Freie Universität Berlin

Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Arbeitsgruppe Software Engineering

Implementierung des Saros-Plugins für Visual Studio Code

Michael Schäfer
Matrikelnummer: 4361169
michael.schaefer@fu-berlin.de

Betreuer: Kelvin Glaß
Eingereicht bei: Prof. Dr. Lutz Prechelt
Zweitgutachter: Prof. Dr. Claudia Müller-Birn

Berlin, 14. August 2020

Zusammenfassung

Saros ist eine Erweiterung für Entwicklungsumgebungen, die initial für Eclipse entwickelt wurde und mittlerweile auch für IntelliJ in einer Alpha Version zur Verfügung steht. Sie ermöglicht die verteilte Paarprogrammierung, wodurch die beteiligten Entwickler nicht mehr physisch am gleichen Ort sein müssen um zusammen zu Programmieren. Visual Studio Code hat seit seiner Veröffentlichung stetig an Popularität gewonnen und durch Microsoft sogar einen eigenen Ableger der verteilten Paarprogrammierung bekommen, Microsoft Live Share. Da Visual Studio Code durch Erweiterungen nahezu jede Programmiersprache unterstützen kann und dazu noch populär ist, liegt es nahe Saros auch für diese Plattform bereitzustellen, um von einem erweiterten Nutzerkreis und Anwendungsmöglichkeiten zu profitieren. Diese Arbeit beschreibt die Entwicklung eines Prototyp, der zeigen soll, dass Saros auch auf dieser Plattform funktioniert und potenzial hat. Desweiteren soll die Grundlage für eine Weiterentwicklung gelegt werden.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

14. August 2020

Michael Schäfer

Inhaltsverzeichnis

1 Einführung **7**
1.1 Was ist Saros? . . . 7
1.2 Motivation . . . 8
1.2.1 Popularität . . . 8
1.2.2 Konkurrenz . . . 8

2 Anforderungen **9**
2.1 Ziel der Arbeit . . . 9
2.2 Funktionsübersicht . . . 9
2.2.1 G1: Konfiguration . . . 10
2.2.2 G2: XMPP - Kontaktverwaltung . . . 10
2.2.3 G3: XMPP - Sitzungsverwaltung . . . 11
2.2.4 G4: XMPP - Allgemein . . . 11
2.2.5 G5: Saros . . . 11
2.3 Abgrenzung . . . 12

3 Visual Studio Code **12**
3.1 Allgemein . . . 12
3.2 Die Extension API . . . 13
3.2.1 Was kann angepasst werden? . . . 13
3.2.2 Was kann nicht angepasst werden? . . . 15
3.3 Die Proposed API . . . 15
3.4 Contribution Points . . . 16
3.4.1 configuration . . . 16
3.4.2 commands . . . 16
3.4.3 menus . . . 16
3.4.4 views . . . 17
3.4.5 viewContainers . . . 17
3.5 Namespaces . . . 18
3.5.1 window . . . 18
3.5.2 workspace . . . 19
3.6 Verheiratung mit Java . . . 19
3.6.1 Möglichkeit 1: Einbettung von Java in Typescript . . . 19
3.6.2 Möglichkeit 2: Saros als eigenständigen Prozess . . . 20

4 Language Server Protocol **20**
4.1 Das Protokoll . . . 22
4.2 Verbreitung . . . 23
4.3 Implementierungen . . . 23

5 Umsetzung **23**
5.1 Vorgehen . . . 23
5.2 Visual Studio Code . . . 24
5.2.1 Anbindung des SLS . . . 24
5.2.2 Ordnerstruktur . . . 24

5.2.3 Warum wurden Ansichten implementiert? 25
5.2.4 Die Assistenten 26
5.2.5 Logging 27
5.2.6 Das Arbeitsbereich Problem 27
5.2.7 Annotationen 27
5.2.8 Gradle 28
5.3 Saros Language Server 28
5.3.1 Konflikte mit anderen Language Servern 28
5.3.2 Das Saros Protokoll 29
5.3.3 Proxies 30
5.3.4 Saros Response 30
5.3.5 Gradle 31

6 Zusammenfassung **31**
6.1 Ergebnis 31
6.2 Ausblick 31
6.2.1 Unabhängigkeit zur Zielumgebung 31
6.2.2 Debug Adapter Protocol 32

Literaturverzeichnis **33**

1 Einführung

1.1 Was ist Saros?

Saros ist eine Erweiterung für Entwicklungsumgebungen, die es ermöglicht die Paarprogrammierung unabhängig vom Standort der beteiligten Programmierer durchzuführen. Initial wurde Saros für die Entwicklungsumgebung Eclipse[1] entwickelt und weist somit den größten Funktionsumfang auf. Darüberhinaus existiert auch eine Saros Umsetzung für die IntelliJ Plattform, nachfolgend nur noch IntelliJ genannt, in einer Alpha Version.

In der klassischen Paarprogrammierung sitzen beide Programmierer zusammen vor dem Rechner und entwickeln gemeinsam eine Lösung zu einem Problem. In der heutigen vernetzten Welt kommt es jedoch immer häufiger vor, dass sich die Programmierer an unterschiedlichen Orten auf der ganzen Welt aufhalten. Dieser Umstand erschwert das physische Zusammenkommen für die paarweise Programmierung, hier kommt nun die verteilte Paar-Programmierung (engl. "Distributed Pair Programming", kurz "DDP") zum Einsatz. Hierbei müssen die Teilnehmenden nicht mehr physisch vor dem selben Rechner sitzen, sondern arbeiten von verschiedenen Orten aus zusammen.

Es gibt natürlich auch Alternativen zu Saros, um verteilt in Paaren zu Programmieren, zum Beispiel könnte ein Teilnehmer über weitere Programme seinen Bildschirm teilen, und je nach Programm kann der Partner auch aktiv den Quelltext über Eingaben verändern. Eine gleichzeitige Bearbeitung wird allerdings in den meisten Fällen nicht möglich sein, da sich der Partner (bei dem gearbeitet wird) die Eingabegeräte mit dem anderen in den meisten Fällen teilen wird. Ein weiterer Nachteil ist auch, dass unter Umständen mehr vom Rechner freigegeben wird als beabsichtigt. Das kann schon damit beginnen, dass der Partner den Bildschirmhintergrund unter Umständen sehen kann oder Emails unbeabsichtigt geteilt werden, wenn diese zum Beispiel über ein Vorschaufenster angekündigt werden. Im schlimmsten Fall werden aber unbeabsichtigt Funktionen des Rechners freigegeben, wie zum Beispiel der Dateizugriff (zum Beispiel bei der Verwendung von Teamviewer[2]). Auch wenn diese Probleme vermieden würden, kann immernoch die Verfügbare Bandbreite die Paarprogrammierung behindern. In den meisten Fällen wird nämlich der gesamte Desktop übertragen, und nicht nur der relevante Ausschnitt (Bild) bzw. die relevanten Daten (Text), wodurch viel Bandbreite verschwendet wird.

Saros verfolgt einen anderen Ansatz. Anstatt den gesamten Bildschirm zu teilen, werden nur die benötigten Daten (der Quelltext) und die Aktionen (z.B. Textänderungen, Selektionen, Dateierstellung etc.) geteilt. Dafür verwendet Saros das Extensible Messaging and Presence Protocol (XMPP), bei dem die Daten im XML[3]-Format ausgetauscht werden. Da nur für die Paarprogrammierung relevante Daten übertragen werden müssen, hat dies den Vorteil dass Bandbreite eingespart werden kann gegenüber Alternativen, die den gesamten Bildschirm übertragen. Damit Saros funk-

[1] https://www.eclipse.org/ide/
[2] https://www.teamviewer.com/de/
[3] Extensible Markup Language

tionieren kann, wird neben der Entwicklungsumgebung, in der Saros läuft, auch ein XMPP-Server benötigt. Der Einsatz des Servers hat den Vorteil, dass der gesamte Datenstrom auch über einen eigenen bereitgestellten Server laufen kann, und man somit die volle Kontrolle über seine Daten hat.

1.2 Motivation

Saros gibt es, wie bereits erwähnt, bereits für Eclipse und in einer frühen Version auch für IntelliJ. Warum sollte man also eine weitere Baustelle aufmachen? Schließlich gibt es durchaus noch genug zu tun bei Eclipse und IntelliJ.

1.2.1 Popularität

Zum einen ist Visual Studio Code, nachfolgend nur noch VSC genannt, in den letzten Jahren populärer geworden. Laut der jährlichen Umfrage der Webseite Stackoverflow[4] hat VSC im Jahr 2019 im Vergleich zum Vorjahr[5] um 15,8% an Popularität zugelegt und belegt mit 50,7%[6] den ersten Platz der beliebtesten Entwicklungsumgebungen 2019. Eclipse liegt 2019 auf Platz 8 und IntelliJ auf Platz 4. Die Umfragen geben allerdings nicht wieder, wie hoch der tatsächliche Einsatz der Entwicklungsumgebungen im Alltag der Programmierer ist, geben aber eine gute Tendenz zu erkennen. Zumal je nach verwendeter Sprache durchaus andere Entwicklungsumgebungen zum Einsatz kommen können, weil diese sich z.B. einfacher integrieren lassen oder mehr Funktionen für eine Programmiersprache bereitstellen. Dies wird insbesondere dann deutlich wenn noch andere Quellen herangezogen werden, wie z.B. die Anzahl an Google Suchen[7], hierbei wäre VSC auf Platz 4 zwischen Eclipse (Platz 2) und IntelliJ (Platz 6). Nichtsdestotrotz ist VSC eine gewisse Popularität nicht abzusprechen.

Entwicklungsumgebung	2018	2019	Zuwachs im vgl. zum Vorjahr
`Visual Studio Code`	34,9%	50,7%	15,8%
`Eclipse`	18,9%	14,4%	-4,5%
`IntelliJ`	24,9%	25,4%	0,5%

Tabelle 1: Anteile der Entwicklungsumgebungen laut Umfragen auf Stackoverflow[5][6]

1.2.2 Konkurrenz

Microsoft ist der Entwickler und Herausgeber der Entwicklungsumgebungen Visual Studio Code und Visual Studio. Schon allein aufgrund der Größe hat Microsoft mehr Kapazitäten seine Lösung der DPP voranzutreiben und zu verbreiten. Mit Live Share hat Microsoft bereits eine Lösung veröffentlicht mit dem verteilte Paarprogrammierung möglich ist und aus meiner Sicht der stärkste Konkurrent zu Saros ist.

Am 7.5.2018 wurde Live Share der breiten Öffentlichkeit vorgestellt[8] und kommt mit

[4] https://stackoverflow.com
[5] https://insights.stackoverflow.com/survey/2018#development-environments-and-tools
[6] https://insights.stackoverflow.com/survey/2019#development-environments-and-tools
[7] https://pypl.github.io/IDE.html, Stand 13.8.2020
[8] https://code.visualstudio.com/blogs/2018/05/07/live-share-public-preview

einer Fülle an Funktionen. Verwendet werden kann es in Microsofts Entwicklungsumgebungen Visual Studio Code und Visual Studio, und ist damit überall dort verwendbar wo diese Entwicklungsumgebungen laufen. Heißt durch VSC ist Live Share neben Windows auch auf Mac OS und Linux verfügbar.

Saros hat jedoch den Vorteil, dass es nicht auf die derzeitigen Entwicklungsumgebungen beschränkt sein muss und Aufbauend auf dieser Arbeit eventuell einfacher auch für andere Umgebungen portiert werden kann.

2 Anforderungen

Saros hat über VSC die Möglichkeit bekannter zu werden und einen neuen Nutzerkreis zu erschließen, jedoch gilt es einige Hürden zu nehmen auf dem Weg zu einer Portierung. Zum Beispiel basiert VSC im Gegensatz zu Eclipse und IntelliJ nicht auf einer JVM-Sprache (wie Java[9] oder Kotlin[10]), sondern auf Typescript[11] (welches in NodeJs[12] ausgeführt wird).

2.1 Ziel der Arbeit

Das Ziel dieser Arbeit ist es Saros prototypisch für VSC zu implementieren, um zu zeigen dass Saros auch in Java fremden Entwicklungsumgebungen wie VSC lauffähig ist und verwendet werden kann. Dazu ist es nötig die Grundfuntkionen von Saros zu implementieren, damit es möglich ist eine Sitzung zwischen Saros-Nutzern aufzubauen. Darauf aufbauend kann dann die Kernfunktionalität umgesetzt werden, das Teilen von Quelltext und das Annotieren von Änderungen. Desweiteren soll der Grundstein für eine Weiterenwicklung gelegt werden, in der es nicht nötig ist erneut Pionierarbeit leisten zu müssen.

2.2 Funktionsübersicht

Zur besseren Übersicht folgt eine grobe Übersicht der Funktionen von Saros, diese orientieren sich an den Funktionen der Oberfläche von Eclipse. Ich teile die betrachteten Funktionen in Gruppen[13] ein und weise diesen dann einzelne Funktionen[14] zu. Zur Bestimmung der Funktionen habe ich folgende Ansätze verfolgt:

- Saros/E[15] im laufenden Betrieb getestet und Notizen gemacht
- Die Dokumentation im Web durchgelesen[16]
- Den Quelltext von Saros/E gesichtet

Daraus ergeben sich die folgenden, grob umrissenen, Funktionen und Gruppen.

[9] https://www.java.com/de/
[10] https://kotlinlang.org/
[11] https://www.typescriptlang.org/
[12] https://nodejs.org/en/
[13] Diese bekommen als Verweis den Prefix G mit ihrer laufenden Nummer
[14] Diese bekommen als Verweis den Prefix F mit ihrer laufenden Nummer
[15] die Eclipse Version von Saros
[16] https://www.saros-project.org/documentation/features.html

2.2.1 G1: Konfiguration

Die Gruppe Konfiguration beinhaltet alles Rund um die Konfiguration von Saros.

F1: Konfigurations-Assistent In der Eclipse Version gibt es Assistenten die durch Konfigurationen führen, beispielweise das hinzufügen eines Benutzers.

F2: Einstellungen Es ist Möglich einzelne Eigenschaften von Saros zu konfigurieren, z.B. die Annotationsfarbe der Sitzungsteilnehmer.

2.2.2 G2: XMPP - Kontaktverwaltung

Die Kontaktverwaltung umfasst die Verwaltung von Kontakten und Konten. Mit Absicht wird zwischen Kontakten und Konten unterschieden, da beide Entitäten unterschiedliche Funktionen haben. Kontakte sind die Personen auf der Kontaktliste, mit Ihnen lässt sich eine Sitzung erstellen. Konten hingegen werden verwendet um sich am XMPP-Server anzumelden und besitzen Kontaktlisten.

F1: Konto hinzufügen Es muss möglich sein neue Konten hinzuzufügen. Nicht erstellen, aktuell wird auch in der Eclipse Variante direkt auf die Webseite verwiesen, da Sicherungsmaßnahmen[17] eingesetzt werden mussten um Missbrauch zu verhindern. Darüber hinaus ist es fraglich das Erstellen in der Entwicklungsumgebung anzubieten, da jedem Nutzer die Wahl des XMPP-Servers freisteht.

F2: Konto entfernen Es muss möglich seine Konten auch wieder zu entfernen.

F3: Aktuelles Konto auswählen Saros unterstützt mehrere Konten gleichzeitig, daher muss es möglich sein, das Konto für die Anmeldung auswählen zu können.

F4: Konto bearbeiten Es muss möglich sein, bereits hinzugefügte Konten zu bearbeiten um z.B. Tippfehler zu korrigieren.

F5: Kontakt hinzufügen Es muss möglich sein Kontakte zur Kontaktliste hinzuzufügen, damit Sitzungen möglich sind.

F6: Kontakt bearbeiten Es muss möglich sein seine Kontakte zu bearbeiten, heißt einen Spitznamen zuweisen zu können. Durch Spitznamen ist es Möglich den Kontakt einfacher wiederzuerkennen, da die verwendeten Adressen zum Teil kryptisch sein können.

F7: Kontakt entfernen Es muss möglich sein, Kontakte auch wieder von seiner Kontaktliste zu entfernen.

[17] in dem Fall Captcha

2.2.3 G3: XMPP - Sitzungsverwaltung

Die Sitzungsverwaltung fasst alle Funktionen zusammen, die Sitzungen erstellen oder manipulieren.

F1: Sitzung erstellen Es muss möglich sein eine Sitzung zu starten.

F2: Sitzung beitreten Es muss für andere Teilnehmer möglich sein, einer Sitzung beizutreten.

F3: Sitzung schließen Es muss möglich sein, eine Sitzung wieder zu schließen.

F4: Teilnehmer einladen Es muss möglich sein, Teilnehmer zu einer bestehenden Sitzung einzuladen.

F5: Teilnehmer entfernen Es muss möglich sein, Teilnehmer die sich bereits in der Sitzung befinden, wieder zu entfernen.

2.2.4 G4: XMPP - Allgemein

F1: Projekt teilen Das derzeitig offene Projekt muss mit den Teilnehmern geteilt werden können.

F3: Chat Es muss möglich sein, sich mit den Sitzungsteilnehmern in einem Chat austauschen zu können.

2.2.5 G5: Saros

Diese Gruppe umfasst allgemeine Saros Funktionen.

F1: Textänderungen Änderungen am Quelltext müssen bei den anderen Teilnehmern angewendet werden, sodass jeder den aktuellen Quelltext hat. Bei Inkonsistenzen muss versucht werden, diese zu beheben.

F2: Annotationen Selektionen von Text müssen bei den anderen Teilnehmern angezeigt werden können.

F3: Zeigerposition Die Position des Zeigers der einzelnen Teilnehmer müssen bei den anderen Teilnehmern dargestellt werden.

F4: Rechteverwaltung Es muss möglich sein, einzelne Rechte ändern zu können, wenn man der Inhaber der Sitzung ist.

F5: Teilnehmer folgen Es muss möglich sein, Teilnehmern zu folgen. Das bedeutet geöffnete Editoren werden auch beim folgenden geöffnet und der sichtbare Bereich entsprechend angepasst.

F6: Sprung zu Position Es muss möglich sein, zu dem Bereich zu springen, der beim Zielteilnehmer gerade sichtbar ist. Nach dem Sprung muss man die gleiche Datei offen haben wie das Ziel, und den gleichen Ausschnitt sehen.

F7: Änderungen farblich nach Teilnehmer hervorheben Änderungen am Quelltext müssen bei den anderen Teilnehmern farblich hervorgehoben werden, um die Änderungen Erkenn- und Unterscheidbar zu machen.

2.3 Abgrenzung

Eine komplette Implementierung der Funktionen im Rahmen dieser Arbeit würde den Umfang sprengen, daher ist es nötig den Funktionsumfang abzugrenzen. Da das Ziel ein Prototyp ist, wird die kleinstmögliche Funktionsmenge implementiert, die Saros basisch nutzbar macht.

Das Minimum enthält also die Funktionen, die mindestens nötig sind, um eine minimale Saros Sitzung starten zu können. Eine minimale Saros Sitzung bedeutet, dass eine Sitzung von mindestens zwei Teilnehmern initiiert wird und zwischen diesen der Quelltext geteilt wird und Änderungen übertragen werden. Um überhaupt eine Sitzung aufbauen zu können, muss es im Vorfeld möglich sein seine Konten und Kontakte zu verwalten.

Zum Minimum gehören die folgenden Gruppen, bzw. Funktionen:

- G2: XMPP - Kontaktverwaltung
- G3: XMPP - Sitzungsverwaltung
- G4/F1: Projekt teilen
- G5/F1: Textänderungen
- G5/F7: Änderungen farblich nach Teilnehmer hervorheben

3 Visual Studio Code

Neben der Popularität hat VSC einen weiteren Vorteil. Es ist durch Erweiterungen vielfältig Anpassbar und Unterstützt das Entwickeln eben dieser durch eine eigene Schnittstelle.

3.1 Allgemein

VSC wurde im Mai 2015 das erste mal öffentlich vorgestellt und im April 2016 offiziell veröffentlicht[7]. Es ist ein Editor für Quelltexte und Unterstützt den Benutzer mit bekannten Funktionen wie z.B. Syntaxhervorhebung oder Debugging.

Es basiert auf dem Electron Framework, dieses ist wie VSC Quelloffen. Electron Anwendungen verwenden einen Chromium Webbrowser und NodeJs als Basistechnologie und es werden die Plattformen Windows, Linux und Mac unterstützt. Diese

Plattformunterstüztung ermöglicht auch VSC über mehrere Plattformen hinweg einsetzbar zu sein. Bekannte andere Electron Vertreter sind die WhatsApp und Slack Desktop Anwendungen[1].

Die Erweiterbarkeit von VSC war schon während der Entwicklung ein wichtiger Aspekt, das geht direkt aus dem ersten Satz der Dokumentation der VSC eigenen Extension API[18] hervor.

> "Visual Studio Code is built with extensibility in mind. From the UI to the editing experience, almost every part of VS Code can be customized and enhanced through the Extension API. In fact, many core features of VS Code are built as extensions and use the same Extension API."[2]

Die Erweiterbarkeit geht sogar so weit, dass einige Kernfunktionen von VSC selbst intern als Erweiterungen umgesetzt wurden.

3.2 Die Extension API

Die Extension API ist die von Microsoft bereitgestellte Schnittstelle zur Erweiterung von VSC. Sie dokumentiert alle Bereiche, die angepasst werden können, gibt Beispiele und stellt die Verfügbare Befehlsreferenz bereit. Da die Erweiterung in einem NodeJs Prozess läuft, ist die verwendete Programmiersprache Typescript[19].

3.2.1 Was kann angepasst werden?

Microsoft teilt die anpassbaren bzw. verwendbaren Funktionen in 6 Gruppen ein[20].

Allgemeine Funktionen Die allgemeinen Funktionen[21] umfassen alle von VSC bereitgestellten Funktionen, die von einer Erweiterung verwendet werden können um Aufgaben zu erledigen.

Bereitgestellt werden unter anderem:

- die Registrierung von Kommandos[22]
- das Speichern und Abrufen von Daten
- das Anzeigen von Meldungen
- das Abfragen von Nutzereingaben
- das Auswählen von Ordern bzw. Dateien
- das Anzeigen einer Fortschrittsanzeige
- die Interaktion mit den Editoren[23]

[18] Application Programming Interface
[19] https://www.typescriptlang.org/
[20] https://code.visualstudio.com/api/extension-capabilities/overview
[21] https://code.visualstudio.com/api/extension-capabilities/overview#common-capabilities
[22] Kommandos sind aufrufbare Aktionen, die Aufgaben ausführen
[23] Editoren sind, analog zu Eclipse, die Fenster über die Quelltexte bearbeitet werden können

Das Erscheinungsbild Während die erste Gruppe die bereitgestellten Funktionien umfasst, beschreiben die nun folgenden Gruppen die tatsächlich anpassbaren Eigenschaften von VSC. Über Erweiterungen ist es möglich das Erscheinungsbild[24] von VSC anzupassen.

Dazu gehört unter anderem:

- farbliche Hervorhebung von Teilen des Quellcodes
- farbliche Akzente von VSC selbst
- eigene Bilder für bestimmte Dateitypen

Deklarative Spracheigenschaften Das umfasst die Grundunterstützung für Programmiersprachen[25]. Deklarativ kann VSC die Handhabung neuer oder bereits bekannter Programmiersprachen beigebracht bzw. erweitert werden. Das Schreiben von Code ist hierfür nicht notwendig.

Folgende Funktionen können darüber z.B. beschrieben werden:

- Syntaxhervorhebung
- Codeschnipsel Vervollständigung
- Klammerung
- Kommentierung
- automatische Einrückung

Dieser Bereich ist für die Arbeit irrelevant, aber für weitergehende Informationen seien die verlinkten Anleitungen [26] ans Herz gelegt.

Programmatische Spracheigenschaften Programmatisch[27] können die deklarativen Eigenschaften um komplexere Funktionen erweitert werden.

Hierzu gehören Beispielsweise[28]:

- Informationsanzeige während Mauszeiger über eine Codezeile ist
- Autovervollständigung
- Sprung zur Definition
- Refaktorisierung

Auch für diesen Bereich hat Microsoft eine tiefer gehende Anleitung bereitgestellt[29].

[24] https://code.visualstudio.com/api/extension-capabilities/overview#theming
[25] https://code.visualstudio.com/api/extension-capabilities/overview#declarative-language-features
[26] https://code.visualstudio.com/api/language-extensions/overview#declarative-language-features
[27] https://code.visualstudio.com/api/extension-capabilities/overview#programmatic-language-features
[28] https://code.visualstudio.com/api/language-extensions/overview#programmatic-language-features
[29] https://code.visualstudio.com/api/language-extensions/programmatic-language-features

Arbeitsumgebung Diese Gruppe umfasst die Erweiterung der Oberfläche[30] von VSC. Ein Großteil der Elemente der Oberfläche von VSC können für eigene Erweiterungen verwendet werden.

Einige Möglichkeiten der Anpassung der Oberfläche von VSC sind:y

- eigene Kontextmenüs
- eigene Oberflächen in der Seitenleiste
- eigene Anzeigen in der unteren Aktivitätsanzeige von VSC
- eigene (virtuelle) Dokumente[31] oder Webseiten[32] anzeigen

Debugging Es können externe Debugger an die Debugging-Oberfläche von VSC angebunden werden[33] oder bereits bestehende erweitert werden um die Erfahrung weiter zu verbessern.

3.2.2 Was kann nicht angepasst werden?

Wie bereits erwähnt, basiert VSC auf Electron und somit wird die Oberfläche über HTML dargestellt. Und genau dieser Zugriff auf die HTML Oberfäche und die daran angebundenen CSS[34] Regeln und Javascript Komponenten ist von vornherein unterbunden, damit die allgemeine Stabilität und Reaktionszeit von VSC sichergestellt ist.

Um das zu erreichen laufen Erweiterungen in einem eigenen NodeJs Prozess[35], welcher keinen direkten Zugriff auf den DOM[36] der Oberfläche hat.

3.3 Die Proposed API

Die "Proposed APIïst, wie der Name schon sagt, die vorgeschlagene Programmierschnittstelle. Mit ihr werden neue Funktionen vorgestellt und getestet ohne die aktuelle Schnittstelle soweit zu verändern, das aktuelle Erweiterungen nicht mehr Funktionieren. Die Entwickler von Erweiterungen können hier neue Funktionen testen und Feedback geben, für den Produktiveinsatz ist sie nicht geeignet.

VSC verhindert auch den Einsatz dieser Schnittstelle in Veröffentlichten Erweiterungen und um sie lokal testen zu können, muss sie erst freigeschaltet und die Definitionen im Quellcode integriert werden.

[30] https://code.visualstudio.com/api/extension-capabilities/overview#workbench-extensions
[31] https://code.visualstudio.com/api/extension-guides/virtual-documents
[32] https://code.visualstudio.com/api/extension-guides/webview
[33] https://code.visualstudio.com/api/extension-capabilities/overview#debugging
[34] Cascading Style Sheets
[35] https://code.visualstudio.com/api/advanced-topics/extension-host
[36] Document Object Model

3.4 Contribution Points

Die "Contribution Points"[37], beschreiben die Schnittstellen die eine Erweiterung der Funktionen von VSC erlauben und über die `package.json` im Bereich "contributes"registriert werden. Die für die Umsetzung relevanten werden nachfolgend kurz beschrieben.

3.4.1 configuration

Über diese Schnittstelle[38] kann definiert werden, welche Einstellungen der Erweiterung dem Nutzer zur Konfiguration bereitgestellt werden. Dabei stehen dem Nutzer zwei Möglichkeiten zur Verfügung diese dann zu verändern. Entweder über die VSC Oberfläche, in den Einstellungen von VSC unter dem Punkt Ëxtensionsöder direkt über die JSON[39]-Datei, die intern auch von der VSC Oberfäche verwendet wird. VSC kennt dabei verschiedene Dateitypen, z.B.:

- number: eine Zahl
- string: ein Text
- boolean: ein Boolescher Wert
- enum: eine Enumeration

Es gibt noch weitere Eigenschaften, wie Titel und Beschreibung, die in der Dokumentation[40] gut beschrieben sind. Über die Schnittstelle `workspace` kann der aktuelle Wert dann abgerufen werden.

3.4.2 commands

In VSC sind viele Funktionen als Kommandos[40] implementiert. Kommandos sind im Prinzip kleine Programme, die über die Oberfläche vom Benutzer oder von Erweiterungen aufgerufen werden können und Aufgaben erfüllen. Die Bezeichnung eines Kommandos im Feld "command"muss eindeutig sein, da sie darüber identifiziert und aufgerufen werden. Der "title"beschreibt den dargestellten Text in Auswahlmenüs und über "category"können Kommandos gruppiert werden.

Wichtig ist es neben der Registrierung in der `package.json`, die Kommandos über den Namensraum `commands` zur Laufzeit zu registrieren. Erst mit der Registrierung zur Laufzeit wird VSC mitgeteilt welches Kommando welchen Code aufruft.

3.4.3 menus

Die Menü-Schnittstelle[41] ermöglicht das Hinzufügen von Menüs zu diversen Standard Ansichten von VSC wie:

- das Kontextmenü des Dateiexplorers

[37] https://code.visualstudio.com/api/references/contribution-points
[38] https://code.visualstudio.com/api/references/contribution-points#contributes.configuration
[39] JavaScript Object Notation
[40] https://code.visualstudio.com/api/references/contribution-points#contributes.commands
[41] https://code.visualstudio.com/api/references/contribution-points#contributes.menus

- das Kontextmenü des Editors
- die Debugleiste
- die Menüs der Versionsverwaltung

Es können aber auch Menüs für Bereiche hinzugefügt werden, die durch die eigene Erweiterung hinzugekommen sind.

Ein Menüeintrag ist immer ein Verweis auf ein Kommando, damit ist die Angabe des Kommandos obligatorisch. Menüeinträge können ähnlich wie Kommandos gruppiert werden und über den Eintrag "when"kann definiert werden, wann der Eintrag ausgewählt werden kann. Wird ein Menüeintrag in einem Kontextmenü ausgewählt, versucht VSC das ausgewählte Objekt an das Kommando zu übergeben. Wird das Kommando also auch für Kontextmenüs verwendet, so kann bei der Registrierung zur Laufzeit das Objekt als Argument der Methode hinzugefügt werden. Kann das Kommando auch unabhängig zum Kontextmenü aufgerufen werden, sollte in dem Fall geprüft werden ob das Argument nicht undefiniert ist um Fehler zu vermeiden.

3.4.4 views

Die Schnittstelle für Ansichten[42] ermöglicht es VSC eigene Ansichten hinzuzufügen. Eine Ansicht muss dabei einem Container zugewiesen werden, folgende Container stehen bereit:

- explorer: die Ansicht des Arbeitsbereiches
- scm: die Ansicht des Versionskontrollsystems
- debug: die Ansicht des Debuggers
- test: die Ansicht der Testanzeige

Darüber hinaus können auch eigene Container hinzugefügt werden, wie im folgenden Abschnitt beschrieben. Analog zu den Kontextmenüs kann auch Ansichten über das Schlüsselwort "when"die Sichtbarkeit gesteuert werden.

An die Ansichten muss ein "TreeDataProviderängebunden werden, um Daten zu veröffentlichen. Aktuell existiert nur diese Art der Ansicht.

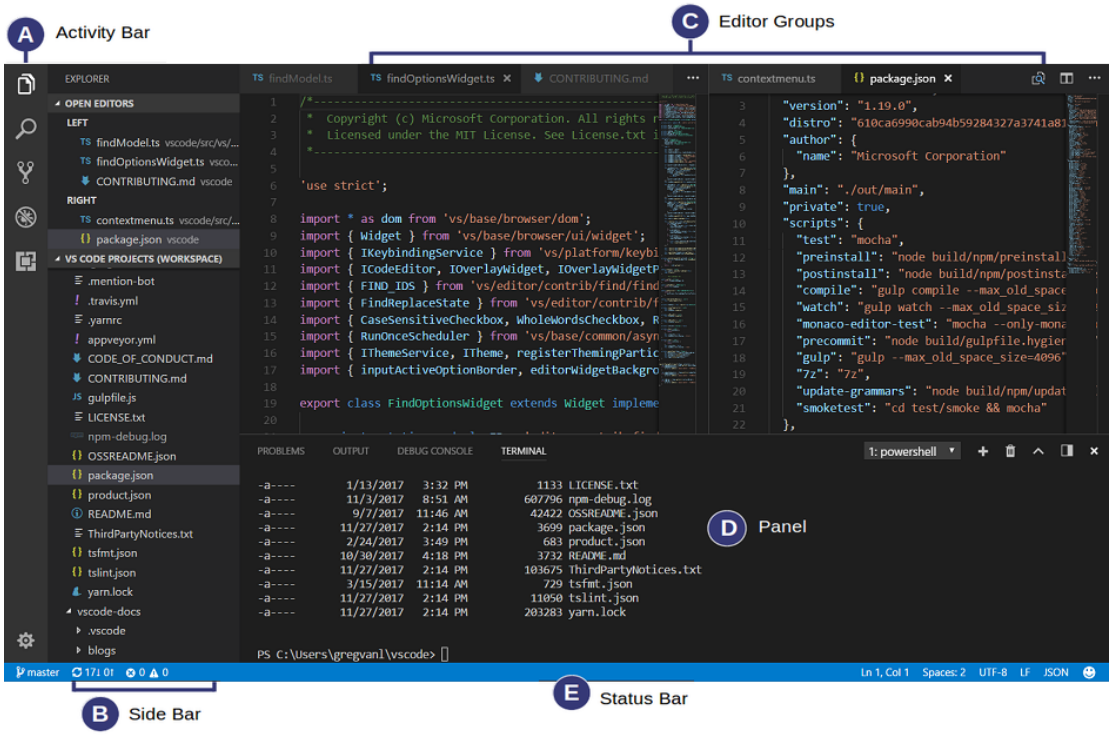
3.4.5 viewContainers

Über die Schnittstelle[43] können eigene Bereiche definiert werden, in denen dann die eigenen Ansichten angezeigt werden können. So ein Bereich muss zu einem von derzeit zwei möglichen Orten gehören, der Äctivity Baröder dem "Panel".

Activity Bar Der Begriff Äctivity Bar"bezeichnet die Seitenleiste auf der linken Seite in VSC (siehe Abb. 1: A). Das hinzufügen eines Containers zur Seitenleiste bewirkt,

[42] https://code.visualstudio.com/api/references/contribution-points#contributes.views
[43] https://code.visualstudio.com/api/references/contribution-points#contributes.views

Abbildung 1: Die Oberfläche von Saros[44]



dass das ausgewählte Bild in der Leiste angezeigt wird und bei Aktivierung den Container sichtbar macht, bzw. ausklappt.

Panel Das "Panel"bezeichnet den Bereich zwischen dem Editor und der unteren Leiste (siehe Abb. 1: D). Das Panel ist im Gegensatz zur Seitenleiste nicht unbedingt bei Start sichtbar und muss unter Umständen erst angezeigt werden, z.B. über das Titelmenü "Terminalünd dann "New Terminal".

3.5 Namespaces

Die Programmierschnittstelle von VSC stellt mehrere Namensräume[45] bereit, über die bestimmte Funktionen von VSC angesprochen werden können. Die für die Umsetzung relevanten werden nun nachfolgend analog kurz beschrieben.

3.5.1 window

Der "window"Namensraum[46] stellt diverse Methoden bereit um mit der Oberfläche von VSC oder dem Nutzer zu interagieren. Anwendungsgebiete sind:

- das Registrieren von Methoden für bestimmte Ereignisse aus der Oberfläche oder dem Editor

[45] https://code.visualstudio.com/api/references/vscode-api
[46] https://code.visualstudio.com/api/references/vscode-api#window

- das Anzeigen von Meldungen
- das Abfragen von Nutzereingaben
- das Anzeigen einer Fortschrittsanzeige
- der Zugriff auf aktive bzw. sichtbare Editoren

Der Namensraum bietet auch Zugriff auf die Editoren, wodurch dann z.B. der Quelltext manipuliert oder Bereiche hervorgehoben werden können.

3.5.2 workspace

Der "workspace"Namensraum[47] bietet Zugriff auf den geöffneten Arbeitsbereich und bietet Zugriff auf diverse Ereignisse im Arbeitsbereich, wie z.B.:

- Erstellung von Dateien/Ordnern
- Löschung von Dateien/Ordnern
- Umbenennungen
- Öffnung/Schließung von Dokumenten
- Speicherung von Dokumenten

Darüber hinaus wird es ermöglicht Dokumente zu öffnen oder auszulesen.

3.6 Verheiratung mit Java

Die größte Herausforderung bei der Umsetzung von Saros für VSC, war das Überbrücken der Sprachbarriere zwischen Typescript (VSC) und Java (Saros). Nach einer Recherche haben sich zwei mögliche Wege herauskristallisiert, wie diese Brücke umgesetzt werden könnte.

Die erste Möglichkeit ist es den gleichen Weg zu gehen, wie damals bei der HTML-Implementierung[5] der Saros Oberfläche. Die zweite Möglichkeit ist es, Saros als eigenen Prozess laufen zu lassen, und einen Weg der Kommunikation zwischen VSC und Saros zu finden.

3.6.1 Möglichkeit 1: Einbettung von Java in Typescript

Dies ist der naheliegenste Weg, da hier analog zur HTML-Implementierung[5] Javascript zum Einsatz kommt. Allerdings läuft hier kein Java basierter Browser, der Java Methoden aufrufen kann, sondern Typescript.

Daraufhin habe ich nach anderen Möglichkeiten gesucht, Java in Typescript einzubetten. Als erstes bin ich auf das NPM[48] Paket ts-java[49] gestoßen. ts-java verspricht aus Java Klassen Typescript Deklarationen zu generieren, damit diese Typensicher

[47] https://code.visualstudio.com/api/references/vscode-api#workspace
[48] ein Paketmanager für NodeJs
[49] https://www.npmjs.com/package/ts-java

in Verbindung mit Node.java[50] aufgerufen werden können. Allerdings ist die letzte Veröffentlichung von ts-java bereits 4 Jahre her, was auf keine gute Unterstützung schließen lässt. Jedoch könnte man Node-java ohne Typsicherheit verwenden, immerhin scheint diese Bibliothek noch aktiv unterstützt zu werden, die letzte Veröffentlichung liegt 7 Monate zurück. Beim Testen des Paketes wurde allerdings eine große Schwäche sichtbar, schon auf meinem System gab es Kompilierfehler bei der Installation des Paketes. Darüberhinaus wird bei der Installation der Code des Paketes für das aktuelle System kompiliert, das bedeutet dass Saros für VSC nur auf der gleichen Plattform laufen würde oder für jedes Zielsystem ein eigenes Paket kompiliert und veröffentlicht werden müsste. Zu viel Aufwand meiner Meinung nach.

Als letztes bin ich auf node-java-rt[51] gestoßen, wird anscheinend auch noch relativ aktiv Unterstützt, die letzte Veröffentlichung liegt ein Jahr zurück. Das Paket basiert allerdings auf node-java, und bringt somit die selben Nachteile mit sich.

3.6.2 Möglichkeit 2: Saros als eigenständigen Prozess

In diesem Szenario müsste Saros ähnlich des Saros-Servers[4] für VSC implementiert werden und dann über eine geeignete Schnittstelle angesprochen werden.

Eine erste Idee war es, Saros in eine Jar-Datei zu kompilieren und über ein Terminal in VSC zu starten. Dann sollte der Austausch wie bei einer CLI erfolgen. Dieser Ansatz hat allerdings einen sehr großen Nachteil, alle Aktionen würden Seriell abgearbeitet werden, was aus meiner Sicht zu einer enormen Verlangsamung der Abarbeitung der Ereignisse von Saros führen würde. Besonders mit steigender Teilnehmerzahl in der Sitzung würde das Problem immer größer werden. Ein weiterer Nachteil ist es, dass Rückgaben von Saros aus Texten extrahiert werden müssen, was sehr Fehleranfällig sein kann.

Durch Tobias Bouschen und Stefan Moll wurde ich auf das "Language Server Protocol"[52], kurz LSP, hingewiesen. Das Protokoll eignet sich gut für eine Client-Server-Architektur, und da es bereits Implementierungen für Java und VSC gibt konnte Ich hier zusätzlich Entwicklungsaufwand für den Prototypen einsparen. Denn die Kommunikation ist bereits für beide Technoilogien implementiert und das Protokoll bringt Funktionen mit, die auch für Saros nützlich sind. Aufgrund dieser Argumente habe ich mich für diese Version entschieden. Die VSC Erweiterung wird als Client und Saros als Server arbeiten und die Kommunikation wird über das LSP laufen.

4 Language Server Protocol

Das Protokoll[53] wurde ursprünglich für VSC entwickelt, und ist mittlerweile in der Version 3.15 verfügbar.

[50] https://www.npmjs.com/package/java
[51] https://www.npmjs.com/package/node-java-rt
[52] https://microsoft.github.io/language-server-protocol/
[53] https://microsoft.github.io/language-server-protocol/specifications/specification-current/

> "The idea behind the Language Server Protocol (LSP) is to standardize the protocol for how tools and servers communicate, so a single Language Server can be re-used in multiple development tools, and tools can support languages with minimal effort."[54]

Es beschreibt also die Art wie Werkzeug und Sprachserver miteinander kommunizieren sollen, dadurch wird es ermöglicht dass für unterschiedliche Sprachen nur der Sprachserver implementiert werden muss, nicht aber die Anbindung an das Werkzeug (hier die Entwicklungsumgebung VSC). Das Werkzeug kümmert sich um das Überwachen der Aktionen und sendet diese an den Sprachserver, welcher die eigentliche Auswertung (also die Logik) vornimmt und die Resultate zurücksendet. Der Austausch findet über JSON-RPC[55] statt, das ist ein Protokoll zum Aufruf entfernter Methoden (RPC: Remote Procedure Call) über das JSON Format.

Durch das Protokoll ist es VSC möglich so viele Sprachen zu unterstützen, denn um eine neue zu Unterstützen muss nur ein Sprachserver implementiert werden, der über LSP angesprochen werden kann. Die Anbindungen innerhalb von VSC müssen nicht angepasst werden, da diese für jede Sprache gleich sind, ein klarer Vorteil.

Aus meiner Sicht könnte man diesen Vorteil auch durchaus auf Saros übertragen. Denn wenn Saros das Protokoll unterstützt, müsste nur noch die Anbindung bei neuen Entwicklungsumgebungen angepasst bzw. implementiert werden. Das aufwändige Implementieren von Saros spezifischen Anbindungen könnte also entfallen.

Natürlich nutzt Saros nicht das gesamte Protokoll, allerdings eine Teilmenge wie z.B. der Austausch und die Verfolgung von Änderungen, eine Kernkomponente von Saros.

Die Spezifikation wird aufgeteilt in 10 Bereiche:

- Base Protocol
- Basic Structures
- General Messages
- Window
- Telemetry
- Client
- Workspace
- Text Synchronization
- Diagnostics
- Language Features

Von diesen 10 Bereichen ist nur "Language Features"für Saros irrelevant, da Saros unabhängig von der verwendeten Programmiersprache Arbeitet und die abgedeckten Funktionen nicht Teil von Saros Funktionen sind (wie z.B. Codevervollständigung

[54] https://microsoft.github.io/language-server-protocol/overviews/lsp/overview/
[55] https://www.jsonrpc.org/specification

oder Code Lens). Daran ist zu sehen, dass Saros den Großteil des Protokolls verwendet, und mMn. das Protokoll für Saros sehr interessant macht.

4.1 Das Protokoll

Das Protokoll ist sehr Umfangreich, sodass nur kurz auf einzelne Besonderheiten eingegangen wird.

DAs Protokoll kennt zwei Arten von Nachrichten. Die erste ist die "Notification Message"[56], sie kommt immer dann zum Einsatz, wenn keine Antwort von der Gegenseite erwartet wird. Die zweite Art ist die "Request Message"[57], auf diese folgt immer eine "Response Message"[58]. Das heißt, sendet man einen "Request", bekommt man nach dem Protokoll immer eine Antwort.

Bei Verbindungsaufnahme von Client zu Server ist der Initialize Request"[59] der erste der mit dem Server ausgetauscht wird. Er dient dem Austausch der Konfigurationen des Clients und des Servers, z.B. verwendete Versionen, Pfad zum Quelltext, unterstützte Funktionen.

Zum beenden des Servers existieren zwei Nachrichten. Der SShutdown Request"[60] fordert den Server nur zum herunterfahren, nicht aber zum beenden auf. Für das Beenden des Prozesses existiert die Exit Notification"[61], welche im Gegensatz zum SShutdown Request"keine Antwort auf die Anfrage liefert.

Während der Laufzeit werden unterschiedliche Nachrichten ausgetauscht. Für das Loggen von Ereignissen existiert die "Log Message"Nachricht, und für das Anzeigen von Nachrichten beim Client durch den Server exisitert die 'Show Message' [62] Nachricht. Wenn Nutzereingaben benötigt werden, wird die SShow Message Request"[63] Nachricht verwendet, die eine Rückgabe liefert.

Zur Überwachung von Text- und Dateiänderungen werden die Nachrichten aus den Bereichen "Workspace"[64] und "Text Synchronization"[65] verwendet.

[56] https://microsoft.github.io/language-server-protocol/specifications/specification-current/#notificationMessage
[57] https://microsoft.github.io/language-server-protocol/specifications/specification-current/#requestMessage
[58] https://microsoft.github.io/language-server-protocol/specifications/specification-current/#responseMessage
[59] https://microsoft.github.io/language-server-protocol/specifications/specification-current/#initialize
[60] https://microsoft.github.io/language-server-protocol/specifications/specification-current/#shutdown
[61] https://microsoft.github.io/language-server-protocol/specifications/specification-current/#exit
[62] https://microsoft.github.io/language-server-protocol/specifications/specification-current/#window_showMessage
[63] https://microsoft.github.io/language-server-protocol/specifications/specification-current/#window_showMessageRequest
[64] https://microsoft.github.io/language-server-protocol/specifications/specification-current/#workspace_workspaceFolders
[65] https://microsoft.github.io/language-server-protocol/specifications/specification-current/#textDocument_synchronization

4.2 Verbreitung

Es gibt für eine große Bandbreite von Programmiersprachen bereits Sprachserver Implementierungen[66] und Entwicklungsumgebungen[67], die diese unterstützen (darunter auch Eclipse). Das zeigt, dass das Protokoll durchaus erfolgreich zu sein scheint und für eine Reihe von Programmiersprachen gibt es auch Bibliotheken[68] um eigene Sprachserver zu entwickeln.

4.3 Implementierungen

Für Java habe ich mich für die Implementierung von Eclipse entschieden, LSP4J[69], denn er Unterstützt das Protokoll in der Version 3.15.0 (bis auf die Fortschrittsnachrichten), scheint sehr ausgereift und hat einige Unterstützer.

Für Typescript in VSC habe ich mich für den Client von Microsoft[70] entschieden, da es nahe liegt hier die Variante zu nehmen, die vom gleichen Herausgeber ist wie die Entwicklungsumgebung. Der Client unterstützt das Protokoll in der derzeit aktuellsten Version 3.16.0.

5 Umsetzung

Die Umsetzung spaltet sich in zwei Teile gemäß der Client-Server-Architektur, die durch die Verwendung von LSP herrührt.

VSC bildet hierbei den Client und kümmert sich um die Überwachung der Änderungen und die Kommunikation mit dem Nutzer. Saros läuft als Java Prozess und bildet den Server, der die Saros Kernaufgaben übernimmt und relevante Aktionen an VSC weiterleitet und Anfragen von VSC verarbeitet.

5.1 Vorgehen

Da zu Beginn vieles Unbekannt war, wie die Art der Verheiratung von Typescript und Java oder allgemein wie bestimmte Funktionen in VSC zu implementieren sind (wie zum Beispiel das Hervorheben der Änderungen), habe ich mich Entschieden bei der Entwicklung des Prototypen inkrementell von Woche zu Woche vorzugehen. Durch die zwei mal wöchentlich vorkommenden Stand-Ups im Saros Team und das wöchentliche Meeting mit meinem Betreuer konnte ich das Gesamtziel im Auge behalten.

Dabei lief die Entwicklung von Client (VSC) und Server (Saros) parallel, beide Programme wurden stetig um die neuen Funktionen erweitert.

Begonnen habe ich mit der allgemeinen Anbindung des Saros Language Servers (von nun an SLS genannt) an VSC. Dann folgten die Verwaltung von Konten, die Ver-

[66] https://microsoft.github.io/language-server-protocol/implementors/servers/
[67] https://microsoft.github.io/language-server-protocol/implementors/tools/
[68] https://microsoft.github.io/language-server-protocol/implementors/sdks/
[69] https://github.com/eclipse/lsp4j
[70] https://github.com/Microsoft/vscode-languageserver-node

waltung von Kontakten und die Sitzungsverwaltung. Darauf Aufbauend kamen das Teilen des Projektes, die Übertragung der Änderungen und zuletzt die Hervorhebung der Änderungen im Client.

5.2 Visual Studio Code

Nachfolgend beschreibe ich kurz die wichtigsten Entscheidungen in VSC.

5.2.1 Anbindung des SLS

Es gibt zwei Möglichkeiten Saros anzubinden.

stdin und stdout Der SLS kan über die Standardeingabe und die Standardausgabe angebunden werden. Das hat den Vorteil, dass der SLS nur seine Ein- und Ausgabeströme an die verwendete Language Server Bibliothek weitergeben muss, und auf Clientseite beide Ströme nur an die Client Implementierung weitergegeben werden muss. Relativ einfach zu implementieren.

Sockets Der SLS kann auch über Sockets an den Client angebunden werden. Hierfür müssen sich allerdings SLS und VSC auf einen Port einigen, über den sich beide dann Verbinden. Die Socket-Ströme können analog zu den Standardströmen an die verwendeten Bibliotheken zur Verbindung weitergegeben werden.

Auf den ersten Blick erscheint die erste Variante die bessere Wahl zu sein, allerdings landen alle Lognachrichten auch auf dem Standardausgabestrom, was das Protokoll kaputt macht. Sobald eine Nachricht, die nicht dem Protokoll entspricht, auf dem Ausgabestrom ausgegeben wird, funktionert der Datenaustausch nicht mehr. Als Übergangslösung könnte man in dem Fall entweder das Logging ausschalten oder sich zwischen Ausgabestrom und Logging schalten um diese herauszufiltern. allerdings erschien dieser Weg mir zu Fehleranfällig und nicht Zielführend, zumal Lognachrichten eine wichtige Daseinsberechtigung haben.

Also habe ich mich für die Sockets entschieden. Um einen freien Port zu ermitteln habe ich das NPM-Paket get-port[71] verwendet, um den Server unabhängig von VSC debuggen zu können habe ich zusätzlich die Konfigurationsmöglichkeit hinzugefügt einen festen Port anzugeben. In dem Fall muss der SLS extern gestartet werden, und VSC versucht sich dann am angegebenen Port mit dem Server zu verbinden. Damit der SLS weiß auf welchen Port er lauschen muss verwende ich Parameter, mit denen der SLS gestartet wird. Der Parameter `-p=PORT`, wobei PORT die Zahl des Ports ist, weist den SLS an diesen Port zu verwenden.

5.2.2 Ordnerstruktur

Ich habe folgende Ordnerstruktur gewählt:

[71] https://www.npmjs.com/package/get-port

commands Dieser Ordner enthält die Teilbereiche äccounts", "contactsünd ßessions". äccounts"beinhaltet alles zur Verwaltung der Konten. "contacts"beinhaltet alles zur Verwaltung der Kontakte. ßessions"beinhaltet alles zur Verwaltung von Sitzungen. Alle drei Bereiche sind gleich aufgebaut, damit bei Anpassungen der Aufwand gering gehalten wird. Die äctivator.tsënthält eine Methode, die alle Kommandos in VSC registriert. Der Ordner "wizardsënthält die Assistenten, die das Einheben und Änderung von Daten erleichtern sollten und ßtepsënthält die einzelnen Schritte, die in den Assistenten verwendet werden können.

Hier ist also die meiste Funktionalität implementiert.

lsp Dieser Ordner enthält alles bezüglich des Language Server Protokolls. Also die Verbindungslogik, die Typescript Klassen zur Verwendung des Protokolls und alles was zur Kommunikation mit dem SLS nötig ist.

types Enthält Implementierungen von Klassen, die in anderen Bereichen verwendet werden.

utils Enthält Hilfsfunktionen, die z.B. den Zugriff auf die Icons vereinfachen und vereinheitlichen.

views Enthält alles, was mit der Oberfläche von VSC zu tun hat.

5.2.3 Warum wurden Ansichten implementiert?

Zusätlich zur Minimalimplementierung habe ich mich während der Entwicklung dazu entschieden diverse Ansichten zu implementieren. Der Hauptgrund ist, dass durch die Fülle an Kommandos, die nötig sind um alle Funktionalität abzubilden, das Benutzen von Saros in VSC schnell sehr unhandlich wurde. Der andere Grund ist, dass in VSC Ansichten relativ komfortabel implementiert werden können und den Aufwand für das Testen der Funktionen vereinfachen und dadurch Zeit einsparen.

Im laufe der Arbeit habe ich drei Ansichten implementiert.

Kontaktliste Das ist die erste Ansicht und zeigt, sobald man mit dem XMPP-Server verbunden ist, die Kontaktliste. Dabei habe ich die verwendeten Bilder aus der Eclipse Variante genommen, die den Online Status und die Saros Funktionalität darstellen. Über diese Ansicht kann man sich mit dem XMPP-Server verbinden und auch wieder trennen. Darüber hinaus können hierüber auch (zusätzlich zu den Kommandos, die immer Verfügbar sind) Kontakte angelegt, angepasst und wieder entfernt werden. Eine Einladung zu einer Sitzung ist auch möglich. Dabei sind die Kommandos auf die Ansicht und Kontextmenüs aufgeteilt.

Sitzungsansicht Das ist die zweite Ansicht, die ich implementiert habe. Diese Ansicht hat mir geholfen die Sitzungsverwaltung und die Verteilung der Annotations-Farben zu testen. Bei bestehender Sitzung, werden hier die Teilnehmer mit ihrer Farbe

angezeigt. Die gleich Ansicht befindet sich auch nochmal im Dateiexplorer, da hier die meiste Zeit während der Paarprogrammierung gearbeitet wird und die Ansicht mir hier sehr nützlich erschien und Informationen über die Sitzung zu erhalten.

Beide Ansichten sind in der Seitenleiste im eigenen SSaros View Containerüntergebracht, erkennbar an dem Saros Symbol. Die dritte Ansicht, die ich implementiert habe ist die Kontoansicht.

Kontoansicht Diese Ansicht dient dazu das aktuelle Konto anzuzeigen und zu wechseln. Diese Ansicht hat sich ab dem Punkt rentiert, an dem ich Saros mit mehreren Konten auf dem gleichen Rechner testen musste. Über einen Klick auf die Ansicht kann man eines der registrierten Konten auswählen und sich somit mit einem anderen Konto am XMPP-Server anmelden. Die Ansicht befindet sich in der unten befindlichen Statusleiste.

Damit die Sitzungsanzeige im Dateiexplorer nur angezeigt wird wenn eine Sitzung besteht oder damit der SSaros View Container"nur nach erfolgreichem Start von SLS angezeigt wird setze ich über das VSC Kommando `setContext` eine Variable auf einen bestimmten Wert. Das hat den Vorteil, dass ich in der `package.json` Zugriff auf diese Variablen habe, was sonst nicht möglich wäre.

5.2.4 Die Assistenten

Das Anlegen oder Bearbeiten von Konten bzw. Kontakten erfordert die Eingabe mehrerer Werte. Bei VSC gibt es kein Formular, sondern nur die Möglichkeit der Einhabe über QuickInput[72] oder QuickPick[73]. Diese Eingabemethoden sind allerdings immer nur für ein Datum gleichzeitig, daher war eine Art Eingabeassistent nötig. Bei den Beispielen für VSC Erweiterungen[74] gibt es bereits eine Beispiel-Implementierung. Die Implementierung musste ich jedoch anpassen, da der Assistent vorab wissen musste wieviele Schritte zur Beendigung nötig sind. Für einen optionalen Schritt war das noch möglich, ab zwei Schritten wurde es allerdings schon sehr kompliziert für mich. Daher habe ich den Quelltext angepasst, dass auch optionale Schritte unterstützt werden und diese dynamisch angezeigt werden. Das war eine besondere Herausforderung, da ein hin- und herspringen möglich war und ich diese Funktionalität gerne beibehalten wollte. Die angedeutete Wiederaufnahme bei verlassen des Assistenten habe ich erweitert, sodass man die Möglichkeit hat den Assistenten wieder aufzurufen, wenn dieser noch nicht abgeschlossen war.

Ein großes Problem hat die aktuelle Implementierung, da das Password der Konten geändert werden können muss, wird dieses vom SLS an VSC im Klartext versendet. Wenn man das verbessern will sollte man das Passwort leer an VSC senden, und bei Antwort prüfen ob es leer von VSC wiederkommt (keine Änderung) oder nicht leer wiederkommt (Passwort wurde geändert). Da sich aber Client und Server auf dem

[72] https://code.visualstudio.com/api/references/vscode-api#QuickInput
[73] https://code.visualstudio.com/api/references/vscode-api#QuickPick
[74] https://github.com/microsoft/vscode-extension-samples/blob/master/quickinput-sample/src/multiStepInput.ts

selben Rechner befinden und unter dem selben Nutzer laufen, habe ich das Problem für den Prototypen als vernachlässigbar eingeschätzt.

5.2.5 Logging

Es gibt zwei Quellen von Log-Nachrichten, beide landen in jeweils separaten Öutput Channels"[75] von VSC. Die Nachrichten von VSC landen im Kanal SSarosünd zeigen im Moment nur die mit dem SLS ausgetauschten Nachrichten. Die Log-Nachrichten von Saros (SLS) landen im Kanal SSaros (Server)ünd zeigen die Log-Nachrichten aus der Java-Implementierung von Saros.

Das Level der Log-Nachrichten kann in den VSC Einstellungen konfiguriert werden, für beide Kanäle separat.

5.2.6 Das Arbeitsbereich Problem

Im Moment erwartet der Prototyp, dass bei Start ein Ordner geöffnet ist. Ist kein Ordner offen oder ist ein Multi-root Workspace[76] offen, würde der SLS abstürzen. Darum habe ich mich für die einfachste Lösung entschieden, in dem Fall die Saros Erweiterung in VSC wieder zu deaktivieren.

Eine weitere Entscheidung ist, geteilte Projekte direkt im von VSC geöffneten Ordner zu speichern. Denn würde man das nicht tun, und dem Nutzer die Wahl lassen, so müsste der Ordner nach Teilen des Projektes von VSC geöffnet werden. Ein Wechsel des geöffneten Ordners bedeutet bei VSC allerdings das Schließen der aktuellen Instanz und das starten einer neuen, in der der neue Ordner geöffnet ist. Da zu dem Zeitraum allerdings eine Sitzung besteht, würde das dazu führen dass der Nutzer, der soeben beigetreten ist, nach Teilen des Projektes die Sitzung wieder verlässt. Denn der SLS läuft als Kindprozess von VSC und wenn VSC für einen Ordnerwechsel geschlossen wird, so wird auch der SLS und somit die Sitzung geschlossen. Den Mehraufwand, den das Abdecken des Falls bedeuten würde, habe ich mit dieser Entscheidung umgangen. Möchte man es allerdings erlauben, so müssten man vor dem Wechsel des Ordners den SLS Kindprozess von VSC lösen und zu einem eigenständigen machen und der SLS müsste nach Trennung der Verbindung erneut in einen Wartemodus für eine erneute Verbindungsaufnahme gehen. Ist der Ordnerwechsel vollzogen, so muss die Verbindung zum alten Server wieder aufgebaut werden, anstatt wie normal einen neuen SLS zu starten.

5.2.7 Annotationen

In Saros werden zur Auswahl der Annotationsfarben nur Zahlen ausgetauscht, nicht aber die Farbwerte. Diesem Beispiel bin ich gefolgt und habe die Definition der einzelnen Farben in den Client ausgelagert. In den Einstellungen von VSC können fünf Farbwerte angegeben werden, diese werden über die in Saros verwendeten Zahlen bei der Annotation ausgewählt und dann zur Hervorhebung der Änderungen verwendet.

[75] https://code.visualstudio.com/api/references/vscode-api#OutputChannel
[76] https://code.visualstudio.com/docs/editor/multi-root-workspaces

Damit man die Farben im Quelltext den Autoren zuordnen kann, habe ich mich dazu entschlossen in der Sitzungsansicht, die zugeordneten Farben neben den Namen darzustellen. Das Problem ist allerdings, dass man in der Ansicht keine Texte einfärben, sondern nur Bilder verwenden kann. Um zu vermeiden, für jeglichen Farbwert ein Bild vorgenerieren zu müssen und damit die Größe der zu veröffentlichenden Erweiterung unnötig zu vergrößern, habe ich mich dazu entschieden diese zur Laufzeit zu generieren und dann in der Ansicht zu verwenden. Um das zu erreichen benutze ich die Bibliothek pureimage[77]. Wenn dem Client die Zahl der verwendeten Farbe mitgeteilt wird, wird der eingestellte Farbwert aus den Einstellungen ausgelesen und geprüft ob das Bild existiert. Existiert es nicht wird es generiert und zurückgegeben für die Ansicht, existiert es wird es direkt der Ansicht zurückgegeben. Um Konflikte mit Schreibrechten zu vermeiden, werden die Bilder im Verzeichnis der Erweiterung gespeichert.

5.2.8 Gradle

Damit sich VSC in das bereits bestehende Build-System einfügt und von den Saros Entwicklern ohne große Probleme verwendet werden kann, habe ich analog zu den bereits bestehenden Projekten eine Gradle[78]-Datei angelegt. Über Gradle ist es nun möglich das Projekt zu bauen oder zur Entwicklung in einem eigenen VSC Prozess zu starten. Die Möglichkeit der Veröffentlichung oder das Zurückziehen der Veröffentlichung habe ich dort ebenfalls implementiert, im Fall der Veröffentlichung wird die aktuelle Version aus der `package.json` ausgelesen und der Resultierenden Datei im Namen angehangen. Da das Projekt nur über NPM gebaut werden kann, wird das Node Gradle Plugin[79] verwendet.

5.3 Saros Language Server

Der SLS ist der zweite große Teil des Protoyps. Bei der Entwicklung des SLS habe ich mich am Saros Server orientiert, zum einen weil ich das Dateisystem bei VSC wie beim Server auch Dateibasiert implementieren wollte. Und zum anderen weil der Server den minimalen Umfang an Funktionen zeigt, die zum Betrieb von Saros nötig sind.

5.3.1 Konflikte mit anderen Language Servern

Während der Entwicklung kam die Frage auf, ob der SLS andere Sprachserver oder selber blockiert werden könnte. Das kann allerdings nicht der Fall sein, denn der SLS kommuniziert über eine eigene Verbindung mit VSC. Die Sprachfunktionen des LSP werden darüber hinaus vom SLS gar nicht verwendet, daher kann es auch hier nicht zu Überschneidungen kommen. Und allgemein Unterstützt VSC mehrere Spachserver, zwar je Datei, aber ein Test hat keine Konflikte zwischen dem Sprachserver und dem SLS offen gelegt.

[77] https://www.npmjs.com/package/pureimage
[78] https://gradle.org/
[79] https://plugins.gradle.org/plugin/com.github.node-gradle.node

5.3.2 Das Saros Protokoll

Das LSP deckt natürlich nicht alle Fälle ab, die Saros benötigt. Denn durch die Trennung von Oberfläche und Saros muss die Anbindung nun über das Protokoll gehen. Dafür wurde das Protokoll erweitert. Die Bibliothek LSP4J delegiert über Atribute die Anfragen an die richtigen Stellen.

Es gibt fünf Endpunkte bei meinem Prototyp auf Serverseite, die durch die entsprechenden Klassen repräsentiert und implementiert werden.

saros/account Das ist der Endpunkt über den Konten hinzugefügt, entfernt und bearbeitet werden können. Desweiteren kann das aktuelle Konto gesetzt und alle gespeicherten Konten abgerufen werden.

saros/contact Das ist der Endpunkt über den Kontakte der Kontaktliste hinzugefügt und wieder entfernt werden können. Auch können Spitznamen zugewiesen und die aktuelle Kontaktliste abgerufen werden.

saros/session Das ist der Endpunkt für die Verwaltung von Sitzungen. Über diesen Endpunkt kann sich Beispielsweise mit dem XMPP-Server verbunden und eine Sitzung gestartet werden. Auch Teilnehmer können eingeladen werden.

Die Benamung der Endpunkte richtete sich nach der bisherigen Einteilung der Funktionen in Konten, Kontakte und Sitzungen. Das LSP definiert bereits zwei Endpunkte, nämlich "workspaceünd "textDocument", beide habe ich implementiert und für Saros angepasst.

workspace Dieser Endpunkt kümmert sich um Änderungen im Arbeitsbereich. Er wird vor allem für das Überwachen von Datei- und Ordneränderungen verwendet. Das bedeutet, wenn eine Datei oder ein Ordner gelöscht bzw. erstellt wird landet das Ereignis hier als Nachricht und wird and den Saros Kern weitergeleitet, damit diese Änderungen über die Sitzung synchronisiert werden können. Da beim Löschen die Datei oder der Ordner bei erreichen der Nachricht nicht mehr existiert und somit nicht mehr festgestellt werden kann ob es sich um eine Datei oder einen Ordner handelte, wird in dem Fall immer eine "FileDeletedActivityän Saros gesendet. Man könnte zwar anhand des Dateinamens raten ob es eine Datei oder ein Ordner war, indem man auf Vorhandensein einer Dateiendung prüft, jedoch haben Dateien nicht immer auch eine Endung. Darüber hinaus ist das Löschen hier so implementiert worden, dass es keinen Unterschied macht.

textDocument Dieser Endpunkt empfängt die konkreten Änderungen an den Dateien, d.h. mit Positionsangabe und dem geänderten Text. Zu Beginn der Arbeit musste ich vom von VSC verwendeten Zeilen+Zeichen Position System auf das Offset.System von Saros umrechnen. Gegen Ende konnte ich das allerdings wieder entfernen und die Daten direkt an Saros zur Synchronisation weiter geben. Dieser Endpunkt ist dafür Verantwortlich Änderungen von VSC entgegenzunehmen und an Saros weiterzuleiten, aber auch Änderungen von Saros entgegenzunehmen und diese dann an VSC

weiterzuleiten. Eine Problematik war dabei zu erkennen welche Änderung von Saros initiiert wurde, und welche vom Nutzer. Denn macht man diese Unterscheidung nicht, dann kommt es zu einer Endlosschleife. Denn wenn von Saros eine Textänderung kommt, dann wird diese an VSC weitergeleitet. Diese von VSC durchgeführte Änderung wird allerdings wieder an SLS gemeldet, was dazu führt dass beim Partner die Änderung durchgeführt wird und das wird dann wieder zurückgespielt und so weiter. Um diese Problematik zu umgehen, wird bei Empfang einer Änderung durch Saros die Änderungserkennung für die Datei für den Zeitraum der Anpassung ausgeschaltet. Nach erfolgter (oder fehlgeschlagener) Anpassung wird die Datei wieder für die Änderungserkennung freigegeben, wodurch vermieden wird das Anpassungen durch Saros als Anpassungen durch den Nutzer gewertet werden. Dem Enpunkt wurde noch eine neue Funktion hinzugefügt, die über LSP hinaus geht, damit Annotation einer Datei abgefragt werden können. Denn bei Wechsel des Editors in VSC werden Annotation des inaktiven Editors entfernt und müssen bei aktivierung wieder neu hinzugefügt werden. Dafür wurde der Endpunkt geschaffen, damit VSC aktuelle Annotation einer Datei abfragen kann.

Auch der Client hat Endpunkte für den Server dazubekommen, um die Funktionen von Saros abbilden zu können. Diese werden dazu verwendet um den Client (also VSC) über Änderungen zu informieren. Dazu gehören:

- Statusänderung der Sitzung
- Statusänderung der Verbindung zum XMPP-Server
- Statusänderung eines Kontakts (z.B. Onlinestatus)
- Annotation einer Datei
- Nutzer tritt einer Sitzung bei
- Nutzer verlässt eine Sitzung

5.3.3 Proxies

Saros verwendet den PicoContainer[80] um seine Abhängigkeiten auflösen zu lassen. Der SLS hat das Problem, dass zwei Komponenten erst nach dem Start zur Verfügung stehen, der Container aber davor bereits alle Abhängigkeiten kennen muss. Um dieses Problem zu umgehen, verwende ich für diese beiden Komponenten Proxies[81], die dann nach dem Start die tatsächlichen Implementierungen zurückgeben. Beziehungsweise Anfragen an diese stellvertretend weiterleiten.

5.3.4 Saros Response

Es kann vorkomment, dass bei einer Funktion von Saros ein Fehler geworfen wird. Zum Beispiel wenn versucht wird sich mit dem XMPP-Server zu verbinden, aber kein Konto zur Verbindung ausgewählt ist. Da das auslösen des Verbindungsversuchs über das LSP läuft, führt das Auslösen eines Fehlers zu Fehlschlagen der Anforderung vom

[80] http://picocontainer.com/
[81] Stellvertreter Entwurfsmuster

Client und erweckt den Eindruck, dass bei der Übertragung etwas schief gelaufen ist. Um zwischen diesen Fällen Unterscheiden zu können habe ich den SSaros Response"für Anfragen ohne Rückgabe und den SSaros Result Response"für Anfragen mit Rückgabe eingeführt. Diese liefern im Saros Fehlerfall einen Fehlermeldung zurück anstatt die Anfrage scheitern zu lassen, worauf der Client je nach Implementierung reagieren kann (z.B. durch Anzeige der Fehlermeldung).

5.3.5 Gradle

Analog zu den anderen Java Projekten habe ich den Bauprozess des SLS über Gradle zur Verfügung gestellt. In diesem Projekt habe ich das google-java-format Gradle-Plugin[82] eingeführt, wodurch die Formatüberprüfung des Quelltextes hier nun auch über Gradle erledigt werden kann, und damit unabhängig zu einer Entwicklungsumgebung.

6 Zusammenfassung

Es war ein spannendes Thema, das mir viel Spaß gemacht hat. Den Umfang habe ich allerdings unterschätzt, was zu erheblichem Zeitdruck gegen Ende der Arbeit geführt hat. Allerdings bin ich mit dem Ergebnis durchaus zufrieden.

6.1 Ergebnis

Das Ziel der Arbeit war ein minimaler Prototyp, anhand dessen festgestellt werden kann ob Saros im Umfeld von Visual Studio Code lauffähig ist und ob es sich lohnt diesen weiter zu entwickeln. Es sollte auch der Grundstein für eine mögliche weiterentwicklung gelegt werden.

Der minimale Funktionsumfang wurde aus meiner Sicht erreicht, es ist möglich sich mit dem XMPP-Server zu verbinden, Konten und Kontakte zu verwalten und eine Sitzung mit mehreren Teilnehmern über die Erweiterung zu führen. Die Kernfunktionalität des Projektteilens ist ebenfalls enthalten und Änderungen werden über alle Teilnehmer hinweg synchronisiert. Der Prototyp weist sogar Funktionen auf, die über den minimalen Umfang hinaus gehen, wie z.B. das Hervorheben der Änderungen und die bereits existierenden Ansichten. Der Prototyp bildet aus meiner Sicht eine gute Grundlage für eine weiterentwicklung.

6.2 Ausblick

6.2.1 Unabhängigkeit zur Zielumgebung

Die Art der Umsetzung des Prototyps birgt die Chance Saros unabhängiger von der Entwicklungsumgebung zu machen und eine Portierung auf andere zu vereinfachen. Bei der Portierung von Saros nach VSC fällt im Vergleich zu den anderen Umsetzungen von Saros auf, dass bestimmte Logiken bei jeder Portierung wieder aufs neue

[82] https://github.com/sherter/google-java-format-gradle-plugin

implementiert werden müssen. Zum Beispiel das Dateisystem[83], denn hier scheint es die gravierensten Unterschiede zwischen den Entwicklungsumgebungen zu geben. Allerdings ist dies aktuell Gegenstand der Master-Arbeit von Tobias Bouschen[84]. Ein weitreres Beispiel ist das Verwalten von Annotationen (mit Ausnahme des Servers), hier haben sowohl die Umsetzung für Eclipse, als auch die für IntelliJ eine eigene Logik für das Verwalten der Annotationen (wie viele werden Angezeigt, wie werden Konflikte sich überlappender Annotationen aufgelöst). Das letzte Beispiel ist das "Gerüst", dass jede Implementierung benötigt. Es müssen diverse Klassen implementiert und im IoC[85]-Container registriert werden. Bei einer Portierung nimmt das Herausfinden der obligatorischen und optionalen Komponenten einige Zeit in Anspruch. Es gab einige Abschlussarbeiten, die zum Ziel hatten die Oberfläche für alle Portierungen zu vereinheitlichen um unter anderem den Entwicklungsaufwand und die Einarbeitungszeit zu reduzieren, siehe z.B. die ersten Abschlussarbeiten von Christian Cikryt[5] und Matthias Bohnstedt[3] oder zuletzt von Marius Schidlack[6]. Die gemeinsame HTML-Oberfläche wurde allerdings, aufgrund von Integrationsproblemen in den unterschiedlichen Entwicklungsumgebungen[86], wieder entfernt. Darüber hinaus ist das Verwenden eines Java basierten Browsers für Eclipse und IntelliJ zwar naheliegend, hätte die Portierung zu einer nicht Java basierten Entwicklungsumgebung erschwert. Die Portierung nach VSC verfolgt einen ähnlichen Ansatz wie die HTML-Oberfläche, es soll doppelt Implementierte Funktionen vermeiden, allerdings ist hier nicht die Oberfläche das Ziel sondern ihr Gegenpart, die Saros Logik. Aufgrund der verwendeten Technologien sind sowohl die Oberfläche als auch für die Entwicklungsumgebung spezifische Elemente strikt von der Saros Logik getrennt. Durch diese strikte Trennung und das verwendete Protokoll, sollte es in späteren Iterationen von Saros möglich sein, bei einer Portierung nur die spezifischen Teile der Entwicklungsumgebung und der Oberfläche zu implementieren bzw. anzubinden.

In der derzeitigen Version ist Saros noch Dateibasiert, wie der Saros Server. Wenn Saros über LSP unabhängiger werden soll, muss die Logik des Dateisystems auch an den Client ausgelagert werden. Z.B. darf die Abfrage des Dateiinhalts nicht mehr über das Dateisystem laufen, sondern das SLS muss vom Client den Inhalt anfordern, welcher dann ggf. dan Inhalt aus dem Dateisystem ausliest. Im Falle von VSC deckt der "workspace"Namensraum diese Funktionen bereits ab, die Client Endpunkte müssten um diese Abfragen erweitert werden.

6.2.2 Debug Adapter Protocol

Es könnte in Zukunft von Saros das Debug Adapter Protocol[87], kurz DAP, unterstützt werden. Dadurch könnte man neben dem Quelltext auch den Zugriff auf den oder die lauifenden Debugger teilen um das Erlebnis der Paarprogrammierung noch weiter zu intensivieren. Denn das Debugging ist ein nicht zu unterschätzender Faktor, bei der lokalisierung von Problemstellen.

[83] Gemeint sind die Interfaces aus dem Namensraum saros.filesystem
[84] http://www.inf.fu-berlin.de/w/SE/ThesisSarosCrossIDE
[85] Inversion Of Control
[86] https://www.saros-project.org/contribute/deprecated/html-gui.html
[87] https://microsoft.github.io/debug-adapter-protocol/

Literaturverzeichnis

[1] Electron.

[2] Extension API.

[3] Matthias Bohnstedt. Entwicklung einer IDE-unabhängigen Benutzeroberfläche für Saros. Masterarbeit, Freie Universität Berlin, Inst. für Informatik, 2015.

[4] Nils Hauke Bussas. Entwicklung eines Server-Prototypen für Saros. Bachelorarbeit, Freie Universität Berlin, Inst. für Informatik, 2014.

[5] Christian Cikryt. Evaluating the Use of a Web Browser toUnify GUI Development for IDE Plug-ins. Masterarbeit, Freie Universität Berlin, Inst. für Informatik, 2015.

[6] Marius Schidlack. Neuimplementierung und Weiterentwicklung der HTML-GUI von Saros. Bachelorarbeit, Freie Universität Berlin, Inst. für Informatik, 2017.

[7] VS Code Team. Visual Studio Code 1.0! https://code.visualstudio.com/blogs/2016/04/14/vscode-1.0, April 2016.