1 Cheat Sheet (TMP)

Dieses CheatSheet wird später wieder entfernt.
Hier ist Text.

Jetzt ist Text nach so nem Doppelslashteil.
Neuer Absatz.
Neue Zeile. Hehe, wo bin ich?

1.1 Untertitel

1.1.1 Das ist die tiefste Titelebene

Ich bin Text.

Figure 1: My caption

Davor ein Bild.
Mehr dazu in Abbildung 1.

1.2 Quellen

Und das wäre ein zweiter Absatz [1].
Wie einer auf 20min sagte:[2]

> Immer mehr europäische Länder verhängen im Kampf gegen das Virus eine Ausgangssperre.

Beachten sie die Fussnote [1]

1.3 Aufzählung

- Erstens
- Zweitens

1. Erstens

[1] Ich bin die Fussnote

2. Zweitens

Erstens

Zweitens

1.4 Tabelle

Col1	Col2	Col2	Col3
1	6	87837	787
2	7	78	5415
3	545	778	7507
4	545	18744	7560
5	88	788	6344

Figure 2: My table

Das war also Tablle 1.4.

1.5 Code

Fogelnd Code in C#

```
public void SendInformation(string msg)
{
    SendMessage("INFO", msg);
}
```

Listing 1: My Caption

Da wär jetzt so code, siehe Listing 1

Dafny Language Server
Bachelor Thesis

Department of Computer Science
University of Applied Science Rapperswil

Spring Term 2020

Authors:	Marcel Hess, Thomas Kistler
Advisors:	Thomas Corbat (lecturer), Fabian Hauser (project assistant)
Expert:	Guido Zgraggen (Google)
Counter reader:	Prof. Dr. Olaf Zimmermann

2 Abstract

Table of Contents

3 Introduction

2do: iwo den Satz "Zielgruppe die HSR Studenten" einbauen. "Messbarkeit von Erfolg."

3.1 Problem Domain

3.2 Relevance

- ...
- ...

3.3 Outlook

Das ist ein Absatz [1].

Und das wäre ein zweiter Absatz. [1]

4 Analysis

4.1 Continuous Integration (CI)

Continuous integration is a verry important part for code quality improvement and colaboration. Unfortunately, the CI process in our student research project extended to almost the entire semester [3].

According to our project plan, we wanted to work on open points regarding the CI initially and have completed the theme accordingly for the remaining duration of the bachelor thesis.

4.1.1 Initial Situation

We achieved in our client CI that code was analyzed with SonarQube and the build failed if it contained TypeScript errors [3]. We did not achieve it within reasonable time headless integration test [3].

On the server side we reached the build process as well as the dafny tests and our own unit tests [3]. Automated integration tests and code analysis by SonarQube remained outstanding [3].

4.1.2 Aimed Solution

According to our research, a major problem was that the scanner for sonarqube does not support any other languages besides C# [4]. This means that in addition to C# in a project, TypeScript (for the client) cannot be analyzed simultaneously. Furthermore there are also single Java files in Dafny project. This also led to conflicts in the Sonar analysis in our student research project [3].

As a simple solution we decided to separate the client (VSCode plugin) and server (Dafny Language Server) into two separate git repositories. This not only simplifies the CI process but also ensures a generally better and clearer separation.

As a result, the client could still be easily analyzed with the previous Sonar scanner. For the Language Server in C# a special Sonar scannerfor MSBuild had to be used, which publishes the analysis in a separate SonarCloud project [2]. Beside the code from our Lanugae server the whole Dafny project code is now analyzed by sonar. This can be very helpful for code reviews.

The only downside is that the code coverage is not analyzed. For .NET OpenCover is a very common tool for code coverage analysis. Unfortunately, it only works on windows and not on our linux CI server [5]. Other tools that works with mono Support .NET Core but not Framework. During our research we came across monocov [6]. This tool would support mono for .NET Framework. Unfortunately this project was archived and has not been supported for almost 10 years [6].
Since we would not gain much added value with sonar code coverage, we decided not to pursue this approach any further. The coverage information is provided by the locally installed IDEs anyway.

For an easier testability of the CI, we now also used local docker. This allows us to test CI customizations efficiently. See the developer documentation for more details [2].

The headless integration tests were a bit more tricky. In consultation with our supervisor, we have removed these tests from the client project and replaced them with own specially written integration tests on the server side.

4.1.3 2do - Kapitelaufteilung komisch

Ich hab hier jetzt in der Analyse auch schon die Lösung vorabgegriffen. Sollen wir das splitten? Bricht das nicht den Lesefluss? Evt besprechen.

5 Design

This chapter contains discussions of fundamental design decisions. It is primarily divided into client and server architecture.

5.1 Client

The client part of our bachelor thesis includes the VSCode plugin written in TypeScript. It starts the Language Server and establishes a connection via LSP.

Since most of the logic should be kept in the IDE independent Language Server, we have made it our goal to keep the client logic to a minimum. This allows to implement a later plugin for another IDE with as little effort as possible.

In the following we will discuss how we achieved this lightweight, and how and why we decided to split the components in the client part like we did.

5.1.1 Initial Situation

Already in our term project we made revisions on the client. We have connected our new Language Server and greatly reduced the unnecessary logic. The following figure 3 gives an impression of the architecture.

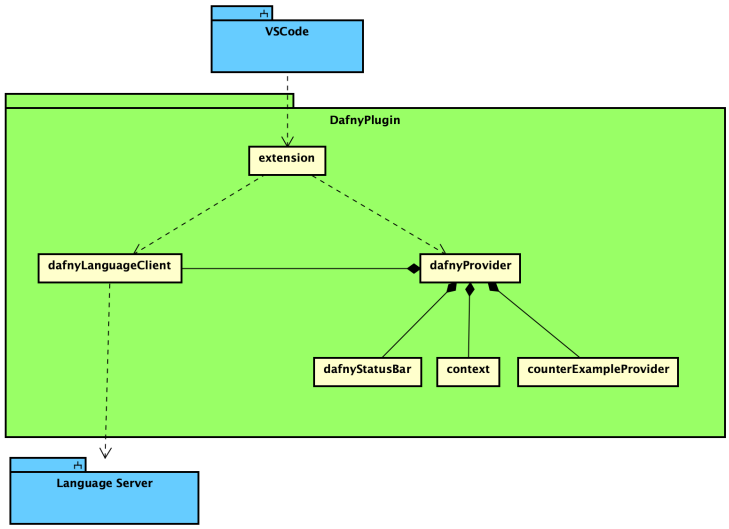

Figure 3: Client Architecture - Term Project

In this simplified representation, the client architecture appears very tidy. Unfortunately, the individual components are very large, almost all members are public and this leads to high coupling and deep cohesion. Furthermore, there are many helper classes that are not grouped into sub-packages. This makes it difficult for other programmers to get into the project. Furthermore, it was difficult to eliminate all dead code due to the non-transparent dependencies.

Because of these problems we decided to redesign the client itself from scratch.

5.1.2 New Architecture

To achieve the goal of a more manageable architecture and to reduce coupling, we have implemented the following measures. As a first step, we aimed to divide all areas of responsibility into separate components. We grouped the components into packages as you can see in figure 4. These packages are discussed in the following sections.

Additionally we detached all logic from the extension class (the main component). This resulted in the root directory containing only a lightweight program entry point and the rest of the logic was split between the created packages.

As a little extra, each component contains code documentation to help other developers get started quickly. This is also helpful because they are displayed as hover tooltips.

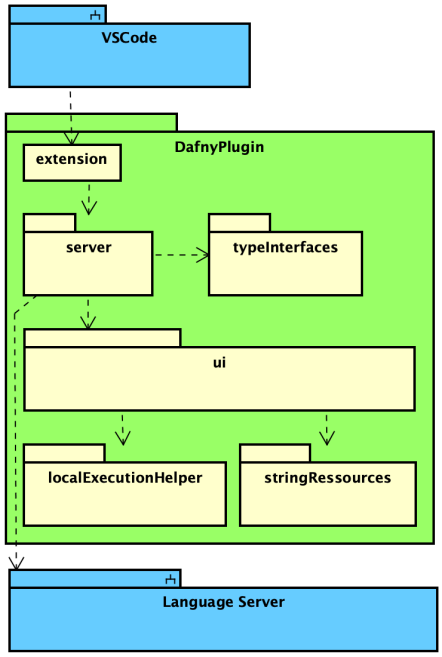

Figure 4: Client Architecture - New Packages

Extension – This component is the aforementioned "main" of the plugin and serves as an entry point. The contained code has been minimized. Only one server is instantiated and started. The logic is located entirely in the server package.

Server – The server package contains the basic triggering of the Language Server and the connection setup. In addition, all server requests, which extend the LSP by own functions, are sent to the server via this package.

TypeInterfaces – In our new architecture we do not use the "any" type. We decide all types, in particular the types created specifically for additional functions such as results for counter examples.

UI – The UI is responsible for all visual displays. Especially VSCode commands and context menu additions. Core components like the status bar are also included in this package.

LocalExecutionHelper – This package contains small logic extensions like the execution of compiled dafny

files. The UI package accesses this package.

StringRessources – All text strings and command definitions are defined in this package. This package is used by the UI package.

In the following chapters the individual components and their contents are described in more detail.

5.1.3 Components

In the following figure 5, the components within the packages are also shown for a more detailed view. The contents of type interfaces and string resources have been omitted for clarity.

It can be seen that only compile and counterexample exist as server accesses. All other features were implemented purely through the LSP protocol without additional client logic as support. This server-side implementation via LSP is a great enrichment. For future plugin developments for other IDEs the effort is automatically eliminated.

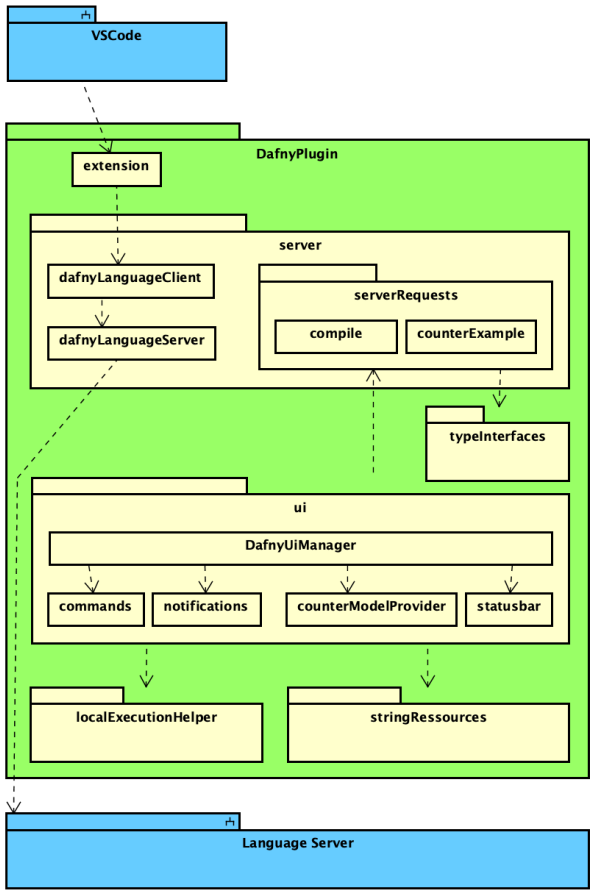

Figure 5: Client Architecture - Components

Figure 6 shows the public methods for the components. All instance variables were set to private. Constructors were not included for simplicity. The contents of type interfaces and string resources were also omitted for clarity

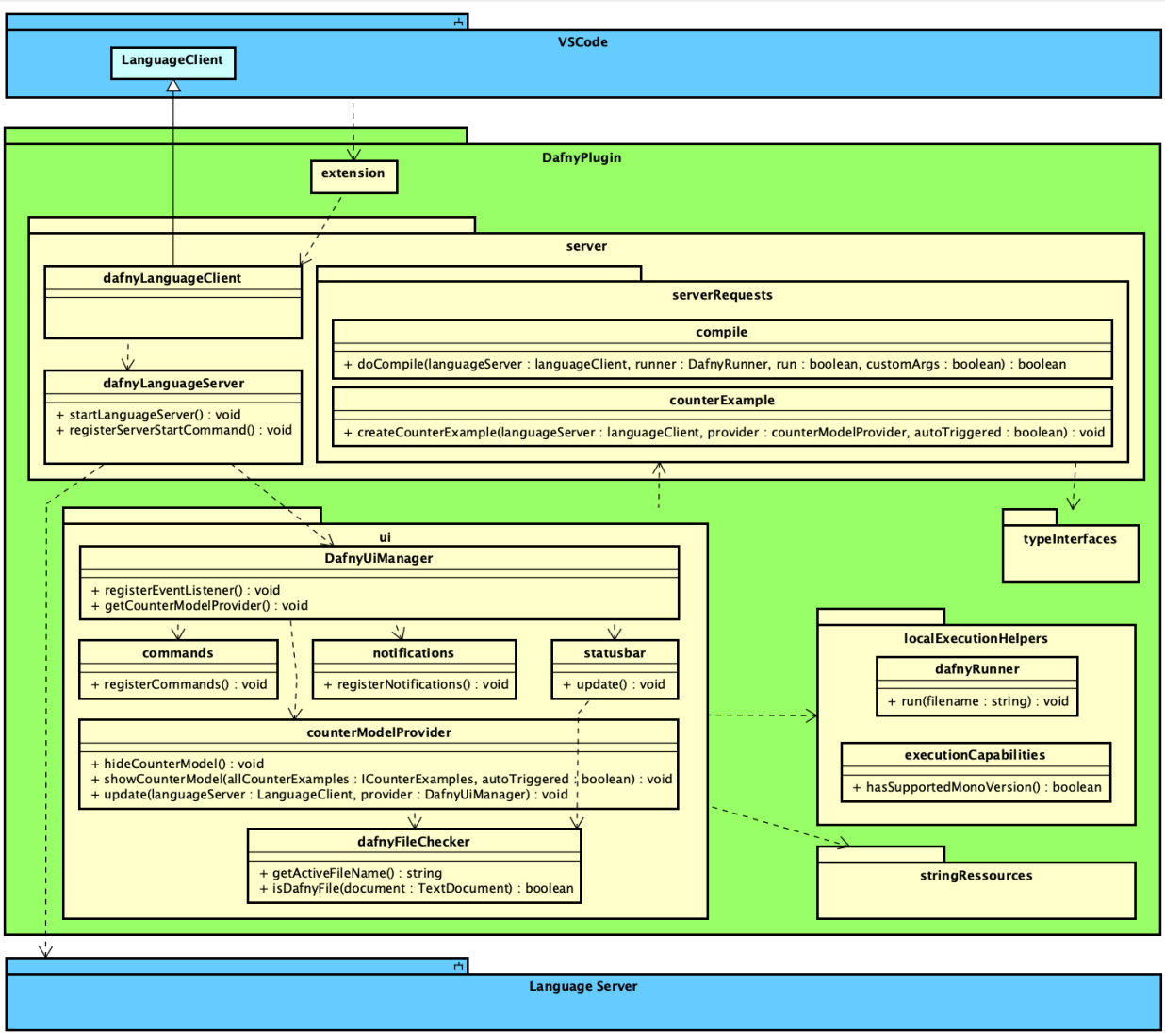

Figure 6: Client Architecture - Components and Public Methods

There is also a compromise that we have made. This distribution has certain upward dependencies, which is not perfect. The UI package accesses the server requests to be exact. Nevertheless, we have decided on this grouping, so that the server access functionality is encapsulated.

5.1.4 Logic

The logic contained in the client has been reduced to a minimum. This mainly with the goal of making porting to other IDEs as easy as possible. This subchapter describes in rough outlines the logic still contained in the client and its benefits.

Server Connection – Starting the Language Server and send API requests. In addition, the client has a simple logic that certain server requests (such as updating the counter example) are sent a maximum of twice per second.

Execute Compiled Dafny Files – The execution of compiled Dafny files is relatively simple. One distinguishes whether the execution of .exe files should be done with mono (on macOS and Linux operating systems) or not.

Notifications – The client allows the Language Server to send certain messages which are displayed directly to the user as a popup. These include the types information, warning and error. The corresponding logic on the client side includes the registration of the interception on the server and the corresponding method calls of the VSCode API to display the messages.

Commands – To enable the user to actively use features (such as compile), the corresponding method calls must be linked to the UI. We have three primary links for this: Supplementing the context menu (right-click), shortcuts and entering commands via the VSCode command line.

Statusbar – The information content for the Statusbar is delivered completely by the Language Server. The client only takes care of the user friendly display with icons and help texts. Therefore certain event listeners must be registered, which react to the server requests. Furthermore the received information is buffered for each Dafny file. This allows the user to switch seamlessly between Dafny files in VSCode without having the server to send the status bar information (like the number of errors in the Dafny file) each time.

Counter Example – The Counter Example has a similar buffer as the Statusbar. For each dafny file in the workspace a buffer stores if the user wants to see the Counter Example. This way the Counter Example is hidden when the user switches to another file and automatically shown again when switching back to the previous Dafny file.

6 Implementation

7 Result

8 Conclusion

9 Project Management

Glossary

IDE Eine integrierte Entwicklungsumgebung für die Herstellung von Software. 14

References

[1] Rafael Krucker and Markus Schaden. *Visual Studio Code Integration for the Dafny Language and Program Verifier*. `https://eprints.hsr.ch/603/`. HSR Hochschule für Technik Rapperswil, 2017.

[2] Marcel Hess and Thomas Kistler. *Developer Documentation*. HSR Hochschule für Technik Rapperswil, 2020.

[3] Marcel Hess and Thomas Kistler. *Dafny Language Server Redesign*. HSR Hochschule für Technik Rapperswil, 2019/20.

[4] *SonarCloud for C# Framework Project*. URL: `https://community.sonarsource.com/t/sonarcloud-for-c-framework-project/17132`. (Accessed: 23.03.2020).

[5] *OpenCover*. URL: `https://github.com/OpenCover/opencover`. (Accessed: 23.03.2020).

[6] *monocov*. URL: `https://github.com/mono/monocov`. (Accessed: 23.03.2020).

Anhang (Entwickler Doku)