EclipseSource

Building

Web-based Modeling Tools

based on

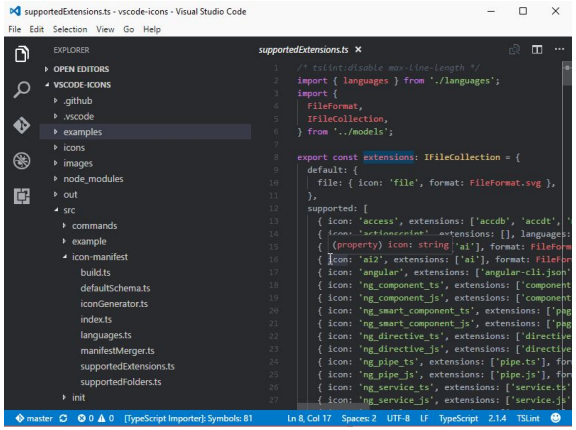
Eclipse THEIA

eclipsecon Europe 2018

Philip Langer & Maximilian Koegel

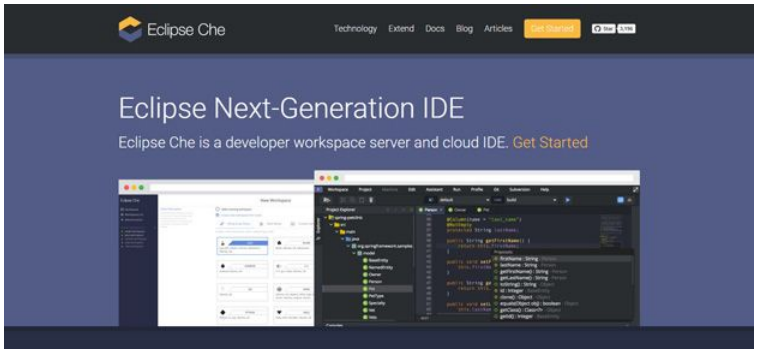
EclipseSource

Why web-based?



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

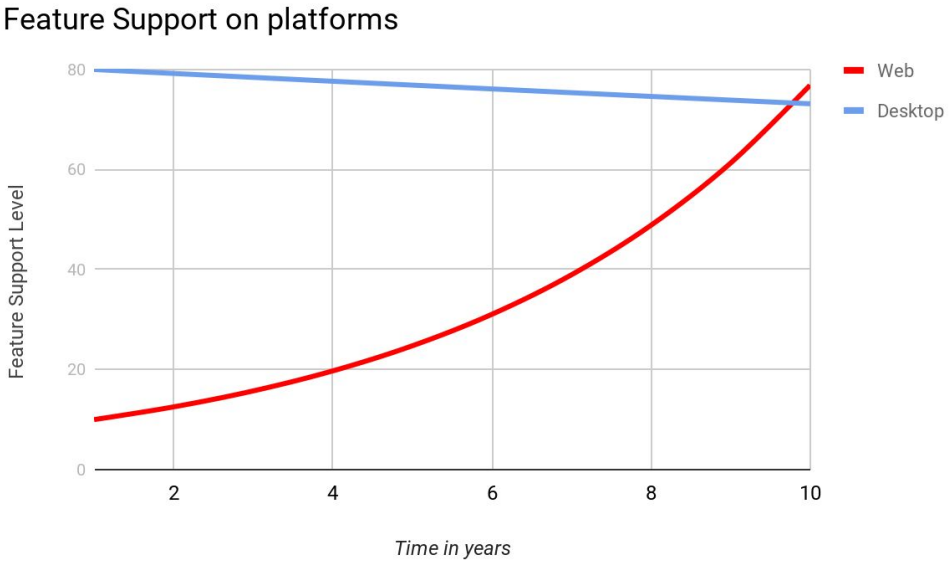
2. Simplified deployment
   - Installing tool vs. opening a link
   - Access from everywhere immediately
   - “Cloud Modeling Tool”



Challenges with web-based technology

- **High uncertainty** about frameworks'…
  - maturity
  - mid-term maintenance
- **Small ecosystem** for modeling tool components:
  - fewer components
  - lower level of abstraction
  - less features

=>Higher cost (order of magnitude)

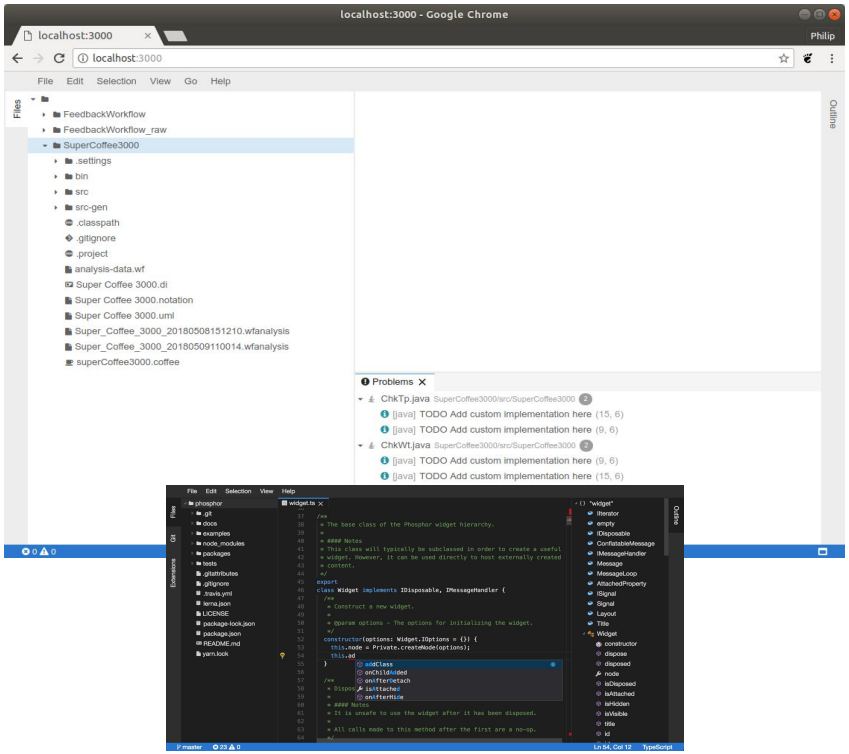


Major strategies to mitigate challenges

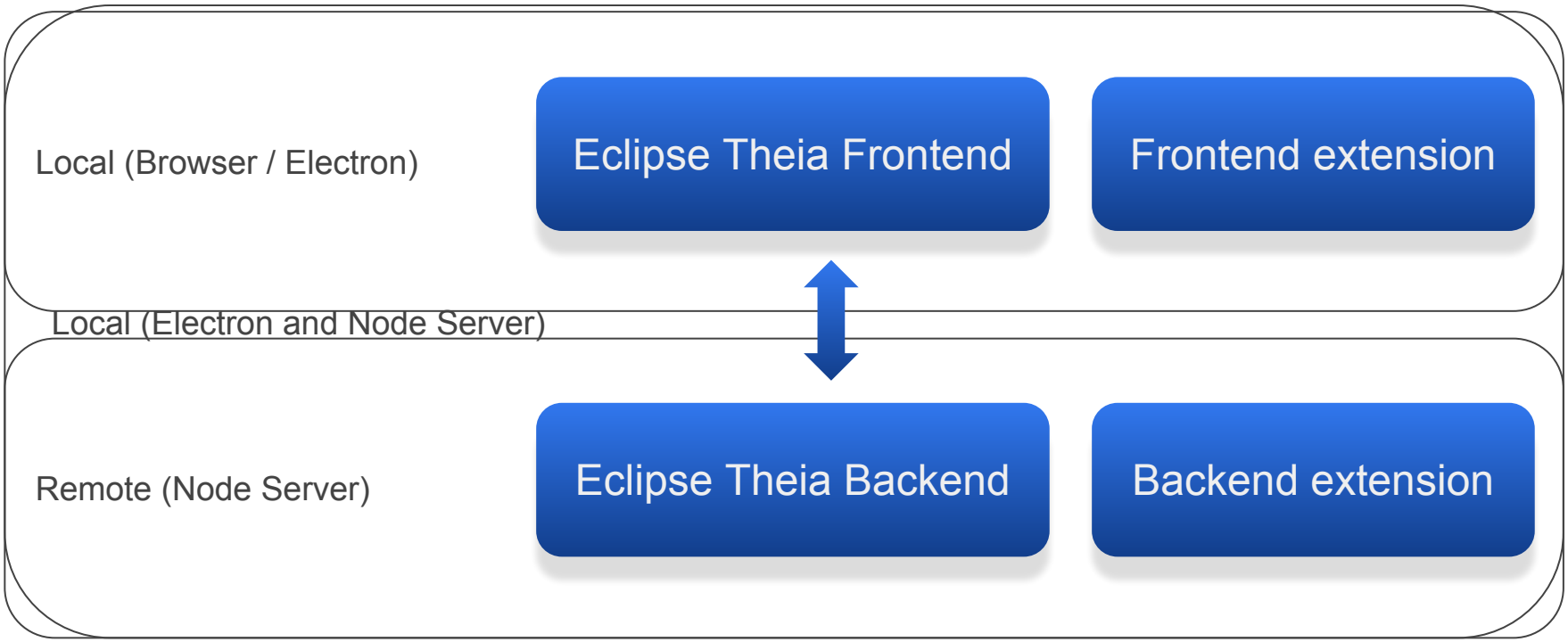
- **Standalone** - Build standalone components
- **Abstraction** - Isolate components from frameworks
- **Declarative** - Use declarative artifacts
- **Services** - Factor-out business logic

Why Eclipse Theia?

- Extensible
- Well-architectured
  - Extensible via DI
  - Extensive use of existing components
- Includes core features
  - Code Editing
  - Console
  - Workspace
  - Windowing
- Deployment
  - Browser-based
  - Desktop-based via Electron
- Open Source and Community

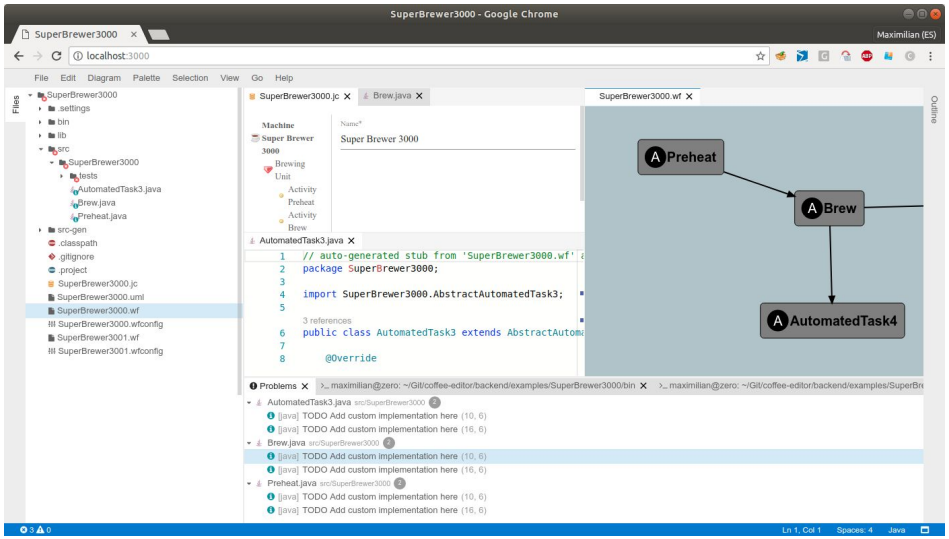


Theia architecture and deployment

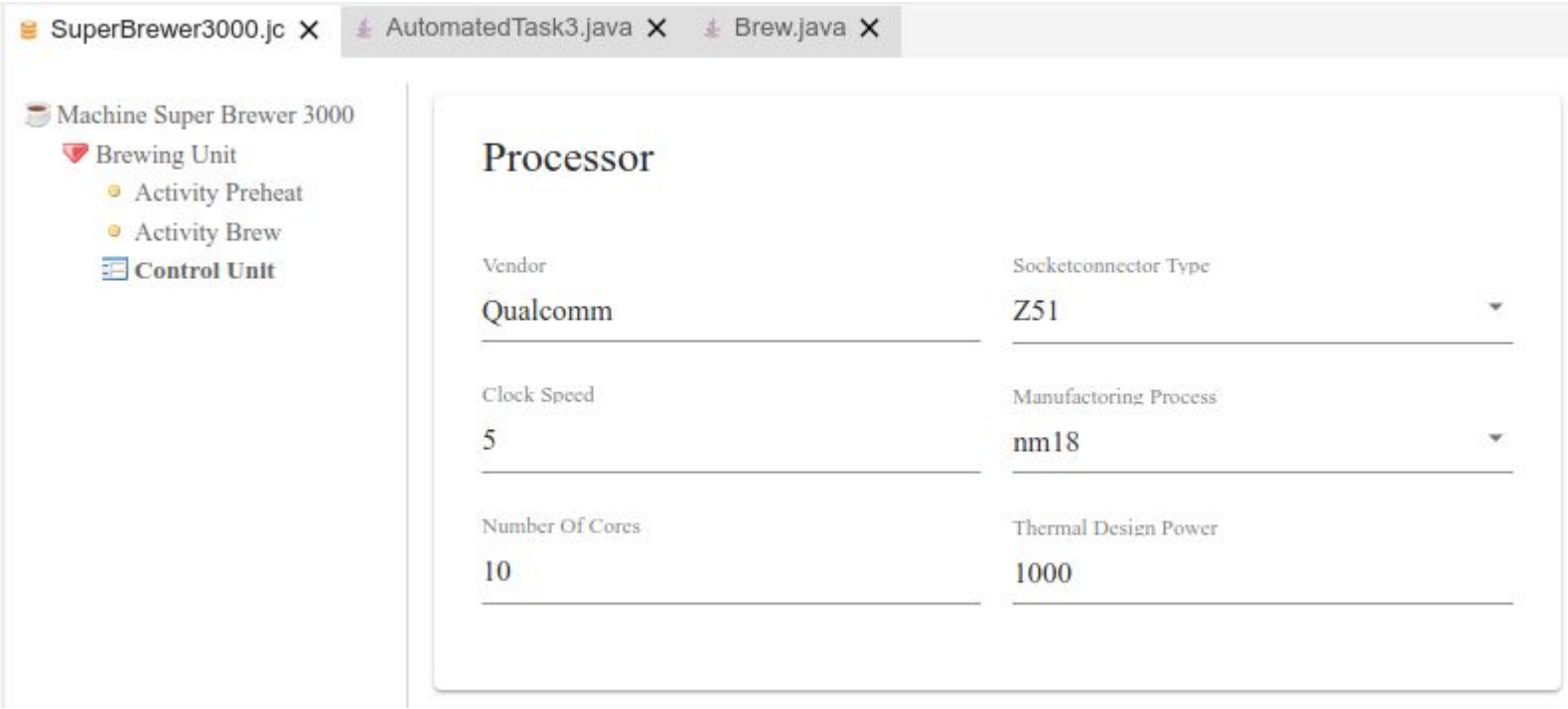


Demo of a web-based IDE

- Example IDE for modeling coffee makers
- Structural model with tree-based visualization
- Code editing
- Behavioral model with graphical visualization
- Analysis of behavior
- Code generation and testing

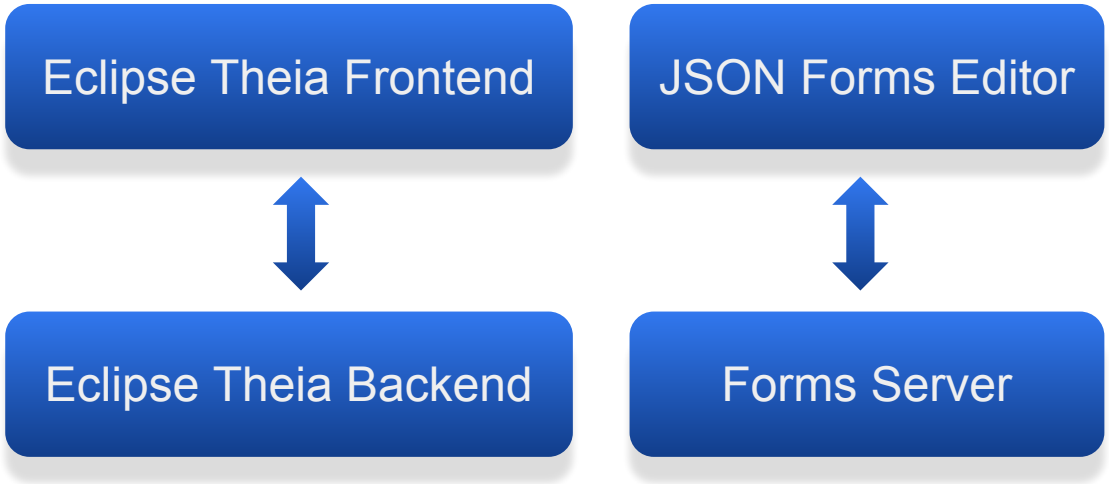


Tree-based editor - Demo



Tree-based editor - Implementation

- Editor based on JSON Forms
  - Declarative Form-based UIs (Excursion)
  - Based on JSON Schema and UI Schema
- Forms Server
  - Supplies Data to client
  - Supplies JSON Schema and UI Schema to client



Excursion: Declarative form-based UIs

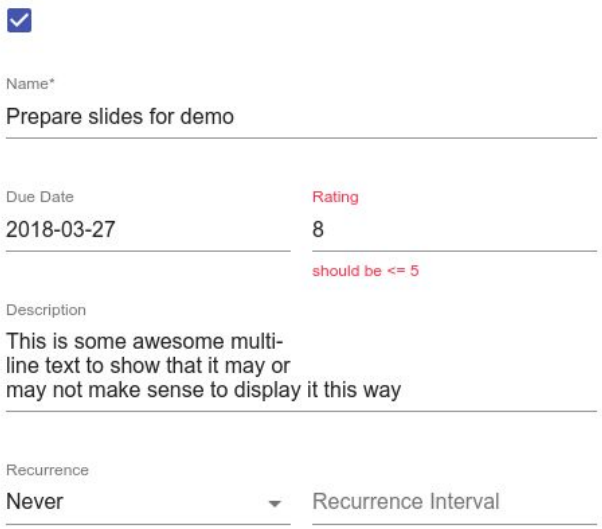
```
1  {
2    "type": "object",
3    "properties": {
4      "name": {
5        "type": "string",
6        "minLength": 1
7      },
8      "description": {
9        "type": "string"
10     },
11     "done": {
12       "type": "boolean"
13     },
14     "due_date": {
15       "type": "string",
16       "format": "date"
17     },
18     "rating": {
19       "type": "integer",
20       "maximum": 5
21     },
22     "recurrence": {
23       "type": "string",
24       "enum": [
25         "Never",
26         "Daily",
27         "Weekly",
28         "Monthly"
29       ]
30     },
```

JSON Schema (Data Schema)

```
1  {
2    "type": "VerticalLayout",
3    "elements": [
4      {
5        "type": "Control",
6        "label": false,
7        "scope": "#/properties/done"
8      },
9      {
10       "type": "Control",
11       "scope": "#/properties/name"
12     },
13     {
14       "type": "HorizontalLayout",
15       "elements": [
16         {
17           "type": "Control",
18           "scope": "#/properties/due_date"
19         },
20         {
21           "type": "Control",
22           "scope": "#/properties/rating"
23         }
24       ]
25     },
26     {
27       "type": "Control",
28       "scope": "#/properties/description",
29       "options": {
30         "multi": true
31       }
32     },
33     {
```

UI Schema

Renderer Set



User Interface

Java code editing - Demo

File Edit Diagram Selection View Go Help

Files

- SuperCoffee3000
  - .settings
  - bin
  - src
    - SuperCoffee3000
      - Brew.java
      - ChkTp.java
      - ChkWt.java
      - KeepTp.java
      - PreHeat.java
      - Push.java
      - RflWt.java
      - WtOK.java
  - src-gen
  - .classpath
  - .gitignore
  - .project
  - analysis-data.wf
  - analysis-data.wfconfig
  - Super Coffee 3000.di
  - Super Coffee 3000.notation
  - Super Coffee 3000.uml
  - Super_Coffee_3000_2018050...
  - superCoffee3000.coffee
  - test.json

ChkWt.java

```
1   // stub generated from 'example1.wf'
2   package SuperCoffee3000;
3
4   import SuperCoffee3000.AbstractChkWt;
5
    0 references
6   public class ChkWt extends AbstractChkWt {
7
8       @Override
        1 reference
9       protected void preExecute() {
10          System.out.prin
11
12      }
13
14      @Override
        1 reference
15      protected void post
16          // TODO Add cus
17
18      }
19
```

```
print(boolean b) : void                PrintStream
print(char c) : void
print(char[] s) : void
print(double d) : void
print(float f) : void
print(int i) : void
print(long l) : void
print(Object obj) : void
print(String s) : void
printf(Locale l, String format, Object... args)
printf(String format, Object... args) : PrintSt
println() : void
```

Problems | Call Hierarchy

Brew.java src/SuperCoffee3000 2

[java] TODO Add custom implementation here (10, 6)

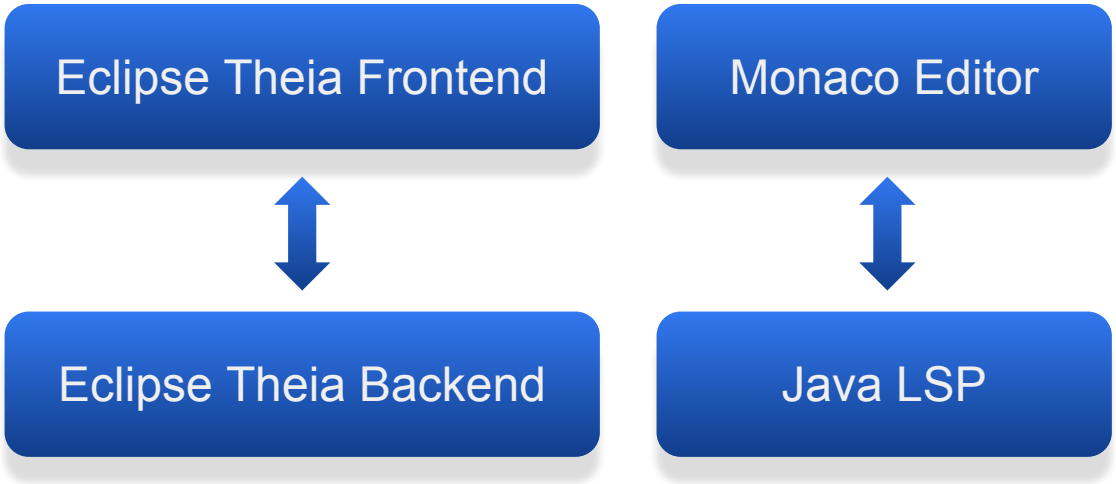
[java] TODO Add custom implementation here (16, 6)

Outline

7 0 | Ln 10, Col 24 | Spaces: 4 | Java

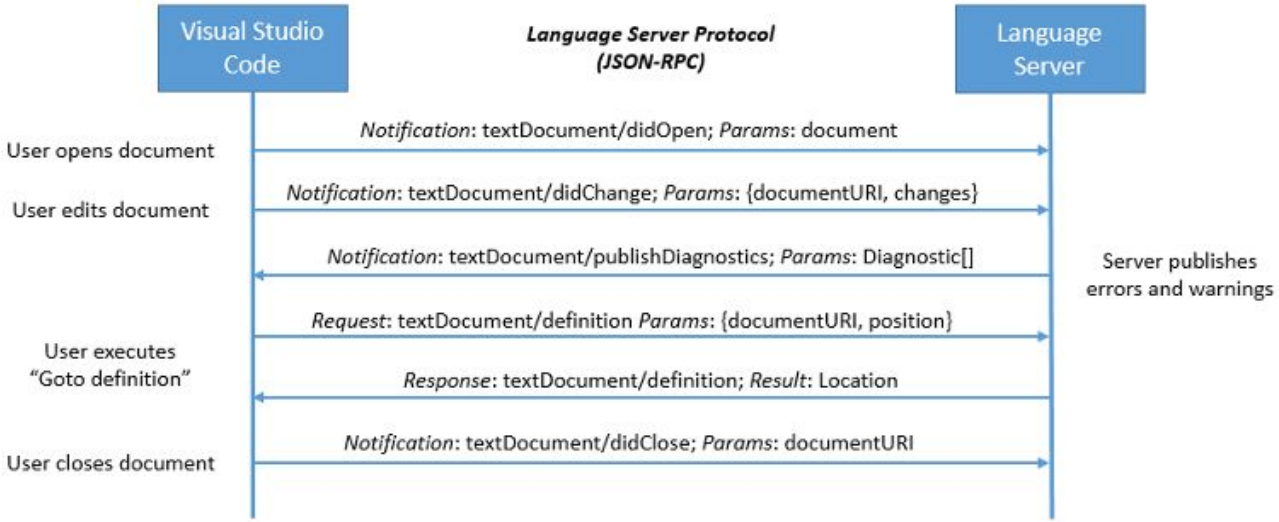
Java code editing - Implementation

- Generic LSP Client (Excursion: **L**anguage **S**erver **P**rotocol)
  - Component reuse: Monaco Editor from VSCode
  - Ships integrated into Eclipse Theia
- Java-specific LSP Server
  - Eclipse JDT Language Server
  - Based on Eclipse JDT

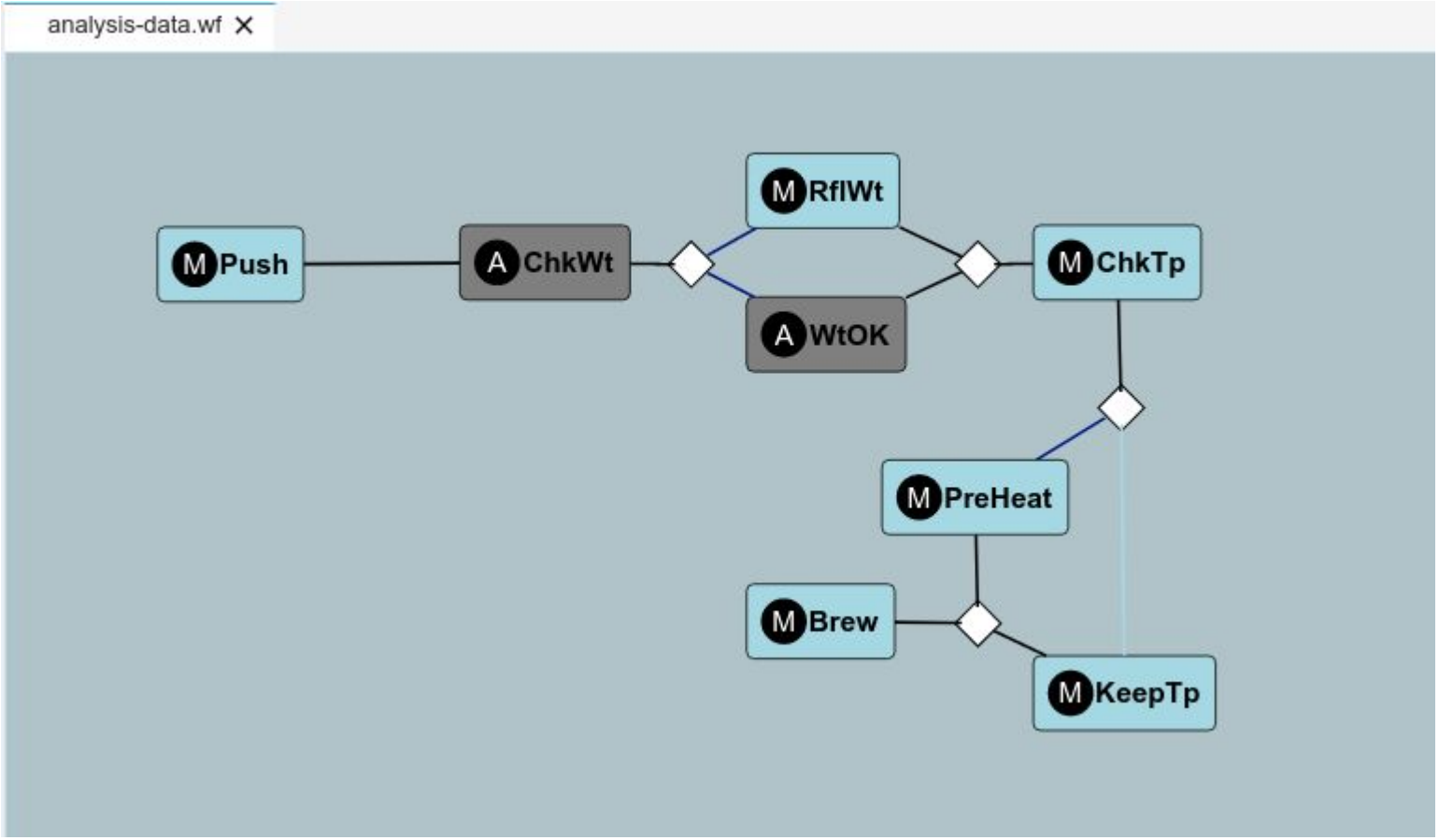


Excursion: Language Server Protocol

- 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



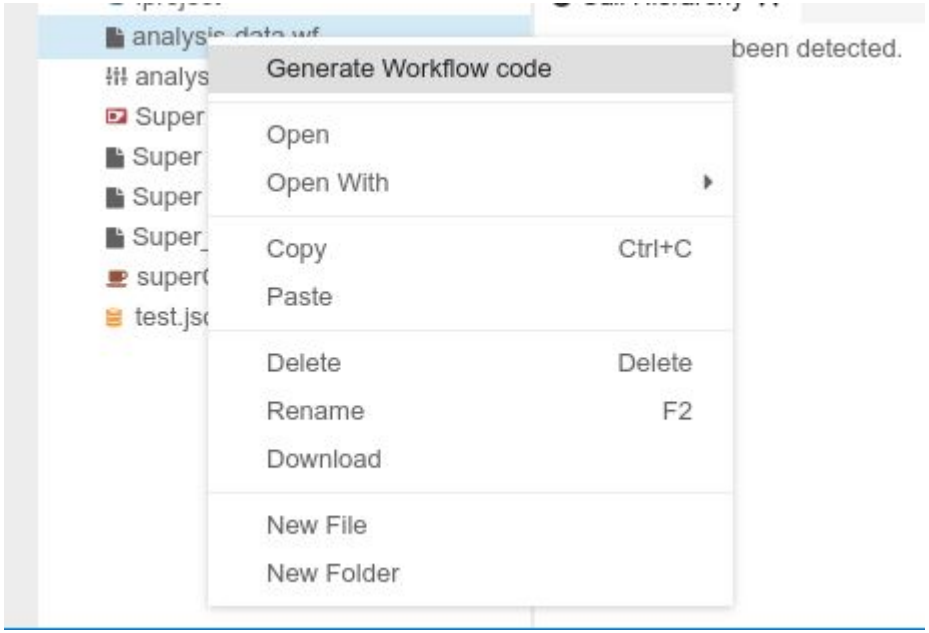
Graphical editor - Demo



Graphical editor - Implementation

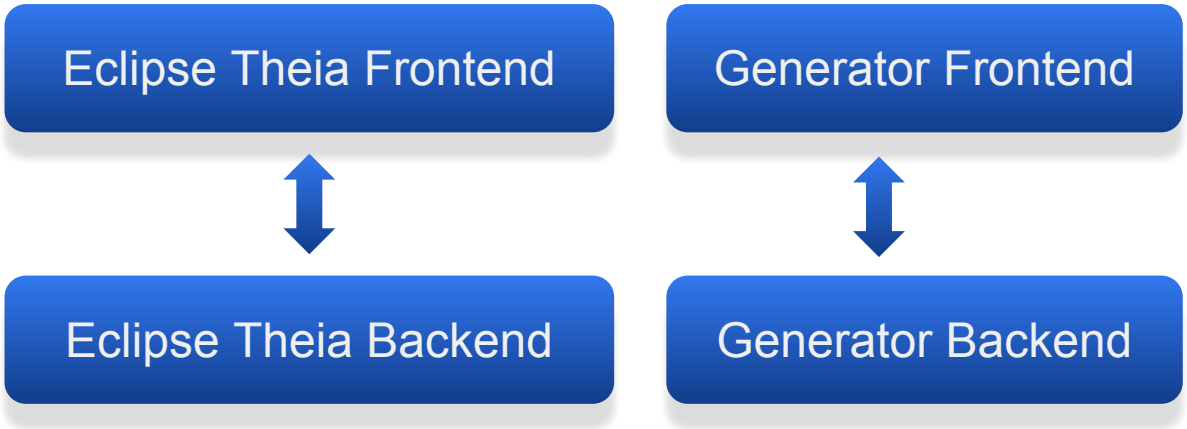
- Adopts concept of LSP for graphical editors
- GLSP Client is graphical-language agnostic
  - Editing of nodes and edges
  - Based on Eclipse Sprotty Framework
- GLSP Server is tool-agnostic
  - Reads in semantic model and layout
  - Handles requests and changes from clients

Code generation and testing - Demo



Code generation and testing - Implementation

- Frontend:
  - Menu contribution to launch generation and test run
- Backend:
  - Code generator based on Eclipse Xtend
  - Test Runner based on JUnit Test Runner
  - Wrapped into custom JSON-RPC Server



Textual editor - Demo

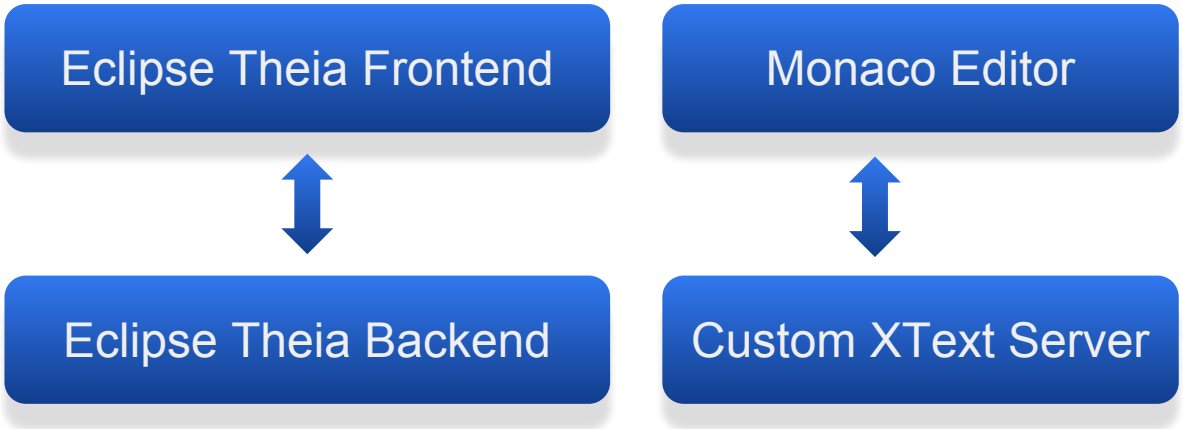
analysis-data.wf ✕ analysis-data.wfconfig ✕

```
1   workflowModel : sc3000
2
3   probabilities
4   low    : 0.1
5   medium : 0.5
6   high   : 0.75
7
8   assertions
9   ChkWt => Brew,
10  ChkTp => C
```

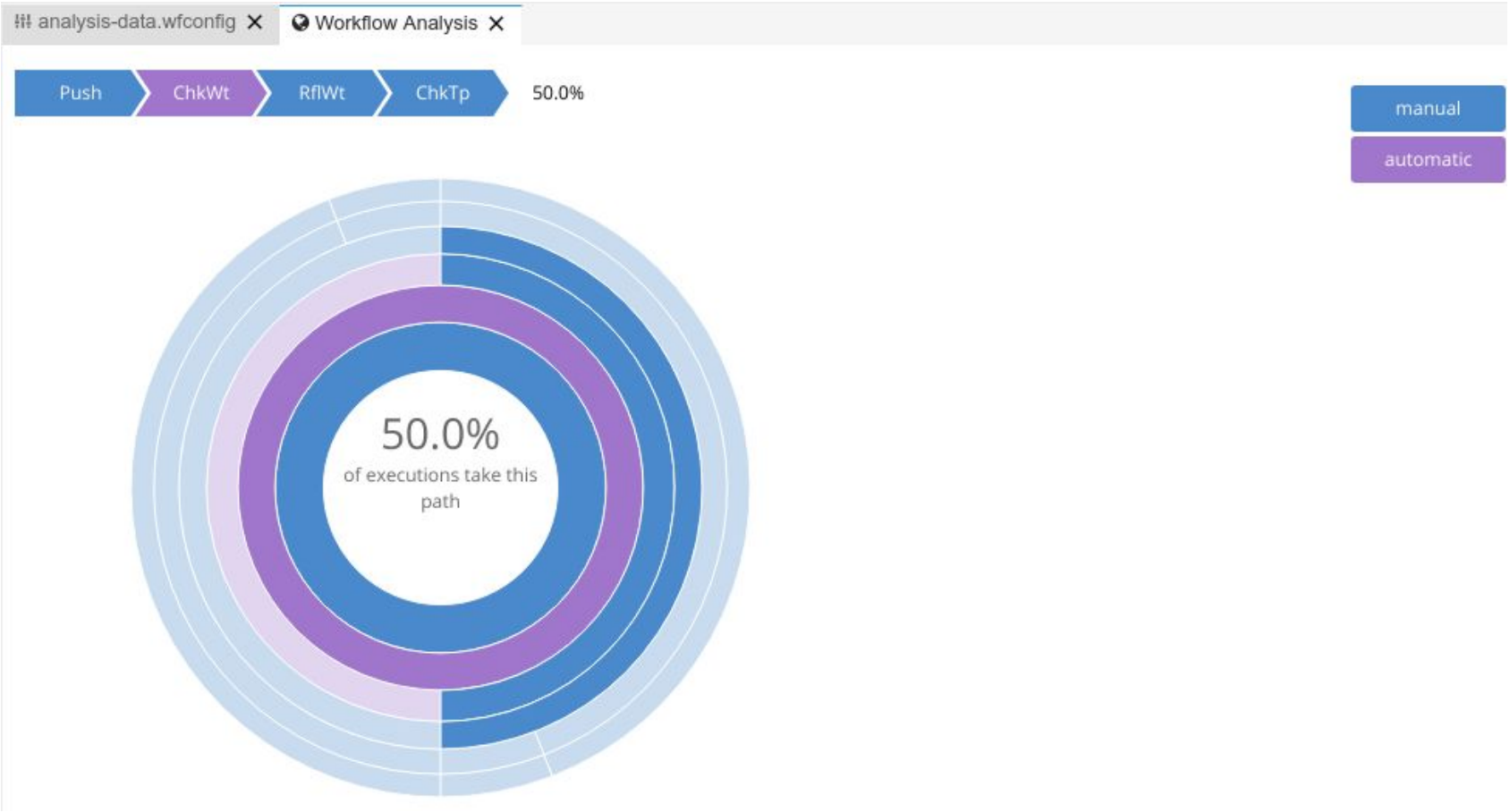
ChkTp
ChkWt

Textual editor - Implementation

- Frontend: Generic LSP Editor (Monaco)
- Backend:
  - LSP Server for the Analysis configuration language
  - Language modeled as XText grammar
  - LSP Server generated from grammer with XText tooling



Workflow analysis view - Demo

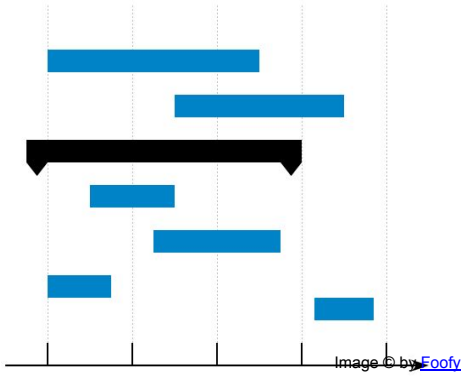


Workflow analysis view - Demo

- Frontend:
  - Menu contribution to trigger analysis
  - D3-based visualization of result from backend
- Backend:
  - JSON-RPC Server
  - Custom business logic to calculate probabilities

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
- Long-term:
  - Migrate use-case by use-case iteratively
  - Deprecate desktop-based solution



→ ECE Talk: “If, when and how? - Strategies towards web-based tooling”

Summary

- Web-based Modeling Tools are feasible
- But usually more costly
- 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
  - 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 2018

OSGi™ Community Event 2018

Evaluate this Session

Sign in and vote at eclipsecon.org

WITH

-1 0 +1