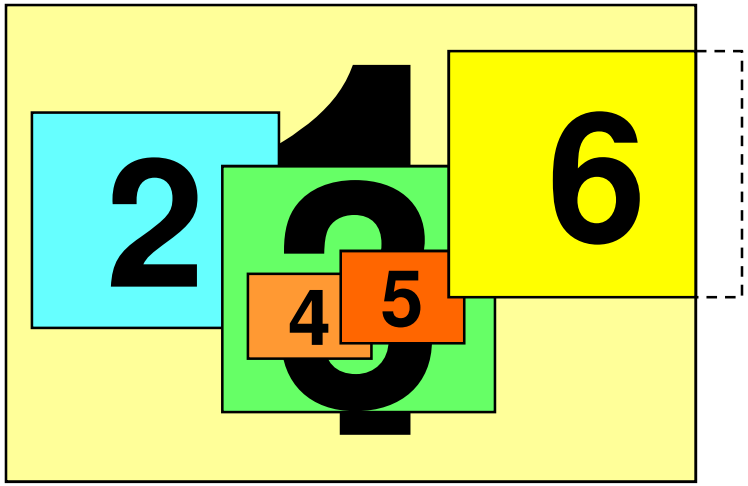
- Are simple lightweight containers
  - Nodes forming a Tree structure
  - Properties: Font, Colors, Borders, Tooltip, etc.
  - Inherit parent's font and color (all the way up to the Canvas)
- Have a Rectangular **Bounds**
- Are not necessarily rectangular

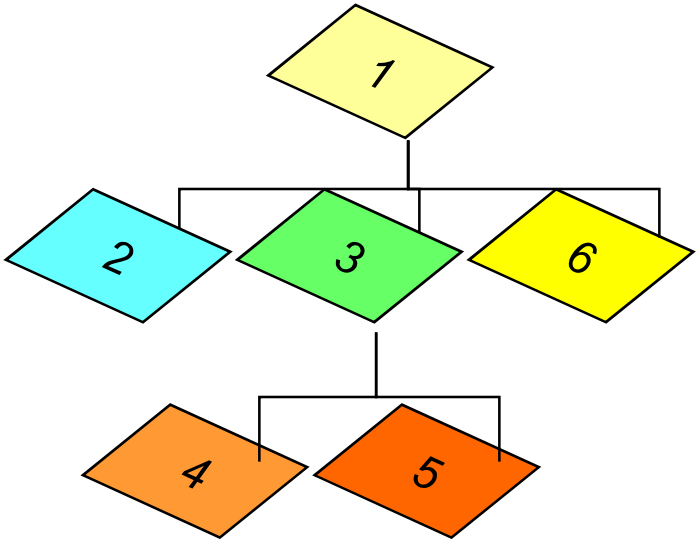


AOT LAB

Painting and Finding Figures

- Figures form a tree
- Painting is pre-order, “LTR”
- Parents clip children
- Last painted is “on top”
- Hit-testing is the opposite





Painting continued

- Override **paintFigure()**, not paint()
  - You don't have to worry about maintaining the state of the Graphics object; change settings as you please
  - Order of painting: parent figure, children figures, parent's border
- Request **repaint()** when any property that affects the appearance is changed
- Double-buffered painting (no flicker)
- Painting will be clipped to the figure's bounds and the parent figure's client area
- You can change child figures' z-order to determine which one gets painted on top
  - Last sibling is painted on top, in case of overlap

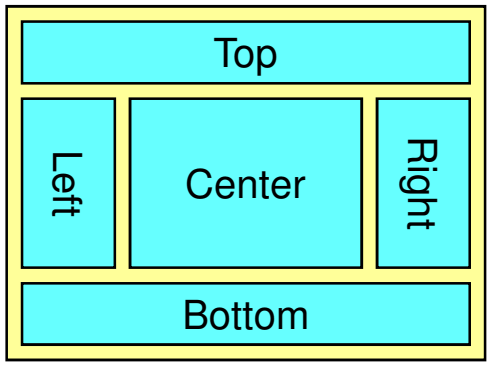
Layout Managers

- Responsibilities
  - Position a Figure's children: `IFigure#setBounds()`
  - Provide the preferred sizes based on children and layout
  - Only invalid figures will layout
- Hints passed during size calculations
  - -1 means not confined in that direction
  - AbstractHintLayout caches sizes based on hints
- Scrollbars' visibility determined by preferred size
- Constraint ≈ SWT's LayoutData
  - Example: XYLayout uses Rectangle constraints

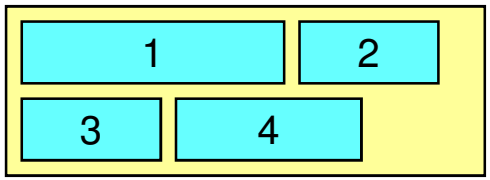
AOT LAB

Layout Managers in Draw2d

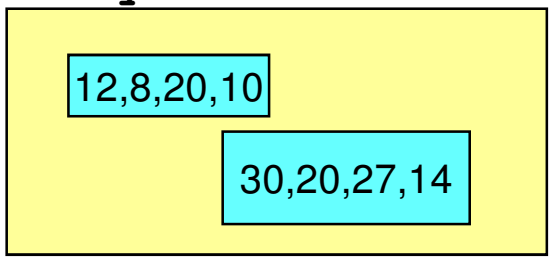
BorderLayout



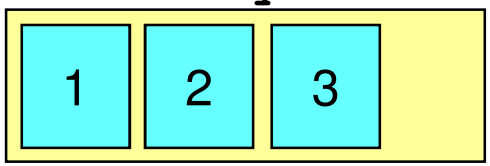
FlowLayout



XYLayout



ToolbarLayout

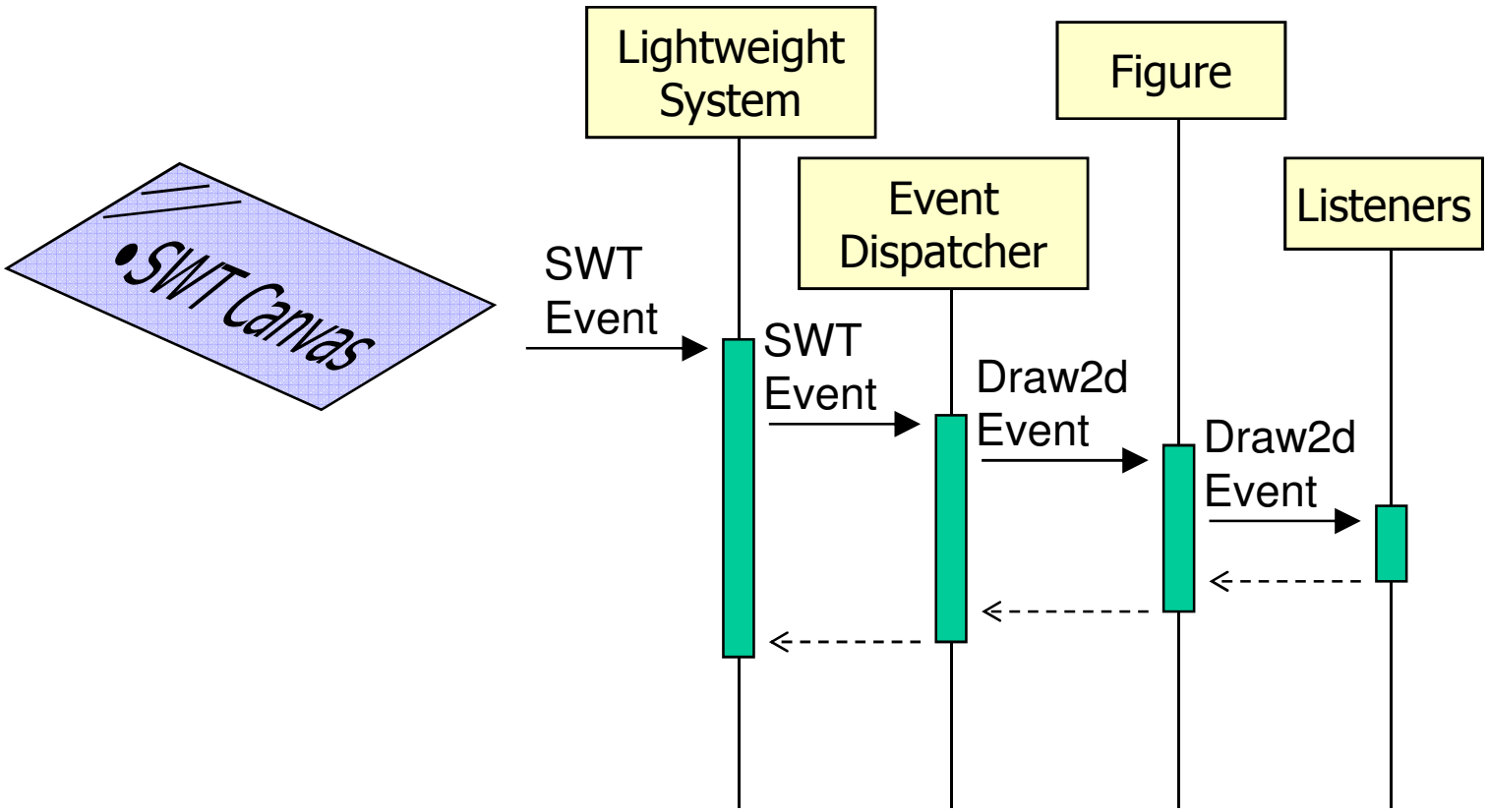


Coordinate Systems

- By default, bounds of parent and children use same coordinates
- Relative Coordinates
  - useLocalCoordinates()
  - Child positioned relative to parent's client area
- Scaling and Translation
  - Viewport
  - Zoom
- Converting coordinates
  - translateToParent() – Translate to parent's coordinate system
  - translateFromParent() – Translate from parent's coordinate system to child's
  - translateToAbsolute() & translateToRelative()

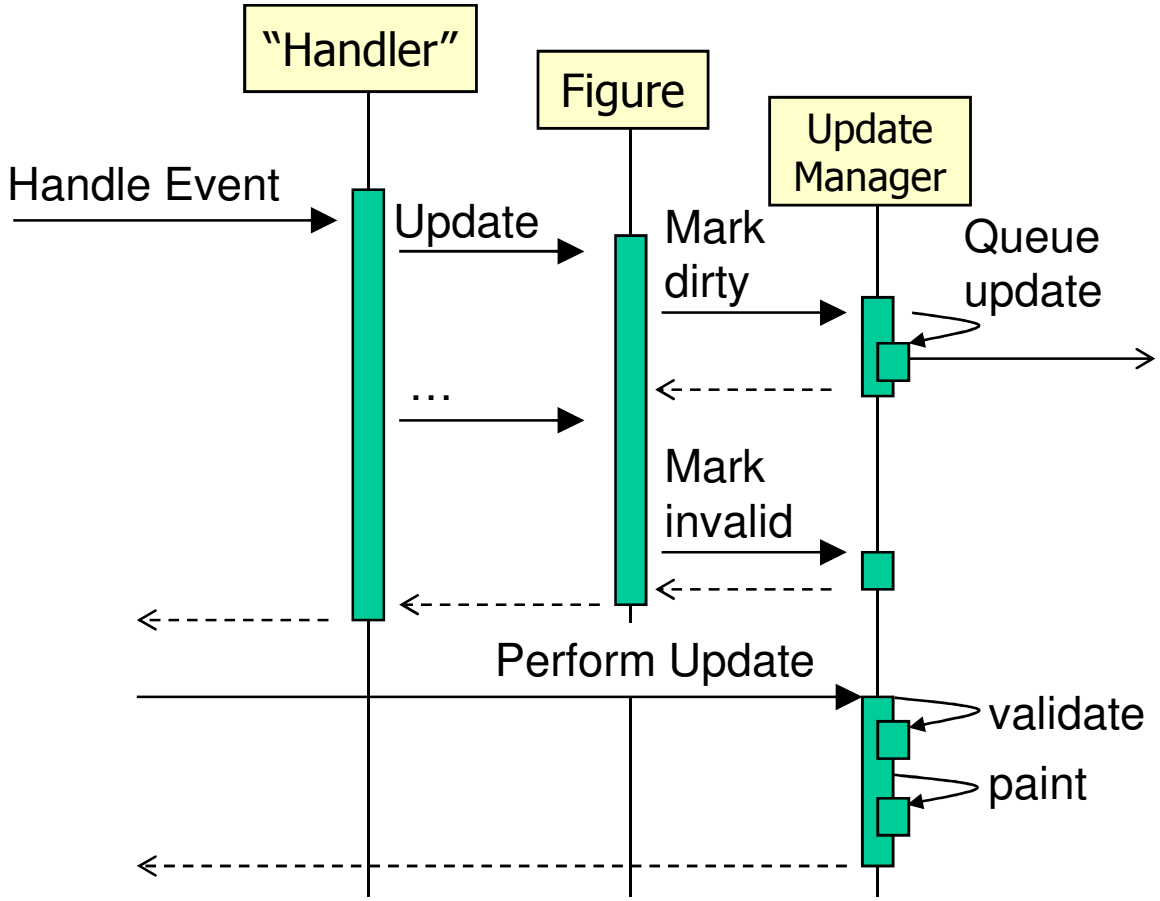
AOT LAB

Draw2d Interaction Sequence



AOT LAB

Making Changes to Figures



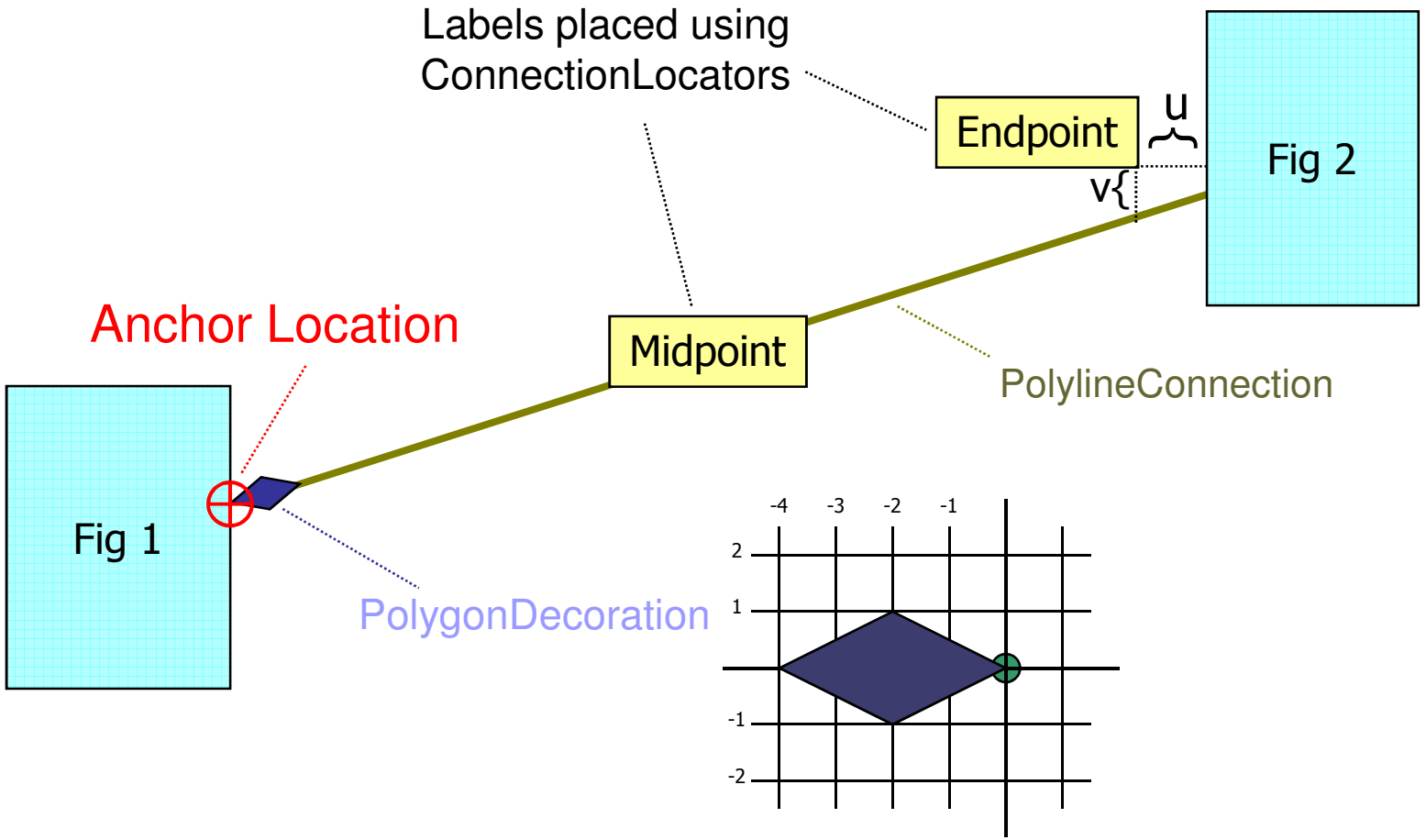
AOT LAB

Connections

- Just Figures with special behavior
- Render a **`PointList`** managed by a **`ConnectionRouter`**
- Have a source and target **`ConnectionAnchor`**
  - Define the start and end points of the connection
  - Return locations using absolute coordinates
  - #getLocation(referencePoint)
  - Context sensitive (ChopBoxAnchor or EllipseAnchor)
- Routers
  - Responsible for setting all points in a connection
  - May use routing constraints such as **`Bendpoint`**
  - Examples: Fan, Bendpoint, Manhattan, ShortestPath
- Connection can have children too
  - Arrowheads, Labels, etc.
- **`DelegatingLayout`**
- Locators place children of a connection
- Connections set their own bounds after validating

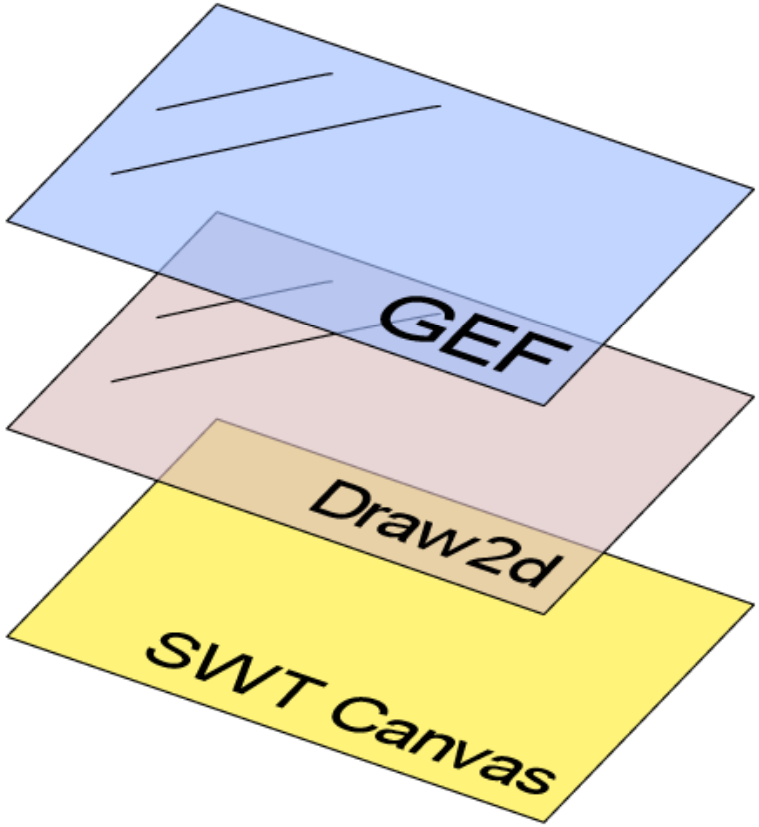
AOT LAB

Connections et al



AOT LAB

The GEF plug-in



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

AOT LAB

What Problems does GEF Solve?

1. Display a Model graphically
2. Allow the User to interact with that model
   - Process user input from Mouse & Keyboard
   - Interpret that input
   - Provide hooks for updating the model
   - Make it undo/redo-able
3. Provide Workbench integration
   - Actions and Menus
   - Toolbars, Contributions
   - Keybindings

◆ Where Can I use GEF?
   - EditorParts, Views
   - Anywhere in the Workbench

Meet the Players

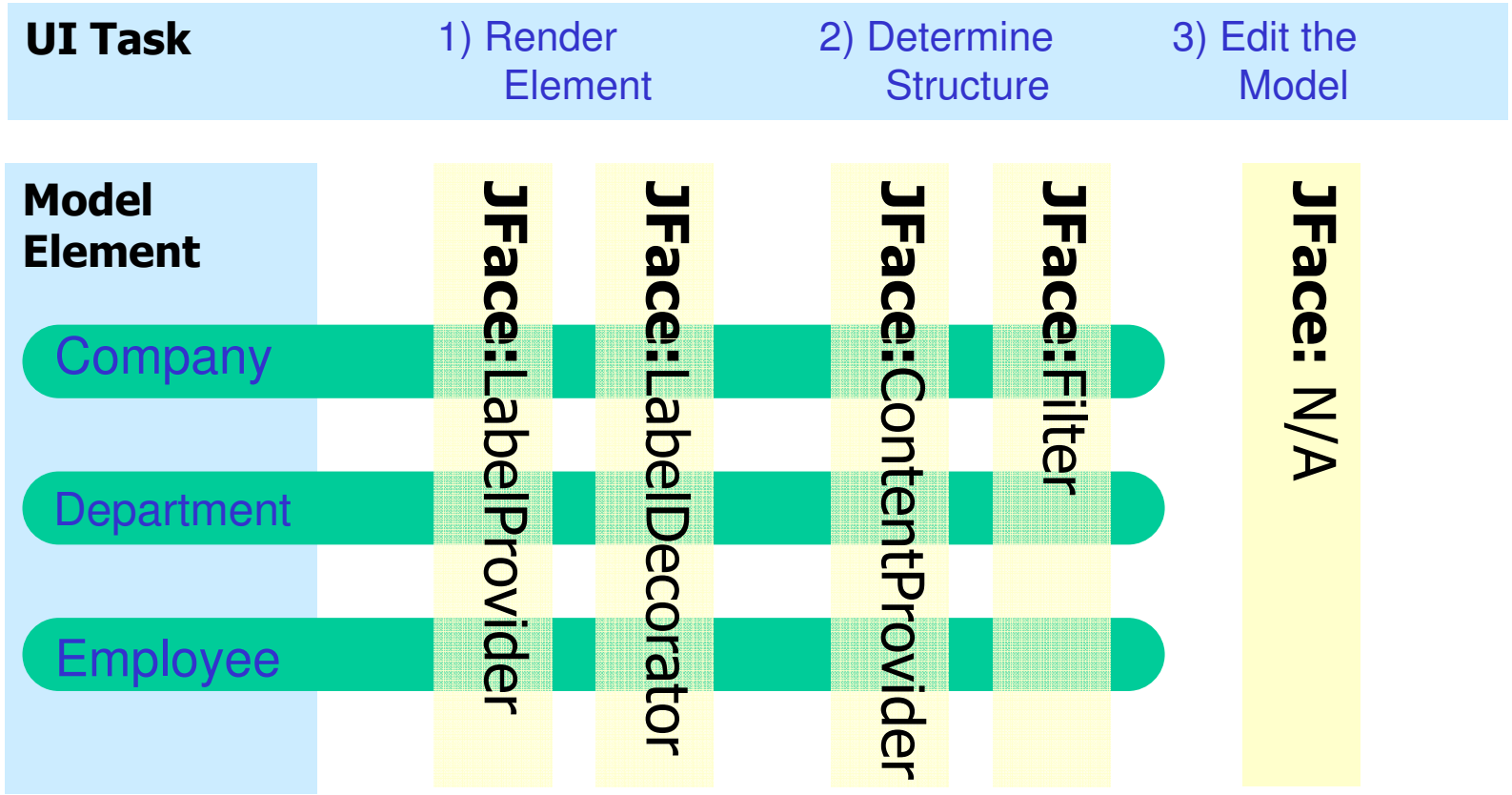
- **Viewer** – foundation for displaying and editing your model
  - Adapter on an SWT `Control`
  - Selection Provider
- **EditPart** – elements inside a viewer
- **Tool** – interprets user input; represents *mode*
- **Palette** – displays available tools
- **CommandStack** – stores Commands for undo/redo

- **EditDomain** – ties everything together

- **GraphicalEditor** – for building an EditorPart

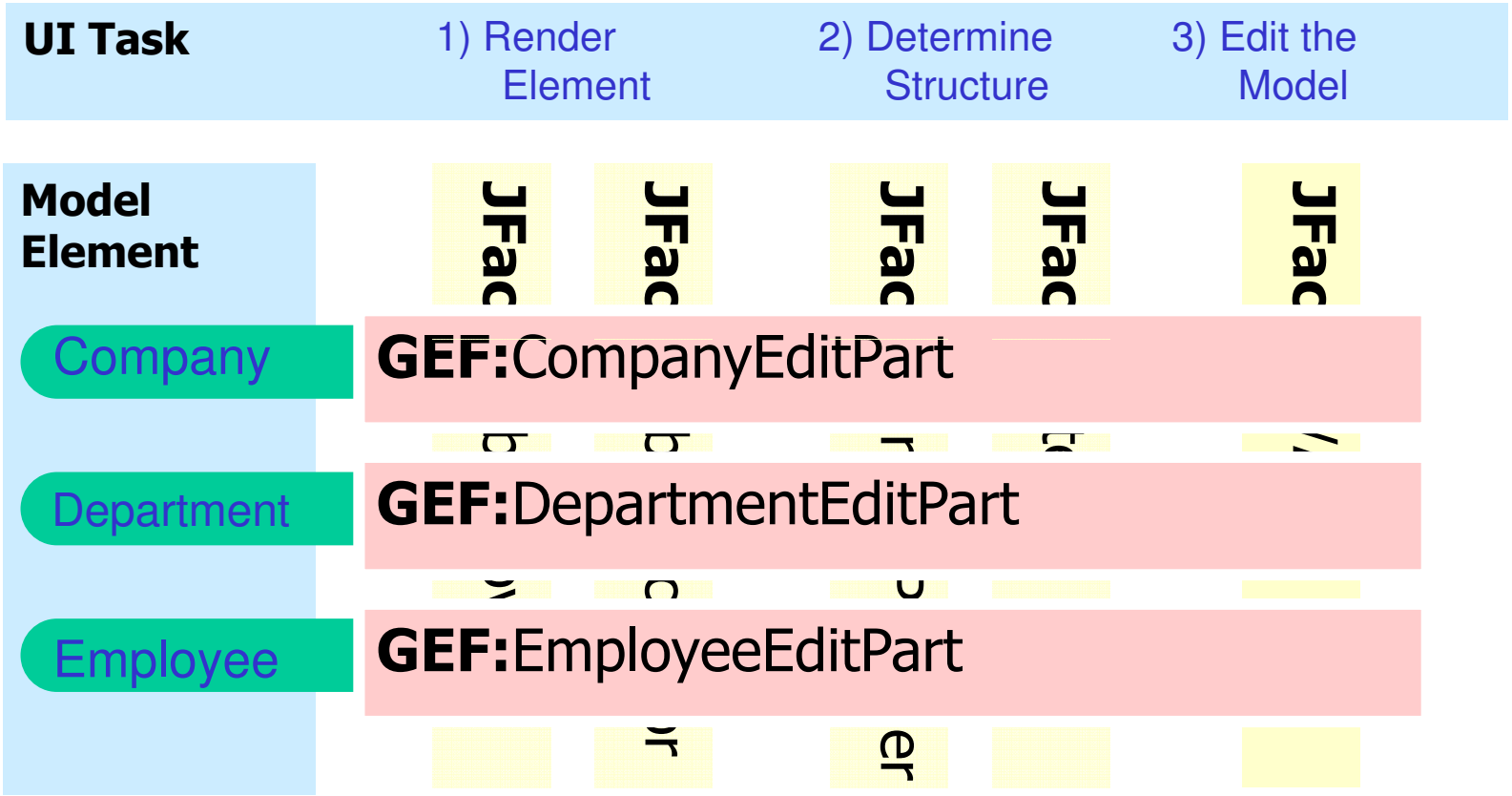
AOT LAB

Comparing GEF to JFace



AOT LAB

Comparing GEF to JFace



AOT LAB

Displaying Your Model Graphically

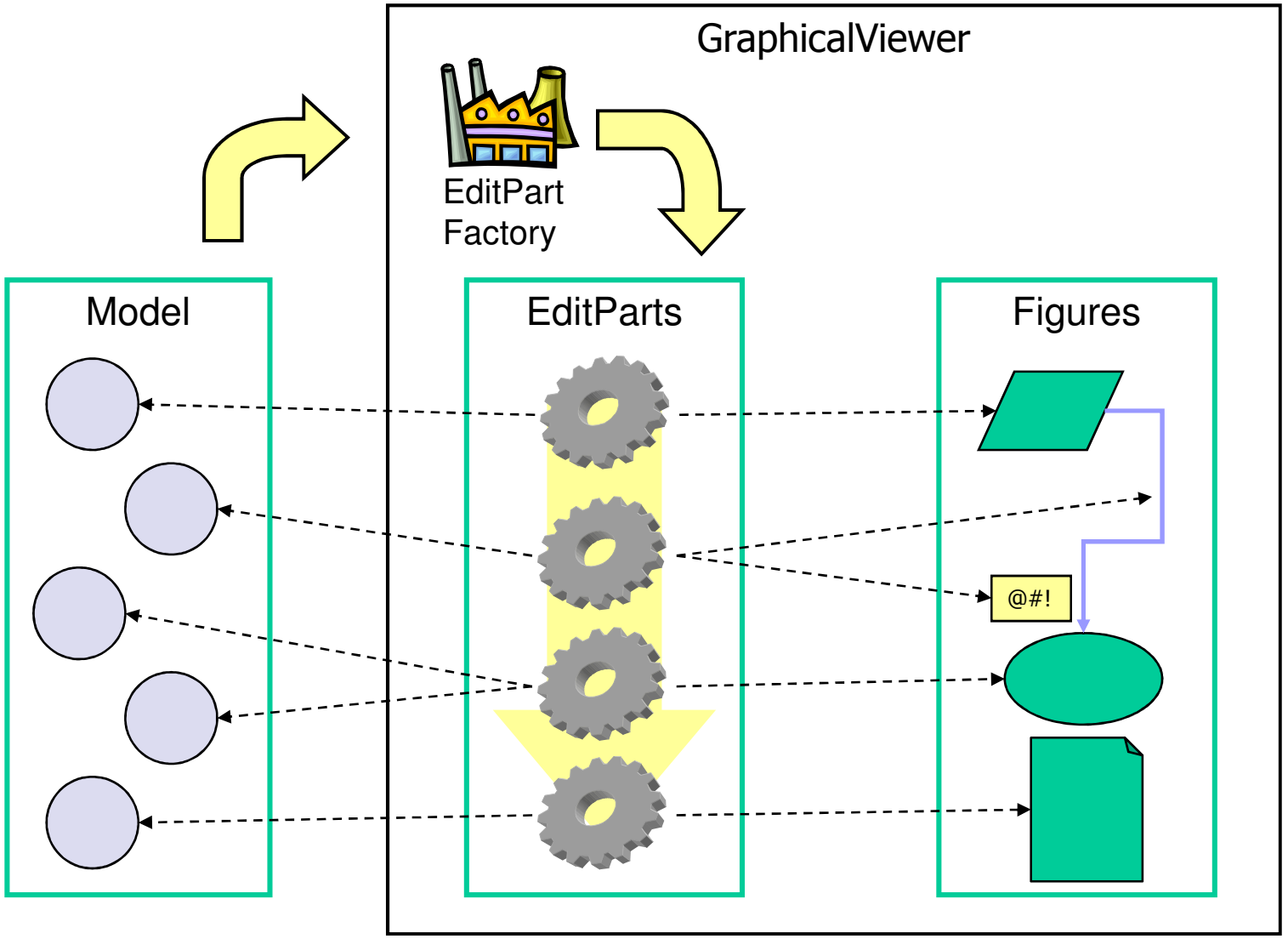
- Setup: Where will the UI appear?
  - EditorPart, ViewPart, or somewhere else?
  - **`GraphicalEditorWithFlyoutPalette`**
  - Use a **`GraphicalViewer`**
    - Manages the Draw2d foundation

1. Create the Viewer
2. Configure the Viewer and create its Canvas
3. Set the viewer's contents

- Next: Mapping the Model into Figures

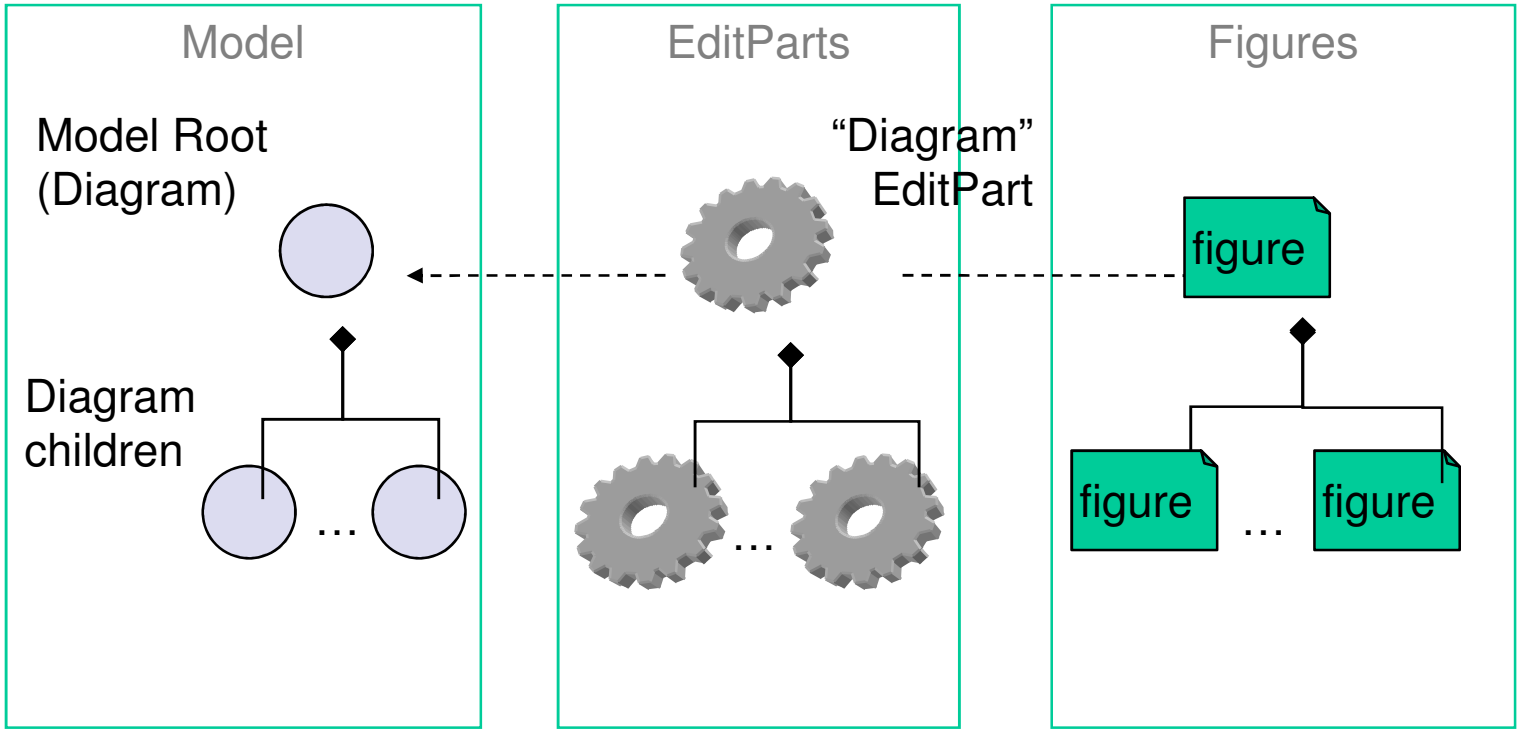
AOT LAB

Populating a Viewer



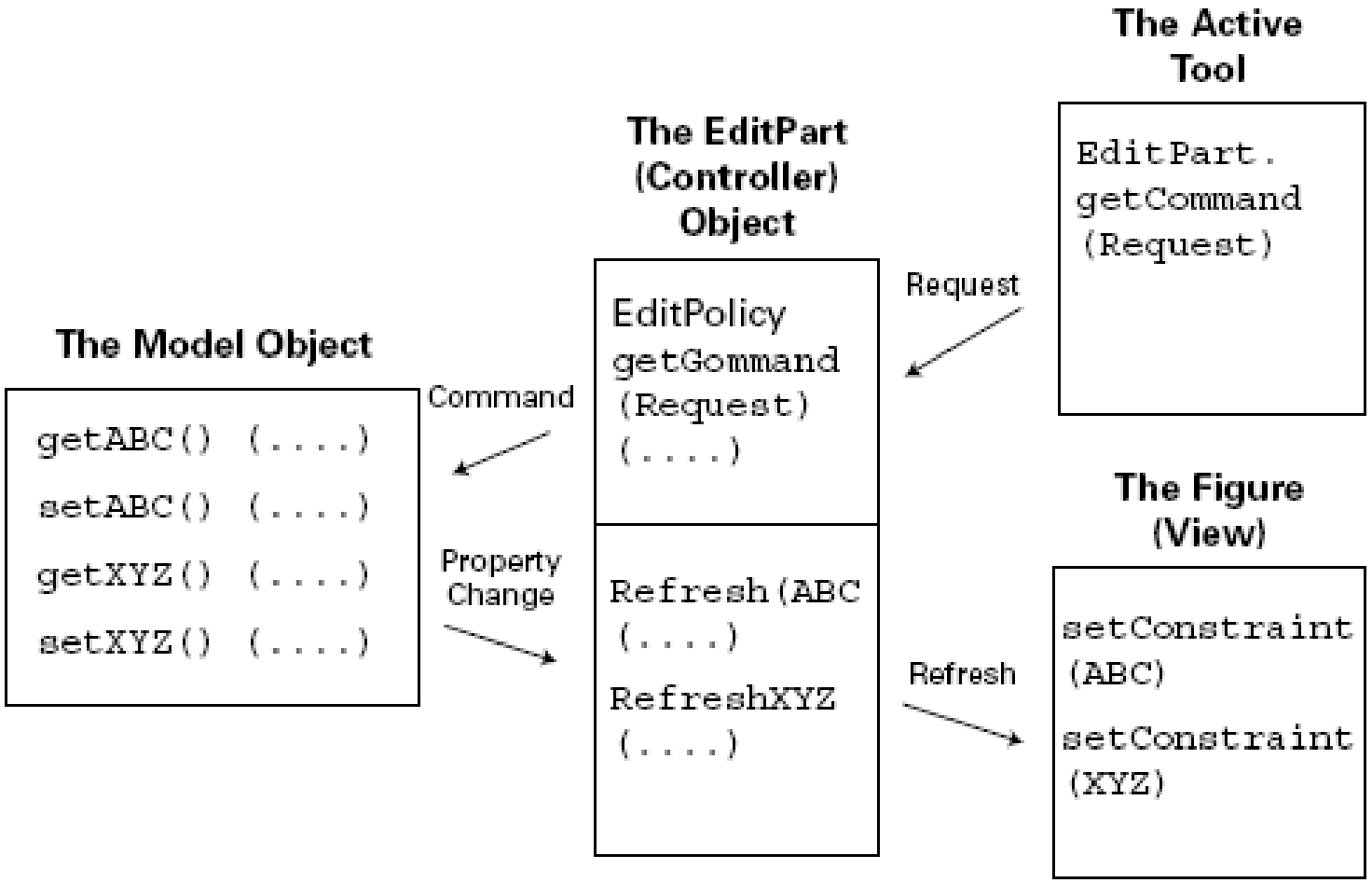
AOT LAB

Containment, and More Containment - MVC



AOT LAB

Model – View - Controller



Extending AbstractGraphicalEditPart

1. createFigure()
   - Just builds the figure
2. refreshVisuals()
   - Reflect the model's state in the view
3. getModelChildren()
   - Determines children structure

- Looking ahead:
  - Changing the model
  - Responding to model changes

AOT LAB

Connection EditParts

- Similarities to children parts:
  - Return a list of model objects: getModelSource/TargetConnections()
  - Factory can create the Connection Parts
- Differences:
  - An anchor must be set for the source and target
  - Target may come before Source
  - Figure gets added to the Connection Layer
- NodeEditPart Interface
- Connection must have a direction

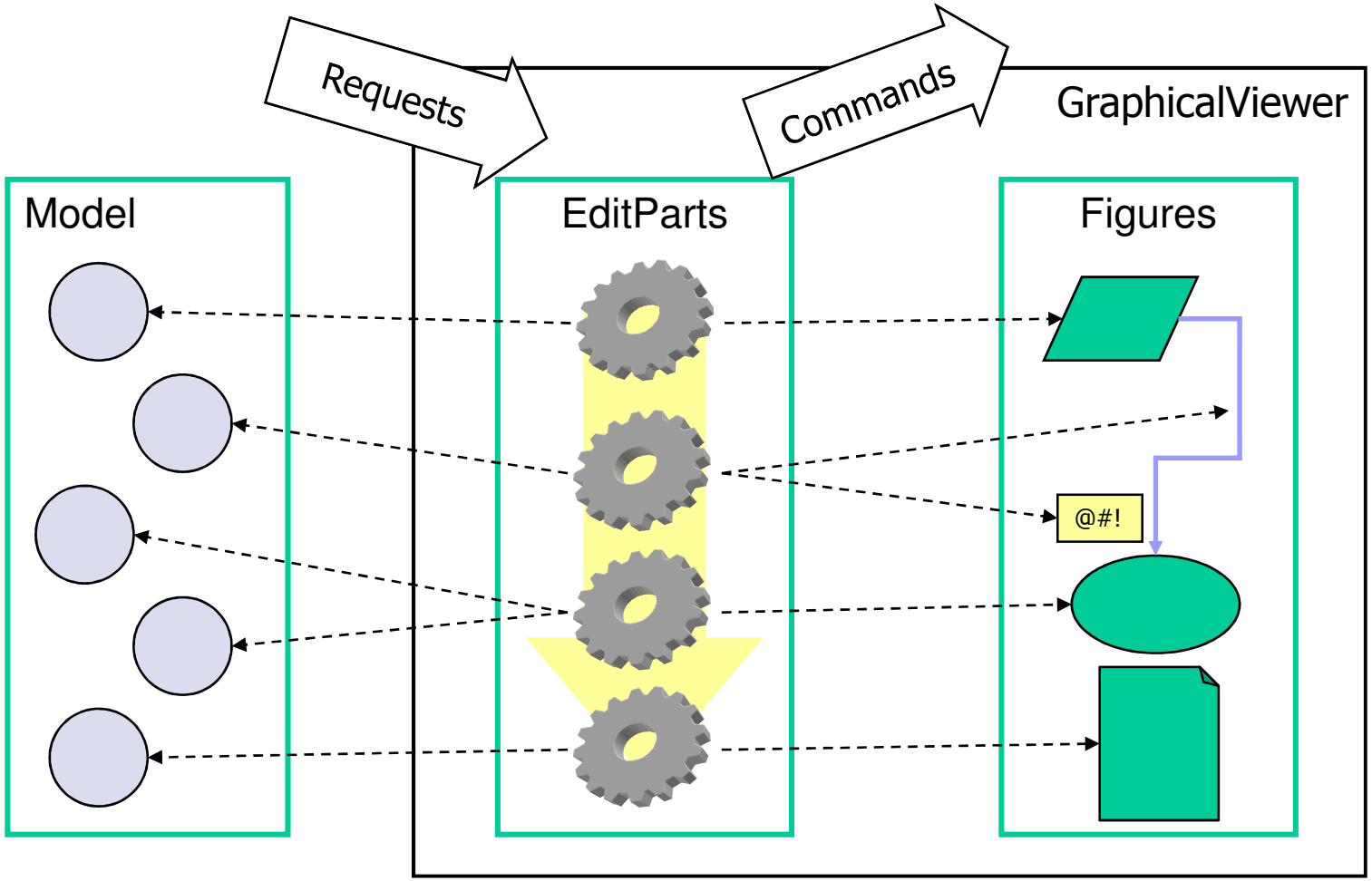
AOT LAB

More about EditParts

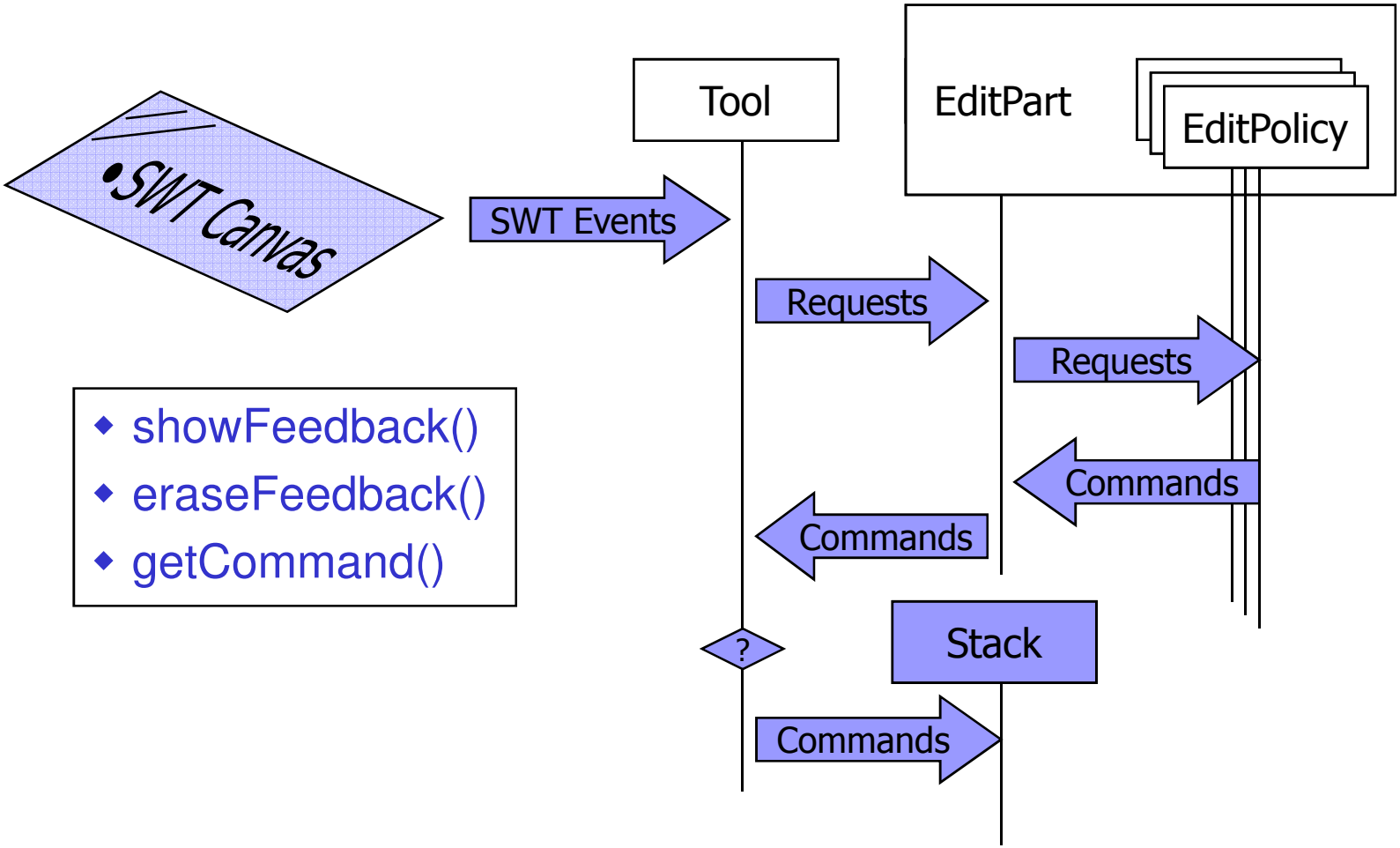
- Deciding on Model → EditPart mapping
  - Unit of “interaction”
  - Selection is comprised of EditParts
  - Can it be deleted?
  - Graphical vs. Tree Viewers
- The Root EditPart

AOT LAB

Editing: Putting the "E" in GEF



Editing and Tools



- showFeedback()
- eraseFeedback()
- getCommand()

EditParts and EditPolicies

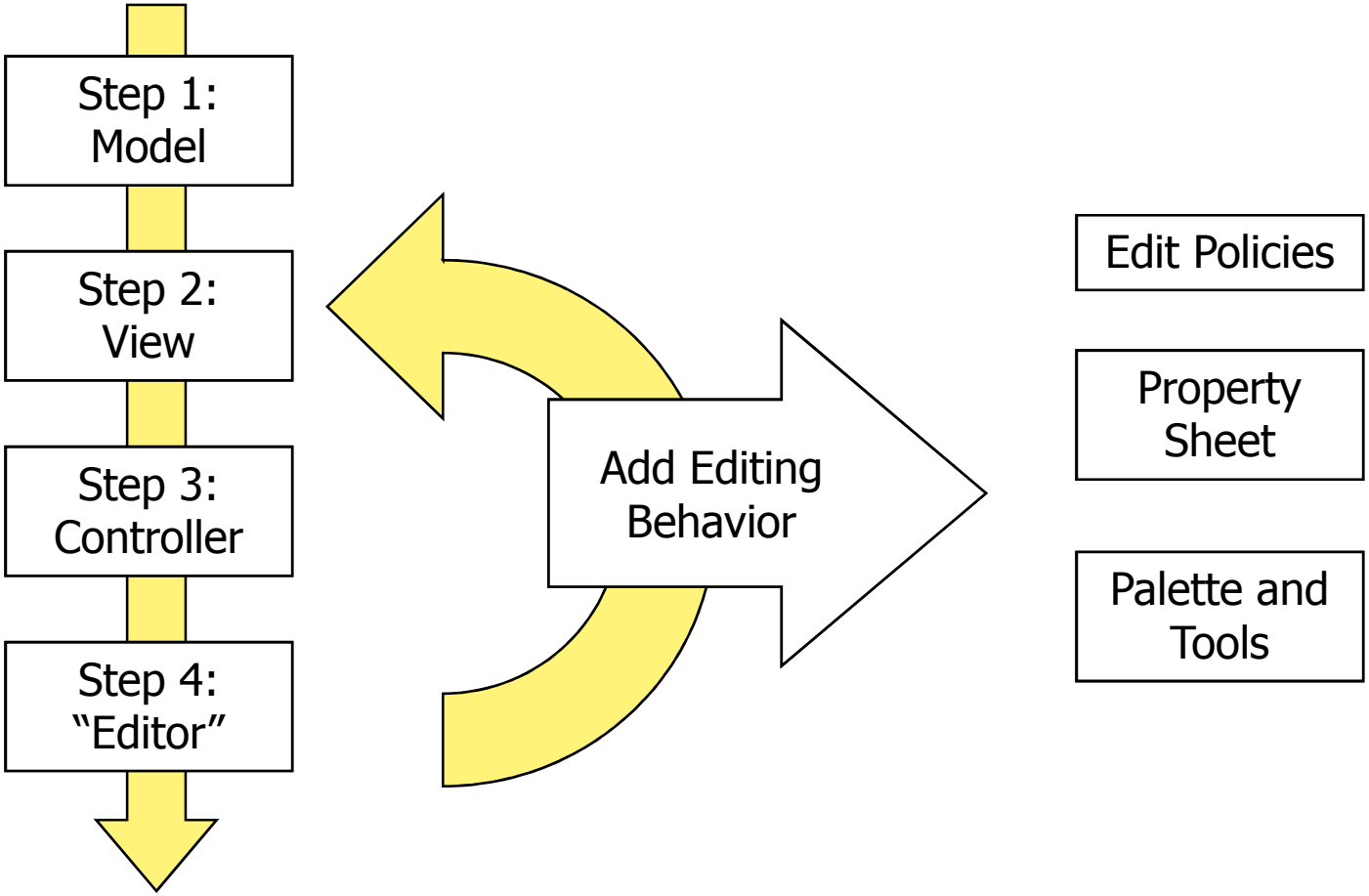
- EditPolicies are installed and maintained by their **host** part
- EditPart responsibility is keep view up-to-date
- EditPolicies offload the editing tasks
  - Avoid limitations of single inheritance
  - Separate unrelated editing tasks
  - Allows editing behavior to be dynamic
  - Keyed using Strings (“Roles”)
  - May contribute to feedback, commands, targeting, etc.
  - Tip: UnexecutableCommand vs. null
  - Examples: BendpointEditPolicy, SelectionEditPolicy, etc.
- Pattern used: “Pool of Responsibility”

Responding to Model changes

- Extend `activate()` to hook listeners
- Extend `deactivate()` to unhook listeners
- Depending on the type of change
  - Add, remove, or reorder Children
  - Update Connections
  - Update the view

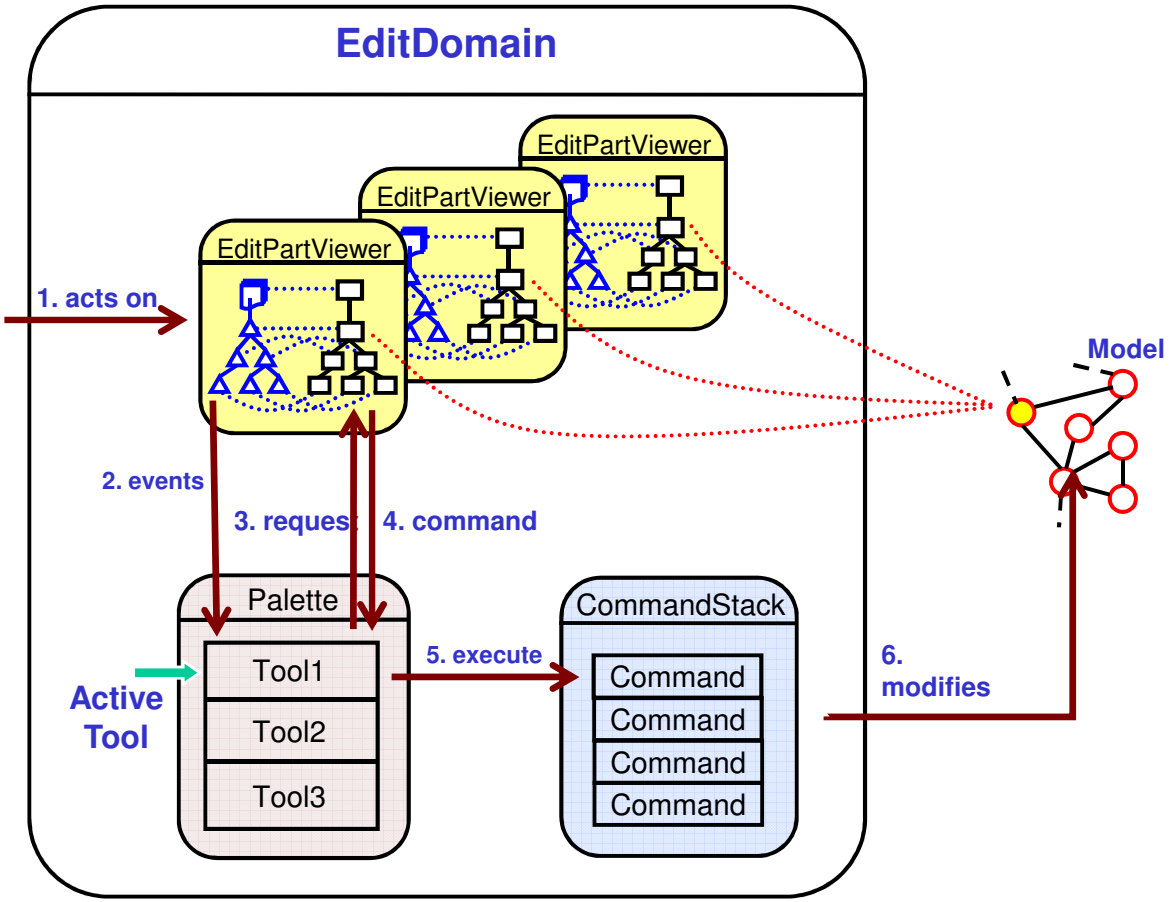
AOT LAB

Building a GEF Application



AOT LAB

The Big Picture

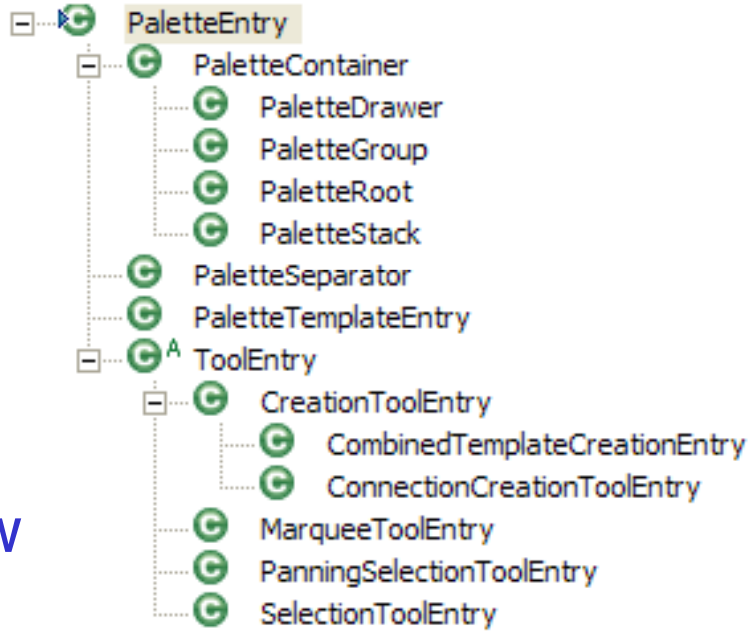


GEF – Conventions and Patterns

- Tools to interpret mouse and keyboard
- Requests to encapsulate interactions
  - Tip: performRequest() for REQ_DIRECT_EDITING and REQ_OPEN
- Absolute coordinates
- Edit Policies for separation of concerns
- Command pattern for undo/redo
- Use of IAdaptable

Palette

- PaletteViewer: Just another GraphicalViewer
  - Tip: #saveState(IMemento)
- Palette Model
  - Templates vs. Tools
  - Drawers, Groups and Stacks
  - Permissions
- PaletteViewerProvider
  - Add drag source listener for DND
- Fly-out Palette and PaletteView
  - GraphicalEditorWithFlyoutPalette
  - PalettePage
  - FlyoutPreferences



Properties View

- Implement `IPropertySource` on
  - The EditPart
  - The Model
    - Or make the model IAdaptable
  - A Custom adapter for combining multiple sources
- GEF provides undo/redo support via commands
- UndoablePropertySheetEntry
  - Auto-Wraps IPropertySource changes in a Command

Outline View

- Use the provided TreeViewer class
- Extend AbstractTreeEditPart
- Use SelectionSynchronizer with multiple viewers
- Editor actions can be reused in the outline
- Use **`ScrollableThumbnail`** as an Overview

Actions

- Editor's ActionRegistry
  - Actions updated as needed
    - Editor does the listening, not the actions
  - It's just a Map
- ActionBarContributor
  - **One** for all editor instances
  - Declared in plugin.xml
  - Use RetargetActions

AOT LAB

Accessibility

- Eclipse is accessible
- GEF is accessible
- IAdaptable#getAdapter(Class)
  - AccessibleEditPart
    - Magnifier and Screen reader API
- Focus indication (Selection Edit Policies)
- Default keyboard handlers
- Accessible Tools
  - AccessibleAnchorProvider
  - AccessibleHandleProvider

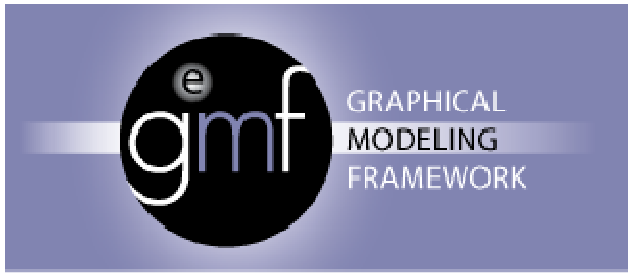
Auto-Scrolling

- During drag operations, including native DND
- Search from target part upwards
- **`AutoExposeHelper`**
  - #detect(Point)
  - #step(Point)
- Not just for scrolling
  - Expanding
  - Page-flipping
- Related: **`ExposeHelper`**
  - Programmatically “reveal” an EditPart

Zooming

- ZoomManager
  - Use a Scalable RootEditPart
  - Tip: available as a property on the viewer
- Action Contributions
  - Zoom-In & Zoom-Out Actions
  - ZoomComboContributionItem
- Ctrl + MouseWheel (in 3.1)
  - MouseWheelZoomHandler

The Future: GMF



http://www.eclipse.org/gmf/

- The Eclipse Graphical Modeling Framework (GMF) provides a generative component and runtime infrastructure for developing graphical editors based on EMF and GEF
- The project aims to provide these components, in addition to exemplary tools for select domain models which illustrate its capabilities

AOT LAB

Agent and **O**bject **T**echnology **Lab**
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma

Graphical Editing Framework (GEF)

Alessandro Negri
negri@ce.unipr.it
http://www.ce.unipr.it/people/negri/