Diagrams in web and space with Eclipse GLSP
(Graphical Language Server Platform)

Philip Langer
planger@eclipsesource.com

Martin Fleck
mfleck@eclipsesource.com
Building domain-specific cloud-based (modeling) tools
Client-server architecture for diagram editors

- Re-use Eclipse Modeling Technologies
  → Modeling language “smarts” runs on a Java-based server

- Migration of existing diagram editors
  → Only rendering & edit UI depends on client

- One diagram != one model
  → Whole-model understanding required for editing

Essentially the same problem
that’s addressed with LSP
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

Eclipse Graphical Language Server Platform (GLSP)

github.com/eclipsesource/graphical-lsp
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
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
   - Connection and interaction to a graphical language server
   - Integrated with Theia (usable stand-alone too)
   - UI for editing support, hooking up the protocol for editing

- Based on ...
  - Eclipse LSP4J
  - Eclipse Sprotty
    - Client implementation for diagram rendering
    - Sprotty’s client-server protocol as a base for model transfer

github.com/eclipsesource/graphical-lsp
Initialization and Rendering with Eclipse Sprotty

open diagram

Client

Start and initialize graphical language server (ws, URI, ...)

Server

load model(s)

translate to "graph"

load

set Graph Model (Sprotty Model)

render diagram

translate "graph" into SVG

Graphical Views for Web-Based Modeling Tools With Theia and Sprotty
Thursday, 10:15 - 10:50, Theater Stage
Client-Server Interaction: Initialization and Editing Tools

Open diagram

Start and initialize graphical language server

Sprotty base protocol to diagram model, bounds, etc.

Request Available Operations

Set Available Operations (operations)

Operation

- id : string
- label : string
- operationKind : OperationKind
- elementType? : string

OperationKind

- CreateNode
- CreateEdge
- Delete
- ChangeBounds
- Move
- Generic

retrieve

render palette update

render diagram, ...
Client-Server Interaction: Operation Execution

Client

- render palette update
- invoke node creation
- render diagram update

Request Available Operations
Set Available Operations (operations)

Execute Create Node Operation (typId, location, ...)
Update Model

Server

- retrieve

Operation
- id : string
- label : string
- operationKind : OperationKind
- elementType? : string

OperationKind
- CreateNode, CreateEdge, Delete, ChangeBounds, Move, Generic

Demo

Client-Server Interaction: Operation Execution

© 2019 EclipseSource | http://eclipsesource.com | Philip Langer & Martin Fleck | Diagrams in web and space with GLSP
Server Framework

```java
public class CreateActivityNodeHandler extends CreateNodeOperationHandler {
    @Override
    protected GNode createNode(Optional<GPoint> point, GraphicalModelState modelState) {
        String nodeType = ModelTypes.toNodeType(ModelTypes.MERGE_NODE);
        return new ActivityNodeBuilder(ModelTypes.MERGE_NODE, nodeType) //
               .position(point.orElse(null)) //
               .build();
    }
}
```

Create Node: 
- `clientld, type, location`

Action Handler: 
- Action Dispatcher
- API
- Client & Model State Mgt
- JSON-RPC Infrastructure

Update Model: 
- Server
Server Framework

- JSON-RPC Infrastructure
- Client & Model State Mgt
- Action Dispatcher
- API
- Update Model
- Create Node (clientId, type, location)

Semantic Model

- Represents

Notation Model
Client-Server Interaction: Undo and redo, delete, ...
Client-Server Interaction: Type Hints

Avoiding server-roundtrip on direct user interaction!
Client-Server Interaction: Type Hints

Client

- initiate creating edge
  - interpret type hints
  - decline creating edge
- initiate creating edge
  - interpret type hints

Server

Set Type Hints

- retrieve
  - NodeTypeInfo
    - elementTypeId: string
    - repositionable: boolean
    - deletable: boolean
    - resizable: boolean
    - containableElementTypeIds: string[]
  - EdgeTypeInfo
    - sourceElementTypeIds: string[]
    - targetElementTypeIds: string[]
    - routable: boolean

Request Type Hints

Execute Create Edge Operation (typeid, source, target)

Update Model

- manipulate model
- render diagram update
- initiate creating edge
  - interpret type hints
Client-Server Interaction: Type Hints

Request TypeHints → Set TypeHints

- Client
  - initiate creating edge
  - interpret type hints
  - decline creating edge

- Server
  - retrieve
  - identify violation

Execute Create Edge Operation (typeld, source, target)

- Client
  - render problem marker

- Server
  - interpret type hints
  - initiate creating edge

NodeTypeHint:
- elementTypeIds: string
- repositionable: boolean
- deletable: boolean
- resizable: boolean
- containableElementTypeIds: string[]

EdgeTypeHint:
- sourceElementTypeIds: string[]
- targetElementTypeIds: string[]
- routable: boolean
Client-Server Interaction: Type Hints

**Request Type Hints**

**Set Type Hints**

**Execute Create Edge Operation** (typeld, source, target)

**Update model** (rejecting operation)

**Render diagram update**

**Initiate creating edge**

**Interpret type hints**

**Decline creating edge**

**Node Type Hint**
- elementTypeId: string
- repositionable: boolean
- deletable: boolean
- resizable: boolean
- containableElementTypeIds: string[]

**Edge Type Hint**
- sourceElementTypeIds: string[]
- targetElementTypeIds: string[]
- routable: boolean

**Retrieve**

**Demo**
Client Framework

- Rendering of graph models
  - Pure Sprotty → all of Sprotty’s rendering flexibility

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

- Theia integration of the GLSP editor (also Eclipse RCP integration available)

- Sprotty extensions to enable editing capabilities
  - Editing tools, tools management, and palette
  - Visual feedback for tools
  - Dirty state handling
  - Command palette for keyboard-oriented actions
  - Direct editing of labels, ...
What else can we do with GLSP? Reuse and Flexibility!

- Integrate with existing model implementations
  - Just let server load and manipulate them
  - The client doesn't need to know

GLSP Server

Load and manipulate a graph model that the client directly understands.

GLSP Client

Arbitrary model source

Modify model source

Generate graph model on change

GLSP Server

JSON Forms

Other Client

Demo

GLSP Client
What else can we do with GLSP? Reuse and Flexibility!

- Integrate with existing editor implementations
  - Reuse a great deal of your work (e.g. model management, editing behavior, ...)

The evolution of Papyrus, an open SysML and UML tool
Tuesday, 14:30 - 15:05, Seminarraum 5
What else can we do with GLSP? Reuse and Flexibility!

- Integrate with existing editor implementations
  - Reuse a great deal of your work (e.g. model management, editing behavior, ...)

An experience report on migrating an industrial-grade IDE to the Cloud
Wednesday, 10:15 - 10:50, Bürgersaal 2
What else can we do with GLSP? Reuse and Flexibility!

- Explore completely different clients
  - Use an existing server and have a client implementation on any device!

...such as an augmented reality device

Property editors in space (AR)
Thursday, 13:00 - 13:35, Theater Stage
Conclusion and Outlook

● Conclusion
  ○ Sprotty-based client allows for developing editors with great flexibility
  ○ GLSP allows for very high reuse when migrating existing diagram editors
  ○ Separation of concerns → more future proof and flexible
  ○ Young technology, but ready for being adopted for several use cases

● Outlook
  ○ Property view support based on JsonForms
  ○ Further editing capabilities on the client
  ○ Identify best balance of powerfulness / simplicity of edit hints

● Get in contact with us to tell us about your use cases!
  ○ planger@eclipsesource.com
  ○ mfleck@eclipsesource.com

eclipse.org/glsp

github.com/eclipsesource/ graphical-lsp