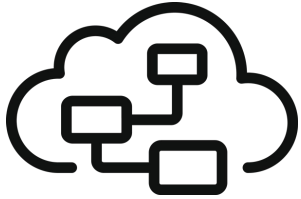




Diagram editors in the web with Eclipse GLSP

(Graphical Language Server Platform)



Philip Langer

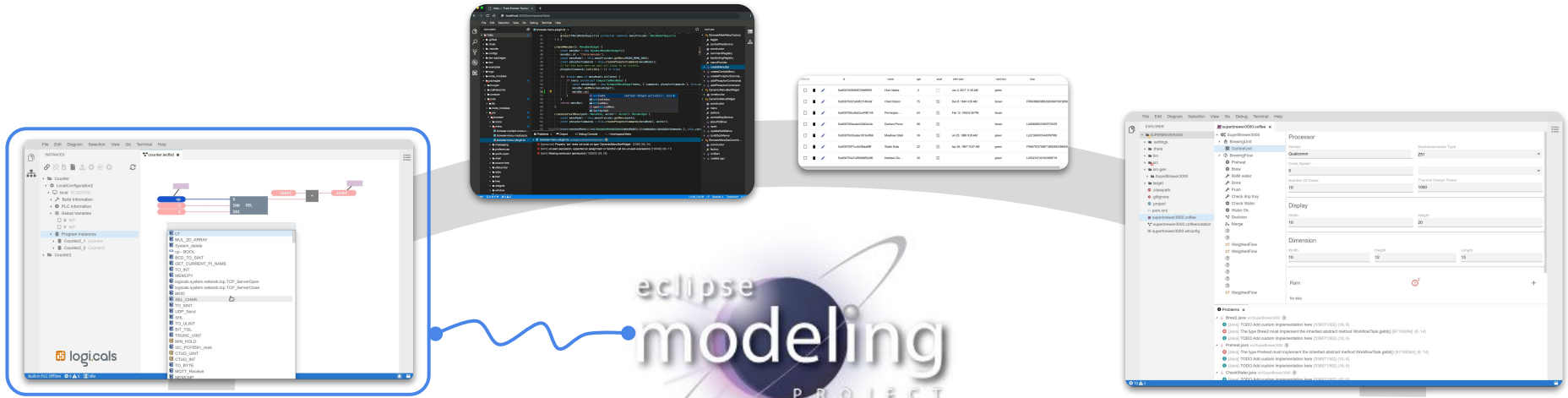
planger@eclipsesource.com

Maximilian Koegel

mkoegel@eclipsesource.com



Building domain-specific web-based (modeling) tools



eclipse
modeling
PROJECT



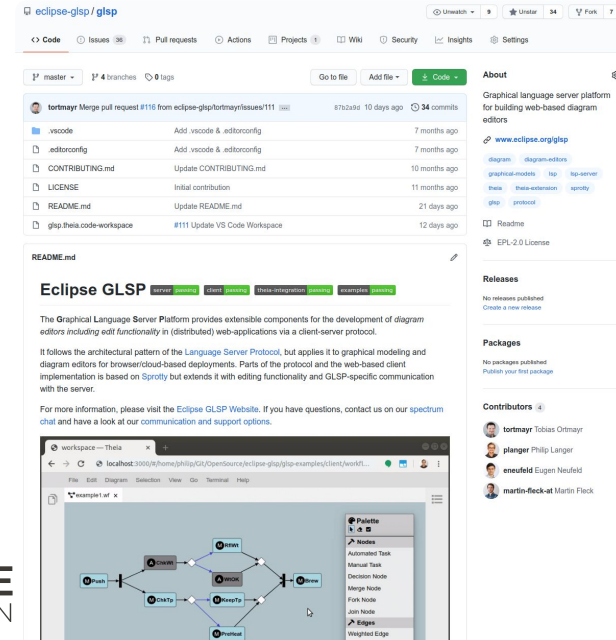
THEIA



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



github.com/eclipse-glsp/glsp

Separation of Concerns with GLSP

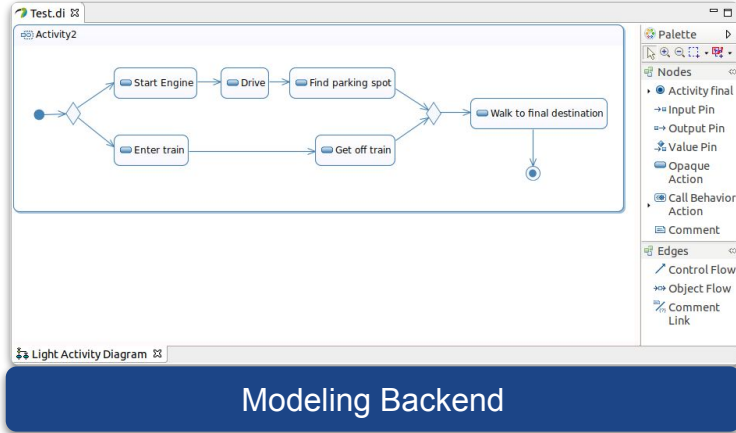


Diagram Rendering

Editing Tools

Visual Feedback

Editing Rules

Commands

Edit Transactions

Live Validation

Model Management

Separation of Concerns with GLSP

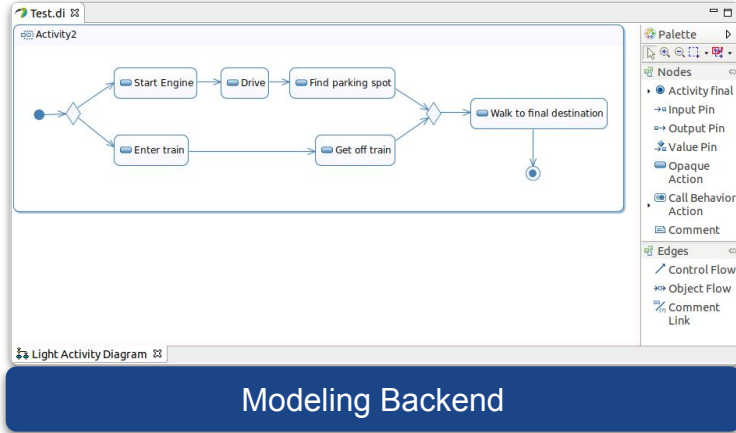


Diagram Rendering

Editing Tools

Visual Feedback

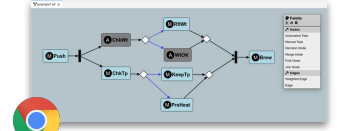
Editing Rules

Commands

Edit Transactions

Live Validation

Model Management

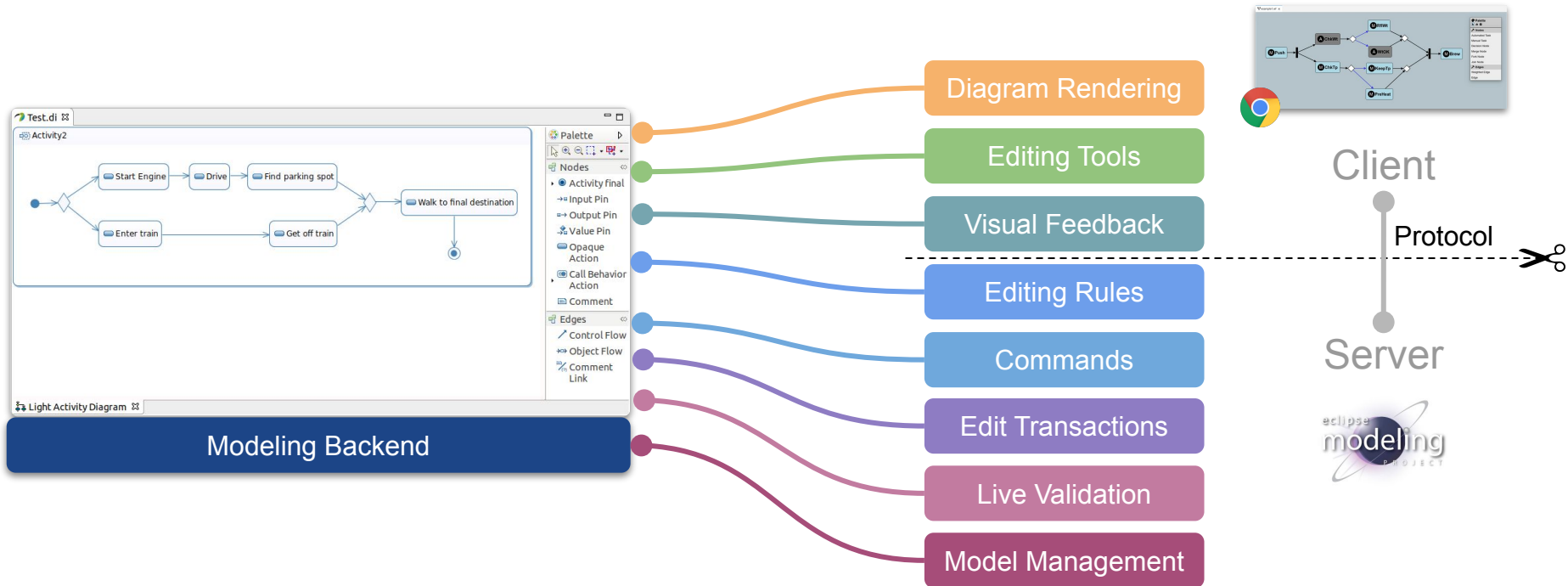


Client

Server

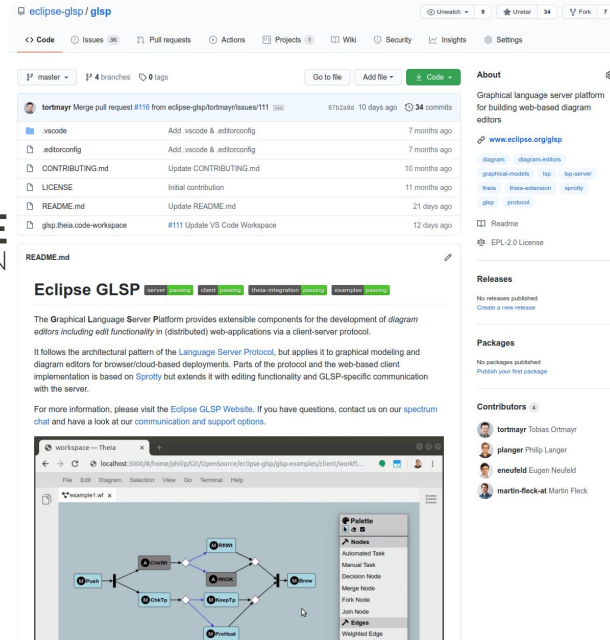


Separation of Concerns with GLSP



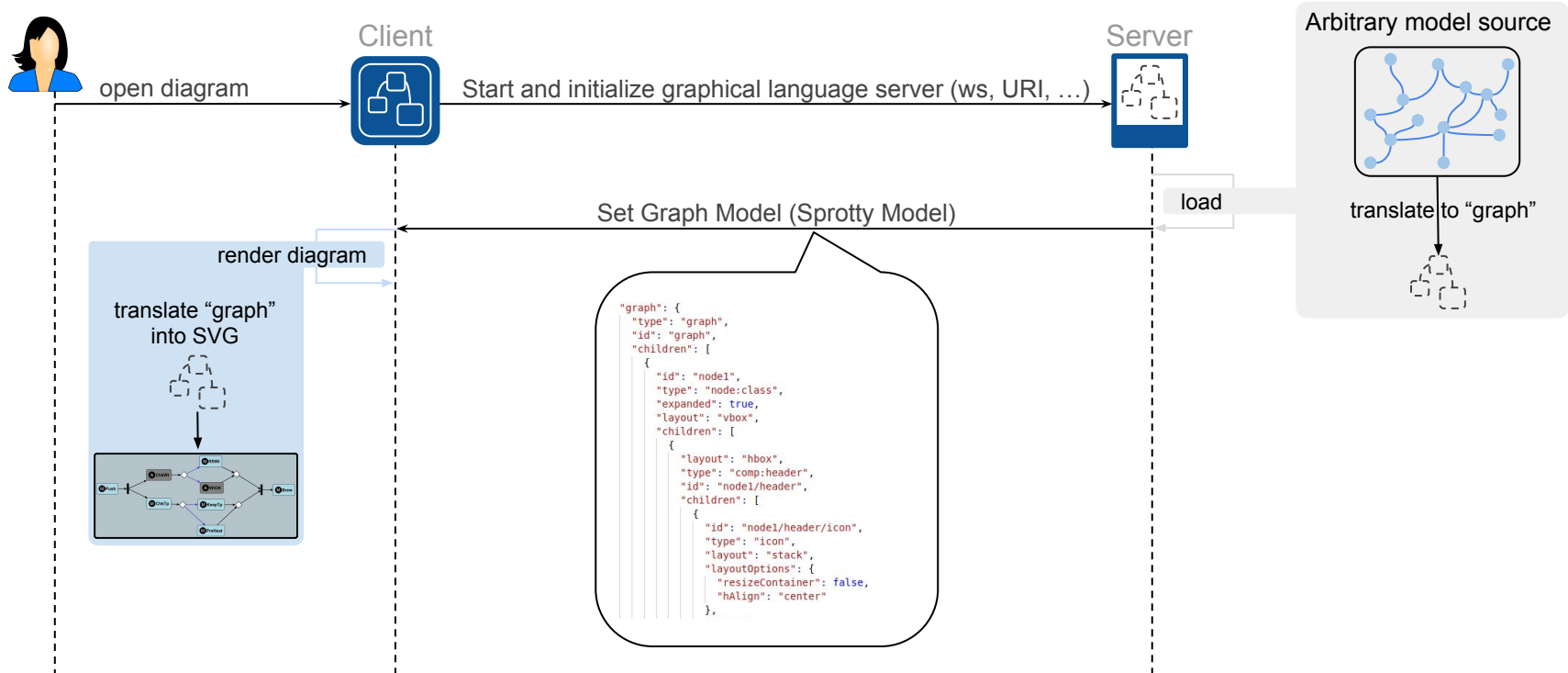
Eclipse Graphical Language Server Platform (GLSP)

- 1 Java-based server framework
 - Standalone server implementation
 - 2 Graphical Language Server Protocol
 - Language config, executing operations, ...
 - 3 Web-based Client framework
 - 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

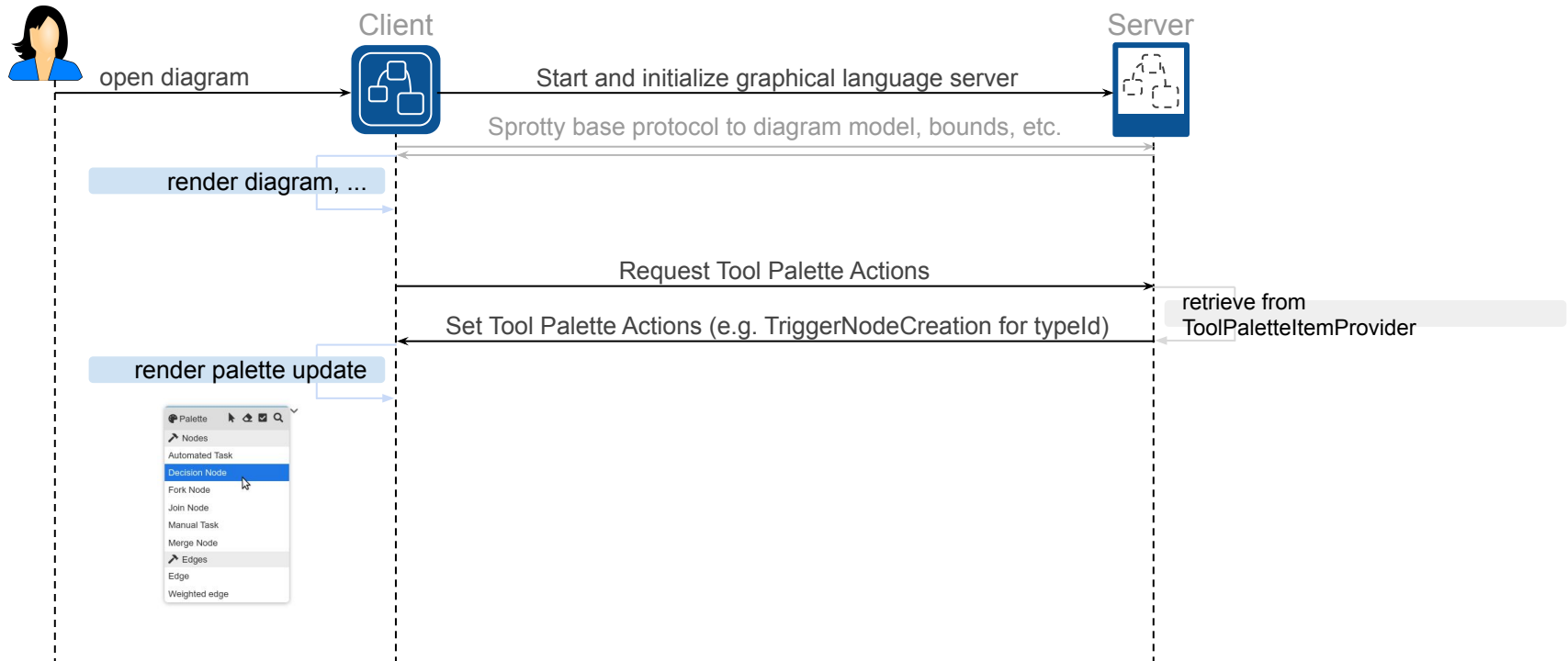


github.com/eclipse-glsp/glsp

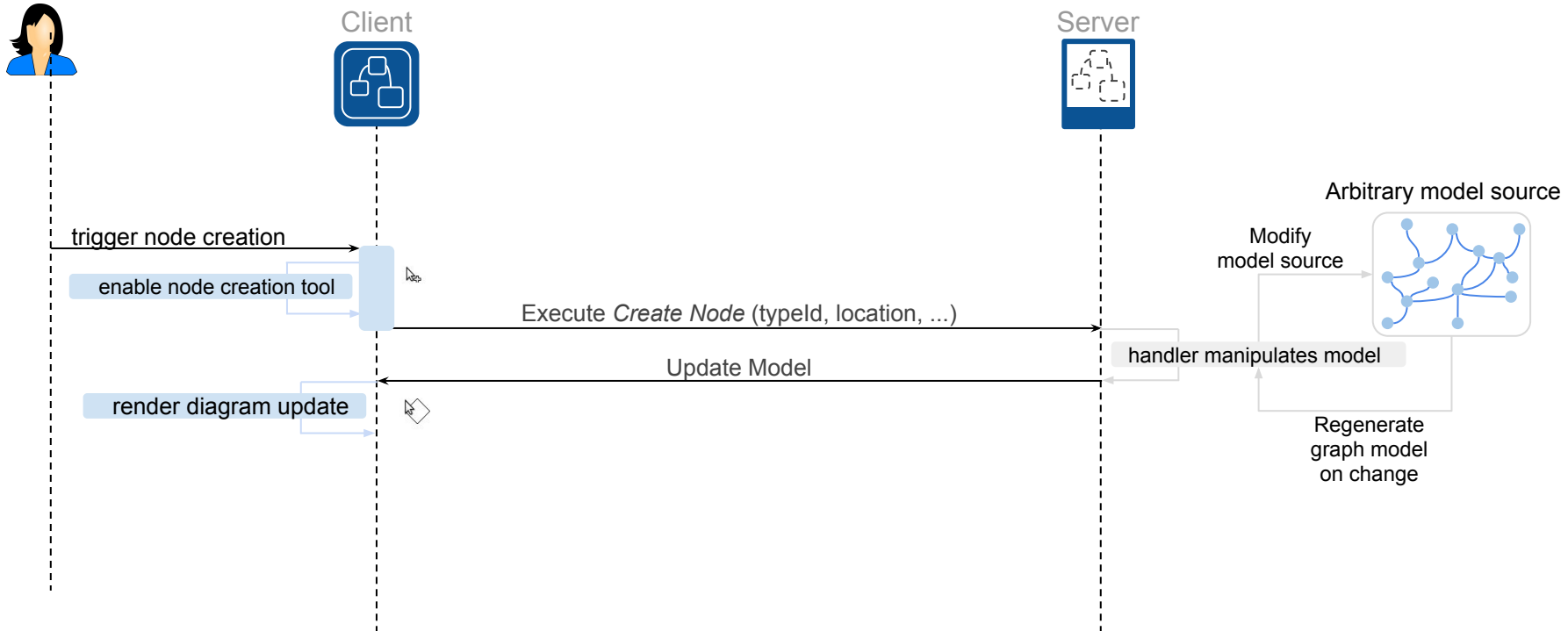
Initialization and Rendering with Eclipse Sprotty



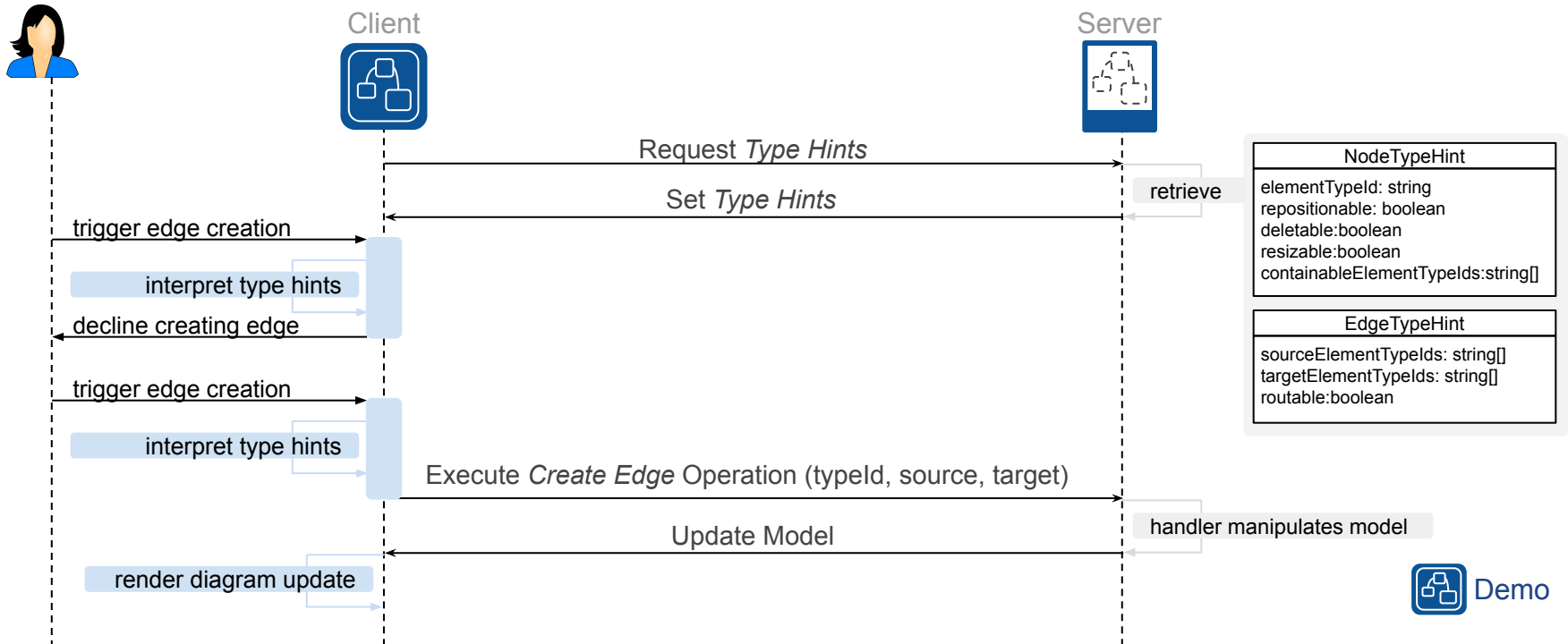
Client-Server Interaction: Initialization and Editing Tools



Client-Server Interaction: Model Manipulation



Client-Server Interaction: Type Hints

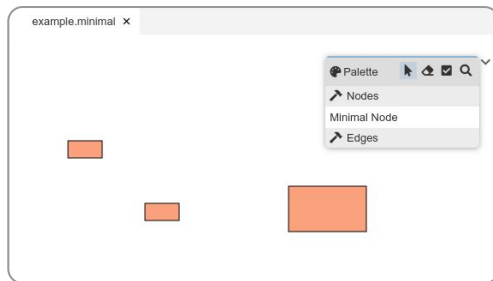


Let's Look Into the Code

- Minimal example available

Client-side Rendering in Sprotty

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



Server Module Configuration

```
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;
  }
}
```



Demo

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

The screenshot shows the logi.cals IEC FBD Editor interface. The main workspace displays a ladder logic diagram with various components like timers (T1, T2), counters (C1), and logic gates (AND, OR). A text box in the diagram reads: "This sample contains a program to count up or down. If 'up' = 'TRUE', counter counts up. 'count' is incremented/decremented by '1' for each cycle." Below the diagram, a table lists variables:

NAME	SECTION	DATA TYPE	USAGE	INITIAL VALUE
ENO	O	BOOL	Unused	
up	L	BOOL	Read	
count	L	INT	Read/Write	
fast	I	BOOL	Unused	

The screenshot shows the Ecere IDE's UML class diagram editor. A class diagram is displayed with the following structure:

```

classDiagram
    class ColorEnum {
        red
        blue
        green
    }
    class AbstractClass {
        color : ColorEnum
    }
    class Class {
        title : EString
    }
    AbstractClass <|-- Class
  
```

Text overlay on the diagram: "Ecere Diagrams in Theia Oct 22nd, 14:00 by Jonas".

The screenshot shows the Ecere IDE's debug console and a state transition diagram. The diagram illustrates a sequence of states and transitions:

```

graph LR
    MPush --> AChkWr
    AChkWr --> MRRWt
    AChkWr --> AWiOk
    MRRWt --> AChkTp
    AWiOk --> AChkTp
    AChkTp --> MKeepTp
    AChkTp --> MPreHeat
    MKeepTp --> MBrew
  
```

Text overlay on the diagram: "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



eclipse.org/glsp



github.com/eclipse-glsp/glsp

Evaluate ~~the~~ Sessions

Sign in and vote at Eclipsecon.org:
WITH

