Building Web-based Diagram Editors

Towards a Graphical Language Server Protocol for Diagrams?

Philip Langer
EclipseSource

Tobias Ortmayr
EclipseSource
Building domain-specific (modeling) tools for various domains

Software and systems engineering
- Hardware producers,
- Mobile networks,
- Firmware, ...

Modeling tools development

Information Systems
- Insurances, Accounting,
- Information management,
- ...
Web-based diagram editors

1. Deployment and integration
   - Browser is all that is needed
   - Installing tool vs opening a link

2. Modern UI technology
   - SWT vs HTML5
   - GEF 3 vs SVG
   - CSS3 Styling

Integration of diagram editors anywhere
- Avoid the smell of an IDE
- More people in the modeling process
- Modern-looking diagrams
- Flexibility in diagram style
- Visual feedback and animations

- Frameworks?
JointJS

SVG-based Diagram Framework in JavaScript

Pros
- Decently implemented MVC architecture
- Nice editing support
- Feature-rich: many shapes, edge routing strategies, 2D function library

Cons
- Not really community-driven open-source framework
- Everything is on the client
Why a client-server architecture is important to us

- One model != one diagram
  - Large models
  - One diagram only shows a part of it
  - Other parts are edited in other diagrams, forms, etc.
  - Prevent loading entire model into browser

- Whole-model understanding required for editing
  - User feedback before/after editing operations
  - Update on outside-of-diagram model changes
  - Live-validation may access outside-of-diagram parts

- Modeling language “smarts” is Java-based
  - Avoid having to re-implement those in JS
  - But re-use them in browser-based implementation

→ Essentially the same problem that’s addressed with LSP
Eclipse Sprotty

SVG-based Diagram Framework in TypeScript

Pros

- TypeScript
  - Not plain JavaScript

- Great and extensible architecture
  - Slim abstractions
  - DI-based configuration

- Truly open-source software
  - Open development
  - Open to contributions
  - Recently became an Eclipse project

- Integration with Xtext / Language Server Protocol
Sprotty & LSP = Client-server Protocol

- Sprotty’s Language Server Extension
  - Visualize models owned by the Language Server

- Sprotty Client-server Protocol
  - Sprotty messages are tunneled through LSP
  - C → S: Request Model
  - S → C: Set/Update Model
  - Bounds
  - Collapse state
  - Selection
  - Pop-ups

- Server manages whole model
- Client doesn’t need to know entire model
- Protocol is front-end oriented (just as LSP)
So what’s missing?

- **Sprotty server implementation**
  - Depends on (textual) language server
  - Textual model is master
  - Graphical model is slave
  
  → Independent diagram server API and impl

- **Client-server protocol**
  - Viewing and navigation capabilities only
  
  → Editing capabilities

- **Client**
  - UI for visualization purposes
  
  → Support and UI for editing
Applying the architectural pattern of LSP to graphical modeling

- Based on Sprotty
  - Client implementation
  - Sprotty’s client-server base protocol

1. Java-based server framework
   - Standalone server implementation
   - Independent from any text language server

2. Extension of Sprotty’s protocol for editing

3. Client framework
   - Server connector (“model source”) decoupled from LSP
   - UI for editing support hooking up the protocol for editing
Server Framework

Framework for building specific diagram servers

- Server infrastructure
  - JSON-RPC communication: Sending and receiving action messages
  - Client and model state management
  - Model manipulation infrastructure

- Extensible DI module
  - Action registry
    - Predefined actions
    - Optionally custom actions
  - Action handler registry
    - Generic action handlers available
    - Custom handlers can be configured
Towards a Graphical Language Server Protocol

- Transferring, updating, and navigating the diagram
  - Already exists in Sprotty
  - Re-used as a base protocol

- Goals
  - Encapsulate modeling language “smarts” on the server
  - Minimize client-server round-trips (esp. on UI interactions)

- Extensions for editing support
  - Available editing operations
  - Request execution of operation
  - Graphical move and resize
  - Drag and drop hints

- Extensions for additional features
  - Execute server action
  - Problem markers

[github.com/eclipsesource/GraphicalServerProtocol]
Towards a Graphical Language Server Protocol

Client

Request Available Operations

- Sprotty base protocol to obtain model, bounds, etc.

Set Available Operations (operations)

- Execute Create Node Operation (elemTypeId, location, ...)
- Update Model

Server

- Operation Kind
  - CreateNode, CreateEdge, Delete, Move, Generic
- Operation
  - id : string
  - label : string
  - operationKind : OperationKind
  - elementType? : string

- render palette update
- invoke node creation
- render diagram update
- invoke node creation
- render diagram update
- retrieve
- manipulate model
- manipulate model
Towards a Graphical Language Server Protocol

Avoiding server-roundtrip on direct user interaction!
Towards a Graphical Language Server Protocol for Diagrams

**Client**

- initiate move
- interpret move hints
- decline move
- initiate move
- interpret move hints

**Server**

- Request Move Hints
- Set Move Hints (hints)
- retrieve DragAndDropHint
- manipulate model

**Process**

1. **initiate move**
2. **interpret move hints**
3. **decline move**
4. **initiate move**
5. **interpret move hints**
6. **Execute Move Operation (elementId, targetContainerId, ...)**
7. **Update Model**
8. **render diagram update**

**Messages**

- **Request Move Hints**
- **Set Move Hints**
- **DragAndDropHint**
  - dragElementClass: string
  - dropElementClasses[]): string
Towards a Graphical Language Server Protocol

1. **Client**
   - initiate move
   - interpret move hints
   - decline move
   - initiate move
   - interpret move hints
   - execute move operation (elementId, targetContainerId, ...)
   - render problem marker

2. **Server**
   - request move hints
   - set move hints (hints)
   - retrieve dragAndDropHint
     - dragElementClass: string
     - dropElementClasses[]: string
   - identify violation
   - update problem marker
Towards a Graphical Language Server Protocol

Client

Server

Request Move Hints

Set Move Hints (hints)

Execute Move Operation (elementId, targetContainerId, ...)

Update model (rejecting operation)

render diagram update

interpret move hints

initiate move

interpret move hints

initiate move

decline move

DragAndDropHint

dragElementClass: string
dropElementClasses[]: string

identify violation

retrieve
Client Framework

- Client-side GLSP Diagram Server implementation
  - Connects to a stand-alone GLSP server instance
  - Handles dispatching events locally or to the server

- Sprotty extensions to enable editing capabilities
  - Palette that enables editing tools (for now very simple)
  - Editing tools, e.g. for adding nodes, edges, etc.
  - Persisting diagram changes

- Editing command handlers
  - Hooking up client-side editing commands with the server
  - Sending and receiving the respective protocol messages
Demo
Current State and Outlook

● So far
  ○ Focus on the server framework & protocol
  ○ Client extensions only as much as necessary

● Next steps
  ○ Enhance generic editing capabilities in the client
  ○ Palette, visual feedback, support for drag and drop hints, etc.
  ○ Problem markers, property views based on JsonForms, etc.

● Collaboration and contribution
  ○ Enhancement of Sprotty with TypeFox
  ○ Client-server protocol definition with TypeFox and Obeo
  ○ Hopefully with you too?

[Links]

- [github.com/eclipsesource/graphical-lsp](https://github.com/eclipsesource/graphical-lsp)
More on related topics at EclipseCon

- **Earlier today -- catch them later on Youtube**
  - If, when and how? - Strategies towards web-based tooling
  - Lucky in the Cloud With Diagrams

- **Tomorrow**
  - Building a Web-IDE based on Eclipse Theia for Smart Home (11:55 Bürgersaal 2)
  - EMF, JSON and I (14:45 Theater Stage)
  - Domain-Specific Languages in the Cloud – With Eclipse Technologies (16:30)

- **Oct 25th**
  - JSON Forms 2.0 (10:45 Theater Stage)
  - Building Web-based Modeling Tools based on Eclipse Theia (11:30 Theater Stage)
Evaluate the Sessions

Sign in and vote at eclipsecon.org

-1  0  +1