Beyond LSP

Getting Your Language into Theia and VS Code

Dr. Jan Köhnlein

@jankoehnlein
The Goal

- Custom domain-specific language
- Rich text editor support
- With diagram support

as VS Code Extension!
VS Code... Isn't this EclipseCon ???
Why VS Code Extension

• >50% IDE market share

• VS Code extensions run in Eclipse Theia
  • new default providing language support
  • Theia is the base for all the fancy new web-IDEs
VS Code Extension

Extension Config

- Manifest
- Editor Config
- Syntax Highlighting
- Grammar
- Snippets

JSON

VSIX
Extension Manifest

- NPM package.json
- extension metadata
- dependencies
- scripts
- VS Code specific extensions
- commands, keybindings, menu entries...
- activation events
- entry module
- language registration
- file extensions
- editor configuration
- highlighting grammar
- snippets

```json
// package.json
{
  "name": "states-extension",
  "displayName": "States Example",
  "description": "An Xtext-based DSL with Sprotty diagrams for statemachines",
  "publisher": "TypeFox",
  "repository": {
    "type": "git",
    "url": "https://github.com/TypeFox/sprotty-vscode"
  },
  "version": "0.0.20",
  "files": [ "syntaxes" ],
  "scripts": {
    "publish": "vsce publish"
  }
}
```

```json
"engines": {
  "vscode": "^1.46.0"
},
"categories": [ "Programming Languages" ],
"contributes": {
  "languages": [{
    "id": "states",
    "aliases": [ "states", "sm" ],
    "extensions": [ ".sm" ],
    "configuration": "./language-configuration.json"
  }],
  "grammars": [{
    "language": "states",
    "scopeName": "source.sm",
    "path": "./syntaxes/states.tmLanguage.json"
  }]
}
```
TextMate Grammar

- grammar für syntax highlighting
- uses regular expressions to map to styles

```json
// states.tmLanguage.json
{
  "name": "States DSL",
  "scopeName": "source.sm",
  "fileTypes": [ "sm" ],
  "patterns": [
    { "include": "#comments" },
    { "name": "keyword.control.states", "match": "\b(statemachine|state|event|=>)\b" },
    { "name": "keyword.operator.states", "match": "\=\>" },
    { "name": "string.quoted.double.states", "begin": "\", "end": "\"" },
  ],
  "repository": {
    "comments": {
      "patterns": [{
        "name": "comment.block.states",
        "begin": "/\*",
        "beginCaptures": {
          "0": { "name": "punctuation.definition.comment.states" }
        },
        "end": "\*/",
        "endCaptures": {
          "0": { "name": "punctuation.definition.comment.states" }
        }
      },
      ....
    },
    ....
  }
}
```
Demo: Declarative Language Services
Language Server Protocol

Language Server

Extension Code

Java/Xtend

Language Client

LSP
Using Xtext to Generate a LS

// States.xtext
grammar io.typefox.examples.theia.states.States
    with org.eclipse.xtext.common.Terminals
generate states "http://www.typefox.io/States"

StateMachine: 
    'statemachine' name=ID 
    (states+=State | events+=Event)*;

State: 
    'state' name=ID transitions+=Transition*;

Event: 
    'event' name=ID;

Transition: 
    event=[Event] '=>' state=[State];
VS Code Extension

Extension Config
- Manifest
- Editor Config
- Syntax Highlighting
- Grammar
- Snippets

Language Server
- Language Server
- Start Scripts

Extension Code
- (De-)activation
- Language Client

LSP

Java/Xtend

VSIX

TS

JSON
Extension Activation

- Registration (JSON)
  - activation events
  - entry module

- Code (TypeScript)
  - (de-)activation functions

// package.json
{
  "name": "states-extension",
  "activationEvents": [
    "onLanguage:states"
  ],
  "main": "./pack/states-extension",
  ...
}

// states-extension.ts
export function activate(context: vscode.ExtensionContext) {
  // start the language server and connect it
}

export function deactivate(): Thenable<void> {
  // disconnect and stop the language server
}
Demo
VS Code Webview

- Generic VS Code GUI component
- Separate IFRAME in the DOM
- can use arbitrary web frameworks
- runs a separate web-application
- communicates with extension via JSON objects
- Runs the diagram client app
Extension Code

Diagram Webview

JSON
Sprotty

- web-based diagramming framework
- reactive architecture
- client server separation

- We use
  - sprotty: diagram client code
  - sprotty-xtext: extend Xtext LSPs with diagrams
  - sprotty-vscode: glue code to integrate with VS Code
Sprotty – SModel

- diagram model
- serializable as JSON
- cross-references via IDs
- extensible
Sprotty – Webview application

- DI container defines
- custom services / configs
- mapping of SModel to views
- behavior using features

```javascript
// di.config.ts

const statesDiagramModule = new ContainerModule(() => {
    rebind(TYPES.ILogger).to(ConsoleLogger).inSingletonScope();
    rebind(TYPES.LogLevel).toConstantValue(LogLevel.warn);
    rebind(TYPES.IModelFactory).to(StatesModelFactory);
    ...
    configureModelElement(context, 'graph', SGraph, SGraphView, {
        enable: [hoverFeedbackFeature, popupFeature]
    });
    configureModelElement(context, 'node', SNode, RectangularNodeView);
    configureModelElement(context, 'label', SLabel, SLabelView);
    configureModelElement(context, 'edge', SEdge, PolylineArrowEdgeView);
    configureModelElement(context, 'html', HtmlRoot, HtmlRootView);
    ...
});
```
Diagram Webview

- DI config
- Views
sprotty-xtext – Language Server Integration

- extends LSP
- adds LSP notifications
  - accept(Action)
  - dispatch(Action)
- LanguageServer replaced by DiagramLanguageServer
- hooks a DiagramGenerator into Xtext model lifecycles
sprotty-xtext – DiagramGenerator

- maps Xtext model elements to Sprotty model
- on text changes
  - Xtext model is reparsed
  - new Sprotty model is created
Language Server

Diagram Language Server

Diagram Webview

DI config

Views

VS Code Diagram Server

Extension Code

Language Client

Sprotty VS Code LSP Extension

LSP

Sprotty

Java/Xtend

Language Client

Diagram Generator

TS

Extension Code

Language Client

Sprotty VS Code LSP Extension

LSP

Sprotty

Java/Xtend

Language Client

Diagram Generator

TS

Extension Code

Language Client

Sprotty VS Code LSP Extension

LSP

Sprotty

Java/Xtend
sprotty-vscode – Wiring it Up

- Glue code for using Sprotty in VS Code
  - abstracts internal communication

- VscodeDiagramServer
  - webview side
  - exchanges actions with the extension

- Sprotty VS Code Extension
  - webview management
  - extension side
  - exchanges LSP messages with the LanguageClient
  - exchanges Actions with the webview
  - converts between Sprotty LSP messages and Actions
generates new SModel

Extension Code

Language Client

Sprotty VS Code
LSP Extension

Diagram Webview

Language Server

Diagram Language Server

Diagram Generator

LSP
Sprotty

JSON

VS Code
Diagram Server

VSIX
Create UpdateModelAction

Language Server

Diagram Webview

Extension Code

Diagram Language Server

Diagram Generator

Language Client

Sprotty VS Code LSP Extension

VS Code Diagram Server

LSP Sprotty

JSON
LSP notify
dispatch(UpdateModelAction)

Language Server
- Diagram Language Server
- Diagram Generator

Extension Code
- Language Client
- Sprotty VS Code LSP Extension

Diagram Webview
- VS Code Diagram Server

VSIX
onNotify
dispatch(UpdateModelAction)

Extension Code

Language Client

Sprotty VS Code
LSP Extension

Diagram Webview

Language Server

Diagram Language Server

Diagram Generator

VS Code
Diagram Server

LSP Sprotty

JSON
sendToWebview (UpdateModelAction)

Extension Code

Language Client

Sprotty VS Code LSP Extension

Diagram Webview

Language Server

Diagram Language Server

Diagram Generator

VS Code Diagram Server

LSP Sprotty

JSON

VSIX
Execute `UpdateModelAction` as a smooth animation

Extension Code

Language Server

Diagram Webview

Language Client

Diagram Language Server

Sprotty VS Code LSP Extension

Diagram Generator

VS Code Diagram Server

LSP Sprotty

JSON
Demo: Diagram View
Development Process
Debugging

- Extension code
  - launch extension from VS Code
  - use host VS Code debugger

- Language Server
  - run language server form Eclipse in socket mode
  - use Eclipse debugger

- Webview
  - in the runtime VS Code hit F1 > Developer: Open Developer Tools
  - also shows error console
webpack – Packaging JS Applications

- standard npm tool
- package multiple JavaScript modules, CSS files, images etc. into a single JavaScript file
- minimizes result
  - removes whitespace
- tree shakes dependencies
- configuration can be tricky
  - use tsconfig.json and webpack.config.json as in the example
  - configures source maps correctly for debugging
Publishing

- VS Code Marketplace
  - ~21300 Extensions (Oct 2020),
  - closed source
  - terms of service only allow access via Microsoft products
  - No on-premise installs

- openVSX
  - Eclipse project
  - can be installed on-premise
  - public instance is run by Eclipse Foundation
  - default for VS Codium, Gitpod, Theia
Demo: Diagram Editing
Thanks for Your Attention
Links

https://theia-ide.org
https://code.visualstudio.com/api
https://microsoft.github.io/language-server-protocol
https://www.eclipse.org/Xtext
https://github.com/eclipse/sprotty
https://github.com/eclipse/sprotty-xtext
https://github.com/eclipse/sprotty-vscode
https://github.com/TypeFox/vscode-xtext-sprotty-example
https://open-vsx.org/