Software Engineering 2

PZotIDE

Version 1.0

Prof. Matteo Rossi

Alessandro Pozone - 10399281

Academic Year 2017 - 2018

POLITECNICO
MILANO 1863

Contents

Chapter 1

Introduction

1.1 Introduction

PZotIDE is a tool made to efficiently write PZot formulas.
It is composed by a text editor and a dependency graph view that allow to edit formulas or dependencies by code and in a graphical way respectively.

For more details about the PZot language we suggest to read the PZot manual.

Chapter 2

Structure

2.1 Components

PZotIDE is composed of two main parts: the IDE created using Theia and a Language Server generated using Xtext that takes care of the syntactic analysis of the code.

2.1.1 Language Server

The language server is generated from the the PZot.xtex file in the "xtext-pzot-language-server" project of the solution. XText and XTend take care of everything and generate a single jar file that contains the Language Server. This server than will communicate through JSON-RPC to the PZot Extension of Theia to provide syntactic analysis.
In the "xtext-pzot-extension" project there are also some scripts that copy the generated jar to the extension.

2.1.2 Theia

Theia is an extensible platform to develop full-fledged multi-language Cloud & Desktop IDE-like products with state-of-the-art web technologies. It is designed to work as a native desktop application as well as in the context of a browser and a remote server. To support both situations with a single source, Theia runs in two separate processes. Those processes are called frontend and backend respectively, and they communicate through JSON-RPC messages over WebSockets or REST APIs over HTTP. In the case of Electron, the

backend, as well as the frontend, run locally, while in a remote context the backend would run on a remote host.
Both the frontend and backend processes have their dependency injection (DI) container to which extensions can contribute.

2.1.3 PZot Extension

The PZot features of the IDE are provided trough the PZot extension. it is composed by a backend extension that initialize the Language Server and that connects to it and by a frontend component that offers the "user visible" features.
Using the dependency injection the following modules are added to the Theia core:

Backend

- LanguageServerContribution

Frontend

- LanguageClientContribution
- PZotUri
- PZotDependencyGraphOpenHandler
- PZotGraphResourceResolver

LanguageServerContribution

As said before here we take care of starting the Language Server and managing the connection

LanguageClientContribution

This module communicates directly with the text editor component offered by Theia, Monaco Editor. We define the colors of the tokens and some special rules like the allowed parentheses in the file that have a .pzot extension. Here we define also the snippets, portions of code that are automatically suggested to the user when they can be useful. In PZot these allow us not to have to write all the symbols of the original language.

PZotUri

This class is useful to understand when we are actually dealing with a PZot file. Theia can actually open any text file but the features of the PZot extension must be available only for files where they are meaningful.

PZotDependencyGraphOpenHandler

Here we tell Theia that the PZot files can be opened also in another way, other than the textual, using a different widget where we will display the actual graph. When this class gets binded to Theia using the Dependency Injection mechanism a new menu option is added in the "Open With" menu of the PZot files. From a code point of view here we tell Theia what kind of files we can manage with this widget and we have the logic that actually creates the widget (PZotDependencyGraphWidget).

PZotGraphResourceResolver

This is a very simple class that basically creates a PZotGraphResource when needed.
The actual flow is that every time someone opens a PZot file using our custom widget, a PZotGraphResource get created using the PZotGraphResourceResolver and then is passed to the graph widget.

PZotGraphResource

This class represents the connection between the document and the graph: here we have the events that are triggered when the document changes and the methods to save our changes to the file. We also have many methods that parse the dependency part of the formula to an Array of PZotGraphItem.
Every PZotGraphItem represents the first item of a dependency clause, the items that depend from it are in the "children" field. All the items of the array are to read as a conjunction.

PZotGraphEngine

Here is the core of the graph representation, where it gets actually rendered. At this level the dependencies are represented by a PZotGraph object that is composed by nodes and edges. Thank to the open source library Cytoscape.js we easily represent the graph. All the nodes are structured in columns, each one representing a time "period".
When a new dependency (PZotGraphItem) is added to the graph it gets analyzed and the necessary nodes and edges are created. Also the upper and lower bound on the period used by this formula are computed. The idea is to render only the necessary periods in order to ease the visualization. Once the graph has all the data it needs, the periods get "normalized" (so to start from 0 and have non negative periods). When the PZotGraph object is passed back to the PZotGraphResource the periods are computed back to the originals.

Chapter 3

Compiling & Usage

3.1 Compiling

To compile the solution first we have to build the Language Server whose executable will be copied into the PZot extension.

```
cd xtext-pzot-language-server &&
./gradlew shadowJar &&
cd ..
```

Then we can build and start the rest of the project issuing the following commands:

```
yarn install &&
yarn rebuild:electron &&
cd pzot-electron &&
yarn start
```

At this point the Electron hosted application will start. Also a browser version of PZotIDE can be built using:

```
yarn install &&
yarn rebuild:browser &&
cd pzot-electron &&
yarn start
```

3.2 Usage

This is the main screen of a PZot file:

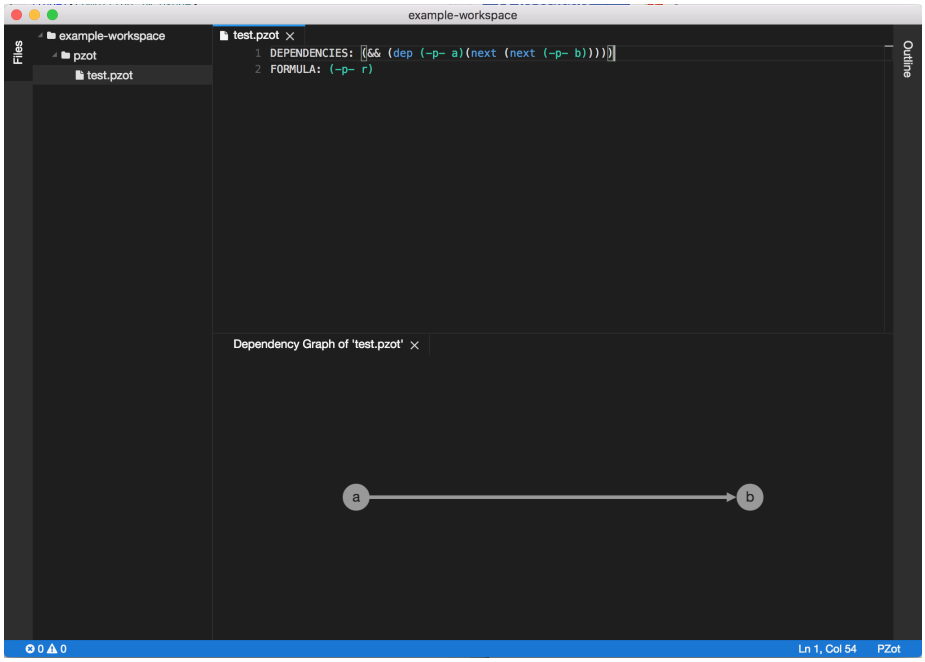

Figure 3.1: Main Screen

On the left side we have the "workspace", a representation of the opened directory with all the files. By clicking the "Files" tab it gets collapsed to give more space to the editor. On the right we have the the text editor on top (Monaco Editor) and the graph widget in the lower part. The widget will appera by right clicking a PZot file and choosing "Open With" and then "Open Dependency Graph".

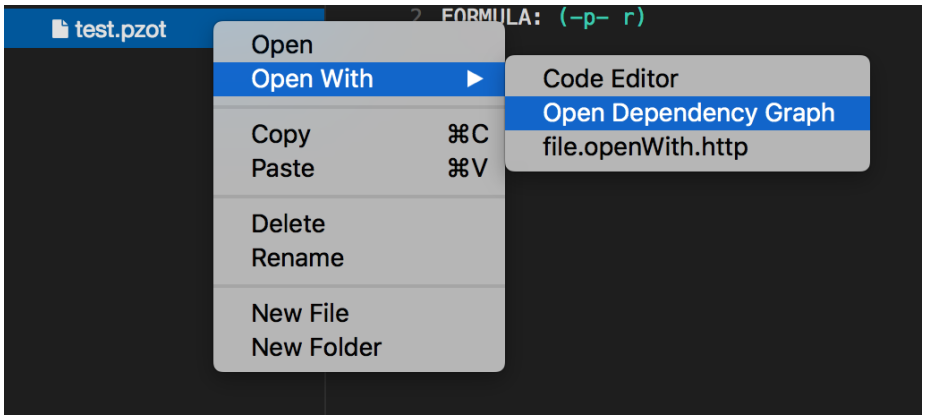

Figure 3.2: Open With Menu

The blu bar at the end of the editor shows the number of errors on the left and the current line and column on right. If we click on the error symbol, the problems panel will open providing us with mode details.

3.2.1 Text Editor

The text editor makes use of the rules defined by the PZot Extension so it will provide auto-complete, color the tokens and offer snippets of code. Some example of the latter are "and" that inserts the formula: "(&& condition condition)" and automatically moves the cursor to the first "condition". Using the tab key we can efficiently move throughout the various "conditions".
Another really useful snippet is "litteral" that adds all the semantic necessary to declare a litteral.

3.2.2 Dependecy Graph Editor

Every time we modify the dependency layer of the formula in the Text Editor the graph gets updated. The columns represent the times of the clauses so it may happen to have the same litteral in more columns. We can also modify the dependecy layer by the graph, the following actions are available:

- **Add New Node**

- **Add New Edge**
- **Remove Node**
- **Remove Edge**
- **Move Nodes Between Periods**

Add New Node

To add a new node we can simply right click on the graph. The new node will have the "period" of the column were it is created.

Add New Edge

To add a new edge we have to hover the pointer on the starting node until a red dot appears. Then we can drag an edge from that dot to the desired node.

Remove Node

To remove a node we just need to right click it. All the edges that belonged to that node are also removed.

Remove Edge

Same procedure as per nodes.

Move Nodes Between Periods

To change the period of a node, we can drag and drop it to desired column.

Chapter 4

Software & Languages

4.1 Software

- **TheiaIDE** - https://github.com/theia-ide/theia
- **Monaco Editor** - https://microsoft.github.io/monaco-editor/
- **XText** - https://www.eclipse.org/Xtext/
- **Electron** - https://electronjs.org/
- **Cytoscape.js** - http://js.cytoscape.org/

4.2 Languages

The project makes use of Xtex language for the definition of the grammar of the Language Server and Typescript in the IDE.