Developing an Editor for Directed Graphs

An Introduction to the Eclipse Graphical Editing Framework

Speaker

- Koen Aers
  - JBoss, a Division of Red Hat
  - JBoss jBPM (http://labs.jboss.org/jbossjbpm)
  - JBoss Tools (http://labs.jboss.org/tools)
    - => Graphical Process Designer (http://labs.jboss.org/jbossjbpm/gpd)

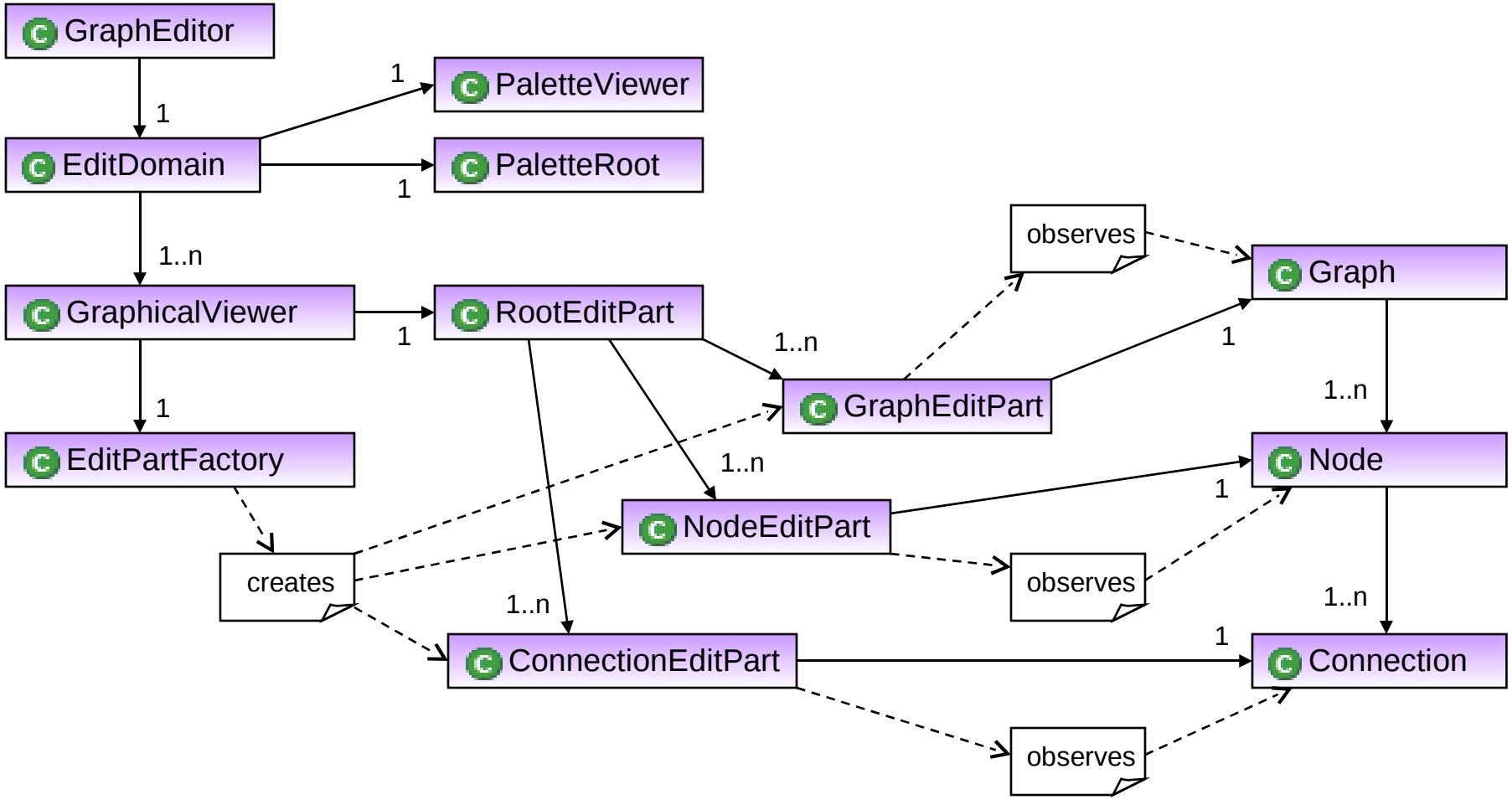
Agenda

- What is GEF?
- GEF Applied : A Graph Editor
  - An Empty Graph Editor
  - Adding the Nodes
  - Doing Things With Nodes
  - Showing Connections
  - Creating New Nodes and Connections
  - Adding Delete Support
- Final Reflections

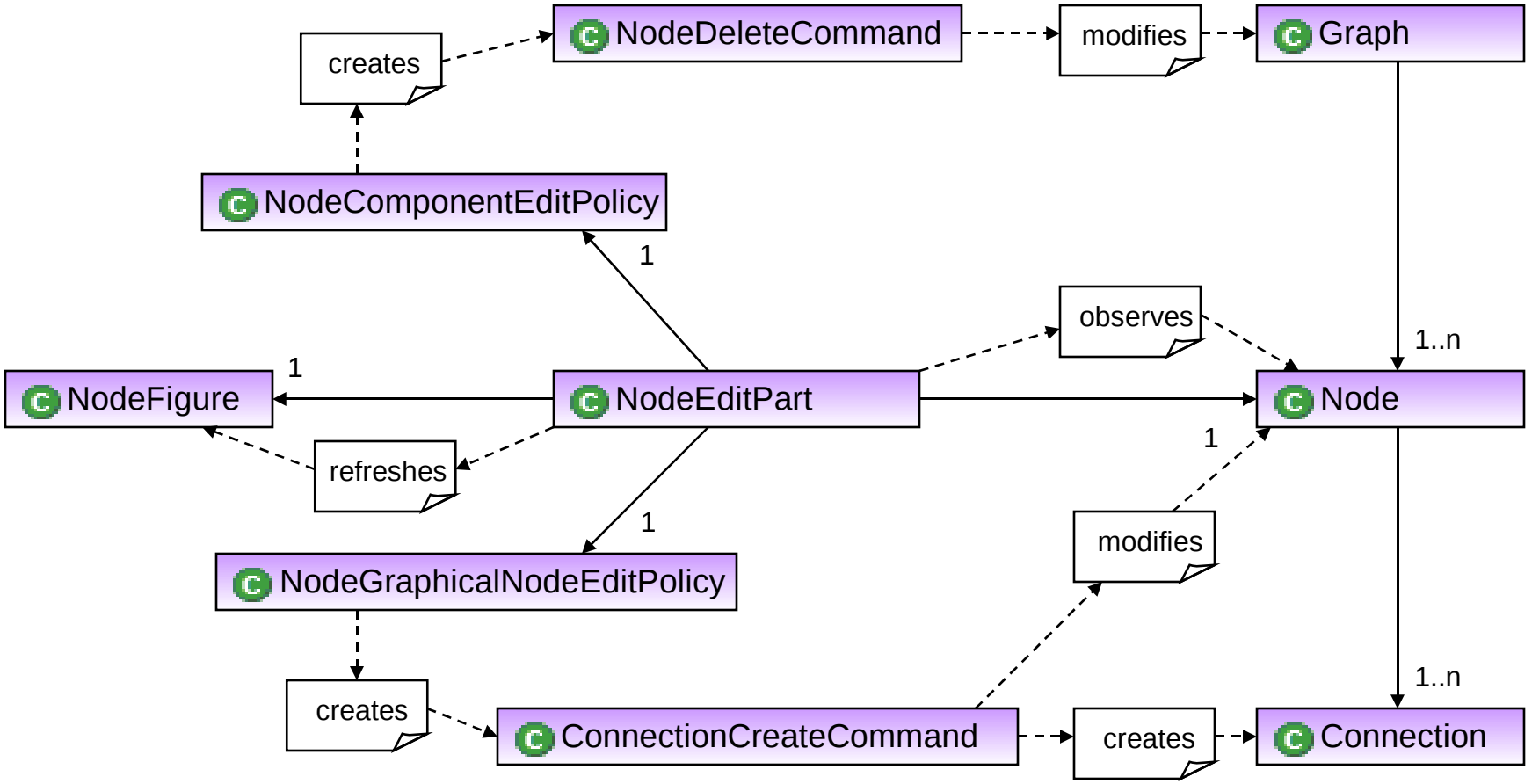
What is GEF?

- Graphical Editing Framework
- Create a Rich Graphical Editor
- Consists of 2 plug-in
  - Draw2D : layout and rendering toolkit for displaying graphics
  - GEF : framework using the old Smalltalk MVC principles
- MVC : Model, Figure, EditPart
  - Input events are translated to requests
  - EditPart has a chain-of-responsibility of so-called EditPolicies
  - EditPolicies translate the requests into GEF Commands when appropriate
  - Commands get executed and result in model changes
  - Model is observed by EditPart
  - When model changes, EditPart refreshes the view

Structure of a Typical GEF Editor



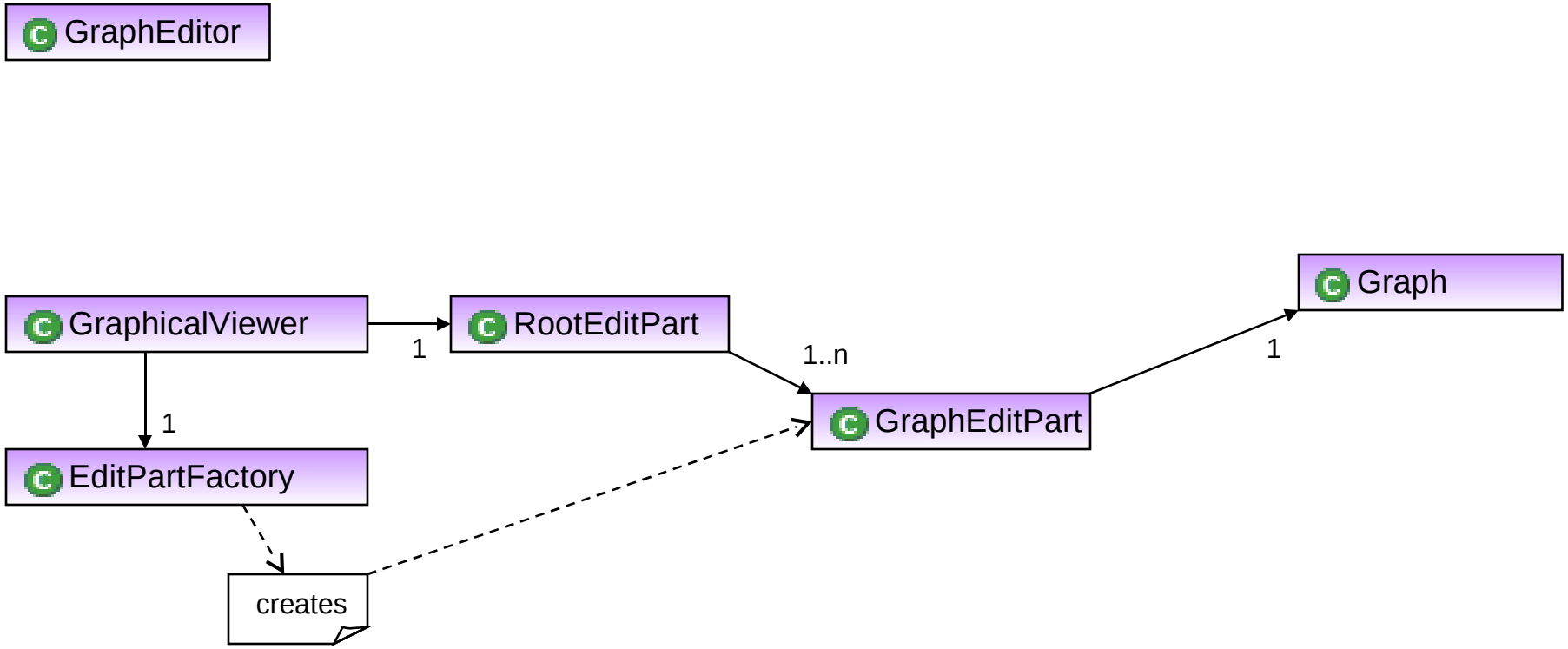
Typical GEF MVC Interactions



An Empty Graph Editor

- Eclipse Plug-in with Editor
- Add a GraphicalViewer
- Add a RootEditPart
- Define the Graph model
- Define the GraphEditPart
- Define and Add the EditPartFactory

An Empty Editor for Directed Graphs



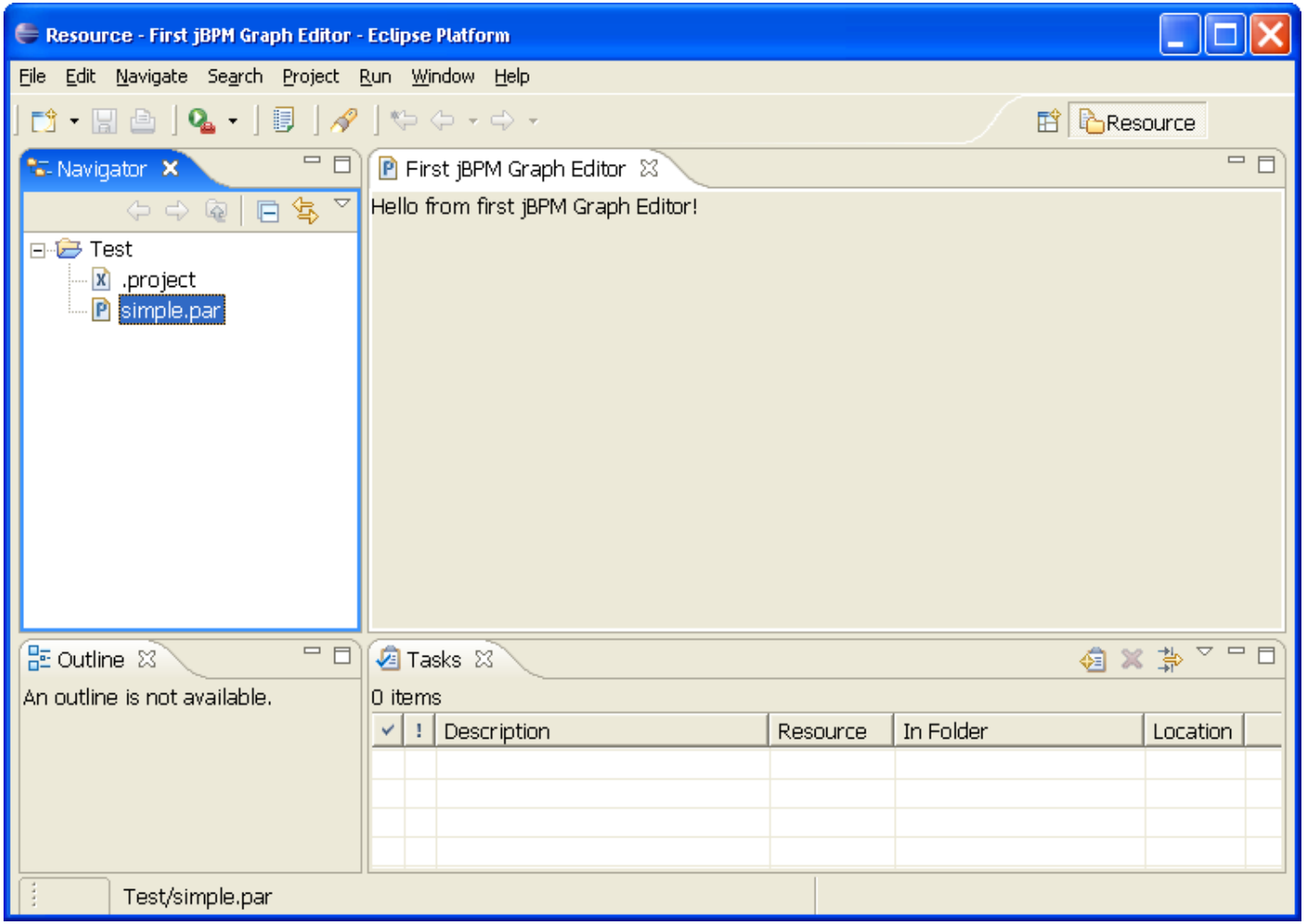
Eclipse Plug-in with Editor

```
<plugin
   id="org.jbpm.graph.ui"
   name="JBoss jBPM Graph Designer"
   version="1.0.0"
   provider-name="JBoss, a Division of Red Hat"
   class="org.jbpm.graph.ui.GraphPlugin">
   …
 <extension point = "org.eclipse.ui.editors">
    <editor
        id = "org.jbpm.graph.ui.editor.GraphEditor"
        name = "First jBPM Graph Editor"
        icon = "icons/full/obj16/par_obj.gif"
        extensions = "par"
        class = "org.jbpm.graph.ui.editor.GraphEditor" >
    </editor>
  </extension>
</plugin>
```

Eclipse Plug-in with Editor

```
public class GraphEditor extends EditorPart {
  …
  public void init(IEditorSite site, IEditorInput input)
      throws PartInitException {
    setSite(site);
    setInput(input);
  }
  …
  public void createPartControl(Composite parent) {
    Label label = new Label(parent, SWT.NONE
    label.setText("Hello from first jBPM Graph Editor!");
  }
  …
}
```

Eclipse Plug-in with Editor

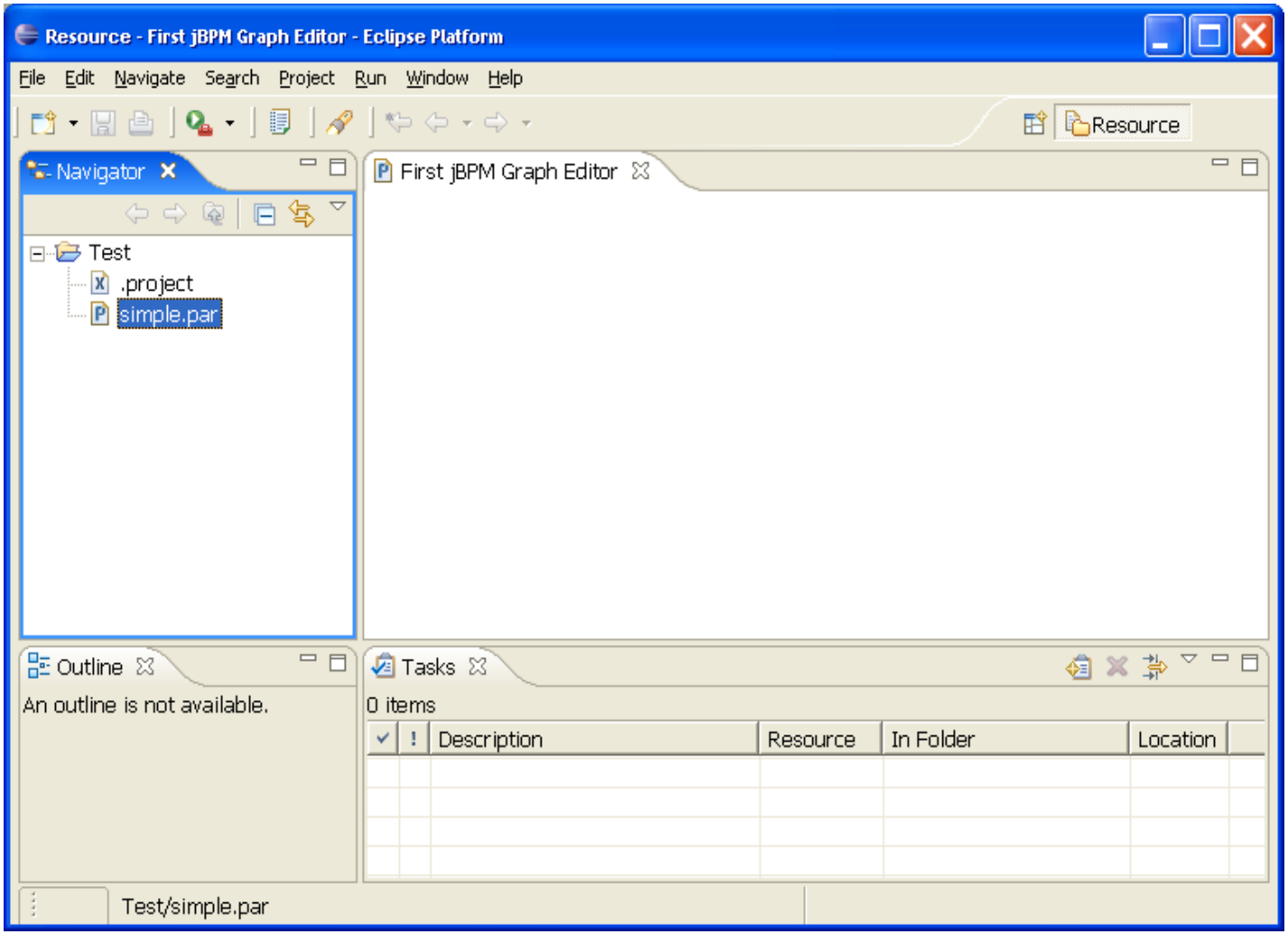


Add an EditPartViewer

- GraphicalViewer is special kind of EditPartViewer
- An adapter on an SWT **Control** that manages the **EditPart**
- Populated by setting its *contents*

```
public void createPartControl(Composite parent) {
  ScrollingGraphicalViewer viewer =
    new ScrollingGraphicalViewer();
  viewer.createControl(parent);

viewer.getControl().setBackground(ColorConstants.white);
}
```

Add an EditPartViewer



Add the RootEditPart

- Bridges the gap between the EditPartViewer and its contents
- Can provide for all kinds of services : zooming, freeform figures, etc.

```
public void createPartControl(Composite parent) {
  ScrollingGraphicalViewer viewer =
    new ScrollingGraphicalViewer();
  viewer.createControl(parent);
  viewer.setRootEditPart(new ScalableFreeformRootEditPart());
  viewer.getControl().setBackground(ColorConstants.white);
}
```

Add the EditPartFactory

- A factory for creating new EditParts
- Used when EditPart of EditPartViewer wants to create a new EditPart
- Used when setting contents of EditPartViewer

```
public void createPartControl(Composite parent) {
  …
  viewer.setRootEditPart(new ScalableFreeformRootEditPart());
  viewer.getControl().setBackground(ColorConstants.white);
  viewer.setEditPartFactory(new GraphEditPartFactory());
  viewer.setContents(new Graph());
}
```

Define the Graph Model

- Initial simplistic model
- Used as the contents of the GraphicalViewer

```
public class Graph {}
```

Define the GraphEditPartFactory

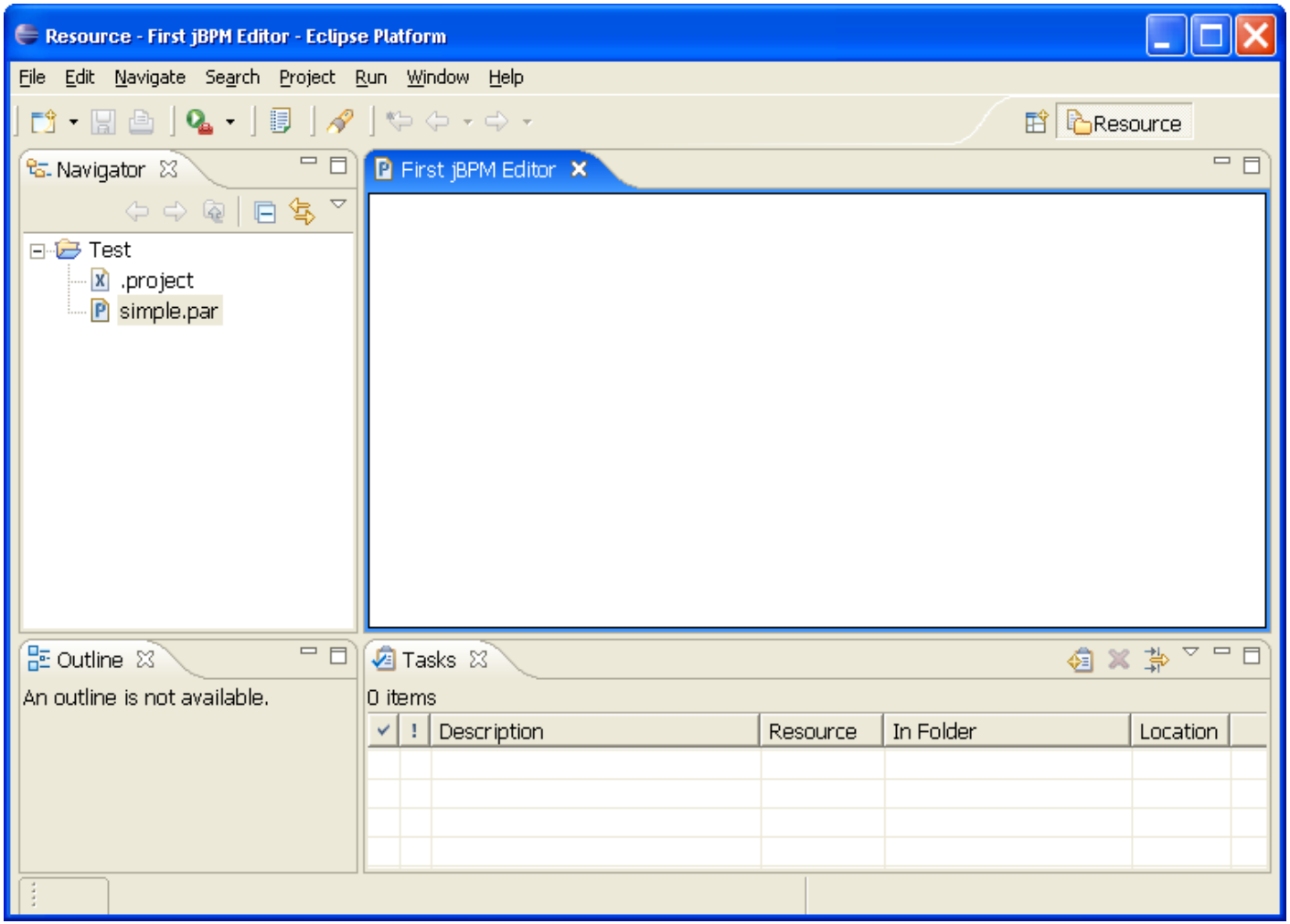
- Only Graph objects are considered
- Return a GraphEditPart instance

```
public class GraphicalEditPartFactory
    implements EditPartFactory {

  public EditPart createEditPart(
      EditPart context, Object model) {
    if (model instanceof Graph) {
      return new GraphEditPart((Graph)model);
    } else {
      return null;
    }
  }
}
```

Define the GraphEditPart

```
public class GraphEditPart
    extends AbstractGraphicalEditPart {

  public GraphEditPart(Graph graph) {
    setModel(graph);
  }

  protected IFigure createFigure() {
    FreeformLayer layer = new FreeformLayer();
    layer.setLayoutManager(new FreeformLayout());
    layer.setBorder(new LineBorder(1));
    return layer;
  }

  protected void createEditPolicies() {}

}
```

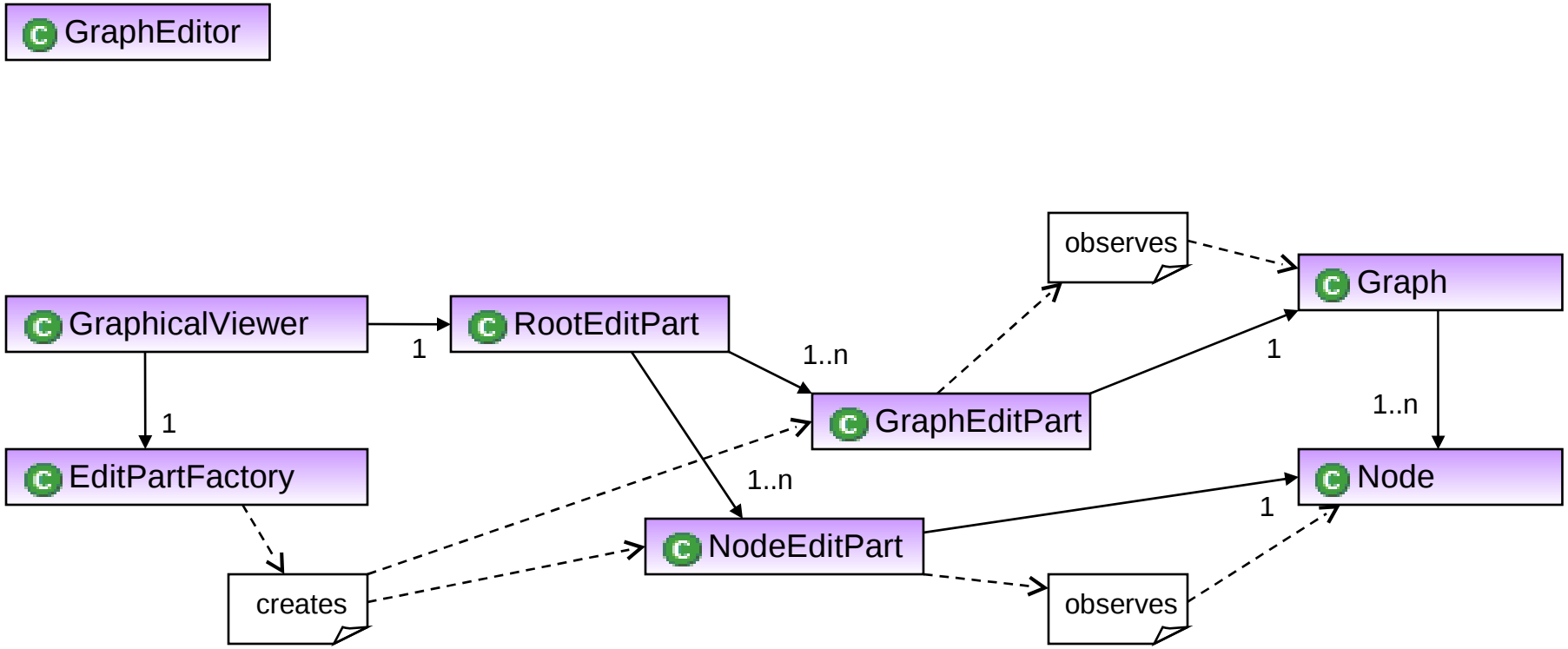
Editor Showing Empty Graph



Adding the Nodes

- The Model Class : Node
- Graphical Representation : NodeFigure
- The Controller : NodeEditPart
- Update the GraphEditPartFactory
- Update the GraphEditPart

Adding Nodes to the Editor



Define the Model : Node

```
public class Node {
  Rectangle constraint;
  String name;
  public Rectangle getConstraint() {
    return constraint;
  }
  public void setConstraint(Rectangle constraint) {
    this.constraint = constraint;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
}
```

Define the Model : Graph Revisited

- Graphs manage a list of nodes

```
public class Graph {
  List nodes
  public List getNodes() {
    if (nodes == null) nodes = new ArrayList();
    return nodes;
  }
  public void addNode(Node node) {
    getNodes().add(node);
  }
}
```

Define the Model : ContentProvider

```
public void createPartControl(Composite parent) {
  …
  viewer.setContents(
    ContentProvider.INSTANCE.newSampleGraph());
}
```

ContentProvider Continued

```
public class ContentProvider {
  public static final ContentProvider INSTANCE =
    new ContentProvider();
  public Graph newSampleGraph() {
    Graph result = new Graph();
    result.addNode(newNode(200, 150, 65, 35, “first”));
    result.addNode(newNode(300, 250, 65, 35, “second”));
    result.addNode(newNode(100, 300, 65, 35, "third”));
    return result;
  }
  …
}
```

ContentProvider Continued

```
public class ContentProvider {
  public static final ContentProvider INSTANCE =
    new ContentProvider();
  …
  private Node newNode(
      int x, int y, int width, int height, String name) {
    Node result = new Node();
    result.setConstraint
      new Rectangle(new Point(x, y),
      new Dimension(width, height)));
    result.setName(name);
    return result;
  }
}
```

Define the View : NodeFigure

- Mostly using the Draw2D framework

```
public class NodeFigure extends Figure {
  private Label label;
  private RectangleFigure rectangle;
  public NodeFigure() {
    setLayoutManager(new XYLayout());
    rectangle = new RectangleFigure();
    add(rectangle);
    label = new Label();
    add(label);
  }
  public Label getLabel() {
    return label;
  }
  …
}
```

Define the View : NodeFigure (ct'd)

```
public class NodeFigure extends Figure {
  …
  public void paintFigure(Graphics g) {
    Rectangle r = getBounds().getCopy();
    setConstraint(
      rectangle, new Rectangle(0, 0, r.width, r.height));

    setConstraint(
      label, new Rectangle(0, 0, r.width, r.height));

    rectangle.invalidate();
    label.invalidate();
  }
}
```

Define the Controller : NodeEditPart

```
public class NodeEditPart extends AbstractGraphicalEditPart {
  public NodeEditPart(Node node) { setModel(node); }
  protected IFigure createFigure() { return new
NodeFigure(); }
  protected void createEditPolicies() {}
  public void refreshVisuals() {
    NodeFigure figure = (NodeFigure)getFigure();
    Node node = (Node)getModel();
    GraphEditPart parent = (GraphEditPart)getParent();
    figure.getLabel().setText(node.getName());
    Rectangle r = new Rectangle(node.getConstraint());
    parent.setLayoutConstraint(this, figure, r);
  }
}
```

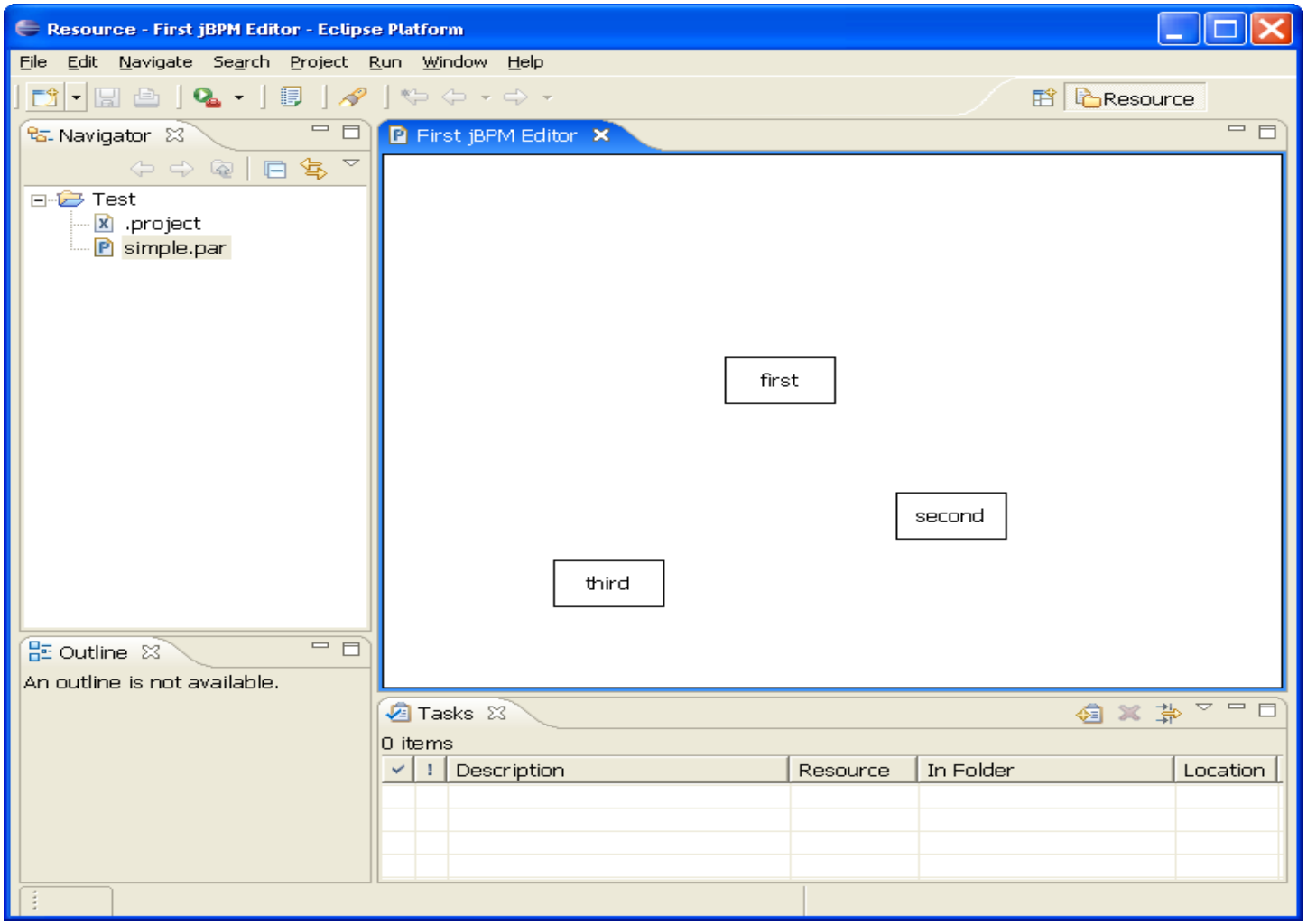
GraphEditPartFactory Revisited

```
public class GraphicalEditPartFactory
    implements EditPartFactory {

  public EditPart createEditPart(
      EditPart context, Object model) {
    if (model instanceof Graph) {
      return new GraphEditPart((Graph)model);
      else if (model instanceof Node) {
      return new NodeEditPart((Node)model);
      else {
      return null;
    }
  }

}
```

GraphEditPart Revisited

```
public class GraphEditPart
extends AbstractGraphicalEditPart {
  public GraphEditPart(Graph graph) {
    setModel(graph);
  }
  protected List getModelChildren() {
    return ((Graph)getModel()).getNodes();
  }
  protected IFigure createFigure() {
    FreeformLayer layer = new FreeformLayer();
    layer.setLayoutManager(new FreeformLayout());
    layer.setBorder(new LineBorder(1));
    return layer;
  }
  protected void createEditPolicies() {}
}
```

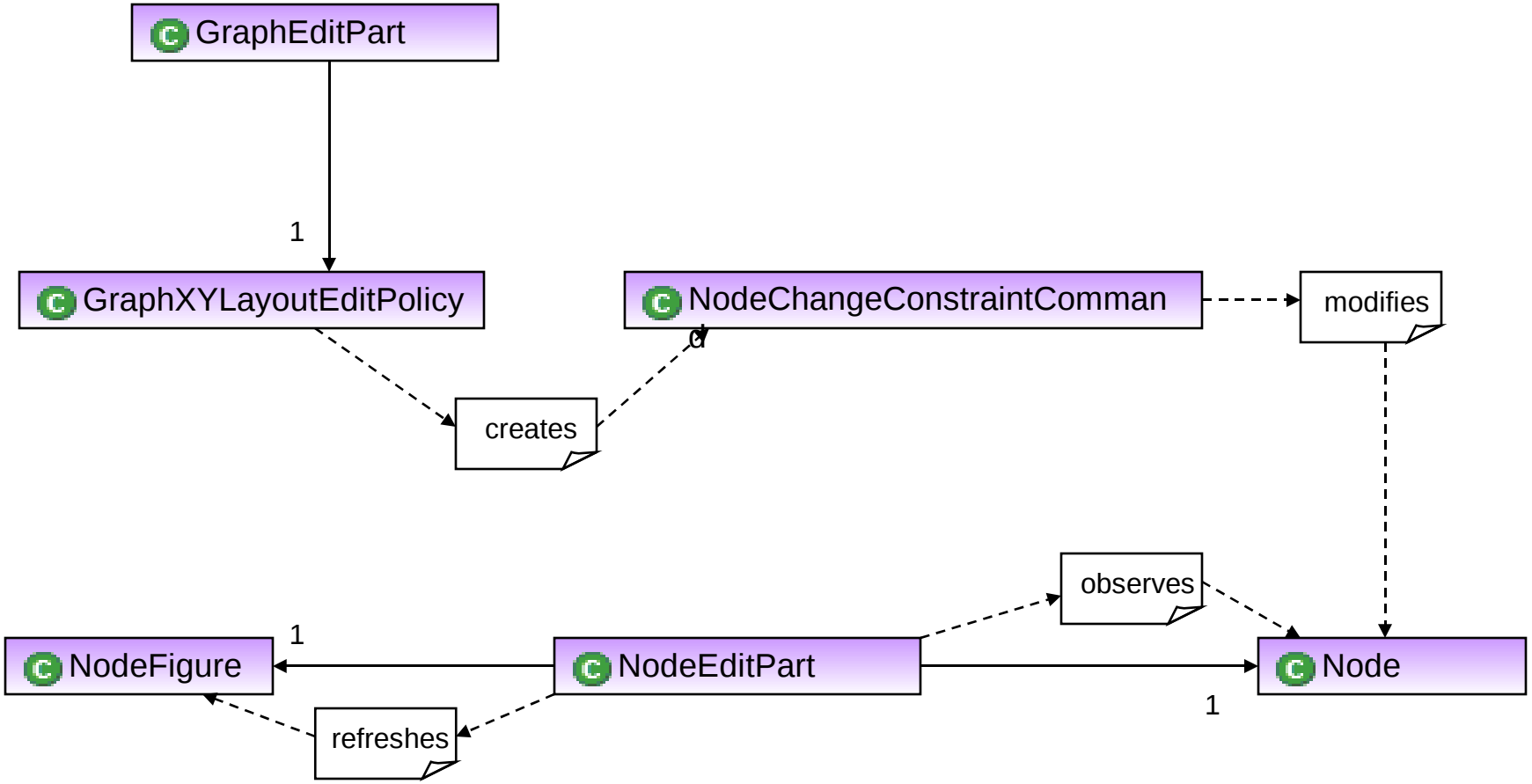
Adding the Nodes : the Result



Doing Things With Nodes

- Adding the Spine : EditDomain
- Transform Requests into Commands : EditPolicy
- Implementing Commands to Modify the Model
- Having the EditParts React to Model Changes : Observer
- Undo and Redo Support: ActionRegistry and ActionBarContributor

The Node Move/Resize Scenario



GEF's Spine : the EditDomain Class

- State of a 'GEF Application'
  - CommandStack
  - One or more EditPartViewers
  - Active Tool
- Tied with an Eclipse IEditorPart

Adding the EditDomain

```
public class GraphEditor extends EditorPart {
  private EditDomain editDomain;
  …
  public void init(IEditorSite site, IEditorInput input)
      throws PartInitException {
    …
    initEditDomain()
  }
  private void initEditDomain() {
    editDomain = new DefaultEditDomain(this);
  }
  public void createPartControl(Composite parent) {
    …
    editDomain.addViewer(viewer);
  }
}
```

NodeChangeConstraintCommand

```
public class NodeChangeConstraintCommand extends Command {
  private Rectangle newConstraint;
  private Rectangle oldConstraint
  private Node node;
  public void execute() {
    if (oldConstraint == null)
      oldConstraint = new Rectangle(node.getConstraint());
    node.setConstraint(newConstraint);
  }
  public void undo() {node.setConstraint(oldConstraint);}
  public void setNewConstraint(Rectangle newConstraint) {
    this.newConstraint = newConstraint;
  }
  public void setNode(Node node) { this.node = node; }
}
```

GraphXYLayoutEditPolicy

```
public class GraphXYLayoutEditPolicy
extends XYLayoutEditPolicy {
  protected Command createChangeConstraintCommand(
      EditPart child, Object constraint) {
    NodeChangeConstraintCommand changeConstraintCommand =
      new NodeChangeConstraintCommand();
    changeConstraintCommand.setNode((Node)child.getModel());
    changeConstraintCommand.setNewConstraint(
      (Rectangle)constraint);
    return changeConstraintCommand;
  }
  // We use a stub implementation for the other methods
  …
}
```

GraphEditPart Revisited Again

- Editparts maintain lists of EditPolicies
- Chain of responsability enabling certain commands

```
public class GraphEditPart
extends AbstractGraphicalEditPart {
  …
  protected void createEditPolicies() {
    installEditPolicy(
      EditPolicy.LAYOUT_ROLE,
      new GraphXYLayoutEditPolicy());
  }
}
```

Node/NodeEditPart Revisited

- Nodes notify changes to listeners : Observable

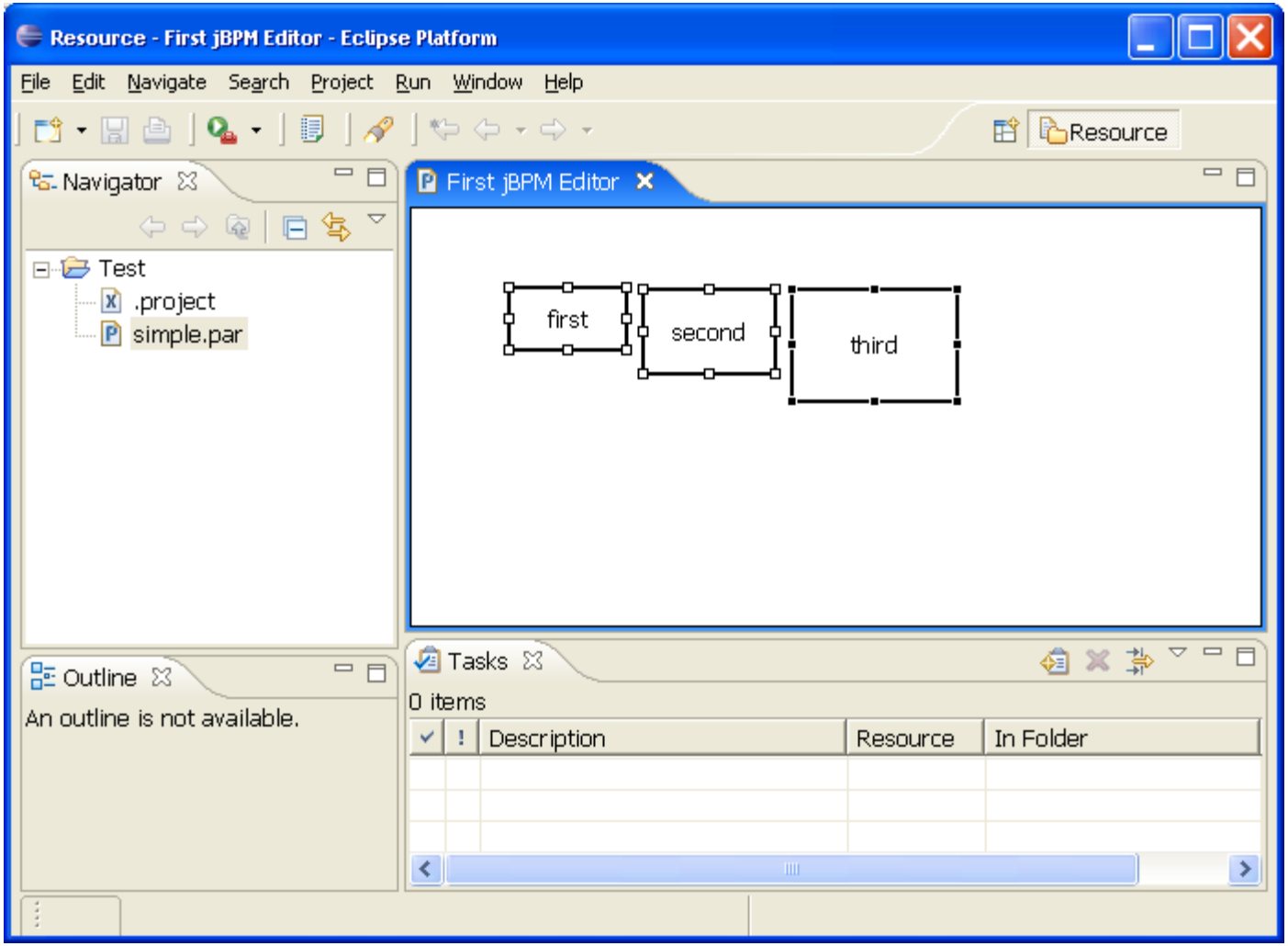
```
public class Node extends Observable {
  …
  public void setConstraint(Rectangle constraint)
    this.constraint = constraint;
    setChanged();
    notifyObservers();
  }
  …
}
```

Node/NodeEditPart Revisited

- NodeEditParts should respond to the changes of the Node : Observer

```
public class NodeEditPart
extends AbstractGraphicalEditPart implements Observer {
  …
  public void activate() {
    if (!isActive()) ((Node)getModel()).addObserver(this);
    super.activate();
  }
  public void deactivate() {
    if (isActive()) ((Node)getModel()).deleteObserver(this);
    super.deactivate();
  }
  public void update(Observable arg0, Object arg1) {
    refreshVisuals();
  }
```

Selectable, Moveable, Resizable Nodes



Undo and Redo Support

- Add an ActionRegistry : container for Editor Actions
- Implement and register the EditorActionContributor
- Keep track of the Command events : CommandStackListener
- Adapt the Editor to the CommandStack class

Add the ActionRegistry

```
public class GraphEditor extends EditorPart {
  private ActionRegistry actionRegistry;
  …
  public void init(IEditorSite site, IEditorInput input)
  throws PartInitException {
    …
    initActionRegistry();
  }
  private void initActionRegistry() {
    actionRegistry = new ActionRegistry();
    actionRegistry.registerAction(new UndoAction(this));
    actionRegistry.registerAction(new RedoAction(this));
  }
  public ActionRegistry getActionRegistry() {
    return actionRegistry;
  }
}
```

Register the ActionBarContributor

- Defines the actions for the editor
- Registered in the plugin.xml

```
<plugin
  …
  <extension point = "org.eclipse.ui.editors">
    <editor
      id = "org.jbpm.graph.ui.editor.GraphEditor“
      …
      class = "org.jbpm.graph.ui.editor.GraphEditor"
      contributorClass=
        "org.jbpm.graph.ui.editor.ActionBarContributor" >
    </editor>
  </extension>
</plugin>
```

Implement the ActionBarContributor

```
public class ActionBarContributor
extends EditorActionBarContributor {
  public void setActiveEditor(IEditorPart targetEditor) {
    IActionBars actionBars = getActionBars();
    if (actionBars == null) return;
    String undoId = ActionFactory.UNDO.getId();
    String redoId = ActionFactory.REDO.getId();
    ActionRegistry actionRegistry =
      ((GraphEditor)targetEditor).getActionRegistry();
    actionBars.setGlobalActionHandler(
      undoId, actionRegistry.getAction(undoId));
    actionBars.setGlobalActionHandler(
      redoId, actionRegistry.getAction(redoId));
    actionBars.updateActionBars();
  }
}
```

Define a CommandStackListener

```
public class GraphEditorListener
implements CommandStackListener {
  private ActionRegistry actionRegistry;
  public GraphEditorListener(
      ActionRegistry registry) {
    this.actionRegistry = registry;
  }
  public void commandStackChanged(EventObject event) {
    Iterator iterator = actionRegistry.getActions();
    while (iterator.hasNext()) {
      Object action = iterator.next();
      if (action instanceof UpdateAction)
        ((UpdateAction)action).update();
    }
  }
}
```

Add a CommandStackListener

- Update the actions in the registry whenever the CommandStack's state changes

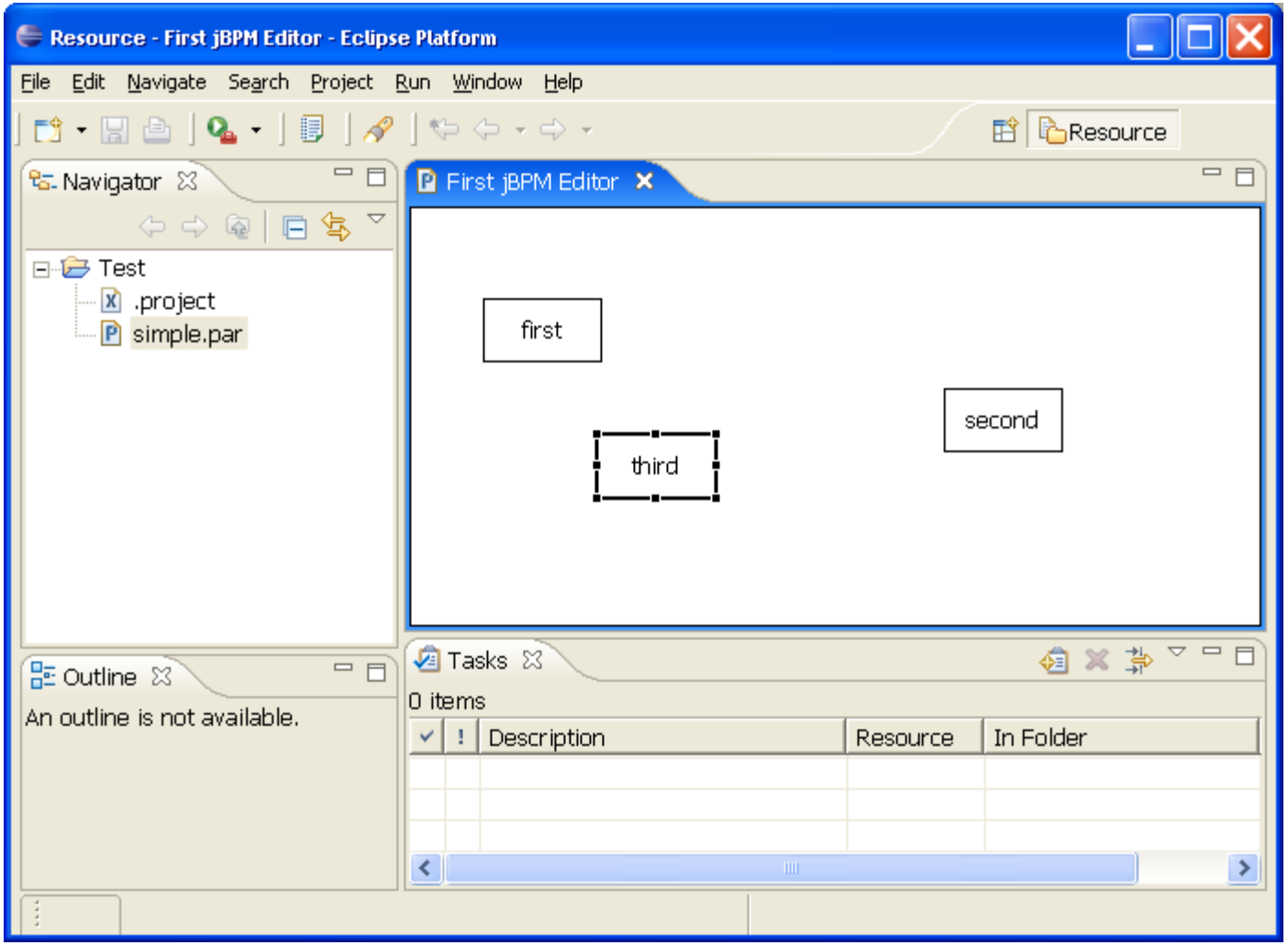
```
public class GraphEditor extends EditorPart {
  …
  public void init(IEditorSite site, IEditorInput input)
  throws PartInitException {
    …
    initGraphEditorListener();
  }
  private void initGraphEditorListener() {
    editDomain.getCommandStack().addCommandStackListener(
      new GraphEditorListener(actionRegistry));
  }
}
```

Adapt the Editor to CommandStack

- Using the IAdaptable interface extended by IEditorPart

```
public class GraphEditor extends EditorPart {
  …
  public Object getAdapter(Class adapter) {
    if (adapter == CommandStack.class) {
      return editDomain.getCommandStack();
    }
    return super.getAdapter(adapter);
  }
}
```

Undo and Redo Demo



Showing Connections

- The Model Class : Connection
- The Controller and Graphical Representation :
  - ConnectionEditPart
  - PolyLineConnection

Add the Connection Model Class

```
public class Connection {
  private Node source, target;
  public Node getSource() { return source; }
  public void setSource(Node source) {
    if (this.source != null)
      source.removeSourceConnection(this);
    this.source = source;
    source.addSourceConnection(this);
  }
  public Node getTarget() { return target; }
  public void setTarget(Node target) {
    if (this.target != null)
      target.removeTargetConnection(this);
    this.target = target;
    target.addTargetConnection(this);
  }
}
```

Update the Node Model Class

```
public class Node extends Observable {
  …
  private List sourceConnections, targetConnections;
  …
  public List getSourceConnections() {
    if (sourceConnections == null)
      sourceConnections = new ArrayList();
    return sourceConnections;
  }
  public List getTargetConnections() {
    if (targetConnections == null)
      targetConnections = new ArrayList();
    return targetConnections;
  }
  …
}
```

Update the Node Model Class (ctn'd)

```
public class Node extends Observable {
  …
  public void addSourceConnection(Connection connection) {
    getSourceConnections().add(connection);
  }
  public void addTargetConnection(Connection connection) {
    getTargetConnections().add(connection);
  }
  public void removeSourceConnection(Connection connection) {
    getSourceConnections().remove(connection);
  }
  public void removeTargetConnection(Connection connection) {
    getTargetConnections().remove(connection);
  }
}
```

Update the ContentProvider

```
public class ContentProvider {
  …
  public Graph newSampleGraph() {
    Graph result = new Graph();
    …
    newConnection(first, second);
    newConnection(first, third);
    newConnection(second, third);
    return result;
  }
  private Connection newConnection(Node source, Node target) {
    Connection result = new Connection();
    result.setSource(source);
    result.setTarget(target);
    return result;
  }
}
```

Define the ConnectionEditPart

- Join source and target EditParts
- Figure is typically a line between the two nodes

```
public class ConnectionEditPart
extends AbstractConnectionEditPart {
  public ConnectionEditPart(Connection connection) {
    setModel(connection);
  }
  protected void createEditPolicies() {}
  protected IFigure createFigure() {
    return new PolylineConnection();
  }
}
```

GraphicalEditPartFactory Revisited

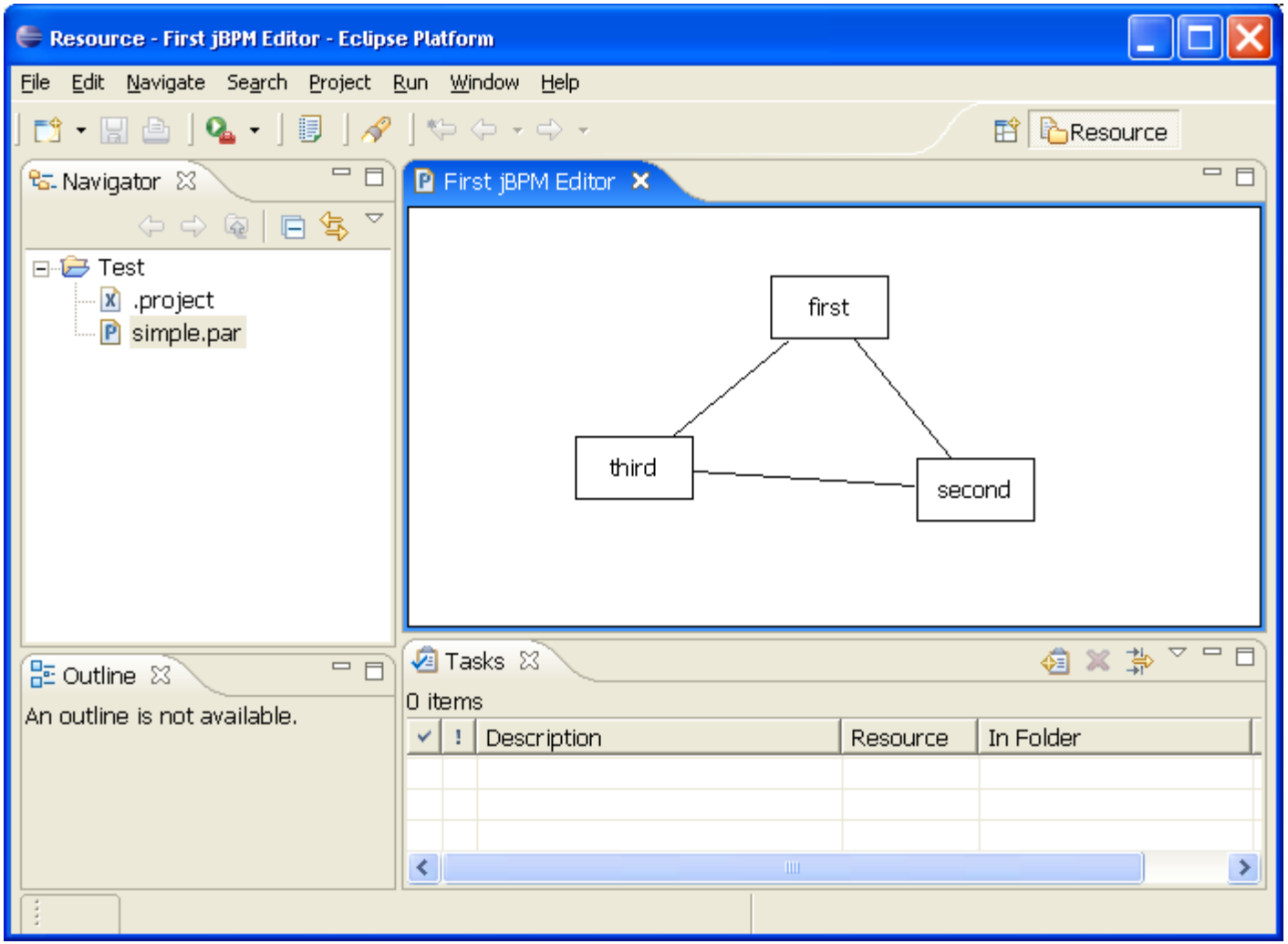
```
public class GraphicalEditPartFactory
implements EditPartFactory {
  public EditPart createEditPart(
      EditPart context, Object model) {
    if (model instanceof Graph) {
      return new GraphEditPart((Graph)model);
    } else if (model instanceof Node) {
      return new NodeEditPart((Node)model);
    } else if (model instanceof Connection){
      return new ConnectionEditPart((Connection)model);
    } else {
      return null;
    }
  }
}
```

NodeEditPart Revisited

- Connections can exist without a model
- Connections cannot exist without a source and a target
- Burden of obtaining the source and target connections is on the NodeEditPart and not on the Node model

```
public class NodeEditPart
extends AbstractGraphicalEditPart implements Observer {
  …
  protected List getModelSourceConnections() {
    return ((Node)getModel()).getSourceConnections();
  }
  protected List getModelTargetConnections() {
    return ((Node)getModel()).getTargetConnections();
  }
}
```

Nodes and Connections



Creating Nodes and Connections

- Splitting the Canvas : SashForm and PaletteViewer
- Adding the Palette : PaletteRoot and its Tools
- Create Nodes : NodeCreateCommand
- Create Connections :
  - GraphicalNodeEditPolicy
  - ConnectionCreateCommand
  - ConnectionAnchor
  - NodeEditPart interface

GraphEditor Revisited Once More

- Refactor createPartControl
  - Add SashForm to parent
  - Add PaletteViewer and GraphViewer

```
public class GraphEditor extends EditorPart {
  …
  public void createPartControl(Composite parent) {
    SashForm form = new SashForm(parent, SWT.HORIZONTAL);
    createPaletteViewer(form);
    createGraphViewer(form);
    form.setWeights(new int[] { 15, 85 });
  }
  …
}
```

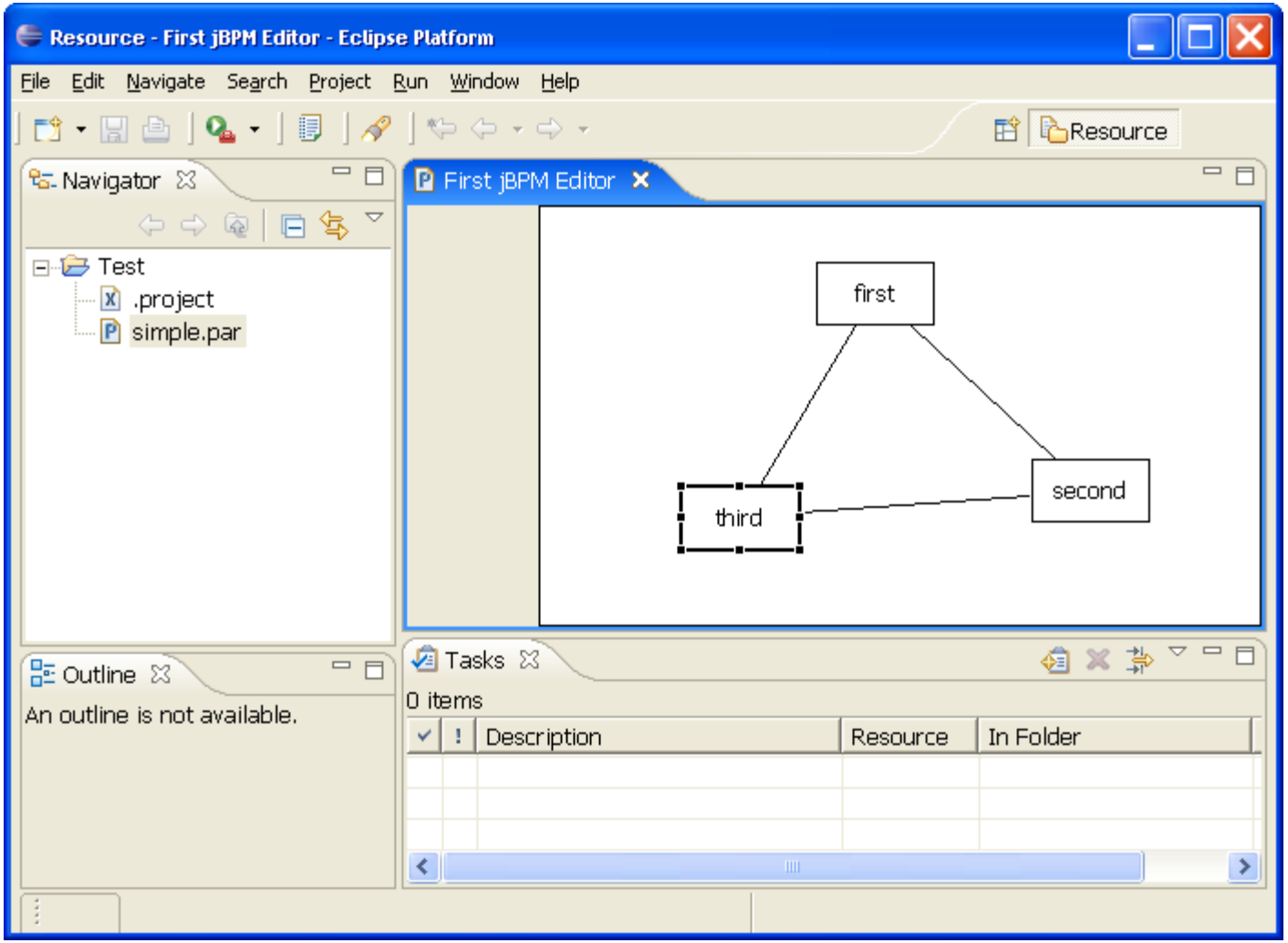
GraphEditor Revisited Once More (ctn'd)

```
public class GraphEditor extends EditorPart {
  …
  private void createPaletteViewer(Composite parent)  {
    PaletteViewer viewer = new PaletteViewer();
    viewer.createControl(parent);
    editDomain.setPaletteViewer(viewer);
    editDomain.setPaletteRoot(new PaletteRoot());
  }
  …
}
```

GraphEditor Revisited Once More (ctn'd)

```
public class GraphEditor extends EditorPart {
  …
private void createGraphViewer(Composite parent) {
    ScrollingGraphicalViewer viewer =
      new ScrollingGraphicalViewer();
    viewer.setRootEditPart(new ScalableFreeformRootEditPart());
    viewer.createControl(parent);
    viewer.getControl().setBackground(ColorConstants.white);
    viewer.setEditPartFactory(new GraphicalEditPartFactory());
    viewer.setContents(
      ContentProvider.INSTANCE.newSampleGraph());
    editDomain.addViewer(viewer);
  }
}
```

Editor with Empty Palette



Populate the Palette : PaletteRoot

```
public class GraphPalette extends PaletteRoot {
  public GraphPalette() {
    PaletteGroup group = new PaletteGroup("Graph Controls");
    SelectionToolEntry entry = new SelectionToolEntry();
    group.add(entry);
    setDefaultEntry(entry);
    group.add(new PaletteSeparator());
    group.add(new CreationToolEntry(
      “Node”, “Creates a new node.”,
      new NodeFactory(), null, null));
    group.add(new ConnectionCreationToolEntry(
      “Connection”, “Creates a new connection.”,
      new ConnectionFactory(), null, null));
    add(group);
  }
}
```

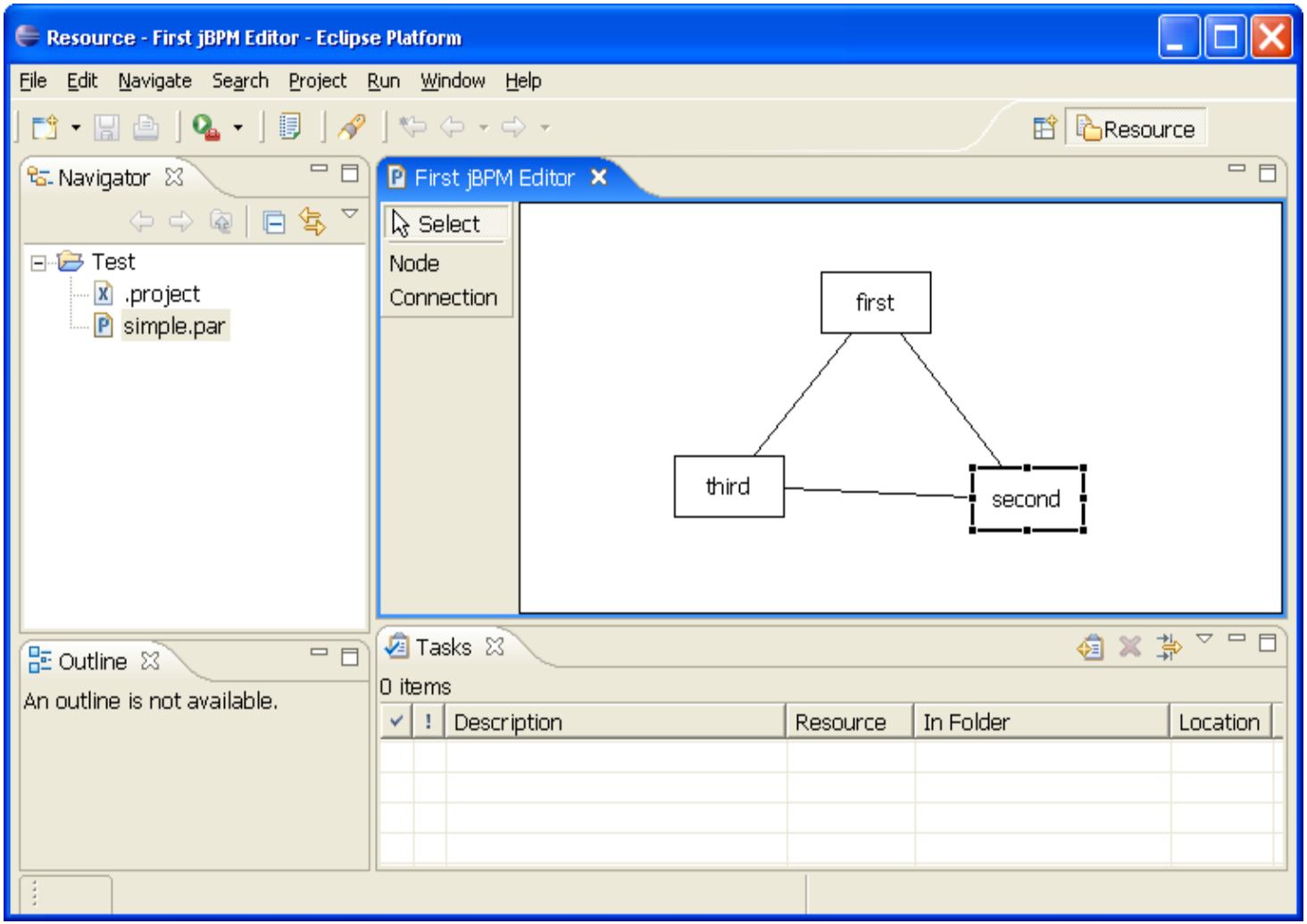
Populate the Palette : CreationFactory

```
public class NodeFactory implements CreationFactor
  public Object getNewObject() { return new Node(); }
  public Object getObjectType() { return Node.class; }
}

public class ConnectionFactory implements CreationFactory {
  public Object getNewObject() { return new Connection(); }
  public Object getObjectType() { return Connection.class; }
}
```

Populate the Palette : GraphEditor

```
public class GraphEditor extends EditorPart {
  …
  private void createPaletteViewer(Composite parent)
{
    PaletteViewer viewer = new PaletteViewer();
    viewer.createControl(parent);
    editDomain.setPaletteViewer(viewer);
    editDomain.setPaletteRoot(new GraphPalette());
  }
  …
}
```

Editor With Populated Palette



Creating New Nodes

- Define NodeCreateCommand
- Implement getCreateCommand
- Make Graph/GraphEditPart Observer/Observable

Define NodeCreateCommand

```
public class NodeCreateCommand extends Command {
  …
  private Node node;
  private Rectangle constraint;
  private Graph parent;
  public void execute() {
    setNodeConstraint();
    setNodeName();
    parent.addNode(node);
  }
  private void setNodeName() {
    node.setName(parent.getNextNodeName());
  }
  private void setNodeConstraint() {
    if (constraint !=
null)node.setConstraint(constraint);
  }
  …
```

Define NodeCreateCommand (ctn'd)

```
public class NodeCreateCommand extends Command {
  private static final Dimension INITIAL_NODE_DIMENSION =
    new Dimension(65, 35);
  …
  public void undo() {parent.removeNode(node);}
  public void setNode(Node node) {this.node = node;}
  public void setLocation(Point location) {
    this.constraint = new Rectangle(
    location, INITIAL_NODE_DIMENSION);
  }
  public void setParent(Graph parent) {
    this.parent = parent;
  }
}
```

Graph Revisited Again

```
public class Graph {
  …
  public String getNextNodeName() {
    int runner = 1;
    while (true) {
      String candidate = "node" + runner;
      if (getNodeByName(candidate) == null) return candidate;
      runner ++;
    }
  }
  private Node getNodeByName(String candidate) {
    for (int i = 0; i < getNodes().size(); i++)
      if (candidate.equals(((Node)getNodes().get(i)).getName()))
        return (Node)getNodes().get(i);
    return null;
  }
}
```

GraphXYLayoutPolicy Revisited

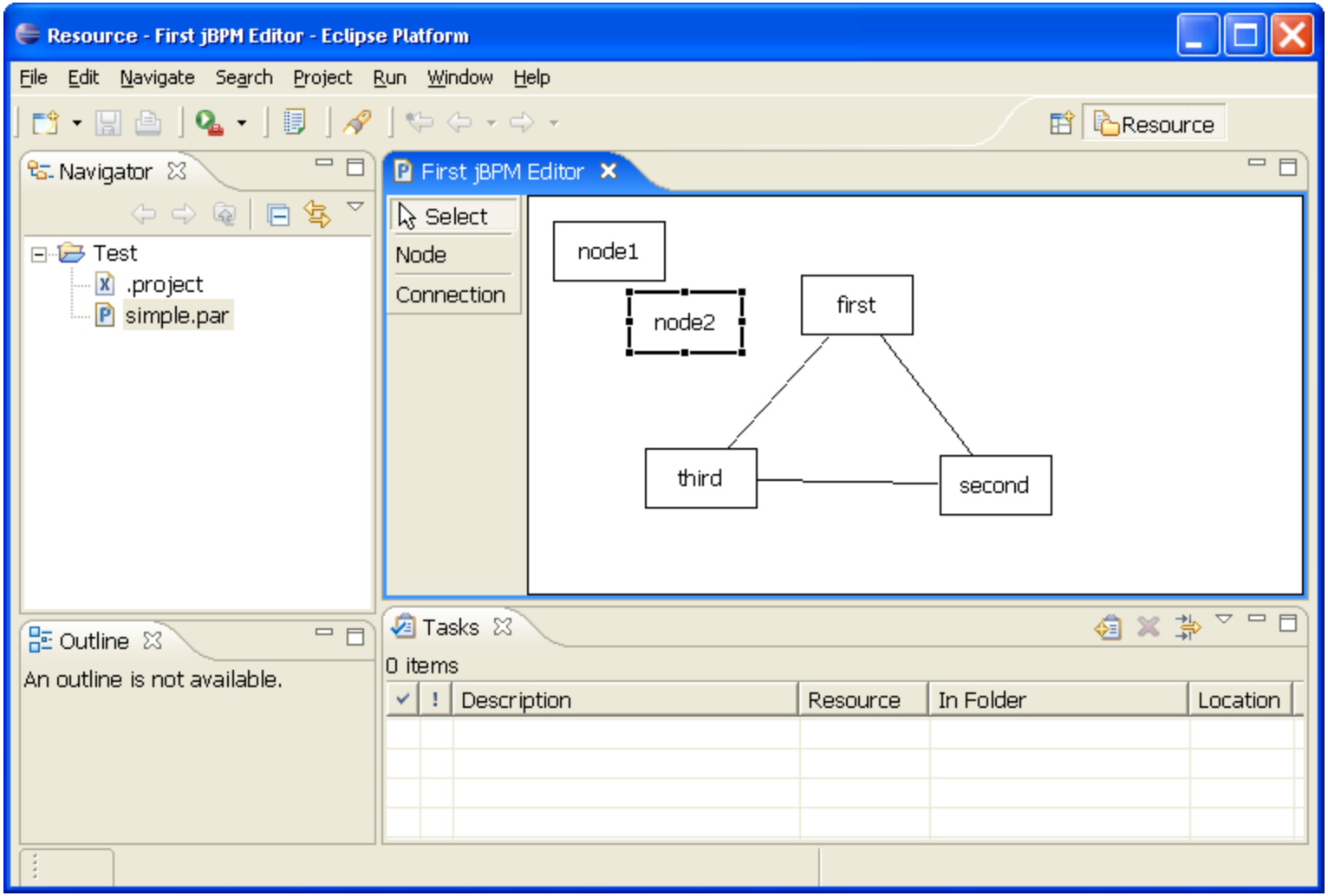
- Implement getCreateCommand
  - Only handle requests to create Nodes

```
public class GraphXYLayoutEditPolicy
extends XYLayoutEditPolicy {
  …
  protected Command getCreateCommand(CreateRequest request) {
    if (request.getNewObjectType().equals(Node.class)) {
      NodeCreateCommand result = new NodeCreateCommand();
      result.setLocation(request.getLocation());
      result.setNode((Node)request.getNewObject());
      result.setParent((Graph)getHost().getModel());
      return result;
    }
    return null;
  }
```

Graph Revisited Once More

```
public class Graph extends Observable {
  …
  public void addNode(Node node) {
    getNodes().add(node);
    setChanged();
    notifyObservers();
  }
  public void removeNode(Node node) {
    getNodes().remove(node);
    setChanged();
    notifyObservers();
  }
  …
}
```

Editor With Some Extra Nodes



Creating New Connections

- Define ConnectionCreateCommand
- Define NodeGraphicalNodeEditPolicy
  - Implement getConnectionCreateCommand
  - Implement getConnectionCompleteCommand
- Make connection changes observable
- Install NodeGraphicalNodeEditPolicy
- Make our NodeEditPart implement the org.eclipse.gef.NodeEditPart interface
  - Obtaining the ConnectionAnchor
- Observe the connection changes of the model
- Get rid of the ContentProvider

Define ConnectionCreateCommand

```
public class ConnectionCreateCommand extends Command {
  private Node source;
  private Node target;
  private Connection connection
  public void setSource(Node source) {
    this.source = source;
  }
  public void setTarget(Node target) {
    this.target = target;
  }
  public void setConnection(Connection connection) {
    this.connection = connection;
  }
  …
}
```

ConnectionCreateCommand (ctn'd)

```
public class ConnectionCreateCommand extends Command {
  …
  public boolean canExecute() {
    return source != null && target != null;
  }
  public void execute() {
    connection.setSource(source);
    connection.setTarget(target);
  }
  public void undo() {
    connection.setSource(null);
    connection.setTarget(null);
  }
}
```

Define NodeGraphicalNodeEditPolicy

```
public class NodeGraphicalNodeEditPolicy
extends GraphicalNodeEditPolicy {
  protected Command getConnectionCreateCommand(
      CreateConnectionRequest request) {
    ConnectionCreateCommand result =
      new ConnectionCreateCommand();
    result.setSource((Node)getHost().getModel());
    result.setConnection((Connection)request.getNewObject());
    request.setStartCommand(result);
    return result;
  }
  …
}
```

Define NodeGraphicalNodeEditPolicy (ctn'd)

```
public class NodeGraphicalNodeEditPolicy
extends GraphicalNodeEditPolicy {
  …
  protected Command getConnectionCompleteCommand(
      CreateConnectionRequest request) {
    ConnectionCreateCommand result =
      (ConnectionCreateCommand)request.getStartCommand();
    result.setTarget((Node)getHost().getModel());
    return
  }
    //Stubs for the remaining methods
}
```

Node Revisited Again

```
public class Node extends Observable {
  …
  public void addSourceConnection(Connection connection) {
    getSourceConnections().add(connection);
    notifyObservers();
  }
  public void addTargetConnection(Connection connection) {
    getTargetConnections().add(connection);
    notifyObservers();
  }
  …
}
```

Node Revisited Again (ctn'd)

```
public class Node extends Observable {
  …
  public void removeSourceConnection(Connection connection) {
    getSourceConnections().remove(connection);
    notifyObservers();
  }
  public void removeTargetConnection(Connection connection) {
    getTargetConnections().remove(connection);
    notifyObservers();
  }
}
```

NodeEditPart Revisited Again

- Installation of the NodeGraphicalNodeEditPolicy

```
public class NodeEditPart
extends AbstractGraphicalEditPart
implements Observer {
  …
  protected void createEditPolicies() {
    installEditPolicy(
      EditPolicy.GRAPHICAL_NODE_ROLE,
      new NodeGraphicalNodeEditPolicy());
  }
}
```

NodeEditPart Revisited Again (ctn'd)

- Observing connection model changes

```
public class NodeEditPart
extends AbstractGraphicalEditPart
implements Observer {
  …
  public void update(Observable observable, Object message) {
    refreshVisuals();
    refreshSourceConnections();
    refreshTargetConnections();
  }
}
```

NodeEditPart Revisited Again (ctn'd)

```
public class NodeEditPart extends AbstractGraphicalEditPart
implements Observer, org.eclipse.gef.NodeEditPart {
  …
  public ConnectionAnchor getSourceConnectionAnchor(
      ConnectionEditPart connection) {
    return ((NodeFigure)getFigure()).getConnectionAnchor();
  }
  public ConnectionAnchor getTargetConnectionAnchor(
      ConnectionEditPart connection) {
    return ((NodeFigure)getFigure()).getConnectionAnchor();
  }
  …
}
```

NodeEditPart Revisited Again (ctn'd)

```
public class NodeEditPart extends AbstractGraphicalEditPart
implements Observer, org.eclipse.gef.NodeEditPart {
  …
  public ConnectionAnchor getSourceConnectionAnchor(
      Request request) {
    return ((NodeFigure)getFigure()).getConnectionAnchor();
  }
  public ConnectionAnchor getTargetConnectionAnchor(
      Request request) {
    return ((NodeFigure)getFigure()).getConnectionAnchor();
  }
}
```

NodeFigure Revisited

- Create the connection anchor lazily

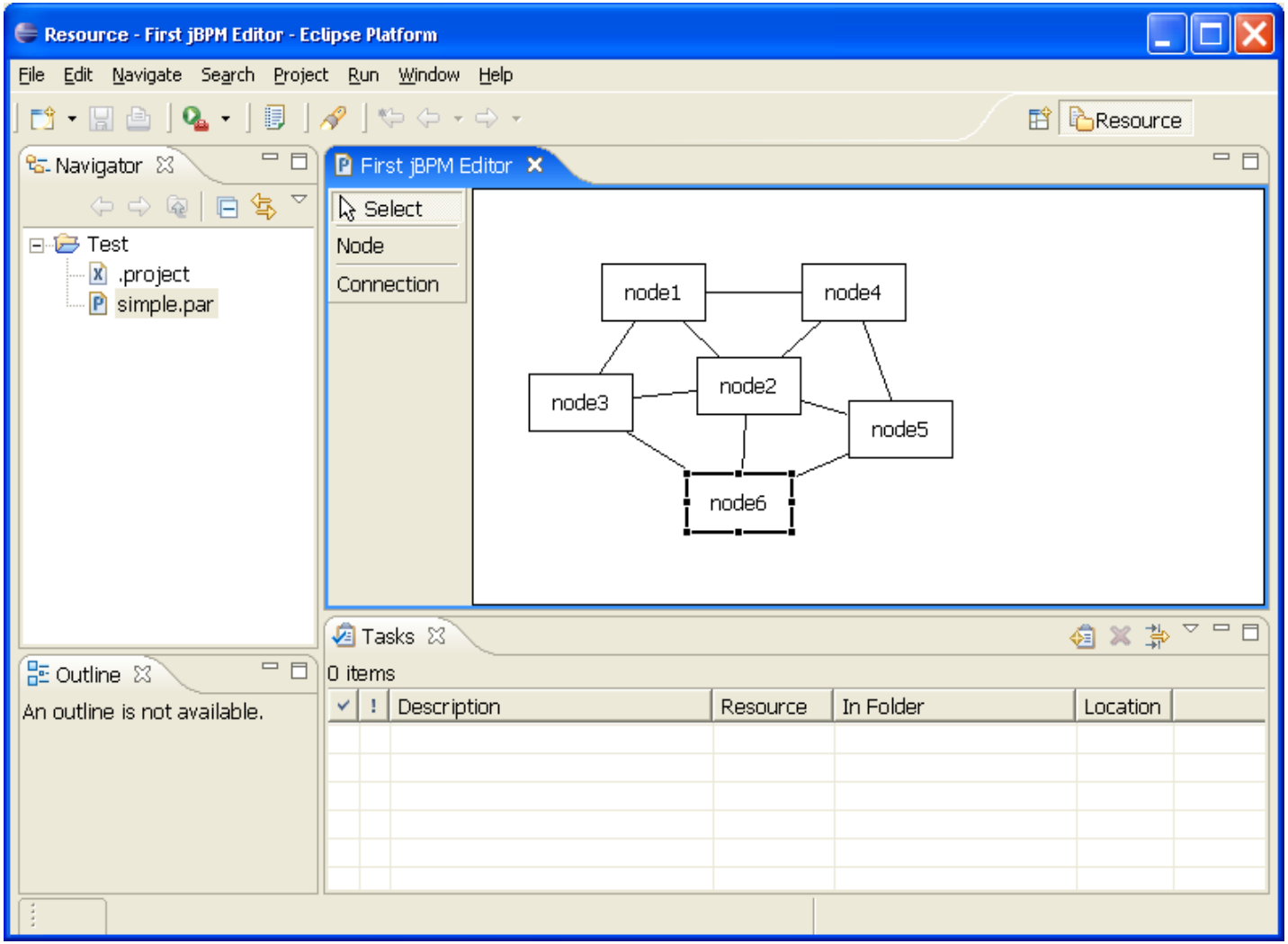
```
public class NodeFigure extends Figure {
  private ConnectionAnchor connectionAnchor;
  …
  public ConnectionAnchor getConnectionAnchor() {
    if (connectionAnchor == null) {
      connectionAnchor = new ChopboxAnchor(this);
    }
    return connectionAnchor;
  }
}
```

Eliminate ContentProvider

- Delete the ContentProvider class
- Modify the createGraphViewer method in class GraphEditor

```
public class GraphEditor extends EditorPart {
  …
  private void createGraphViewer(Composite parent) {
    ScrollingGraphicalViewer viewer =
      new ScrollingGraphicalViewer();
    viewer.setRootEditPart(new ScalableFreeformRootEditPart());
    viewer.createControl(parent);
    viewer.getControl().setBackground(ColorConstants.white);
    viewer.setEditPartFactory(new GraphicalEditPartFactory());
    viewer.setContents(new Graph());
    editDomain.addViewer(viewer);
  }
}
```

GraphEditor With Connection Support



Delete Support

- Include DeleteAction in ActionRegistry
- Selection Support :
  - SelectionProvider
  - SelectionListener
- Update ActionBarContributor
- Deleting Nodes :
  - NodeDeleteCommand
  - NodeComponentEditPolicy
- Deleting Connections :
  - ConnectionDeleteCommand
  - ConnectionEditPolicy
  - ConnectionEndpointEditPolicy

ActionRegistry Revisited

```
public class GraphEditor extends EditorPart {
  private ActionRegistry actionRegistry;
  …
  private void initActionRegistry() {
    actionRegistry = new ActionRegistry();
    actionRegistry.registerAction(new UndoAction(this));
    actionRegistry.registerAction(new RedoAction(this));
    actionRegistry.registerAction(
      new DeleteAction((WorkbenchPart)this));
  }
  …
}
```

GraphViewer Is SelectionProvider

```
public class GraphEditor extends EditorPart {
  …
  private void createGraphViewer(Composite parent) {
    ScrollingGraphicalViewer viewer =
      new ScrollingGraphicalViewer();
    viewer.setRootEditPart(new ScalableFreeformRootEditPart());
    viewer.createControl(parent);
    viewer.getControl().setBackground(ColorConstants.white);
    viewer.setEditPartFactory(new GraphicalEditPartFactory());
    viewer.setContents(new Graph());
    editDomain.addViewer(viewer);
    getSite().setSelectionProvider(viewer);
  }
}
```

Define the SelectionListener

```
public class GraphEditorListener
implements CommandStackListener, ISelectionListener
  …
  public void commandStackChanged(EventObject event) {
    updateActions();
  }
  public void selectionChanged(
      IWorkbenchPart part, ISelection selection) {
    updateActions();
  }
  …
}
```

Define the SelectionListener (ctn'd)

```
public class GraphEditorListener
implements CommandStackListener, ISelectionListener
  …
  private void updateActions() {
    Iterator iterator = actionRegistry.getActions();
    while (iterator.hasNext()) {
      Object action = iterator.next();
      if (action instanceof UpdateAction)
        ((UpdateAction)action).update();
    }
  }
}
```

Add the SelectionListener

```
public class GraphEditor extends EditorPart {
  …
  private void initGraphEditorListener() {
    GraphEditorListener graphEditorListener =
      new GraphEditorListener(actionRegistry));
    ISelectionService selectionService =
      getSite().getWorkbenchWindow().getSelectionService();
    editDomain.getCommandStack().addCommandStackListener(
      graphEditorListener);
    selectionService.addSelectionListener(
      graphEditorListener);
  }
}
```

Update ActionbarContributor

```
public class ActionBarContributor
extends EditorActionBarContributor {
  public void setActiveEditor(IEditorPart targetEditor) {
    …
    String deleteId = ActionFactory.DELETE.getId();
    actionBars.setGlobalActionHandler(
      deleteId, actionRegistry.getAction(deleteId));
    actionBars.updateActionBars();
  }
}
```

Define NodeDeleteCommand

```
public class NodeDeleteCommand extends Command {
  private Node node;
  private Graph graph;
  private List connections;
  private Map connectionSources, connectionTargets;
  public void setNode(Node node) { this.node = node; }
  public void setGraph(Graph graph) { this.graph = graph; }
  public void execute() {
    detachConnections();
    graph.removeNode(node);
  }
  public void undo() {
    graph.addNode(node);
    reattachConnections();
  }
  …
}
```

Define NodeDeleteCommand (ctn'd)

```
public class NodeDeleteCommand extends Command {
  …
  private void detachConnections() {
    connections = new ArrayList();
    connectionSources = new HashMap();
    connectionTargets = new HashMap();
    connections.addAll(node.getSourceConnections());
    connections.addAll(node.getTargetConnections());
    for (int i = 0; i < connections.size(); i++) {
      Connection connection = (Connection)connections.get(i);
      connectionSources.put(connection,connection.getSource());
      connectionTargets.put(connection,connection.getTarget());
      connection.setSource(null);
      connection.setTarget(null);
    }
  }
  …
```

Define NodeDeleteCommand (ctn'd)

```
public class NodeDeleteCommand extends Command {
  …
  private void reattachConnections() {
    for (int i = 0; i < connections.size(); i++) {
      Connection connection = (Connection)connections.get(i);
      connection.setSource(
        (Node)connectionSources.get(connection));
      connection.setTarget(
        (Node)connectionTargets.get(connection));
    }
  }
}
```

Define NodeComponentEditPolicy

- Override the createDeleteCommand method

```
public class NodeComponentEditPolicy
extends ComponentEditPolicy {
  protected Command createDeleteCommand(GroupRequest request) {
    NodeDeleteCommand deleteCommand = new NodeDeleteCommand();
    deleteCommand.setGraph(
      (Graph)getHost().getParent().getModel());
    deleteCommand.setNode((Node)getHost().getModel());
    return deleteCommand;
  }
}
```

Install NodeComponentEditPolicy

- Responsibility of NodeEditPart

```
public class NodeEditPart extends … {
  …
  protected void createEditPolicies() {
    …
    installEditPolicy(
      EditPolicy.COMPONENT_ROLE,
      new NodeComponentEditPolicy());
  }
}
```

Selecting Connections

- Install ConnectionEndpointEditPolicy
  - Primary SelectionEditPolicy for showing focus on connections
  - All ConnectionEditParts need one

```
public class ConnectionEditPart
extends AbstractConnectionEditPart {
  …
  protected void createEditPolicies() {
    installEditPolicy(
      EditPolicy.CONNECTION_ENDPOINTS_ROLE,
      new ConnectionEndpointEditPolicy());
  }
  …
}
```

Define ConnectionDeleteCommand

```
public class ConnectionDeleteCommand extends Command {
  private Node source, target;
  private Connection connection
  public void setConnection(Connection
    this.connection = connection;
  }
  public void execute() {
    if (source == null) source = connection.getSource();
    if (target == null) target = connection.getTarget();
    connection.setSource(null);
    connection.setTarget(null);
  }
  public void undo() {
    connection.setSource(source);
    connection.setTarget(target);
  }
}
```

Define ConnectionEditPolicy

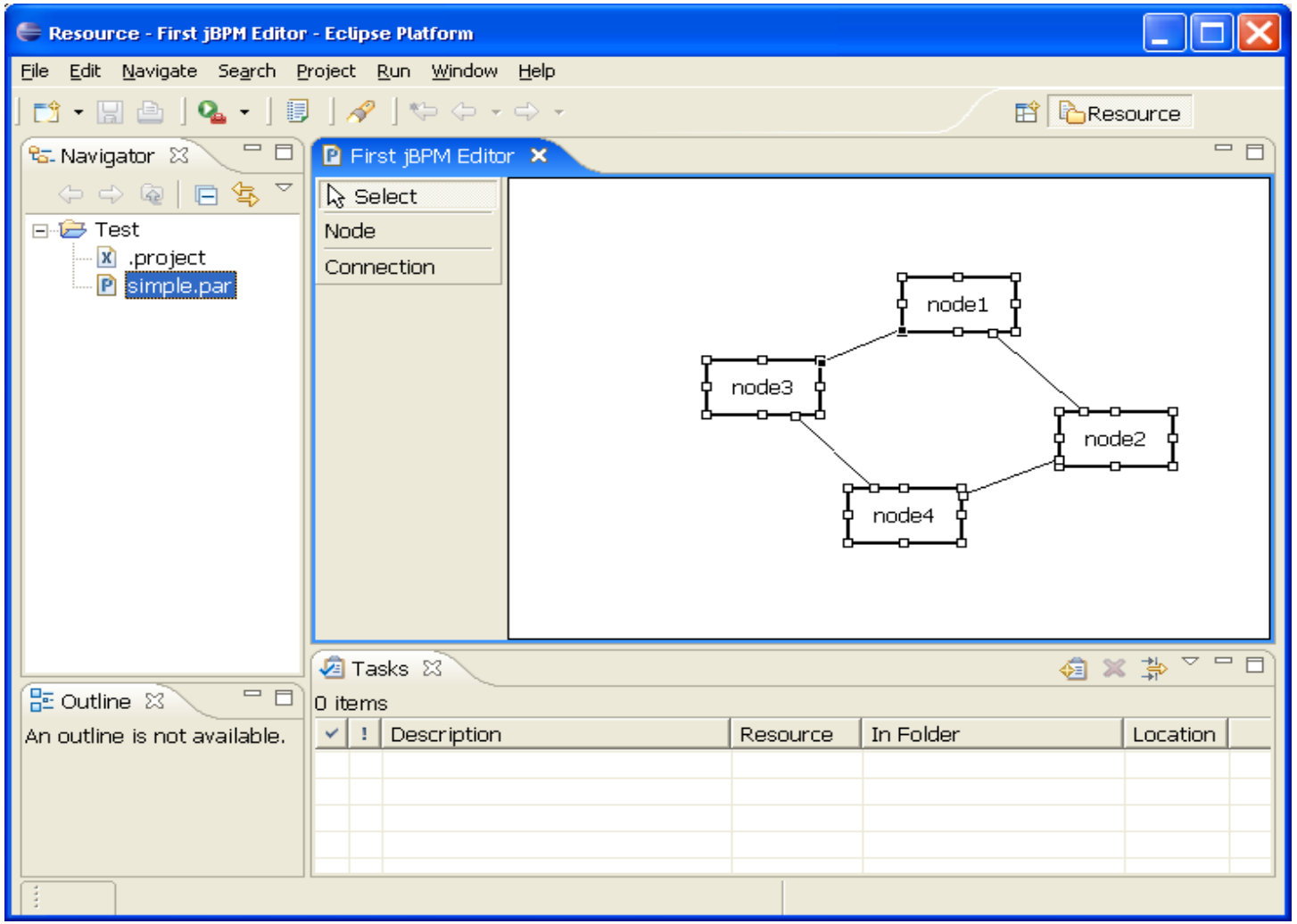
- Default model-based EditPolicy for Connections
- Only knows about the model and its basic operations
  - Single default operation : DELETE

```
public class ConnectionEditPolicy
extends org.eclipse.gef.editpolicies.ConnectionEditPolicy {
  protected Command getDeleteCommand(GroupRequest request) {
    ConnectionDeleteCommand result =
      new ConnectionDeleteCommand();
    result.setConnection((Connection)getHost().getModel());
    return result;
  }
}
```

Install ConnectionEditPolicy

```
public class ConnectionEditPart
extends AbstractConnectionEditPart {
  …
  protected void createEditPolicies() {
    installEditPolicy(
      EditPolicy.CONNECTION_ROLE,
      new ConnectionEditPolicy());
    installEditPolicy(
      EditPolicy.CONNECTION_ENDPOINTS_ROLE,
      new ConnectionEndpointEditPolicy());
  }
  …
}
```

Basic Functional Graph Editor



What's Next?

- Saving and loading the model
  - By serialization or with XML representation
- Provide an outline view
- Support for a grid, zooming, guides, …
- Make actions available through context menu
- Provide an extension point to plug-in custom node types

Final Thoughts

- Steep and long learning curve
  - Starting from scratch is not easy
  - No books available
  - Starting small is mandatory to fully understand
- Very rich framework
  - Lots of predefined functionality
  - Do very complex things with almost no code
- Use code and javadocs to find details
  - Overwhelming for newbie

Questions?