Tutorial #23
GEF In Depth

Randy Hudson
Pratik Shah

IBM Rational Software
Research Triangle Park, NC

Agenda

- Start things off
- What is GEF?
- GEF Demo
- Draw2d
  - Overview
  - Example
- GEF
  - Overview
- Break
- Hands-on Activity: Shapes Example

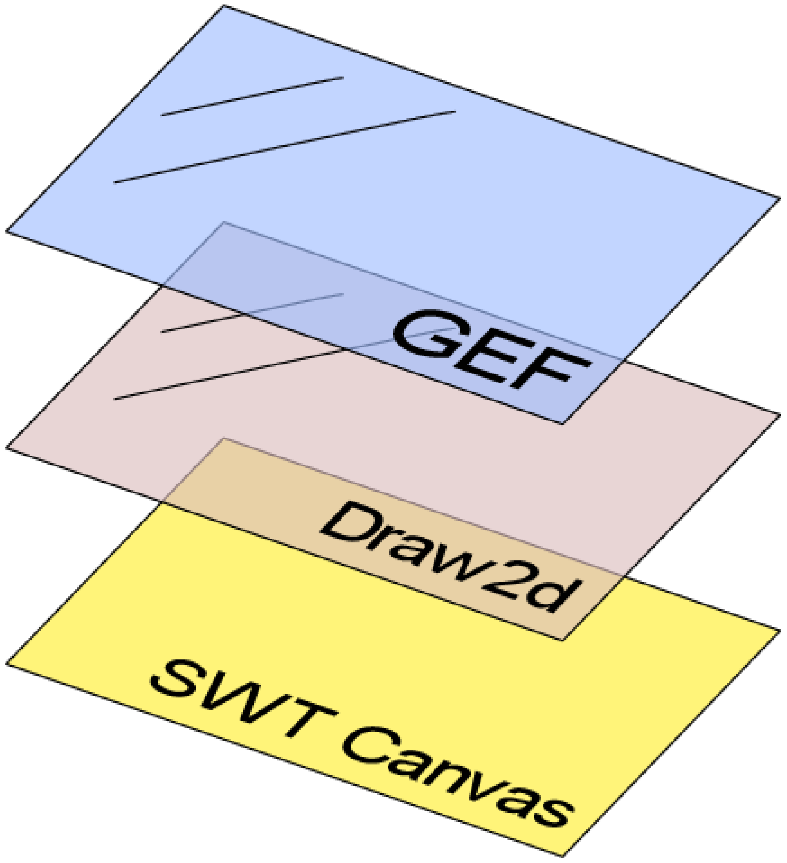
Starting things off…

- This is an interactive session: feel free to ask questions
- Goals for this tutorial
  - Introduce GEF and Draw2d
  - Highlight main features
  - Show how to find answers easily
  - Provide hands-on experience
  - Steer you around common mistakes
- Tell us about your GEF Plans
  - Level of experience with GEF?

Before we begin…

- Newer version of these slides are available
  - GEF Home Page -> Click on Documentation
  - http://www.eclipse.org/gef/reference/articles.html
- You're going to need:
  - Eclipse 3.1M5a
  - GEF SDK 3.1M5
  - Files for tutorial: unzip into your workspace before launching Eclipse, and then from within Eclipse, Import -> Existing Project Into Workspace
    - http://www.eclipse.org/gef/conference/activity.zip
- We have CDs

What is GEF?

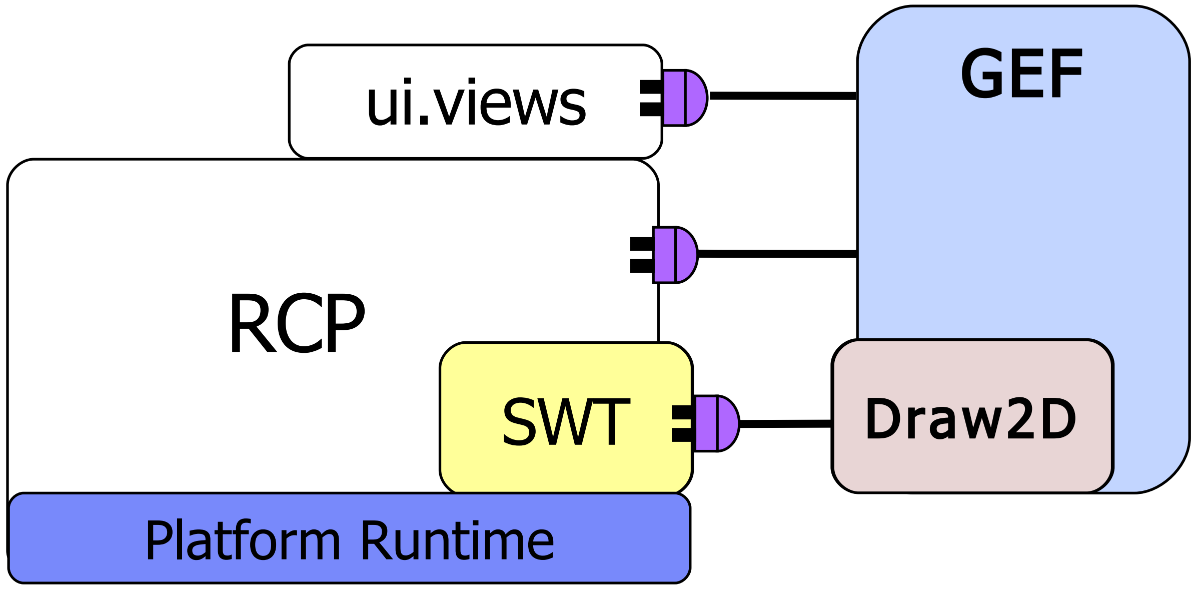


- Interaction Layer
- Model-to-View mapping
- Workbench Integration

- Rendering
- Layout
- Scaling

- Native (SWT) Layer

GEF Components & Dependencies



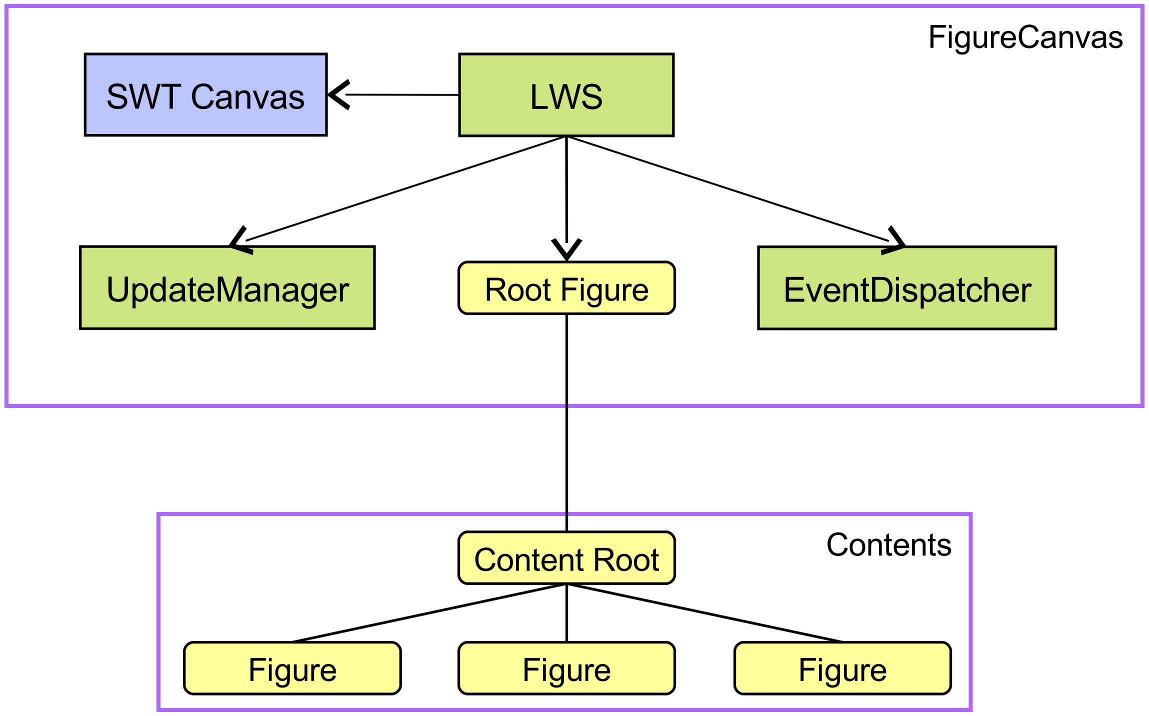
GEF Demo

- Features
  - Move, Resize, Create, Bend, Connect
  - Delete, Undo, Redo, Direct-editing
  - Overview & Zooming
  - Palette Viewer and workbench view
  - Palette Customization
  - Rulers & Guides
  - Snap-To-Grid or Geometry
  - Shortest Path Connection Router
  - Keyboard Accessibility
  - Directed Graph Layout

Draw2d Overview

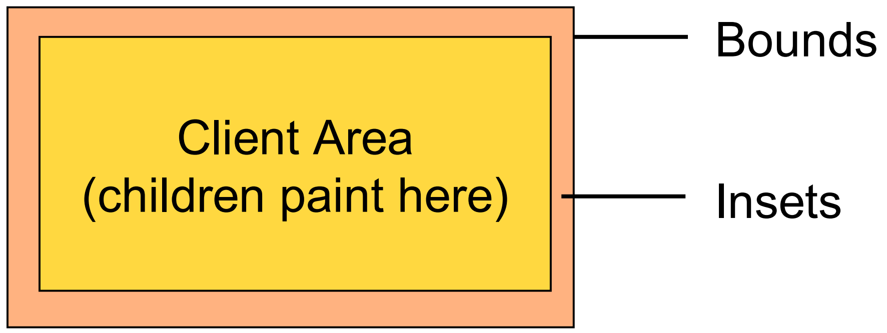
- Draw2d : SWT :: Swing : AWT
- Lightweight Toolkit built on top of SWT
- Concepts borrowed from Swing and others
- Built to support GEF function

Draw2D Lightweight System

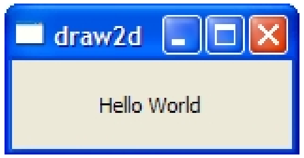


Figures

- Simple lightweight container: building block
  - Fonts, Colors, Opacity, Borders, Layouts, Visibility, Bounds, Tool-tip
  - Inherit parent's font and color (RootFigure gets it from the Canvas)
- Can be non-rectangular
- Can be nested



Hello World



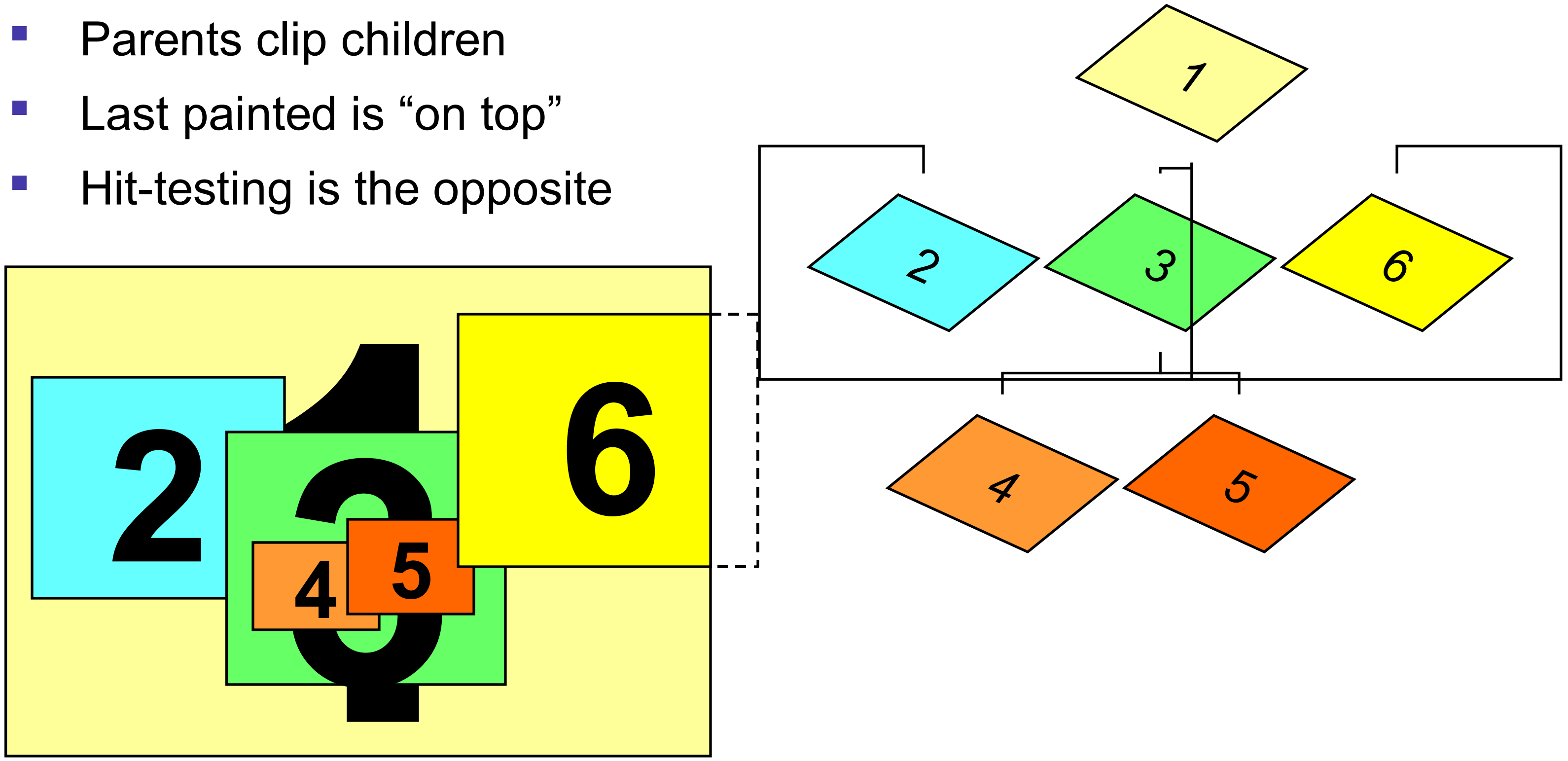
```
Display d = new Display();
Shell shell = new Shell(d);
shell.setLayout(new FillLayout());

FigureCanvas canvas = new
FigureCanvas(shell);
canvas.setContents(new Label("Hello World"));

shell.setText(“draw2d”);
shell.open();
while (!shell.isDisposed())
    while (!d.readAndDispatch())
        d.sleep();
```

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



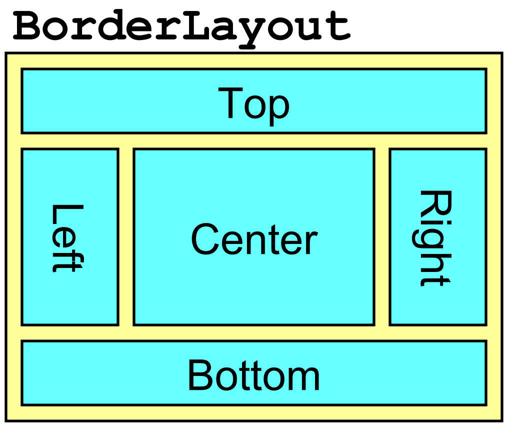
Painting continued

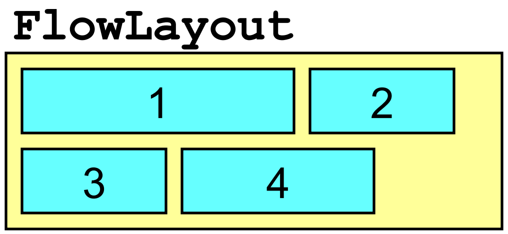
- Override paintFigure(), not paint()
  - You don't have to worry about maintaining the state of the Graphics object; change settings as you please
  - Order of painting: parent figure, children figures, parent's border
- Request repaint() when any property that affects the appearance is changed
- Double-buffered painting (no flicker)

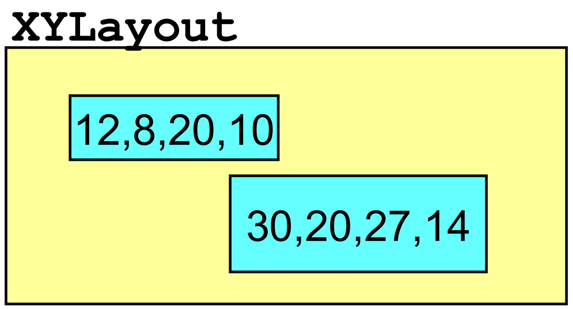
LayoutManagers

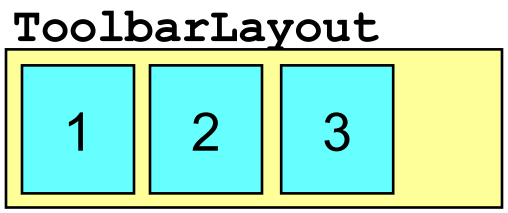
- Responsibilities
  - Position a Figure’s children: `IFigure#setBounds()`
  - Provide the preferred sizes based on children and layout
  - Only invalid figures will layout
- Hints passed during size calculations
  - -1 means not confined in that direction
  - AbstractHintLayout caches sizes based on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

Layout Managers in Draw2d





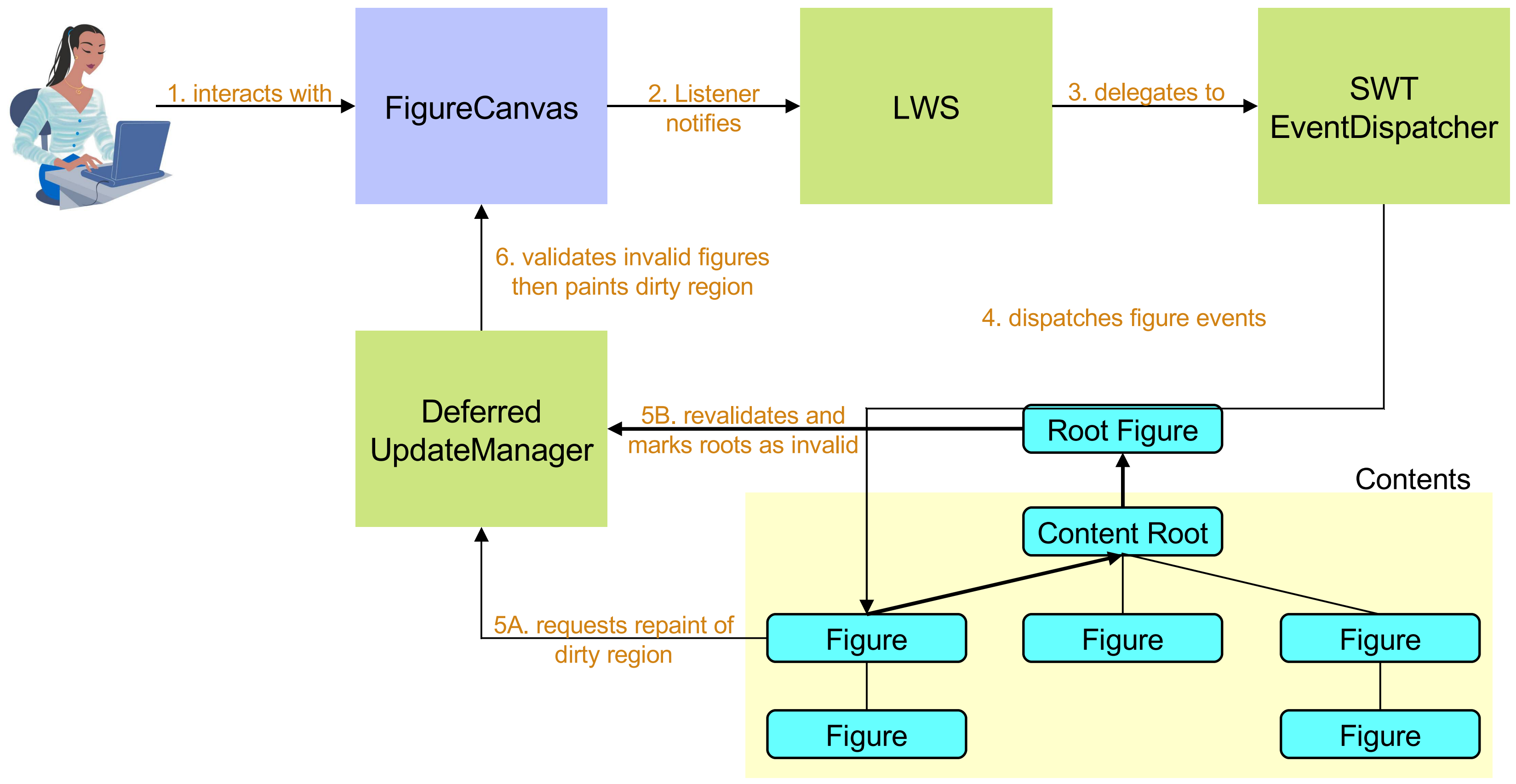




Using Layout Mangers

- Layouts place children within the parent’s “client area”
- Borders can be used to modify the client area
- The included layouts can get you far
- Nest Layouts/Figures to get results
- Layout Managers must **not** be shared
- Borders may be shared (singletons)

User Interaction Scenario in Draw2d



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()

Displaying Text

- **`org.eclipse.draw2d.Label`**
  - Icon + Text
  - Truncation, Alignment
  - Tip: Use ImageFigure if you only have Images
- Improved “Rich” Text features in 3.1
  - **`org.eclipse.draw2d.text`** package
  - Content which wraps itself like a paragraph
  - Margins, borders and padding
  - Mixing non-textual components with text
  - API for finding offset based info
  - BiDi – Arabic, Hebrew, etc.

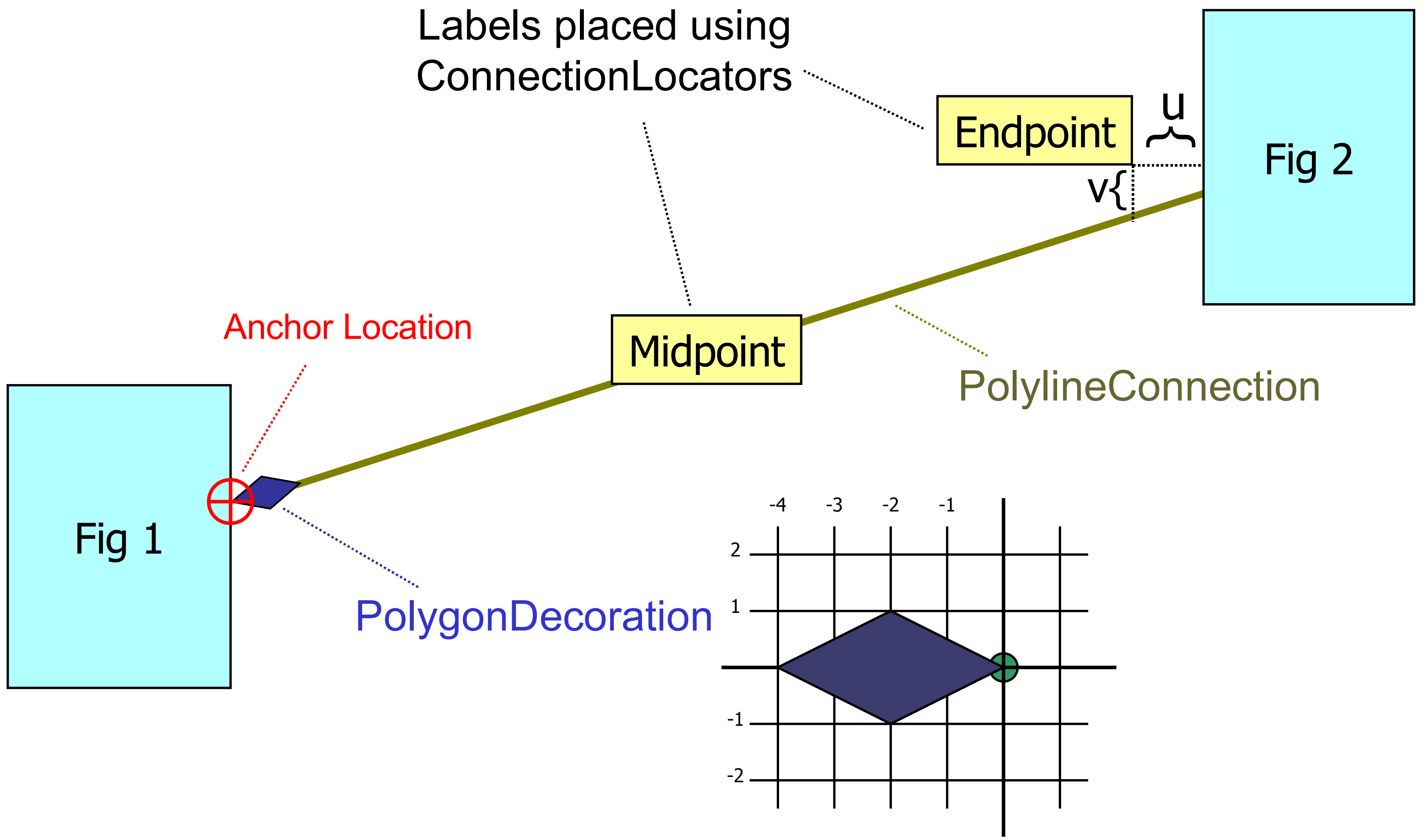
Connection extends **IFigure**

- 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

Connections continued

- Connection can have children too
  - Arrowheads, Labels, etc.
- **`DelegatingLayout`**
  - Locator constraints position each child
- Connections set their own bounds after validating
  - Do not call setBounds() on connections

Connections et al

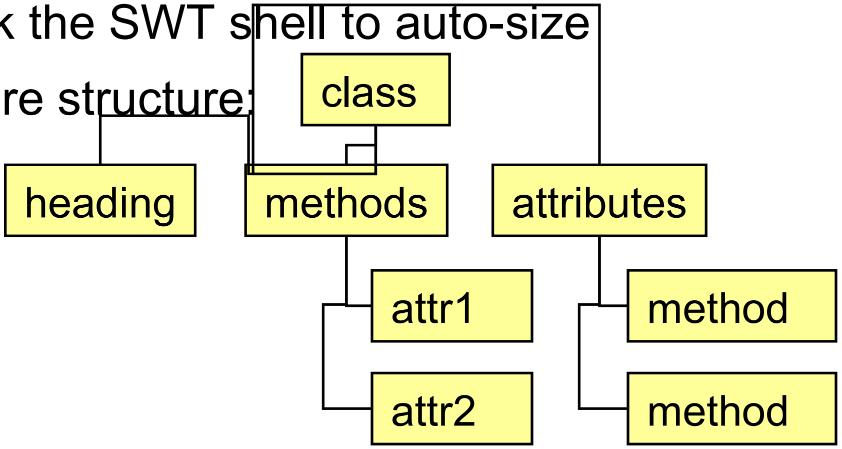


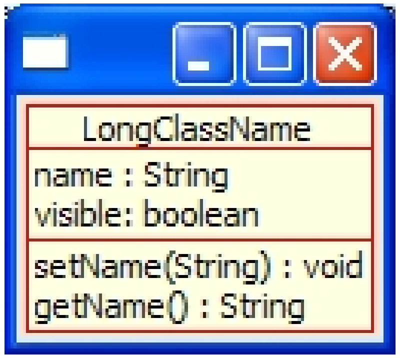
Avoiding Common Mistakes

- Do not modify Objects returned by reference (e.g., bounds)
- Use revalidate() and repaint() appropriately
- `Polyline` is not a regular Shape
- Know your coordinate systems
- Use LineBorder as a debugging tool
- Can’t ask for preferred sizes without Fonts
- news://news.eclipse.org

Draw2d Exercise

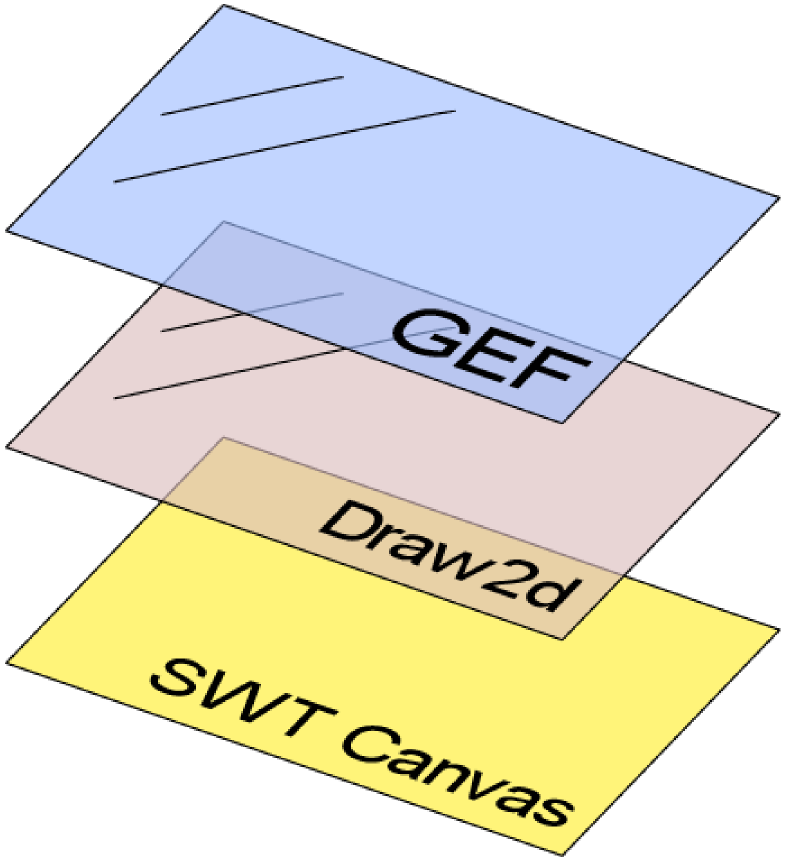
- Build the following representation of a UML Class
- Use **only** the provided classes:
  - **`ToolbarLayout`**
  - **`LineBorder`**
  - **`MarginBorder`**
  - **`Label`** (not SWT's)
  - **`ColorConstants`**
- Pack the SWT shell to auto-size
- Figure structure:





The Solution

The GEF plug-in



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

What Problems does GEF Solve?

- Display a Model graphically
- 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
- Provide useful Workbench function
  - Actions and Menus
  - Toolbars, Contributions
  - Keybindings
- Where Can I use GEF?
  - EditorParts, Views
  - Anywhere in the Workbench

Displaying a Model Graphically

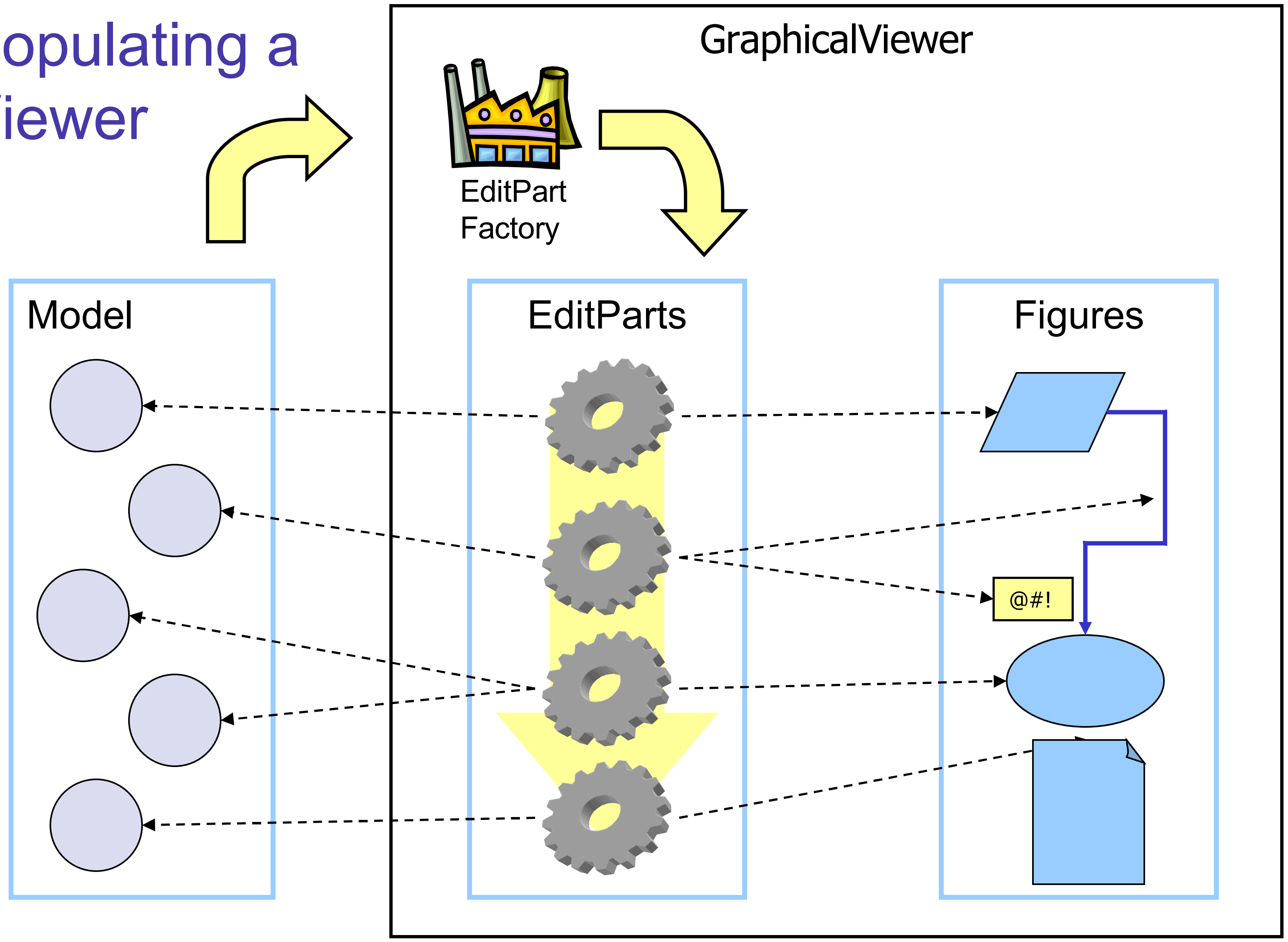
- Figures are not your model
- Lots of models to choose from (EMF, DOM)
- GEF works with any model
- Model requirements:
  - Notification mechanism
  - Persist and restore state
  - Commands which operate on the model

Displaying a Model Graphically

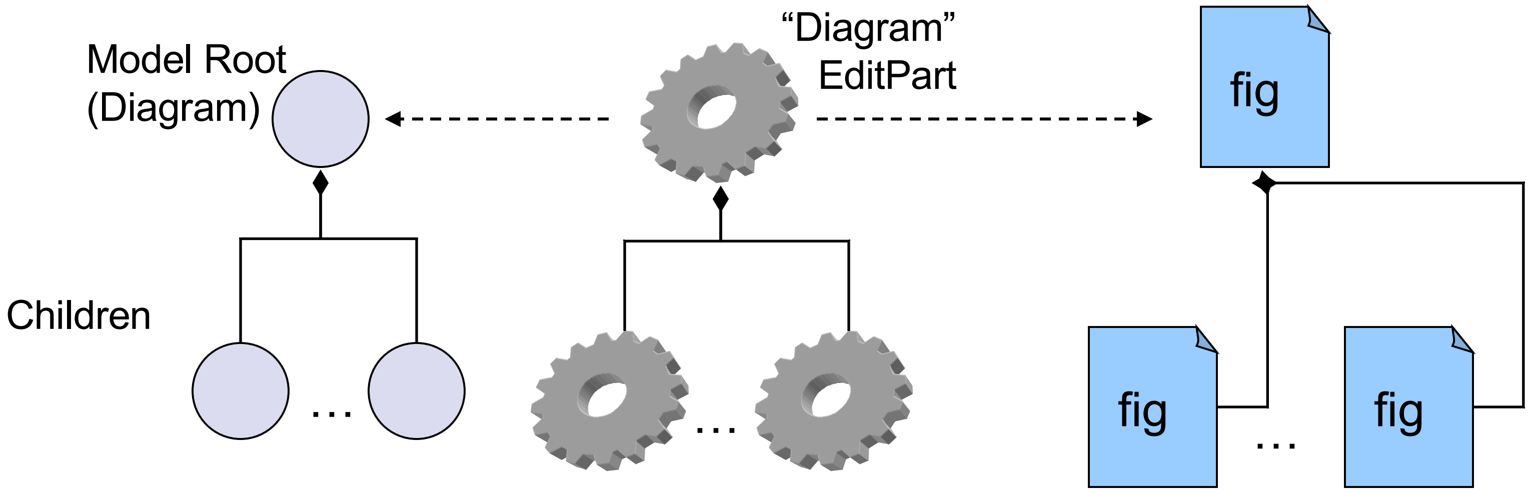
- **`GraphicalViewer`**
  - Viewer == Adapter for a **`Control`**
  - Manages Draw2d's Figures
- Construct a Viewer and Canvas
- **`TreeViewer`** uses Tree
- Set the model input for the Viewer

- Next: Map the Model into Figures

Populating a Viewer



Graphical EditParts and Containment



Extending AbstractGraphicalEditPart

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

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

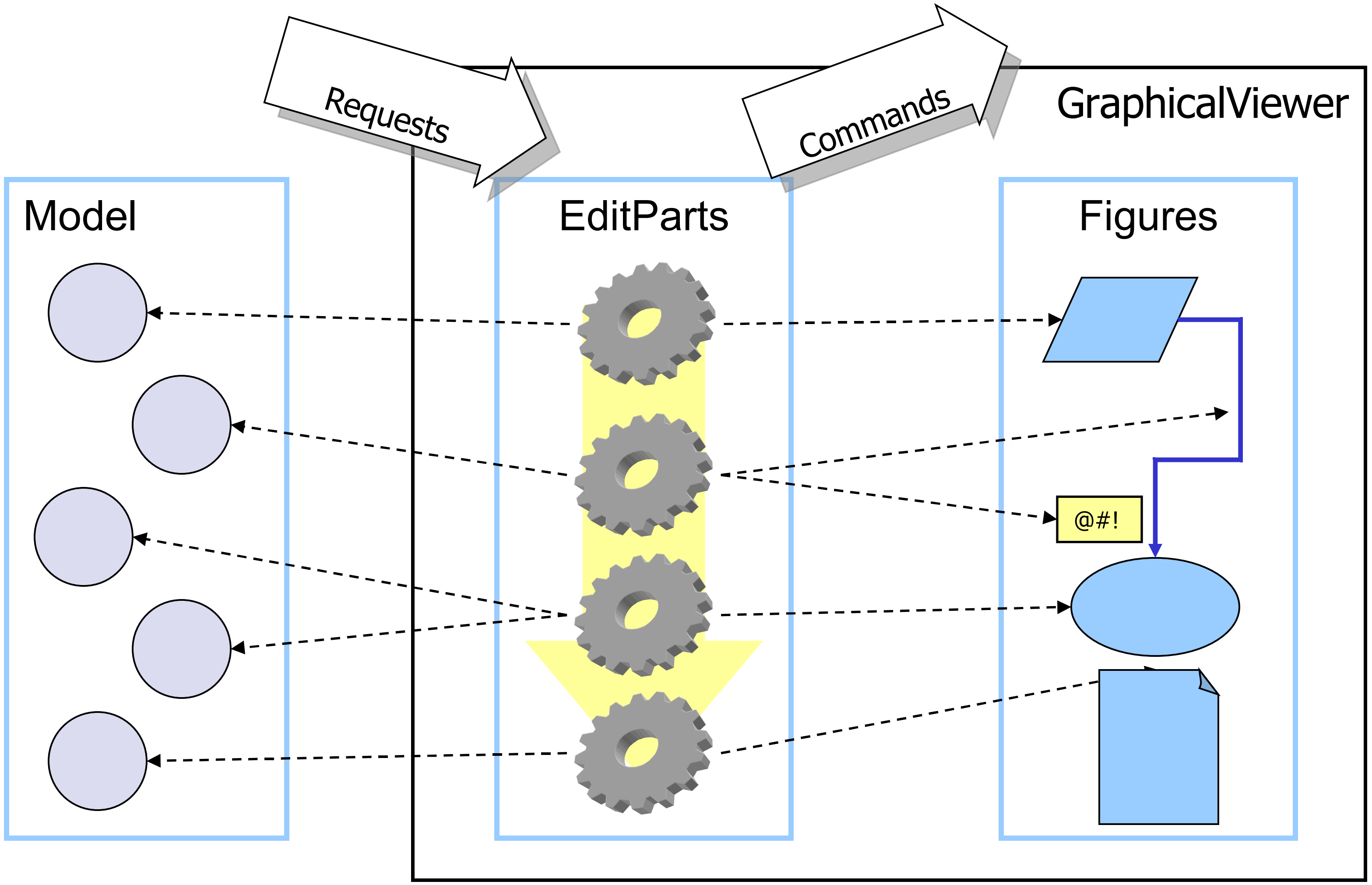
Connection EditParts

- 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

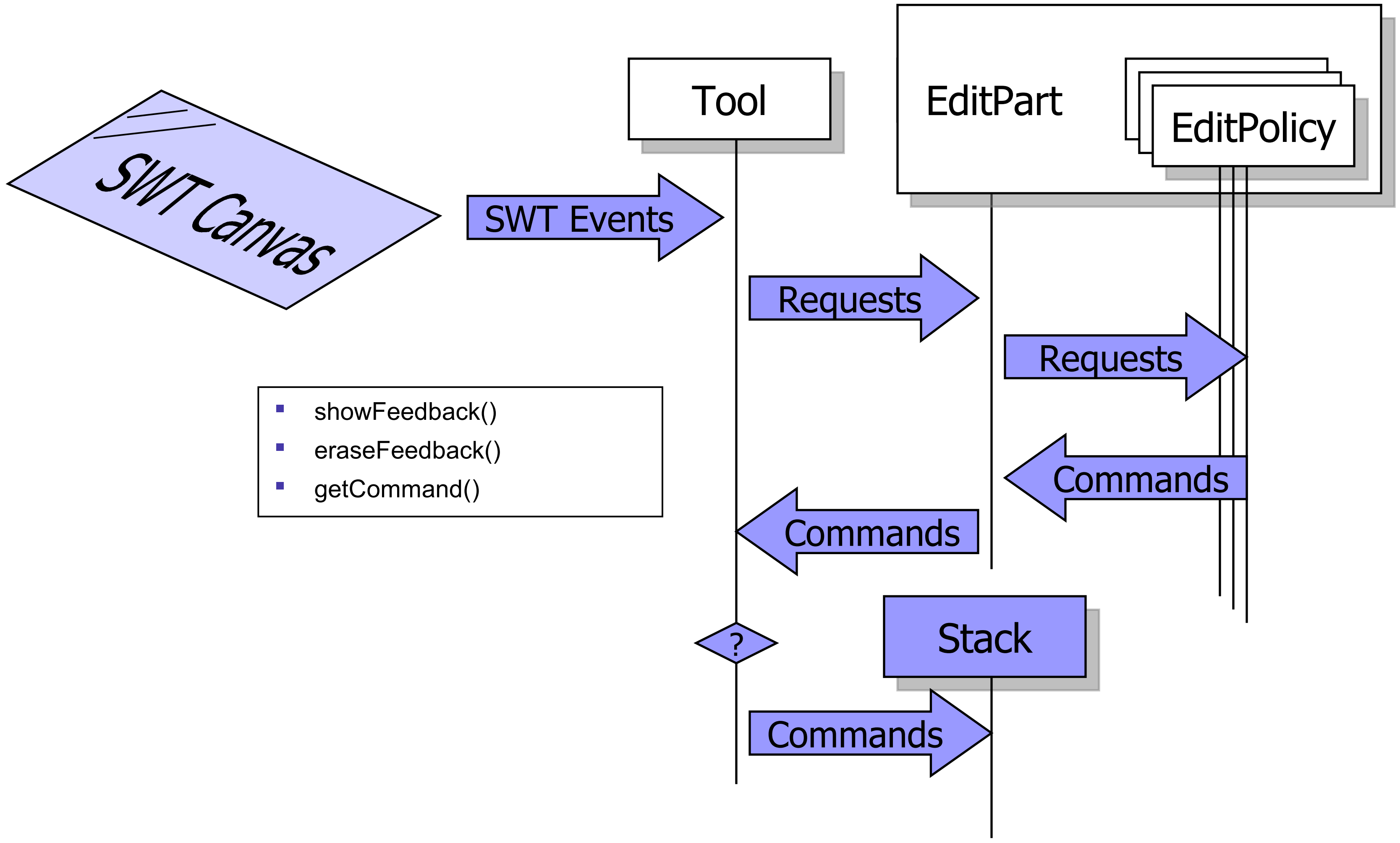
More about EditParts

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

Editing: Putting the “E” in GEF



Editing and Tools



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

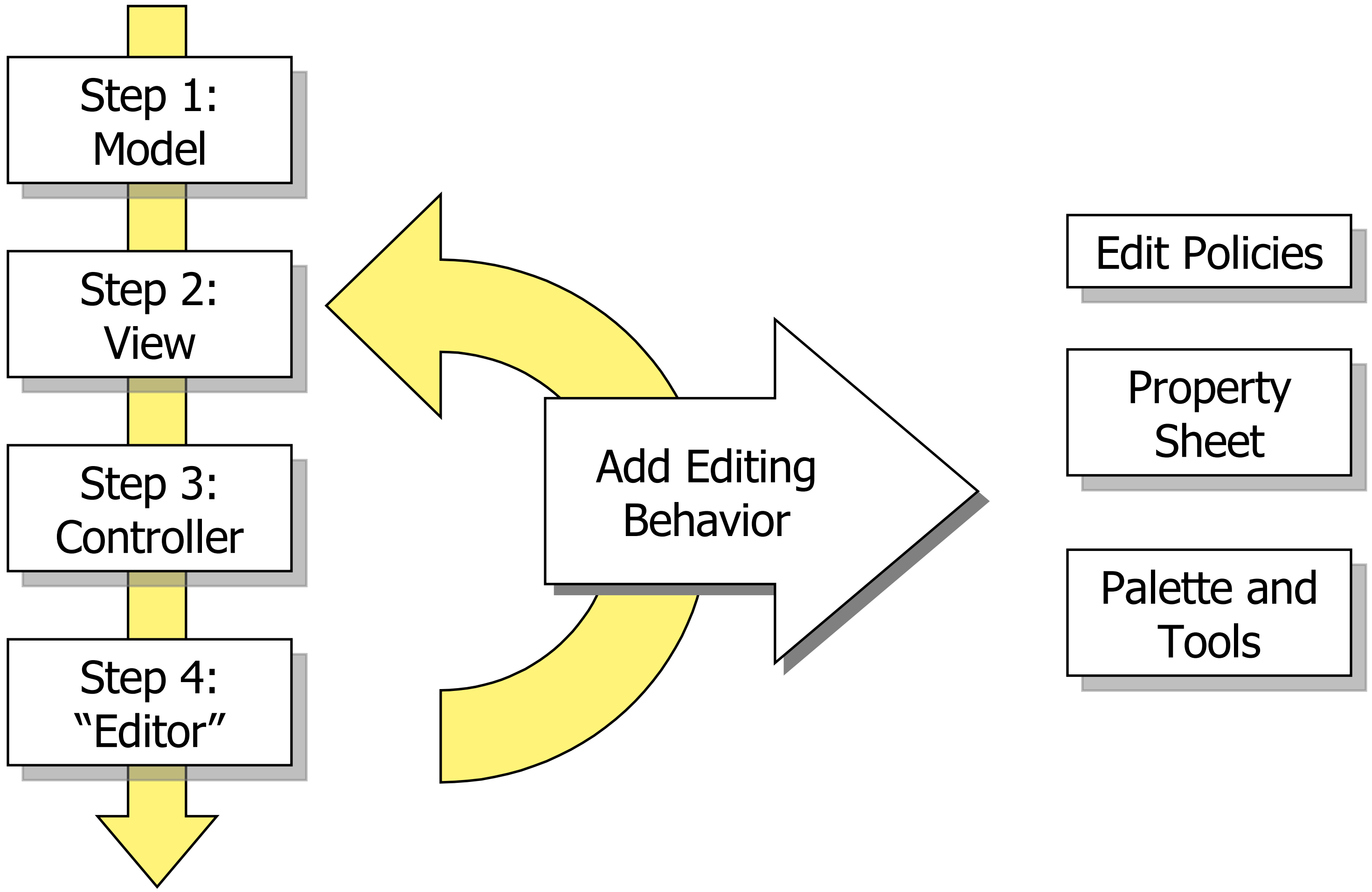
EditParts and EditPolicies

- EditPolicies are installed and maintained by their **host** part
- EditPart responsibility is keep view up-to-date
- EditPolicies handle 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”

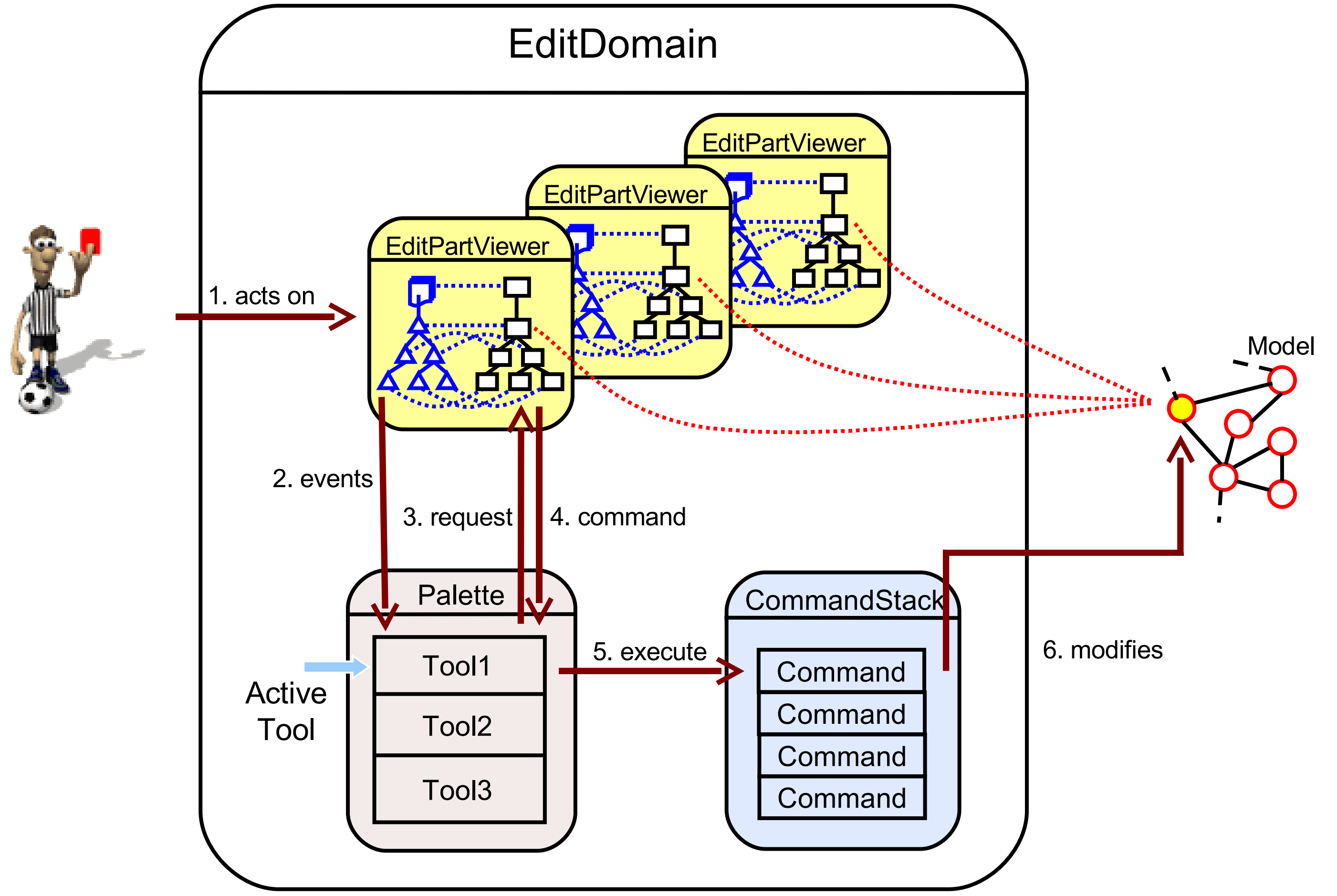
Responding to Model changes

- 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

Building a GEF Application



The Big Picture

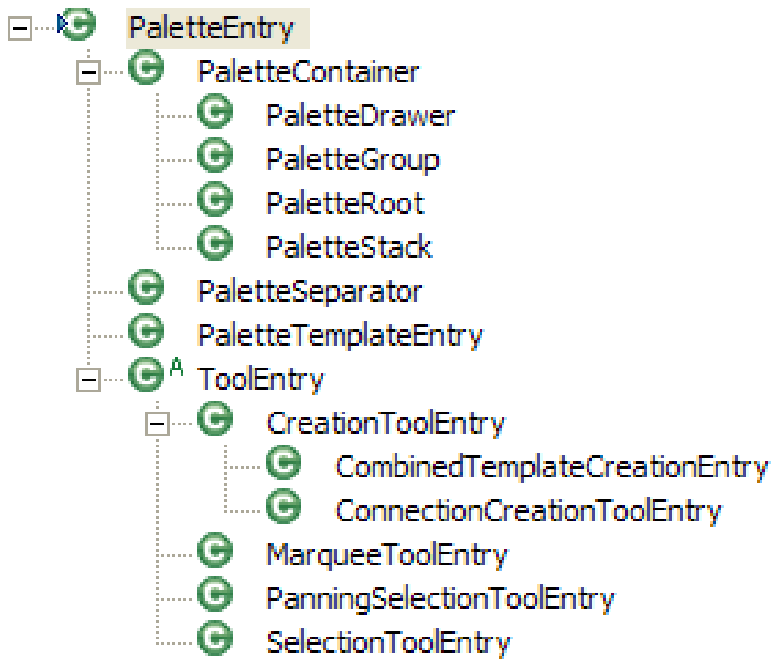


GEF – Conventions and Patterns

- 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



```
PaletteEntry
    PaletteContainer
        PaletteDrawer
        PaletteGroup
        PaletteRoot
        PaletteStack
    PaletteSeparator
    PaletteTemplateEntry
    ToolEntry
        CreationToolEntry
            CombinedTemplateCreationEntry
            ConnectionCreationToolEntry
        MarqueeToolEntry
        PanningSelectionToolEntry
        SelectionToolEntry
```

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

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

BREAK

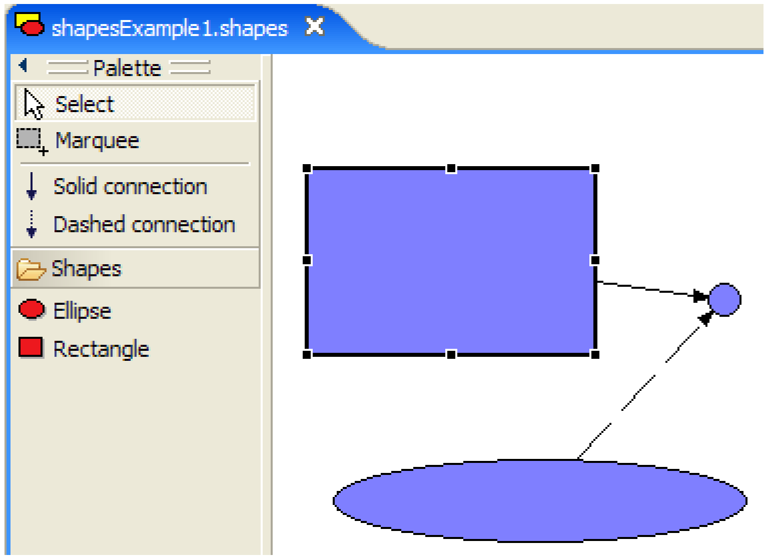
10 minutes

Set-up

- Install Eclipse 3.1M5a and GEF SDK 3.1M5
- Start with a clean workspace
- Unzip activity.zip into your workspace before launching Eclipse, and then from within Eclipse, Import -> Existing Project Into Workspace
  - http://www.eclipse.org/gef/conference/activity.zip
- Switch to the Java Perspective

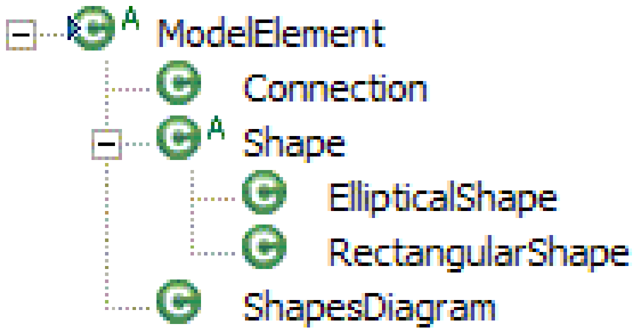
Hands-on Activity: Shapes Example

- Create your own plug-in
  - Based on the simple Shapes example contributed by Elias Volanakis
  - Skeleton provided for the plug-in

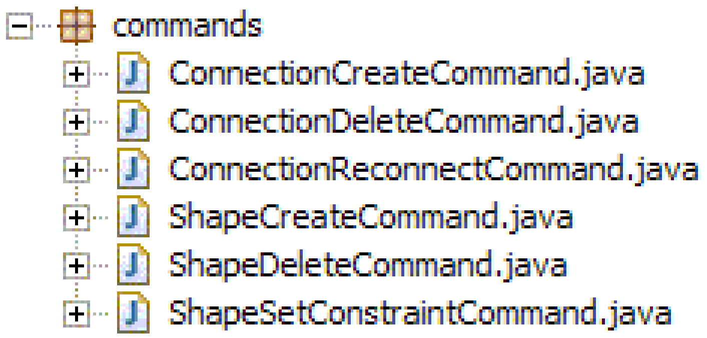


Model

- Provided!
- So are the Commands
- ModelElement implements IPropertySource and Serializable



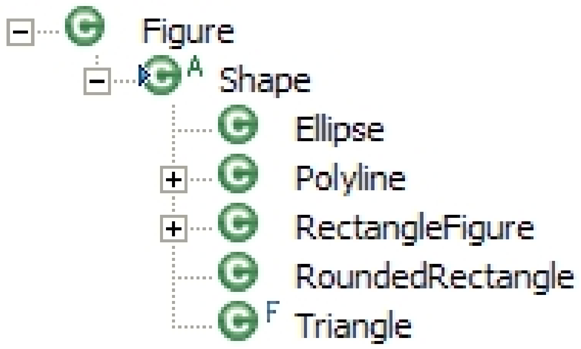
```
ModelElement
    Connection
    Shape
        EllipticalShape
        RectangularShape
    ShapesDiagram
```



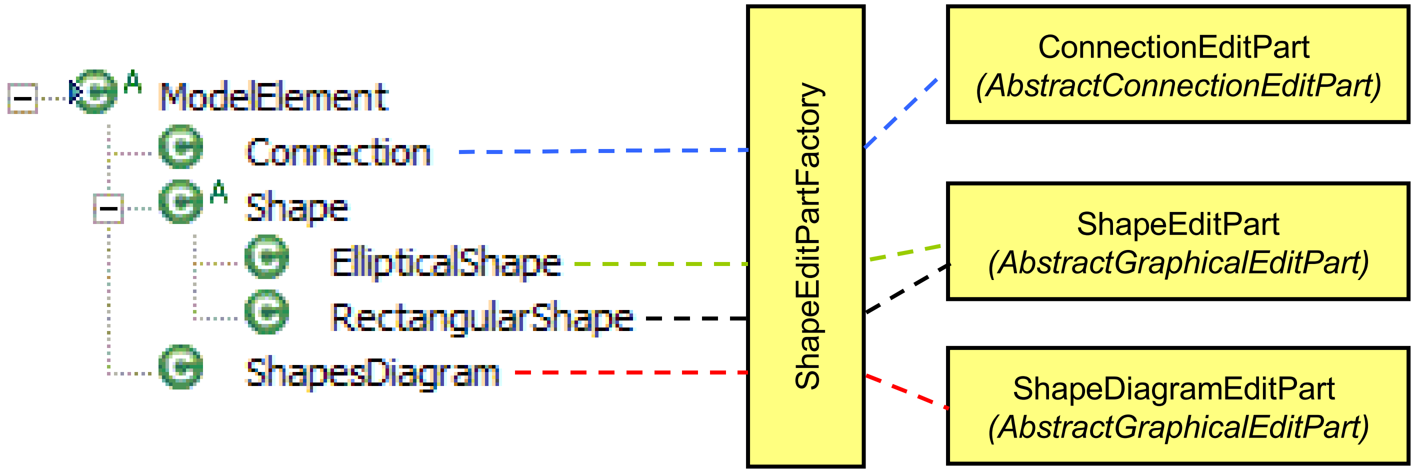
```
commands
    ConnectionCreateCommand.java
    ConnectionDeleteCommand.java
    ConnectionReconnectCommand.java
    ShapeCreateCommand.java
    ShapeDeleteCommand.java
    ShapeSetConstraintCommand.java
```

View

- Available in Draw2d
- Basic Shapes: RectangleFigure and Ellipse



Controller


ModelElement
Connection
Shape
EllipticalShape
RectangularShape
ShapesDiagram
ShapeEditPartFactory
ConnectionEditPart
(AbstractConnectionEditPart)
ShapeEditPart
(AbstractGraphicalEditPart)
ShapeDiagramEditPart
(AbstractGraphicalEditPart)

ShapeDiagramEditPart

- Listen to the model only when the EditPart is active
  - activate()
  - deactivate()
  - propertyChange()
- createFigure()
  - FreeformLayer with FreeformLayout if you want to scroll into negative co-ordinates; Figure with XYLayout otherwise
- getModelChildren()
- createEditPolicies()
  - COMPONENT_ROLE: RootComponentEditPolicy (provided by GEF) to prevent deletion of the content EditPart
  - LAYOUT_ROLE: Subclass XYLayoutEditPolicy
    - For creating, moving and resizing children
    - createChildEditPolicy() : default will do

ShapeEditPart

- Listen to the model for changes
- Create RectangleFigure or Ellipse based on the model instance
- Implement NodeEditPart to support connections
- ChopboxAnchor or EllipseAnchor (return the same anchor in all methods)
- Edit Policies
  - COMPONENT_ROLE: Sub-class ComponentEditPolicy to provide delete support
  - GRAPHICAL_NODE_ROLE: Sub-class GraphicalNodeEditPolicy
- getModelSourceConnections() and getModelTargetConnections()
- refreshVisuals()
  - ((GraphicalEditPart)getParent()).setLayoutConstraint(…)

ConnectionEditPart

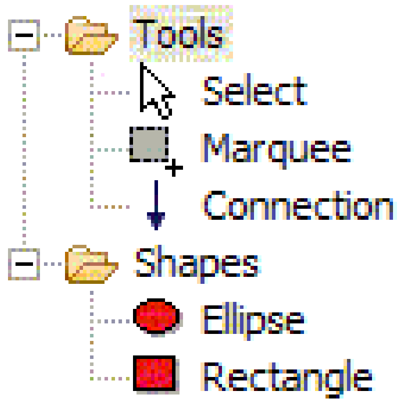
- Listen to model for changes
- Create figure
  - PolylineConnection with a target decoration and proper lineStyle
- Edit Policies
  - CONNECTION_ENDPOINTS_ROLE: ConnectionEndpointEditPolicy (provided by GEF) to select the connection's endpoints
  - CONNECTION_ROLE: Sub-class ConnectionEditPolicy to support delete

Bring It All Together - ShapesEditor

- Functionality not related to GEF is provided
  - We've added TODO:Tutorial to mark missing functionality
- Constructor: Provide a new DefaultEditDomain
- configureGraphicalViewer()
  - RootEditPart
  - EditPartFactory
- initializeGraphicalViewer()
  - Set the viewer's contents
- Palette
  - getPaletteRoot() and getPalettePreferences()
  - Delegate to ShapesEditorPaletteFactory

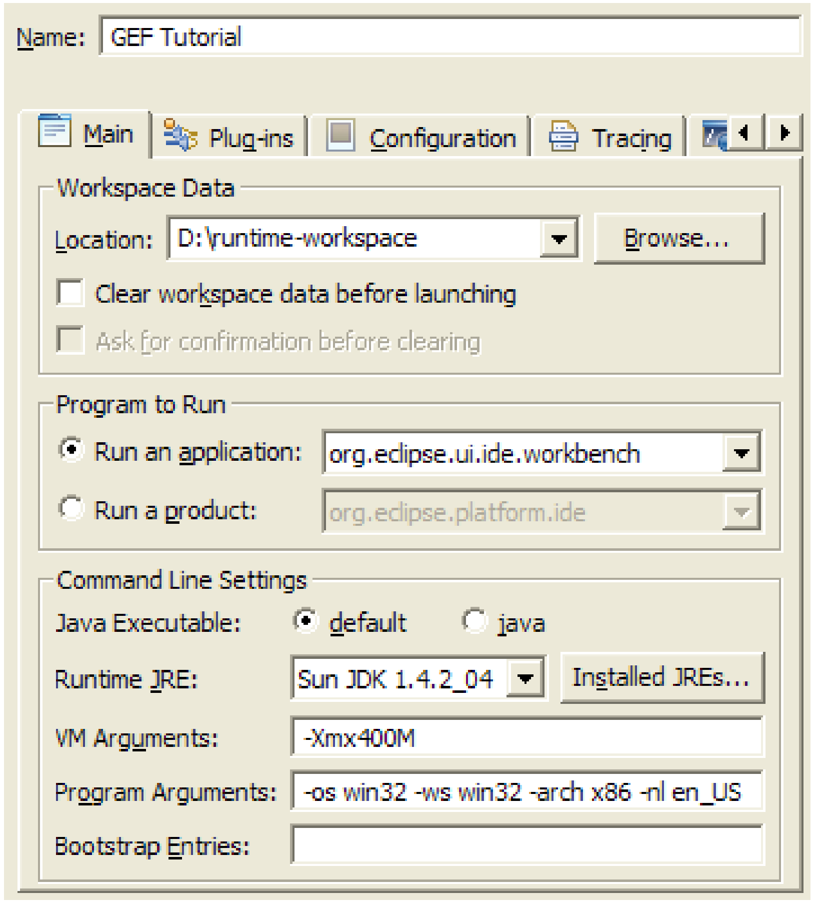
Bring It All Together - ShapesEditorPaletteFactory

- createPalettePreferences()
  - ShapesPlugin#getDefault()#getPreferenceStore()
- createPaletteRoot()
  - Tools Group has PanningSelectionToolEntry, MarqueeToolEntry and ConnectionCreationToolEntry
  - Shapes Drawer has CombinedTemplateCreationEntry for the two shapes
    - The templates can be null for now (used for DND)
  - Tip: Do not forget to provide a default tool (usually Selection)



Test Drive

- Launch the Runtime Workbench
  - Run menu -> Debug… -> Click on Entire Application -> Click New
- Test functionality
  - Create a Simple Project and a new Shapes example using the wizard
  - Click and drop from palette (notice drag and drop doesn't work yet)
  - Select, move, resize, connect parts
  - Properties View
  - Select a shape and hit '.' (the period key); then try '/' (the slash key)
  - Bring up the Palette View
  - Right-click on the palette
  - Drag the shapes into negative region
- From here, we'll incrementally add desired features



Name: GEF Tutorial

Workspace Data
Location: D:\runtime-workspace
[ ] Clear workspace data before launching
[ ] Ask for confirmation before clearing

Program to Run
[✓ Run an application: org.eclipse.ui.ide.workbench]
Run a product: org.eclipse.platform.ide

Command Line Settings
Java Executable: [✓ default] java
Runtime JRE: Sun JDK 1.4.2_04
VM Arguments: -Xmx400M
Program Arguments: -os win32 -ws win32 -arch x86 -nl en_US
Bootstrap Entries:

(Native) Drag-n-Drop from Palette

- Add DragSourceListener to the Palette and DropTargetListener to the graphical viewer
- ShapesEditor
  - createPaletteViewerProvider() - TemplateTransferDragSourceListener
  - initializeGraphicalViewer() – TemplateTransferDropTargetListener
- ShapesEditorPaletteFactory
  - Change CombinedTemplateCreationEntries' to return the model classes as templates
- Sub-class TemplateTransferDropTargetListener to provide the CreationFactory: use SimpleFactory

Context Menu

- ShapesEditorContextMenuProvider#buildContextMenu()
  - Add Undo, Redo, Delete and Save actions from the Editor’s ActionRegistry
- ShapesEditor#configureGraphicalViewer()
  - Create and hook the context menu

Actions

- ShapesEditorActionBarContributor
  - All ShapesEditor instances share this Contributor
  - buildActions() – Add RetargetActions for undo, redo, delete
  - contributeToToolBar() – Add undo and redo retarget actions to the toolbar
    - Do not create new actions here!
- Register the contributor in plugin.xml where you register your editor
  - contributorClass= "org.eclipse.gef.examples.shapes.ShapesEditorActionBarContributor"

Connection Routing

- ShapesEditor#initializeGraphicalViewer()
  - Get the connection layer from the RootEditPart
  - Create a new ShortestPathConnectionRouter and add it to the connection layer
    - Tip: Do this after the viewer's contents have been set
  - Add the router's layoutListener to the content EditPart's content pane (i.e., the figure that is going to have the connections)
  - Voila!

Outline

- ShapesOutlinePage extends ContentOutlinePage
  - Set-up as inner class in ShapesEditor since it shares a few things with it
  - Use the GEF TreeViewer
  - Since we're creating a new viewer, we'll need to define a new EditPartFactory and new EditParts
- EditPart
  - Extend AbstractTreeEditPart
  - Much simpler than earlier ones
    - No Edit Policies needed since we're only supporting selection
  - Listen to the model for changes
  - We do not show connections in the outline, so no need to worry about those
  - Override getImage() and getText()

ShapesOutlinePage

- Override createControl()
  - Outline viewer should share EditDomain/CommandStack with the Editor's viewer
  - Provide the EditPartFactory
  - Register the context menu for this viewer
  - Register viewer with the SelectionSynchronizer
    - Tip: Don't forget to remove the viewer from the SelectionSynchronizer when the outline page is disposed – override dispose()
  - Set the contents
- Override init()
  - Register Editor's actions as GlobalActionHandlers (undo, redo, delete)
  - pageSite.getActionBars().setGlobalActionHandler(…)
- Hook with the Editor
  - ShapesEditor#getAdapter()

Other Features

- Customizer for the Palette
- Zooming (already supported, but no means to manipulate it)
- Grid
- Snap To Geometry
- Rulers & Guides

That’s All, Folks!

- Feedback Questionnaire
- Upcoming GEF session on Wednesday
  - ShortestPathConnectionRouter
  - New EMF-Based Example
  - WYSIWYG Text Editing
- For further help with GEF
  - http://www.eclipse.org/gef
    - Newsgroup (Do not use the Mailing List)
    - Documentation
    - FAQs
  - Tip: Look at GEF examples to see how they handle similar problems