Diagram editors in the web with Eclipse GLSP
(Graphical Language Server Platform)

Philip Langer
planger@eclipsesource.com

Maximilian Koegel
mkoegel@eclipsesource.com
Building domain-specific web-based (modeling) tools
Eclipse Graphical Language Server Platform (GLSP)

Applying the architectural pattern of LSP to graphical modeling

- Enable development of web-based diagram clients
  - Or clients in any technology
  - Decouple client implementation from modeling language implementation

- Encapsulate language know-how on the server
  - Reuse of existing frameworks & diagram implementation
  - Management of large models

- Front-end focused on rendering & user interaction
  - Everything else is obtained from the server
  - With the minimum amount of roundtrips
Separation of Concerns with GLSP

Diagram Rendering
Editing Tools
Visual Feedback
Editing Rules
Commands
Edit Transactions
Live Validation
Model Management

Modeling Backend
Separation of Concerns with GLSP

Modeling Backend

Diagram Rendering
Editing Tools
Visual Feedback
Editing Rules
Commands
Edit Transactions
Live Validation
Model Management

Client

Server
Separation of Concerns with GLSP

Modeling Backend

Diagram Rendering
Editing Tools
Visual Feedback
Editing Rules
Commands
Edit Transactions
Live Validation
Model Management

Client

Protocol

Server
Eclipse Graphical Language Server Platform (GLSP)

1. Java-based server framework
   - Standalone server implementation

2. Graphical Language Server Protocol
   - Language config, executing operations, ...

3. Web-based Client framework
   - Diagram editing, visual feedback, server communication, ...

4. IDE / Tool platform integration
   - Theia, VSCode, and Eclipse

   - Based on ...
     - Eclipse LSP4J
     - Eclipse Sprotty
       - Rendering & Sprotty’s client-server protocol for model transfer
Initialization and Rendering with Eclipse Sprotty

**Client**

- open diagram
- Start and initialize graphical language server (ws, URI, …)
- render diagram

**Server**

- translate to “graph” into SVG

**Set Graph Model (Sprotty Model)**

```
{ "graph": { "type": "graph", "id": "graph", "children": [ { "id": "node1", "type": "nodeClass", "expanded": true, "layout": "plain", "children": [ { "layout": "Nbox", "type": "compHeader", "id": "modelHeader", "children": [ { "id": "modelHeaderIcon", "type": "icon", "layout": "stack", "layoutOptions": { "resizeContainer": false, "HAlign": "center" } ] } ] } ] } }
```
Client-Server Interaction: Initialization and Editing Tools

- **Open diagram**
- **Client**: Start and initialize graphical language server
  - Sprotty base protocol to diagram model, bounds, etc.
- **Server**
- **Render diagram**, ...
- **Request Tool Palette Actions**
- **Set Tool Palette Actions** (e.g. TriggerNodeCreation for typeId)
- **Render palette update**
- **Retrieve from ToolPaletteItemProvider**
Client-Server Interaction: Model Manipulation

trigger node creation → enable node creation tool → trigger node creation

Client

Execute Create Node (typeid, location, ...)

Update Model

Server

render diagram update

client manipulates model

Arbitrary model source

Modify model source

Regenerate graph model on change
Client-Server Interaction: Type Hints

**Client**
- Client sends a request to retrieve type hints.
- Server sends back the type hints.
- Client interprets the type hints.
- Client triggers edge creation.
- Client declines creating edge.
- Client triggers edge creation.
- Client interprets type hints.
- Client renders diagram update.

**Server**
- Server retrieves type hints.
- Server handles manipulates model.

### Request Type Hints
- NodeTypeHint
  - elementTypeId: string
  - repositionable: boolean
  - deletable: boolean
  - resizable: boolean
  - containableElementTypeIds: string[]

### Set Type Hints
- EDGE_TYPE_HINT
  - sourceElementTypeIds: string[]
  - targetElementTypeIds: string[]
  - routable: boolean

### Execute Create Edge Operation (typeId, source, target)
- Update Model
Let’s Look Into the Code

- Minimal example available

**Client-side Rendering in Sprotty**

```java
const minimalDiagramModule = new ContainerModule({bind, unbind, tsBound, rebind}) => {
    rebind(TYPES.ILogger).to(ConsoleLogger).inSingletonScope();
    rebind(TYPES.LogLevel).toConstantValue(LogLevel.warn);
    const context = {bind, unbind, tsBound, rebind};
    configureModelElement(context, 'graph', GLSPGraph, SGraphView);
    configureModelElement(context, 'node', RectangularNode, RectangularNodeView);
};
```

**Server Module Configuration**

```java
public class MinimalGLSPModule extends DefaultGLSPModule {
    @Override
    protected void configureDiagramConfigurations(final MultiBindConfig<DiagramConfiguration> config) {
        config.add(MinimalDiagramConfiguration.class);
    }

    @Override
    protected void configureOperationHandlers(final MultiBindConfig<OperationHandler> config) {
        super.configureOperationHandlers(config);
        config.add(MinimalCreateNodeOperationHandler.class);
    }

    @Override
    protected Class<? extends ModelFactory> bindModelFactory() {
        return JsonFileModelFactory.class;
    }
}
```
Tool Platform Integrations of GLSP

- A good modeling tool is not just a diagram editor!
  - Seamless integration with an tool / application platform of your choice
  - Flexibly combined with other capabilities of the tool
Tool Platform Integrations of GLSP

● Beyond just showing the diagram editor in a tool!
  ○ Styling, commands, menus, keyboard shortcuts, other views, ...
  ○ Navigation across editors (diagram and non-diagram)
  ○ Problem markers
  ○ Copy & Paste

● Status
  ○ Full support in Eclipse Theia
  ○ VSCode: basic integration, but growing functionality
  ○ Eclipse RCP: basic integration, further features on demand
Other Cool Stuff

Commercial IEC FBD Editor by logi.cals

Ecore Diagrams in Theia
Oct 22nd, 14:00 by Jonas

Debugging via DAP
by Hansjörg Eder
Conclusion

● Eclipse Sprotty: flexible & modern rendering

● GLSP unlocks very high reuse on the server
  ○ Migrating existing diagram editors
  ○ Hooking up EMF models

● Integratable with tool platforms and domain-specific tools
  ○ Integration with Theia, VSCode, or Eclipse
  ○ Avoiding lock-in effect with specific tool / app platform

● Get in contact with us to tell us about your use cases!
  ○ planger@eclipsesource.com
  ○ mkoegel@eclipsesource.com
Evaluate the Sessions

Sign in and vote at Eclipsecon.org: