GEF - The Graphical Editing Framework

Presented by Arieh Bibliowicz
www.vainolo.com - @vainolo

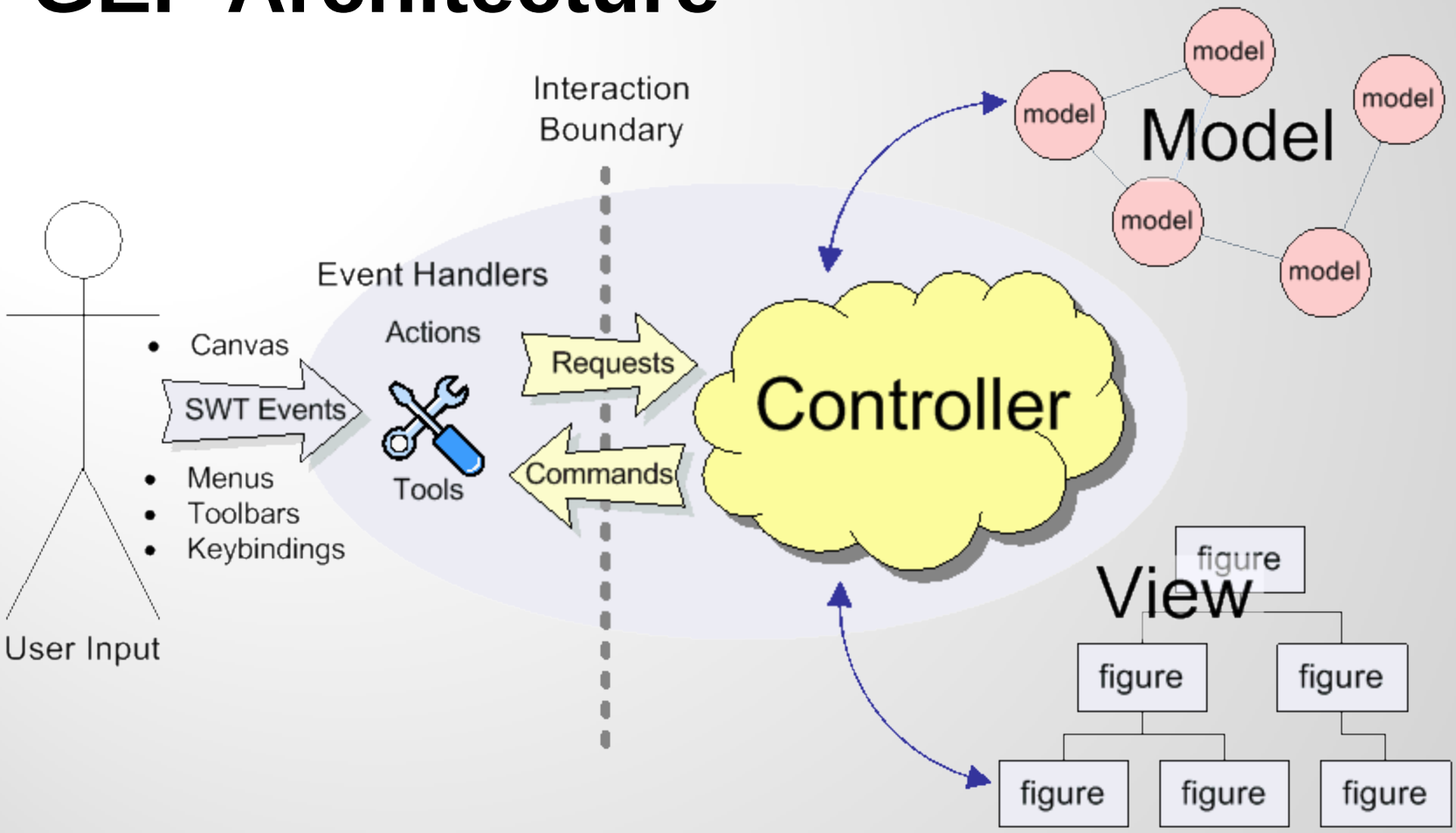
What is GEF

- GEF - The Eclipse Graphical Editing Framework, is a set of eclipse plug-ins that help you create graphical editors for the eclipse platform
- Based on MVC pattern
- Very complex... but worth the effort

Requirements

- Eclipse IDE for Java Developers (Juno SR1): http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/junosr1
- GEF SDK, Eclipse SDK and Eclipse Plug-in Devevelopment Environment

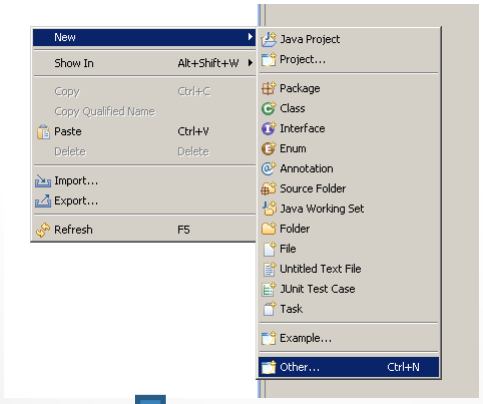
GEF Architecture

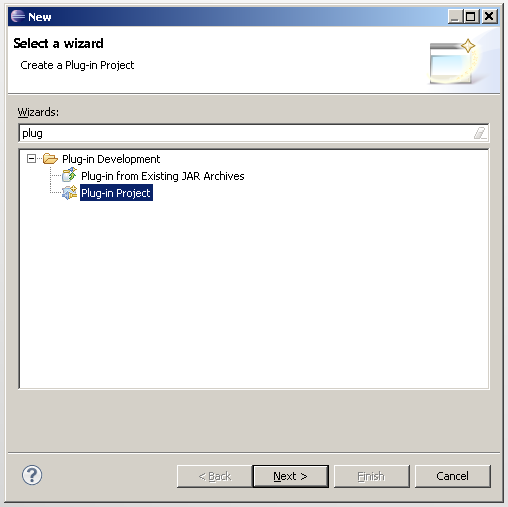


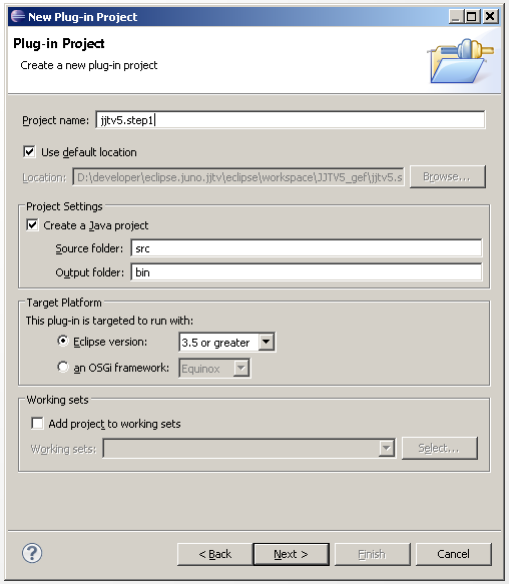
Step 1 - "Working" editor

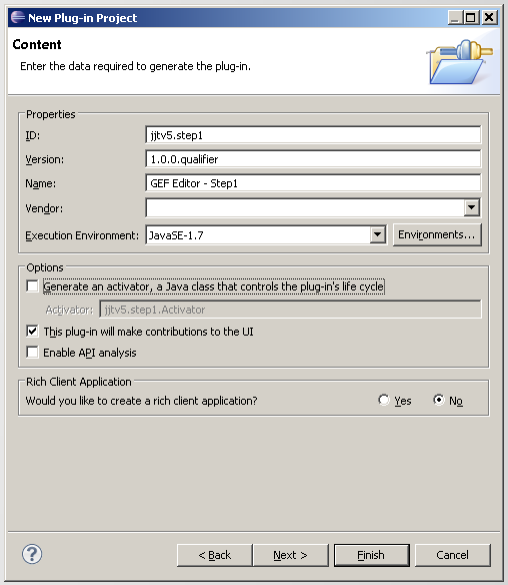
works - but doesn't do much

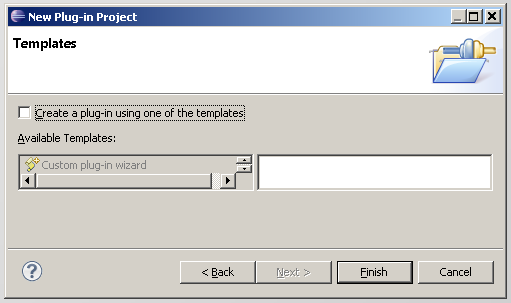
Creating an Eclipse Plug-in Project



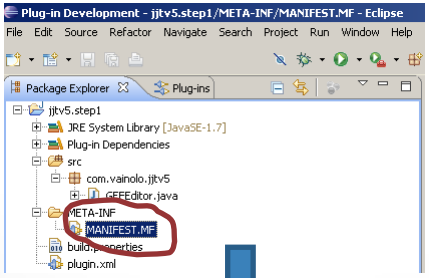


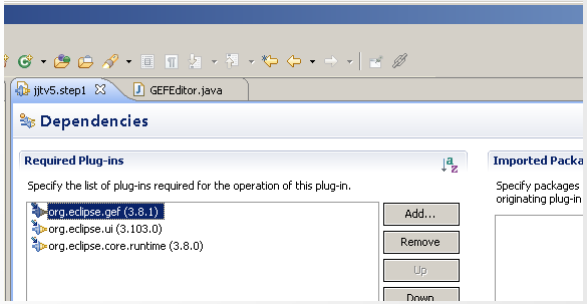


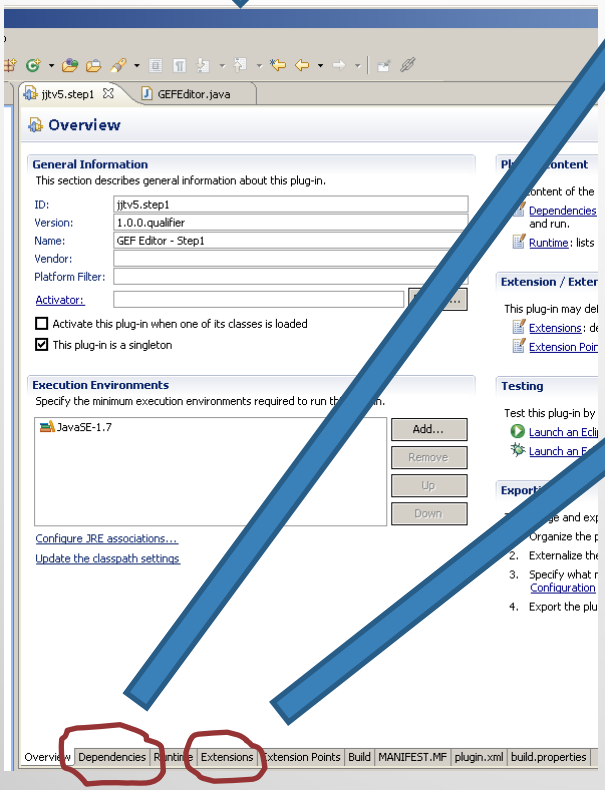


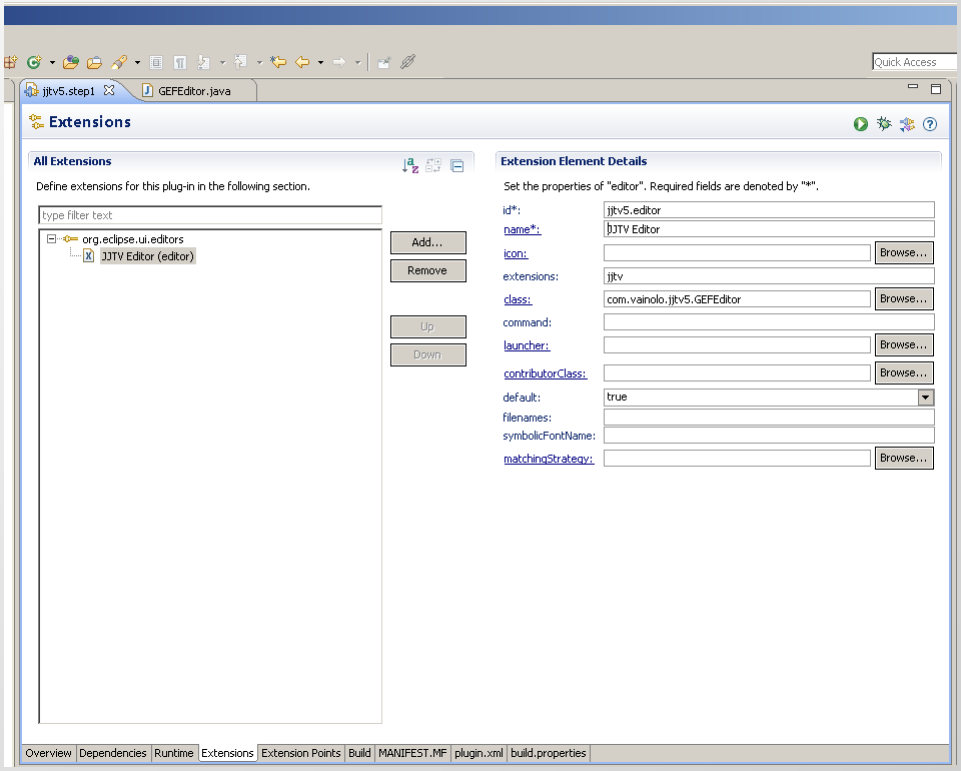


Configuring the Plug-in







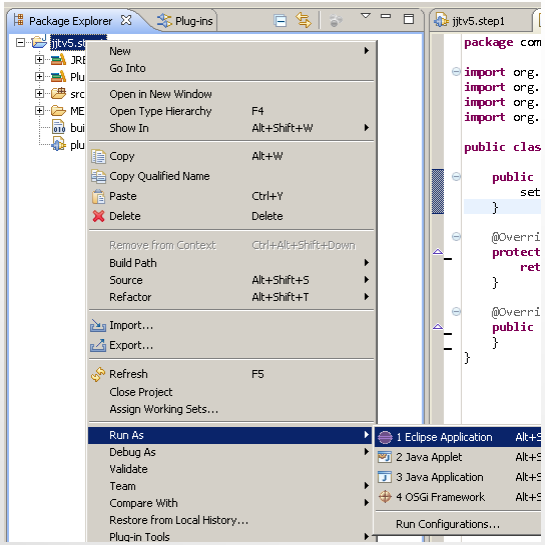


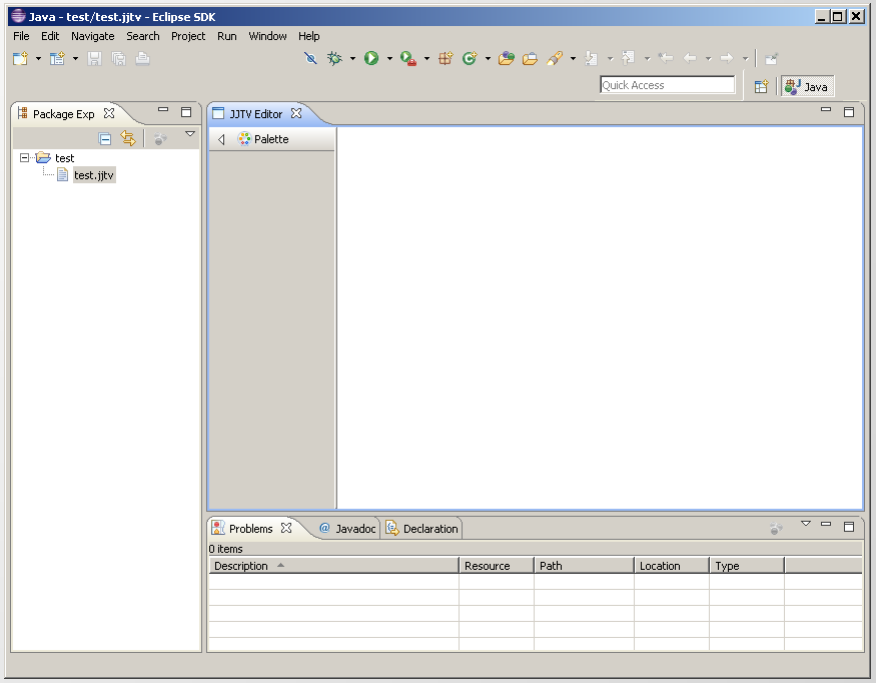
Implement the Editor Class

- Extend one of GEF's editor classes. Usually, `GraphicalEditorWithFlyoutPalette`, implementing it's abstract functions and setting the `EditDomain` (which stores the "collective" state of the GEF application

```
public class GEFEditor extends GraphicalEditorWithFlyoutPalette {
  public GEFEditor() {
    setEditDomain(new DefaultEditDomain(this));
  }
  @Override
  protected PaletteRoot getPaletteRoot() {
    return null;
  }
  @Override
  public void doSave(IProgressMonitor monitor) {
  }
}
```

Let’s Check What We Have





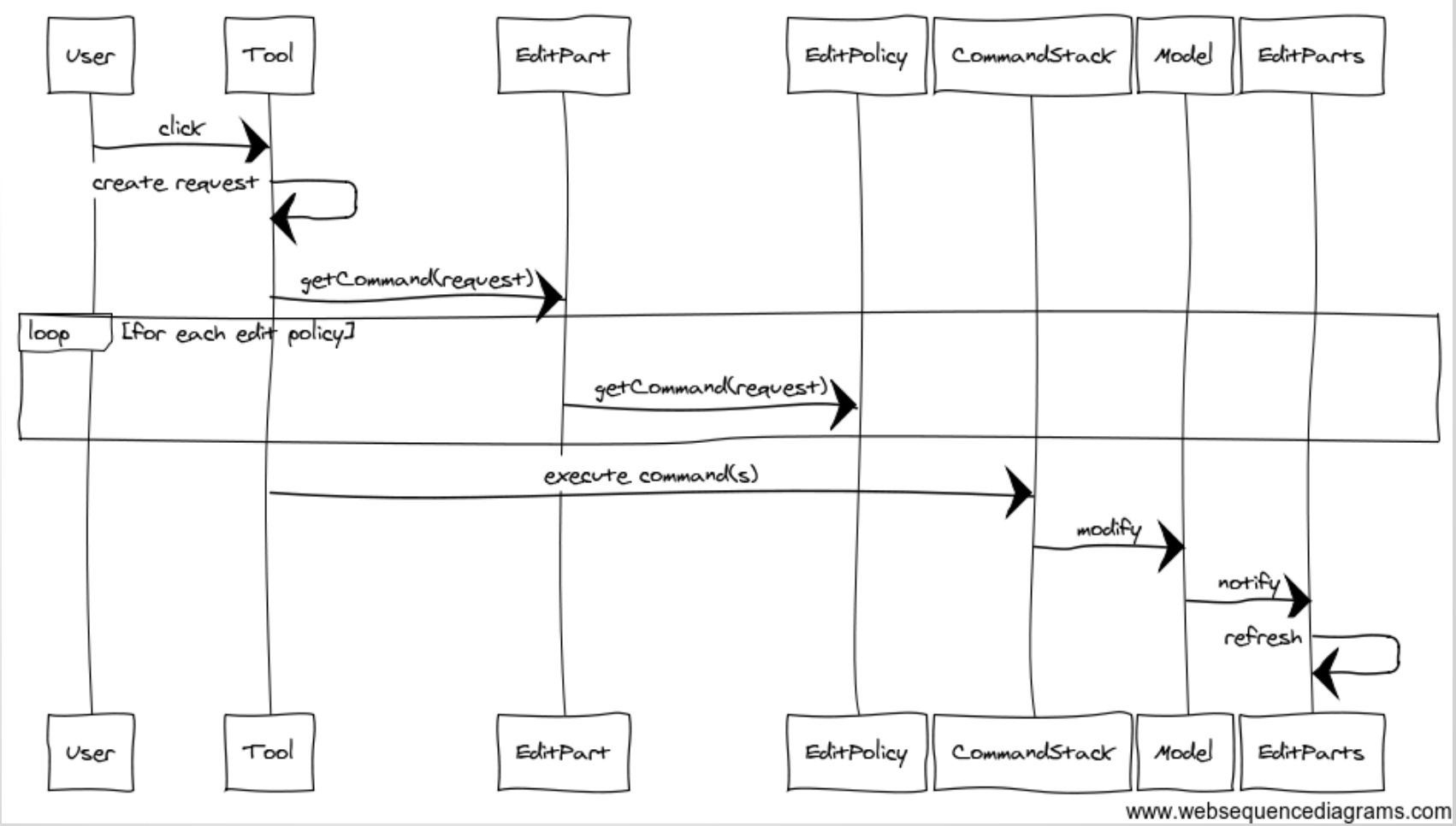
Step 2 – Adding Nodes

our first GEF baby steps

Adding New Elements

- Figures in the diagram are created by tools (there are other options… but not today ☺)
- Tools are located in the palette of the editor. Each tool fires different request that can be interpreted by the editor. Useful built-in tools:
  - Creation tool – creating new edit parts.
  - Connection creation tool – creating new connections between existing edit parts.
- When a tool fires a request it is sent to all `EditPart`s for interpretation. This is done by helper `EditPolicy` classes
- If an `EditPolicy` wants to handle a request, it creates a `Command`, which modifies the model
- The `Command` is applied to the `CommandStack` and the model is changed
- The `EditPart`s must observe model changes. When a change is observed, the `EditPart`s refresh their visual representation

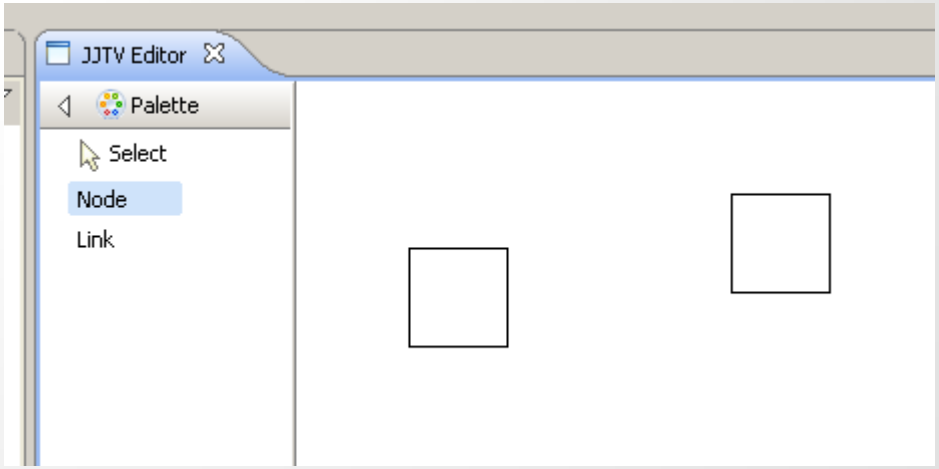
Let's Visualize This



And Now in Code

- A sample model consisting of a `Canvas`, `Node`s and `Link`s. All of them `Observable`
- Factories for each model entity that is created by the `CreationTool`
- `EditPart`s for each of the model entities, and an `EditPartFactory` that creates new `EditPart`s when new model entities are created by the `CreationTool`
- An `XYLayoutPolicy` which handles the `CreateRequest` sent by the `CreationTool`
- `Command` to update the model
- The `CanvasEditPart` returns it's model children (`Node`s), and the framework creates an `EditPart` for each `Node` returned
- `EditPart`s create `Figure`s to display in the diagram

So Now What Do We Have?



Step 3 – Adding Connections

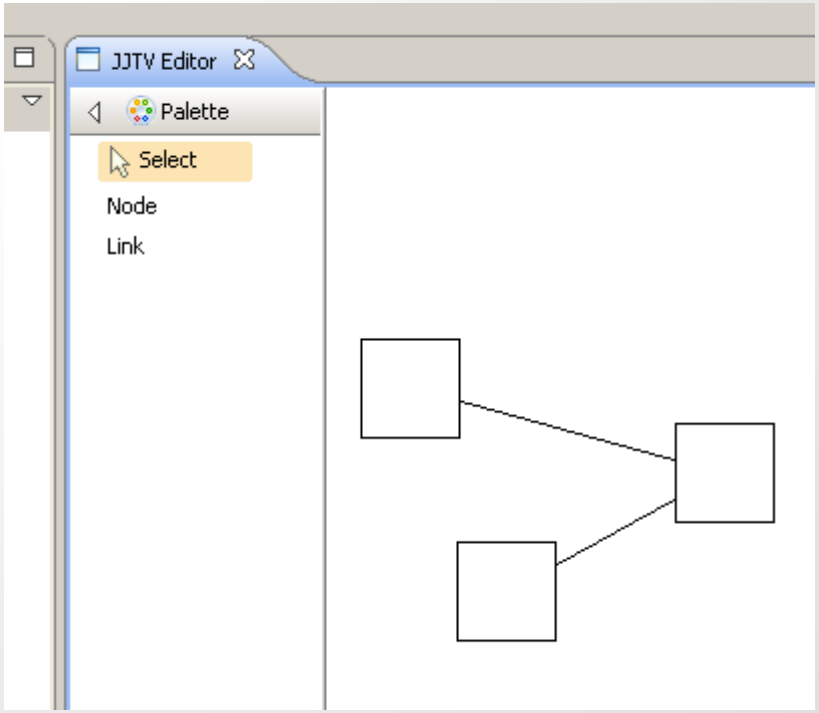
Connections Overview

- Similarly to the way an `EditPart` returns model children, it can also return model connections
- A model connection must connect between two model entities that are represented visually
- GEF creates a `ConnectionEditPart` for each connection model object that is returned (using a provided factory) and attaches them to the nodes using `ConnectionAnchors`. The anchors are provided by source and target edit parts.
- Connections are created in two steps – source selection and then target selection

And Now in Code

- Had to rename `NodeEditPart` to `MyNodeEditPart` (GEF has it's own `NodeEditPart` interface that we now have to use). It was also extended to Observe the model (to detect when new links are added) and to provide Anchors for the connections
- `GraphicalNodeEditPolicy` that handles connection creation
- `CreateLinkCommand` to update the model
- Created `LinkEditPart` to manage links and updated `EditPartFactory` to create `LinkEditPart`

Show Me the Connections



Step 4 – Moving Things Around

Moving and Resizing

- These operations are handled by the `XYLayoutPolicy`, in the `createChangeConstraintCommand` method
- The command is executed, the model changes and the view is updated (Magic!)
- When a node is moved, the connections are moved with it (no need to handle this. More magic)
- GEF deals with all of the details of adding handles to the figures that can be resized, making them selectable, etc.. (A LOT of magic)

Small Changes to the Code

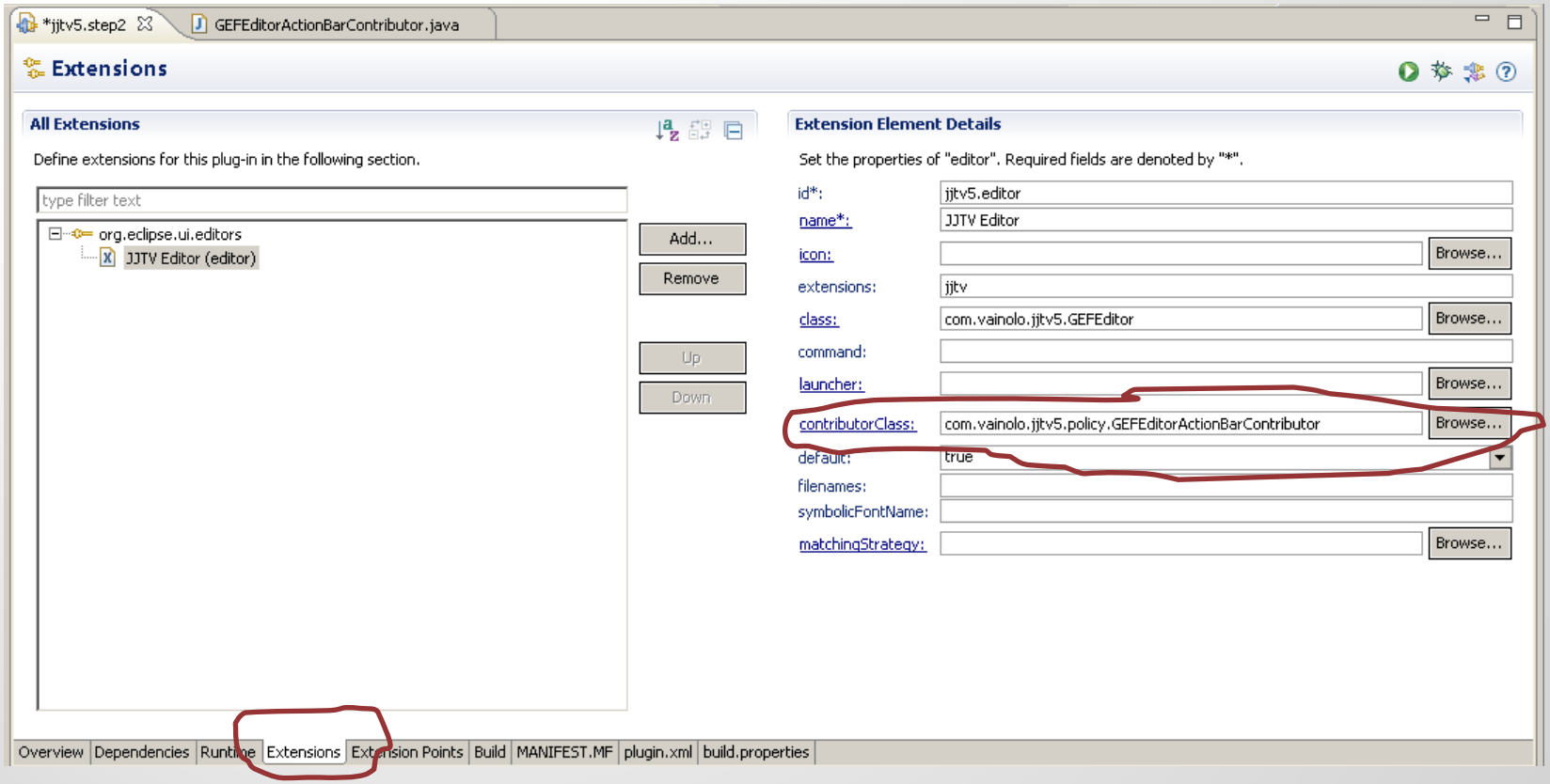
- Override the `createChangeConstraintCommand` in the `CanvasXYLayoutEditPolicy`, creating a command that updates the Nodes
- `ChangeNodeConstraintsCommand` to update the nodes

Step 5 –Undo, Redo and Delete

Undo and Redo

- Remember – all commands are stored in the command stack
- Undo information is stored in the command themselves. Undoing is just like executing, but backwards. Redo is executing the command again. Delete is a special request that must be handled by the edit part. Done in a `ComponentEditPolicy` for nodes and in a `ConnectionEditPolicy` for links.
- Undo and Redo are GEF tool operations, so their treatment is different. They are caused by user clicks/key presses, captured by the workbench and forwarded to the editor
  - But we must tell the workbench how to do this

Contributing to the Toolbar



MANIFEST.MF

What Happens in the Code

- GEFEditorActionBarContributor class that makes the actions available in the editor and adds them to the toolbar
- Override `undo` method in all commands (and changes to model to support this)
- `ComponentEditPolicy` to for node delete request, `ConnectionEditPolicy` for link delete request, `DeleteNodeCommand` to delete nodes and `DeleteLinkCommand` to delete links
  - Note: when deleting a node we manually take care to delete all incoming/outgoing links. GEF does magic, but up to a point…
- Installed the edit policies in the node and link `EditPart`s

Conclusion

- GEF is a very powerful framework to create graphical editors that run on the eclipse platform
- High learning curve, but lots of magic behind the scenes
  - Snap to grid/geometry
  - Guidelines to place elements in the diagram
  - Multi level hierarchy of edit parts
  - Layouts
- Resources:
  - http://www.eclipse.org/gef
  - http://wiki.eclipse.org/GEF/Articles,_Tutorials,_Slides
- Tutorial code:
  - https://github.com/vainolo/JJTV5_gef

Thank You