

Building

Web-based Modeling Tools

with



Theia and Che

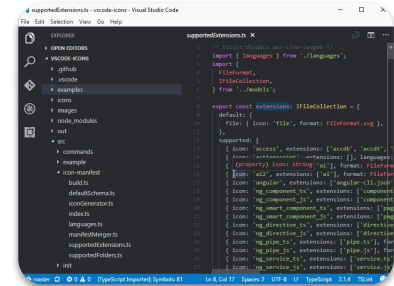


Philip Langer & Maximilian Koegel

EclipseSource

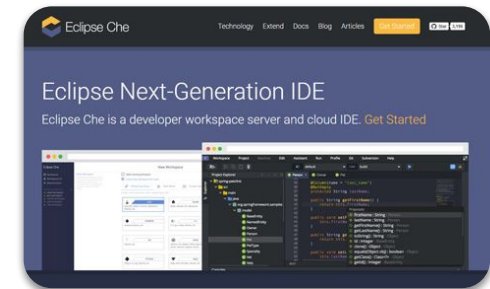
Why web-based tools?

- **Accessibility**
 - No client installation
 - Access through a web link
 - Simple client updates
 - Physical resource sharing
- **Usability**
 - Modern UI look and feel
 - SWT vs. HTML5
 - GEF 3 vs. SVG
- **Maintainability**
 - Room for evolution
 - Availability of developers



```

support@theia:~/workspace$ cat support/extensions.json
{
  "name": "Eclipse Che",
  "version": "1.0.0",
  "description": "Eclipse Che",
  "author": "Eclipse Che",
  "license": "EPL",
  "repository": "https://github.com/eclipse-che/che",
  "bugs": "https://github.com/eclipse-che/che/issues",
  "homepage": "https://www.eclipse.org/che",
  "keywords": "Eclipse Che",
  "categories": "IDE",
  "engines": {
    "vscode": "^1.47.0"
  },
  "activationEvents": [
    "*"
  ],
  "main": "index.js",
  "browser": "index.js",
  "icon": "resources/icon.png",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "devDependencies": {
    "vscode": "1.47.0"
  },
  "dependencies": {
    "vscode": "1.47.0"
  },
  "publisher": "Eclipse Che"
}
    
```



A prototypical (modeling) tool

The screenshot shows the SuperBrewer3000 modeling tool interface. On the left, a vertical list of callout boxes points to specific features in the interface:

- Workbench:** Points to the overall IDE environment.
- Forms:** Points to the configuration panels for the Processor, Display, and Dimension components.
- Diagrams:** Points to the main process flow diagram on the right.
- Text:** Points to the code editor at the bottom right.
- Analysis:** Points to the Workflow Analysis window at the bottom center.
- Generator:** Points to the code editor, indicating the tool's ability to generate code.

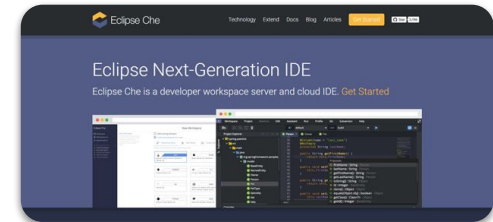
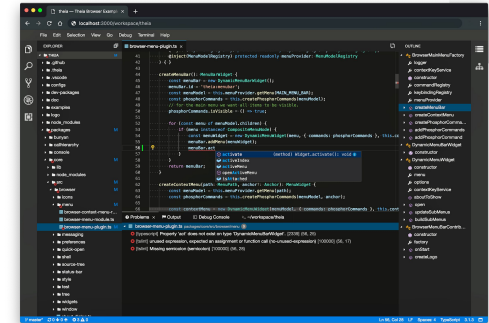
The interface includes the following components:

- EXPLORER:** A file tree on the left showing the project structure.
- Component Properties:** Panels for Processor (Vendor: Qualcomm, Clock Speed: 5), Display (Width: 10, Height: 20), and Dimension (Width: 10, Height: 12, Length: 13).
- Process Flow Diagram:** A state machine diagram with nodes like 'Push', 'Check Water', 'Water Ok', 'Refill water', 'Check drip tray', 'Preheat', and 'Brew'.
- Workflow Analysis:** A window showing a path analysis: 'Push > Check Water > Refill water > Check drip tray' with a 50.0% completion rate.
- Code Editor:** Shows the generated Java code for the 'Brew' class.

Key enablers for building domain-specific, web-based tools

- Eclipse Theia
 - Extensible cloud IDE
 - Default frontend for Eclipse Che
- Eclipse Che
 - Kubernetes-native IDE platform
 - Management of workspaces and dev environments
- Monaco & Language server protocol (LSP)
 - Protocol enabling the separation of editor (front-end) and language implementation (back-end)
 - Feature-rich and broadly adopted (VS Code)

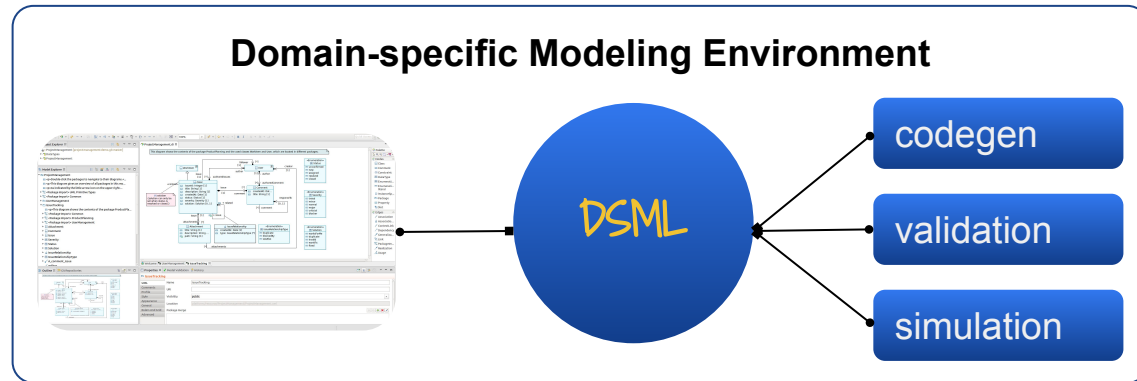
 THEIA



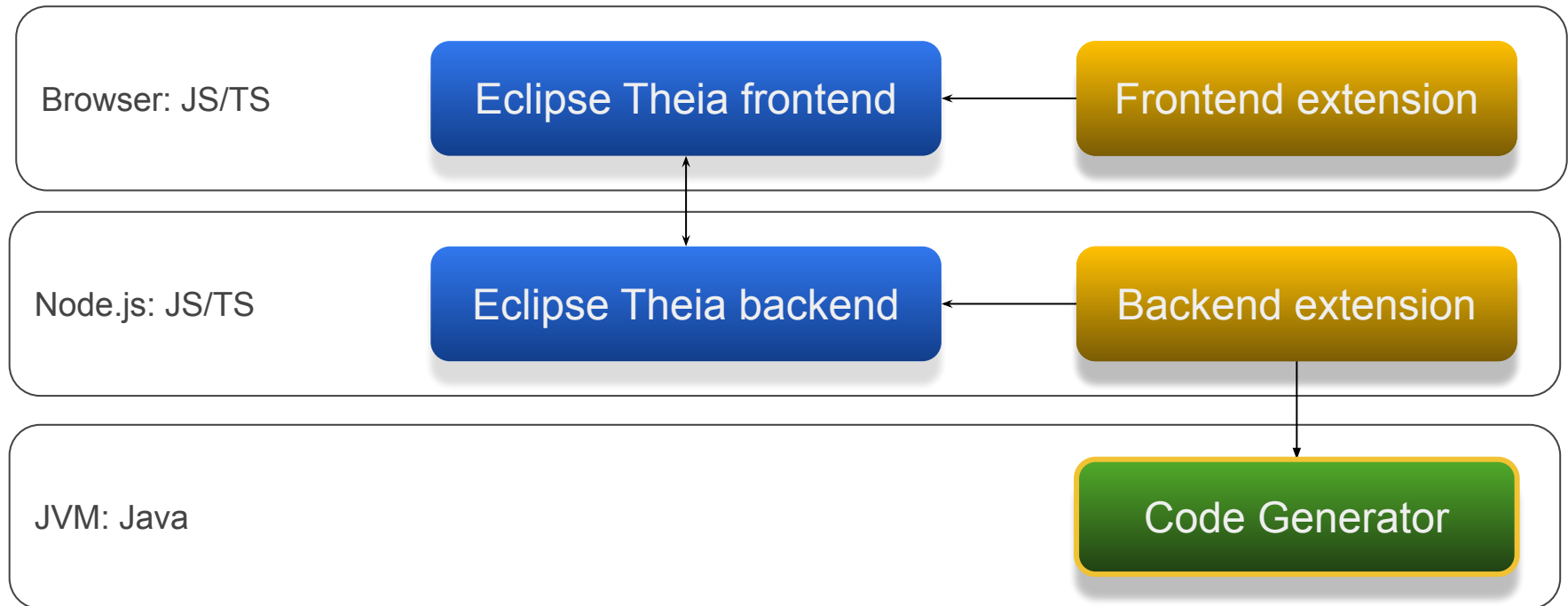
⇒ **Related talk: “Eclipse Theia and Che, explained and explored!”, Today 17:00, Theater Stage**

Reinventing the wheel?

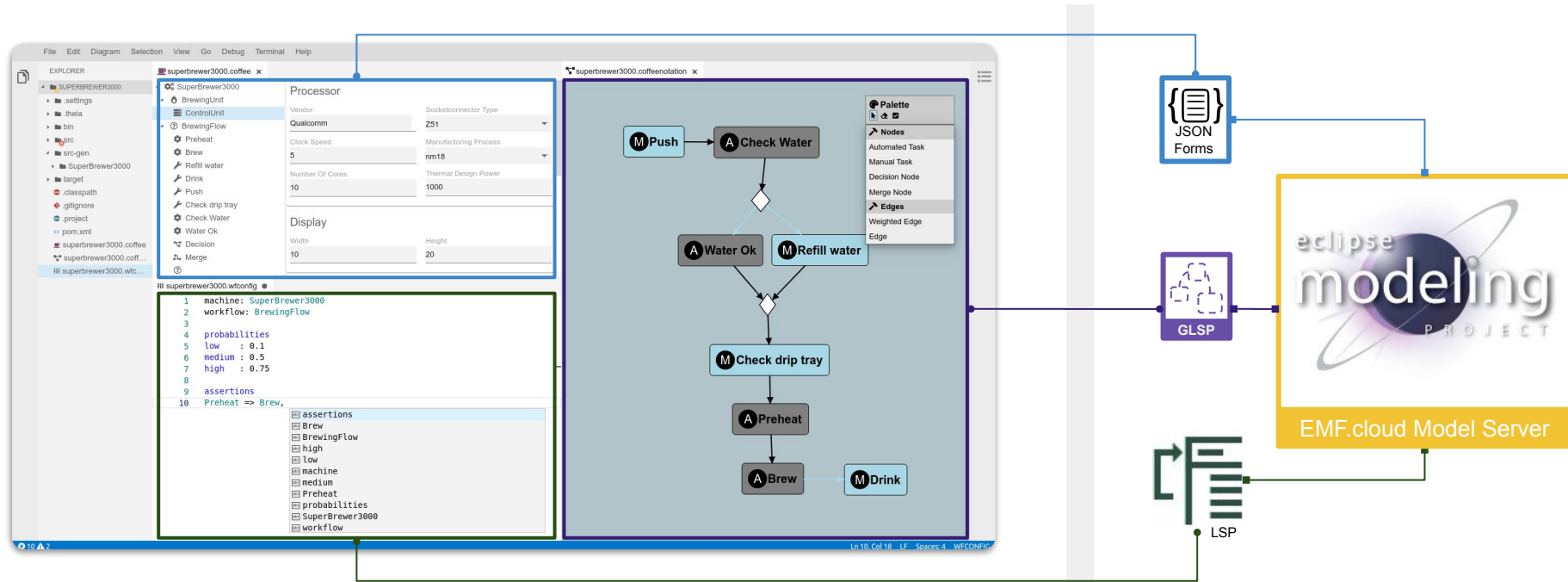
- Which components can be reused?
- What needs to be reimplemented?
- How do we separate frontend and backend functionality?



Typical reuse example: Code Generation

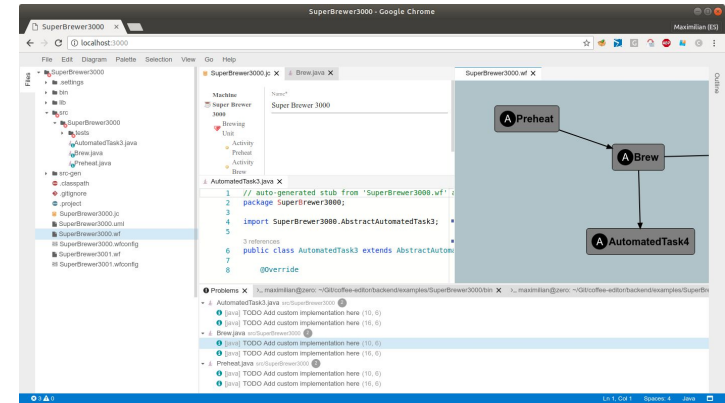


Overall tool architecture



Demo overview

- One connected model for coffee makers
 - Structural model
 - Behavioral model
- Example IDE with the following features:
 - Tree-Editor for structural modeling with forms
 - Graphical editor for behavioral model
 - Code generation
 - Working with source code
 - Textual Modeling
 - Model Analysis
 - Multi-User Support



Demo: Form-based editing

superbrewer3000.coffee x

- SuperBrewer3000
 - BrewingUnit
 - ControlUnit**
 - BrewingFlow
 - Preheat
 - Brew
 - Refill water
 - Drink
 - Push
 - Check drip tray
 - Check Water
 - Water Ok
 - Decision
 - Merge
 - Flow
 - Flow
 - WeightedFlow
 - WeightedFlow
 - Flow
 - Flow
 - Flow
 - Flow
 - Flow
 - Flow
 - WeightedFlow

Processor

Vendor	Socketconnector Type
Qualcomm	Z51
Clock Speed	Manufacturing Process
5	
Number Of Cores	Thermal Design Power
10	1000

Display

Width	Height
10	20

Dimension

Width	Height	Length
10	12	13

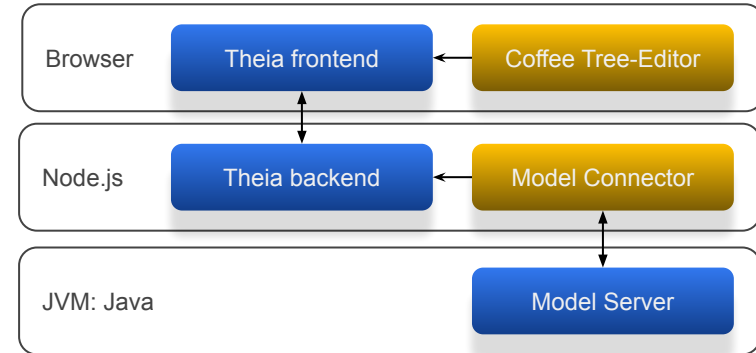
Ram

No data

!² +

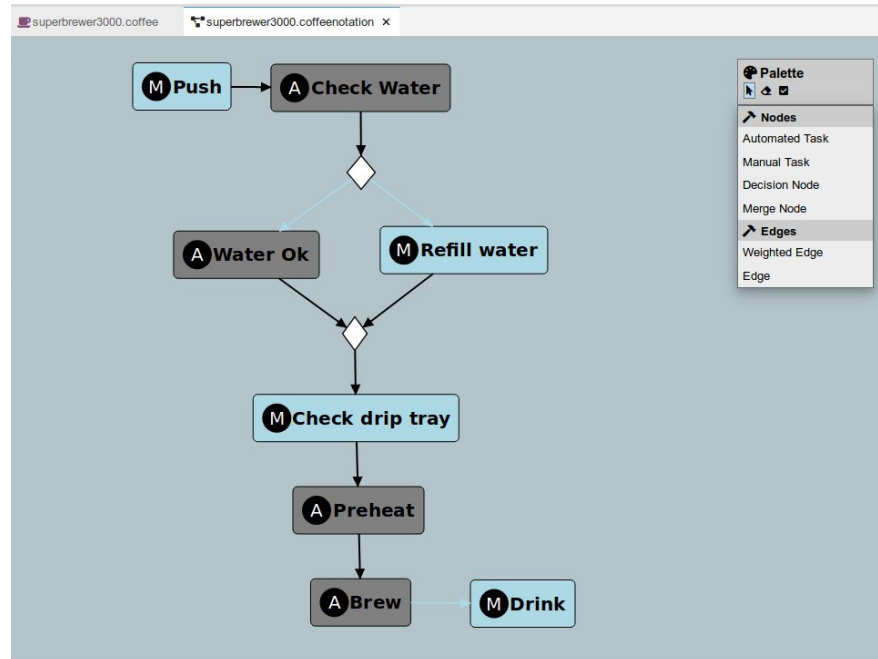
Form-based editing

- Tree
 - Based on Theia Tree Widget
 - LabelProvider and ContentProvider
- Detail-Form
 - Based on JSON Forms
 - Declarative approach: JSON + UI Schema
- Synchronization
 - Based on EMF.cloud Model Server
 - Typescript-based Model Server client API
 - Push changes as commands
 - Subscribe to updates



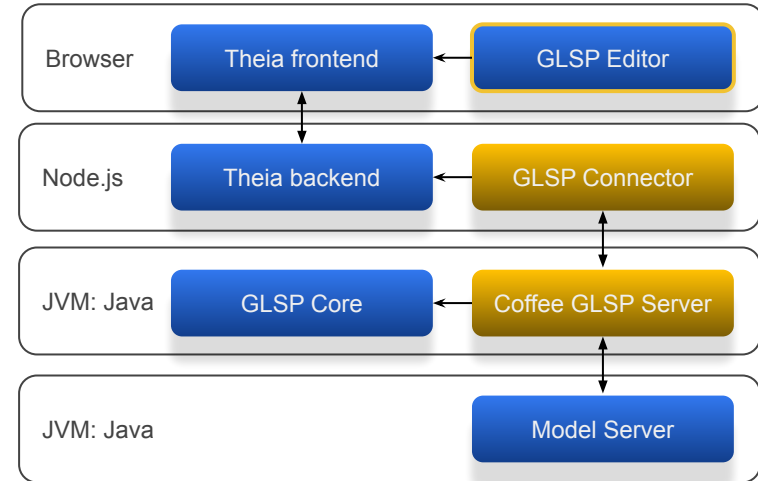
⇒ **Related talk: “Property editors in space”, Thursday 1pm, Theater Stage**

Demo: Graphical Modeling



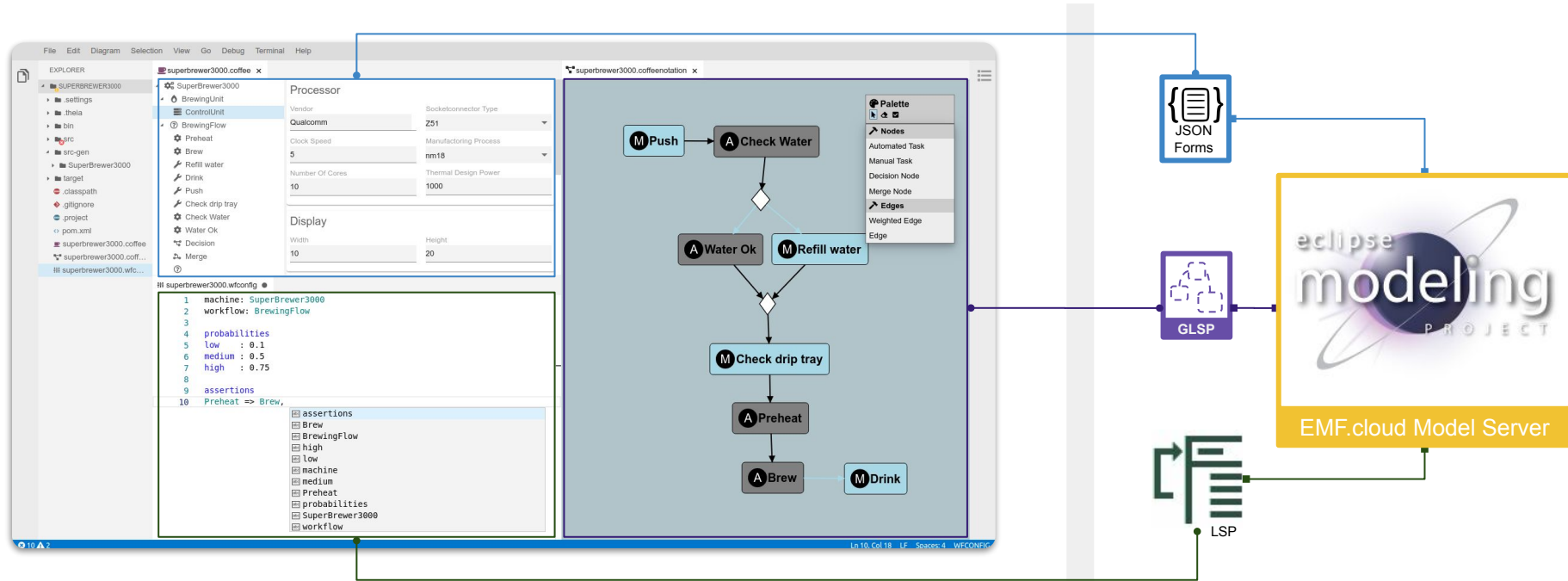
Graphical Modeling

- Graphical Language Server Platform (GLSP)
 - LSP for Graphical Editors
 - GLSP client:
 - generic
 - renders graphical visualization
 - GLSP server:
 - specific to DSL
 - maps model to graphical visualization
 - synchronization with model server
 - Based on Eclipse Sprotty

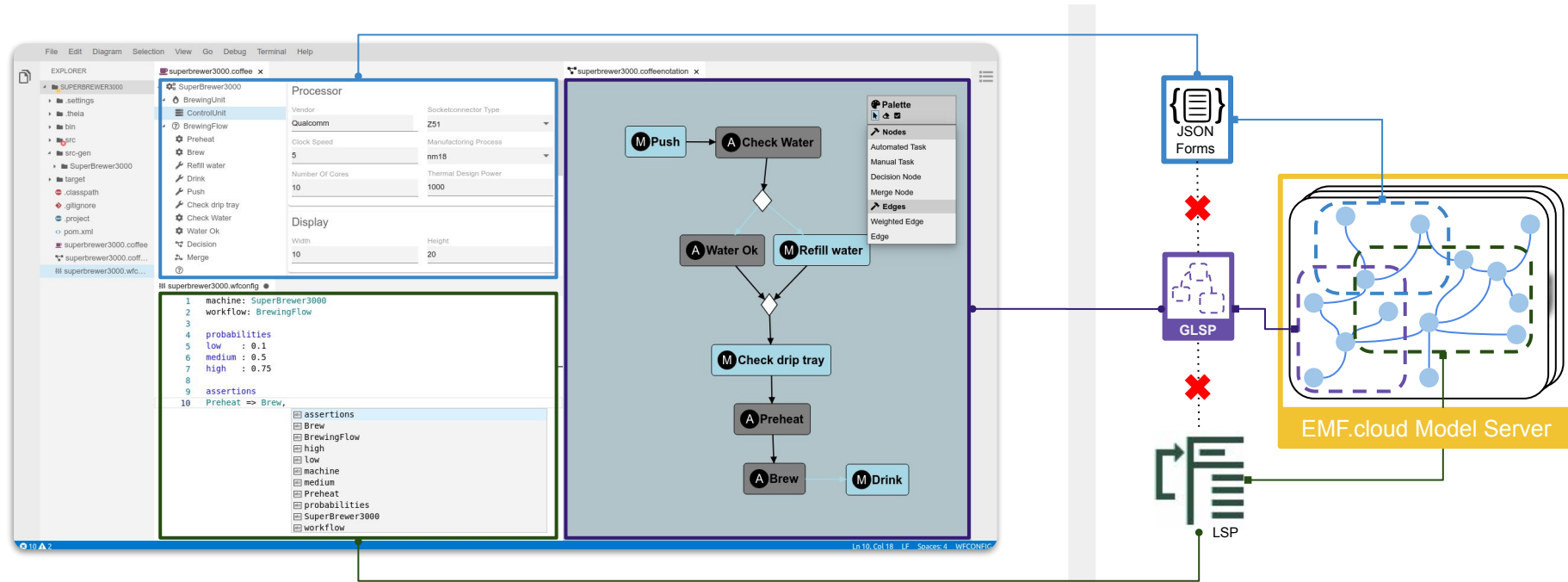


⇒ **Related talk: “Diagrams in web and space with GLSP”, Thursday 15:10pm, Bürgersaal 2**

Demo: Model Server



Model Server



Model Server

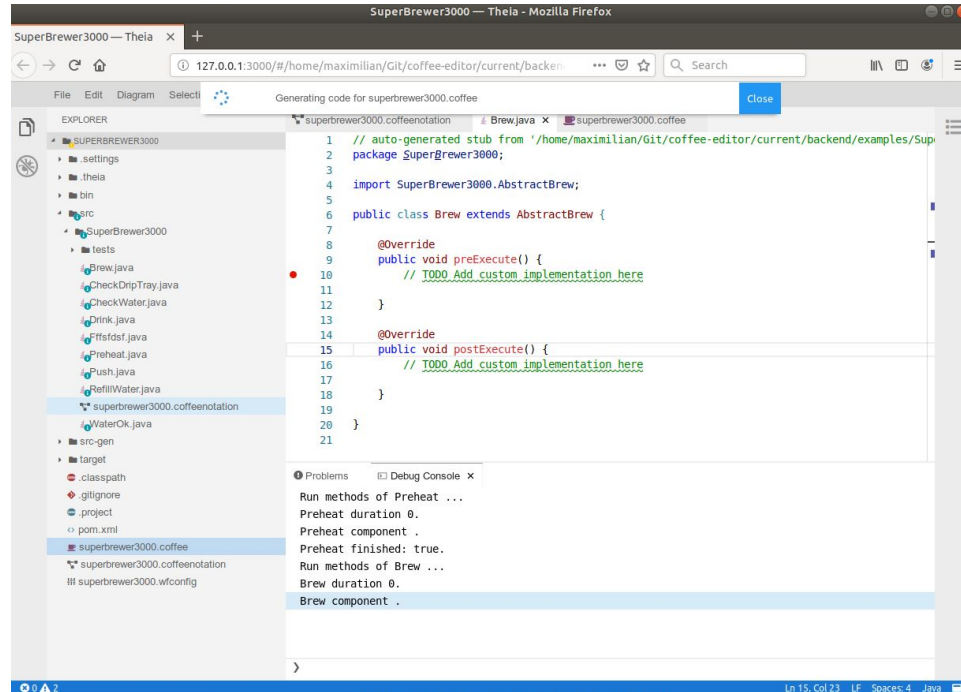
- Component of EMF.cloud Eclipse project
- Features
 - Command-based change interface
 - Notification mechanism via sockets
 - Convenient model access with client APIs



EMF.cloud Model Server

⇒ **Related talk: “Lifting the greatness of EMF into the cloud with EMF.cloud”, Wednesday 12:35pm**

Demo: Generators - Model to Text



Generating code for superbrewer3000.coffee

```

1 // auto-generated stub from '/home/maximilian/Git/coffee-editor/current/backend/examples/Sup
2 package SuperBrewer3000;
3
4 import SuperBrewer3000.AbstractBrew;
5
6 public class Brew extends AbstractBrew {
7
8     @Override
9     public void preExecute() {
10         // TODO: Add custom implementation here
11     }
12
13
14     @Override
15     public void postExecute() {
16         // TODO: Add custom implementation here
17     }
18
19 }
20
21

```

Problems | Debug Console

```

Run methods of Preheat ...
Preheat duration 0.
Preheat component .
Preheat finished: true.
Run methods of Brew ...
Brew duration 0.
Brew component .

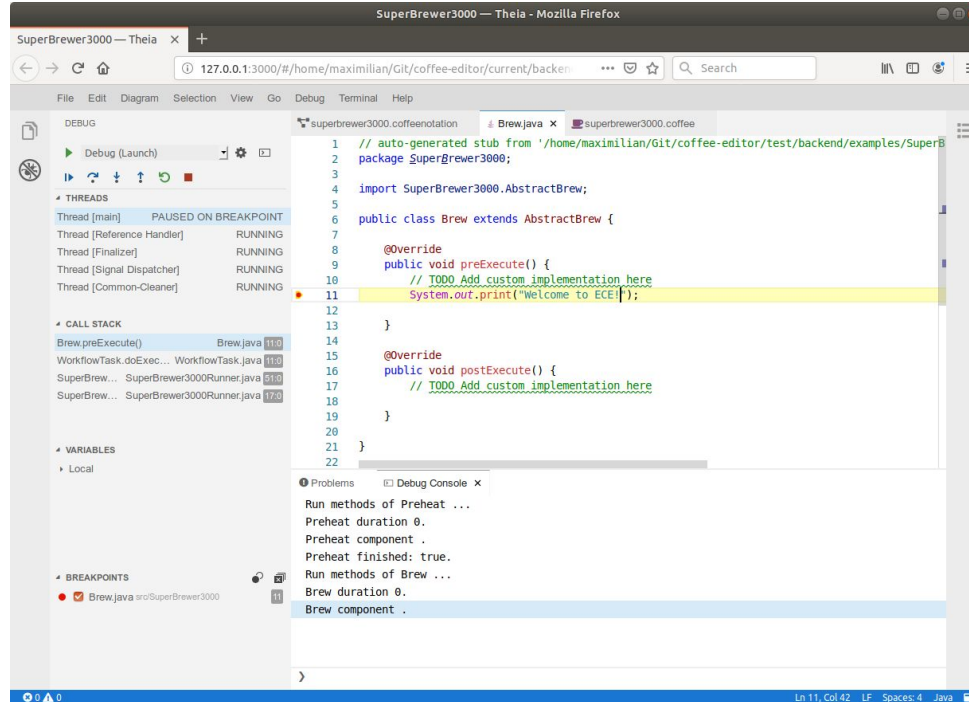
```


Generators: Model to Text

- Based on Eclipse Xtend
- Generator jar build with Maven
- Launched on demand via CLI
- Generates into selected Theia workspace folder



Demo: Working with source code



The screenshot shows the Eclipse IDE interface. The main editor displays a Java source file named `Brew.java` with the following code:

```

1 // auto-generated stub from '/home/maximilian/Git/coffee-editor/test/backend/examples/SuperB
2 package SuperBrewer3000;
3
4 import SuperBrewer3000.AbstractBrew;
5
6 public class Brew extends AbstractBrew {
7
8     @Override
9     public void preExecute() {
10         // TODO: Add custom implementation here
11         System.out.println("Welcome to ECE!");
12     }
13
14
15     @Override
16     public void postExecute() {
17         // TODO: Add custom implementation here
18     }
19
20 }
21 }
22

```

A breakpoint is set at line 11. The left sidebar shows the **DEBUG** view with the following sections:

- DEBUG**: Debug (Launch) button.
- THREADS**: Thread [main] PAUSED ON BREAKPOINT; Thread [Reference Handler] RUNNING; Thread [Finalizer] RUNNING; Thread [Signal Dispatcher] RUNNING; Thread [Common-Cleaner] RUNNING.
- CALL STACK**: Brew.preExecute() (Brew.java:110); WorkflowTask.doExec... (WorkflowTask.java:110); SuperBrew... (SuperBrewer3000Runner.java:210); SuperBrew... (SuperBrewer3000Runner.java:170).
- VARIABLES**: Local.
- BREAKPOINTS**: Brew.java src/org/SuperBrewer3000 (11).

The bottom panel shows the **Debug Console** with the following output:

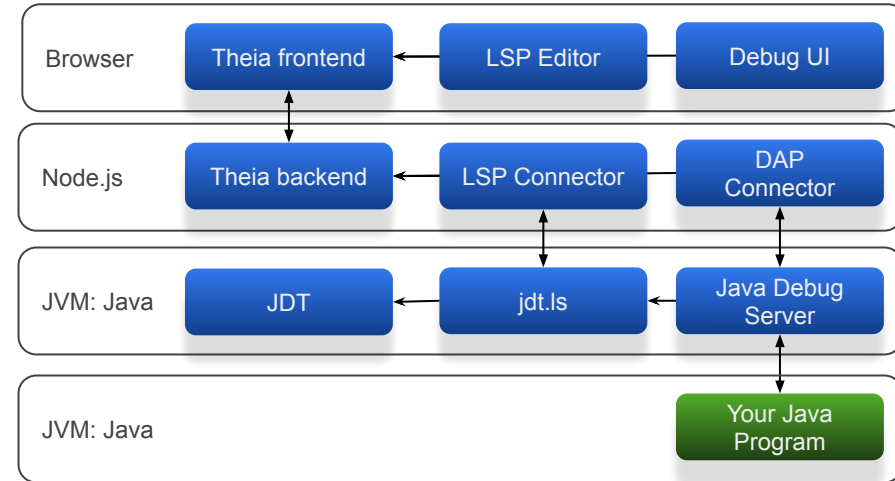
```

Run methods of Preheat ...
Preheat duration 0.
Preheat component .
Preheat finished: true.
Run methods of Brew ...
Brew duration 0.
Brew component .

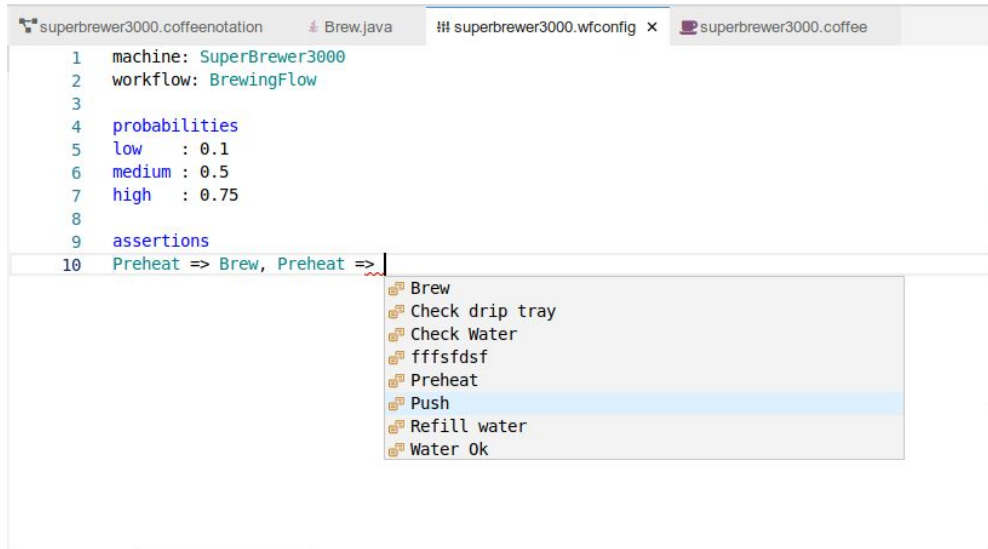
```

Working with source code

- Enablers:
 - Language Server Protocol (LSP)
 - Debug Adapter Protocol (DAP)
- Theia Code Editor
 - Monaco-based (VS Code)
 - Uses LSP to “understand” language
- Theia Debug Extension (DAP)
 - Uses DAP to support language debug



Demo: Textual Modeling

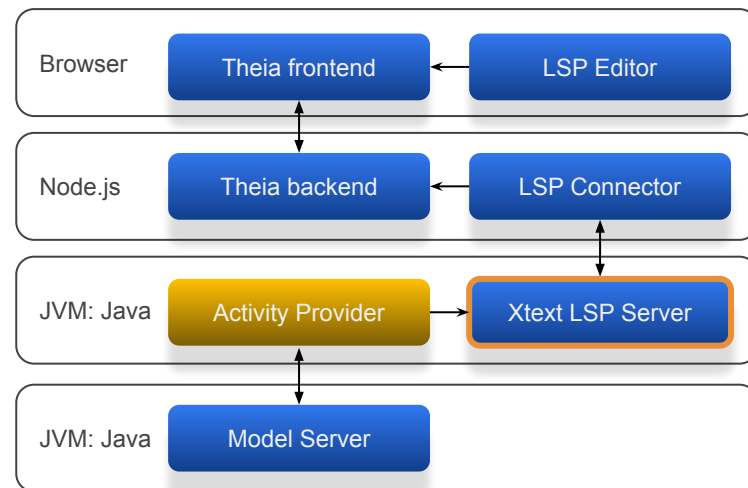


```
superbrewer3000.coffeenotation Brew.java #! superbrewer3000.wfconfig x superbrewer3000.coffee
1 machine: SuperBrewer3000
2 workflow: BrewingFlow
3
4 probabilities
5 low : 0.1
6 medium : 0.5
7 high : 0.75
8
9 assertions
10 Preheat => Brew, Preheat =>
```

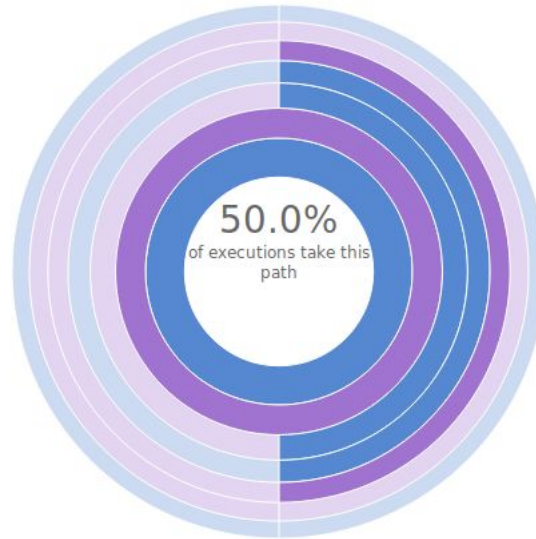
- Brew
- Check drip tray
- Check Water
- fffsdfs
- Preheat
- Push
- Refill water
- Water Ok

Textual Modeling

- Frontend: Theia Editor (Monaco)
- Backend:
 - DSL modeled as XText grammar
 - XText LSP Server for DSL

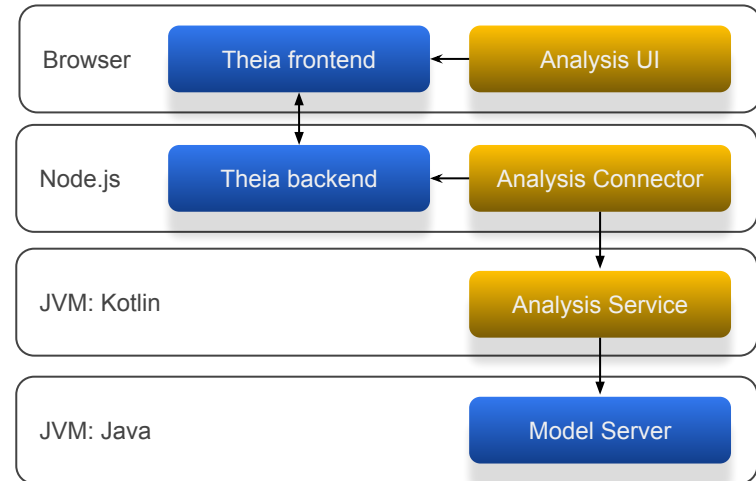


Demo: Model Analysis

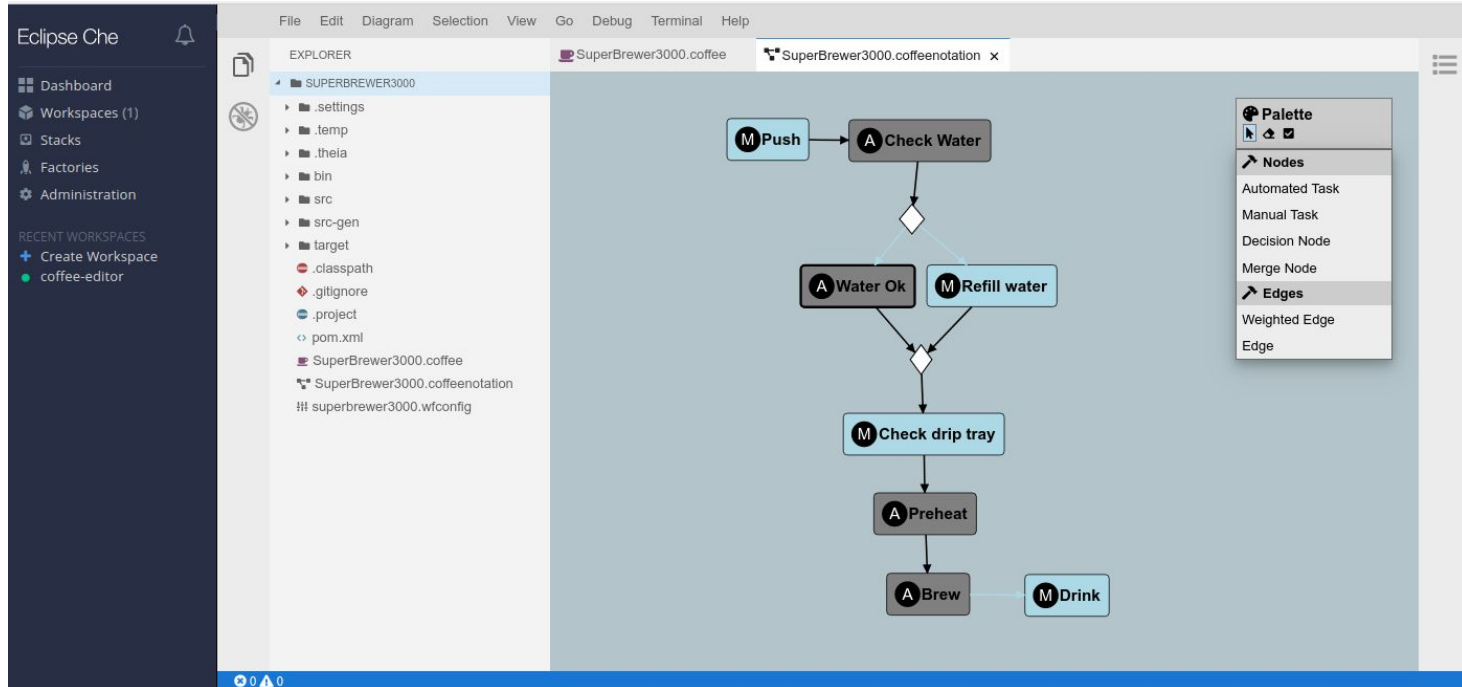


Model Analysis

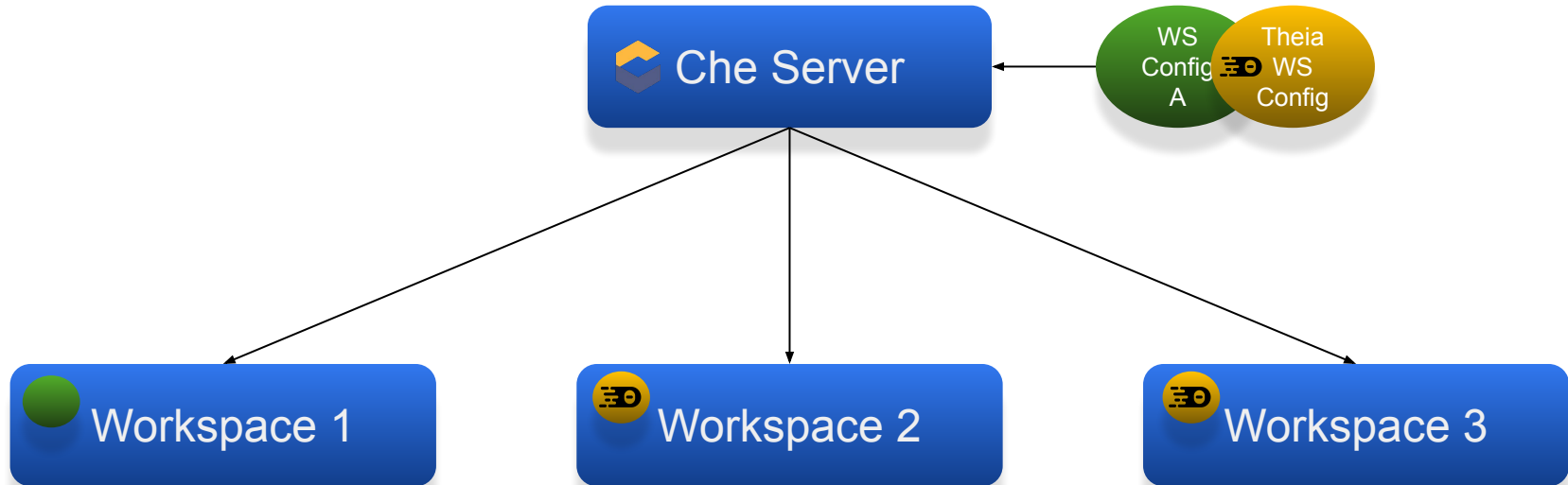
- Frontend:
 - Uses D3 to visualize analysis results
- Backend:
 - Fetches data from model server
 - Calculates analysis result from data



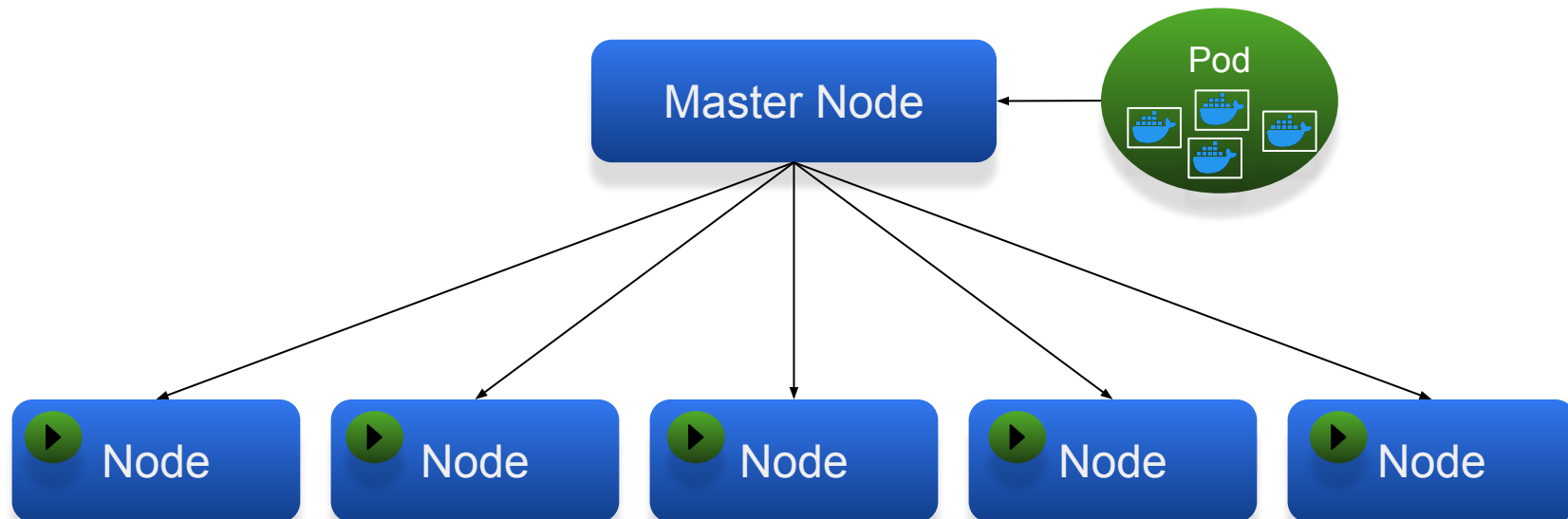
Multi-User Support



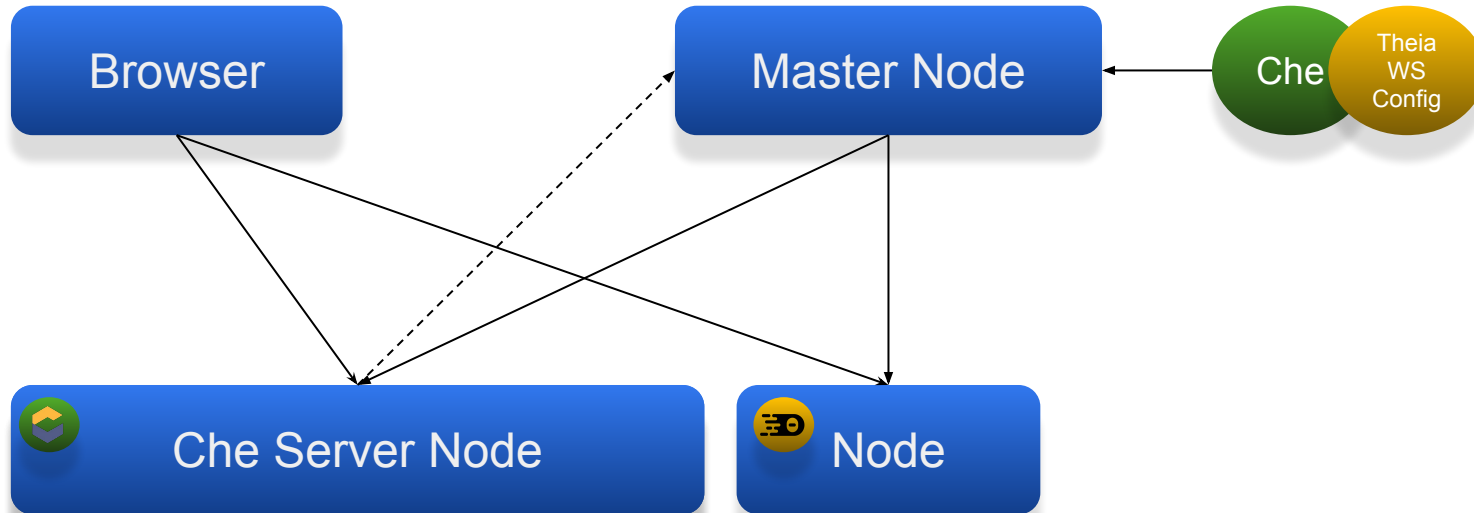
Running Theia on Che



Excursion: Running applications on Kubernetes



Excursion: Running Che and Theia on Kubernetes



Summary

- Web-based modeling tools are feasible today
- Reuse and migration easier than expected
- Web technology can leverage unique advantages
 - Modern UI and styling
 - Zero installation for users
 - Enables “cloud” business models
- There is open-source components
 - Eclipse Theia and Eclipse Che
 - EMF.cloud, LSP, GLSP, JSON Forms, XText, Sprotty and D3
 - Existing business logic can often be reused
- Demo code available: <https://github.com/eclipsesource/coffee-editor>



→ **Important now:** Define strategy and timeplan, build POC

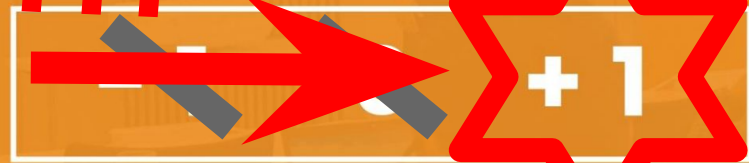
 eclipsecon
Europe 2019  OSGi™
Community Event 2019
LUDWIGSBURG, GERMANY | OCTOBER 21 - 24, 2019

Please

EVALUATE THIS SESSIONS

Sign in and vote using the conference app or eclipsecon.org

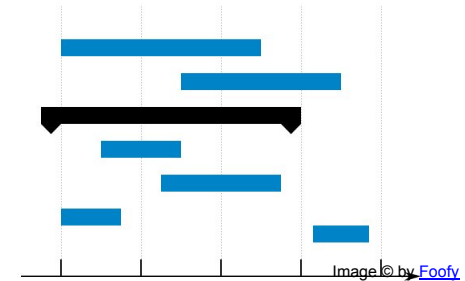
WITH



Thank you!

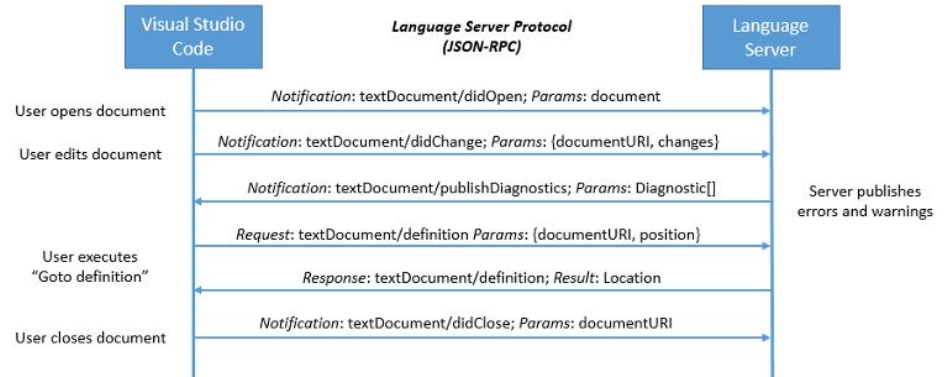
Towards a migration strategy

- Now: Define a strategy and timeplan, build POC
- Short-term: Consider for architectural decisions
- Mid-term:
 - Prepare architecture for migration iteratively
 - Migrate high-value use cases iteratively
 - Single-source components, enable reuse
- Long-term:
 - Migrate use-case by use-case iteratively
 - Deprecate desktop-based solution



Excursion: Language Server Protocol

- Separation of concerns
 - Tooling for editing code and textual DSLs
 - Language smarts: auto-completion, refactoring support
- Advantages
 - LSP-Client is language-agnostic
 - LSP-Server is tool-agnostic



Excursion: Separation of Concerns with GLSP

