Freie Universität Berlin

Masterarbeit am Institut für Informatik der Freien Universität Berlin

Arbeitsgruppe Software Engineering

Saros-Server: Realisierung Nutzer-unabhängiger Sitzungen für den Produktiveinsatz

Victor Brekenfeld
Matrikelnummer: 4681675
victor.brekenfeld@fu-berlin.de

Betreuer:	Franz Zieris
Eingereicht bei:	Lutz Prechelt
Zweitgutachter:	Barry Linnert

Berlin, 31.01.2019

Zusammenfassung

Saros ist ein Werkzeug, das verteilte Paarprogrammierung ermöglicht. Es basiert traditonell auf *host-basierten* Sitzungen. Im Rahmen dieser Arbeit geht es um die Weiterentwicklung des Saros-Servers, welcher jene resultierende Limitierung der ständigen Anwesenheit des einmalig festgelegten Sitzungs-Hosts zu entfernen versucht. Basierend auf vorherigen Arbeiten zu dem Thema soll das Projekt fertiggestellt werden in dem Sinne, dass eine minimale läufige Version in die Saros-Codebasis vollständig integriert wird und so den praktischen Einsatz dieses neuen Tools ermöglicht. Hierzu zählen insbesondere eine Lösung für das Projekt-Sharing, welches traditionell nur durch den Host vergenommen wurden konnte und die Aufarbeitung der mittlerweile veralteten Patchsets der vorherigen Arbeiten. Eben diese Minimalimplementierung wurde erreicht. Weitere Arbeiten, die jedoch für den tatsächlichen Produktiveinsatz noch wünschenswert wären, sind dargestellt.

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.

31.01.2019

Victor Brekenfeld

Inhaltsverzeichnis

1 Einführung

1.1 Saros

"Distributed Pair Programming" (DPP) oder auch verteilte Paar- oder Gruppenprogramming ermöglicht es Entwicklern die vielfältigen Vorteile von Paarprogrammierung über eine aktive Netzwerkverbindung zu nutzen. Laurie Williams und Robert Kessler nennen beispielsweise "Quality, Time, Morale, Trust and Teamwork, Knowledge Transfer" sowie "Enhanced learning" in ihrem Buch "Pair Programming Illuminated" [1, S. 4].

Julia Schenk, Lutz Prechelt und Stephan Salinger kamen in einer Studie aus dem Jahr 2014 zu dem Ergebnis, dass, wenn richtig eingesetzt, DPP mindestens genauso effektiv sein, wie die klassische Paar Programmierung [2]. Sie zeigen hierbei allerdings auch einige Stolpersteine auf, die den Einsatz erschweren können.

Das Konzept des Distributed Pair Programming kann mit einer Vielzahl von technischen Lösungen umgesetzt werden, bspw. Remote-Desktop-Verbindungen (vergl. Remote Pair Programming) oder verteilten IDEs. Letzteres bietet zusätzliche Möglichkeiten der Integration wie beispielsweise gleichzeitiges Editieren von Quellcode. Saros ist ein Werkzeug, das es ermöglicht bis zu fünf Entwicklern sich in einer Saros-Sitzung zusammenzufinden und wird in seiner ausgereiftesten Version als IDE-Plugin für Eclipse (Saros/E) bereitgestellt, um eben eine solche verteilte IDE bereitzustellen. Eine Implementierung für JetBrains IntelliJ (Saros/I) ist in aktiver Entwicklung.

1.2 Motivation

Die Saros IDE-Plugins ermöglichen es synchronisiert mit anderen Nutzern zu Arbeiten. Das heißt Nutzer können live sehen, wie andere Nutzer an Projekten arbeiten. Das schließt natürlich Änderungen am Code ein, aber erstreckt sich auf weitere Details, wie Text-Cursor oder Markierungen anderer Nutzer, die ebenfalls eingeblendet werden. Nicht zuletzt werden weitere Funktionen zur Unterstützung der gemeinsamen Arbeit angeboten, wie wie Chat-Funktion.

Um gemeinsam an einem Projekt oder mehreren Projekten zusammenzuarbeiten, müssen sich alle Teilnehmer in einer Saros-Sitzung zusammen finden. Einer der Teilnehmer agiert hierbei als Host und lädt alle weiteren Nutzer in seine Sitzung ein und füg gemeinsam genutzte Projekte der Sitzung hinzu. Da Saros-Sitzungen Host-basiert sind, sind alle Teilnehmer dieser vom Host abhängig. Verlässt er die Sitzung, ob beabsichtigt oder unbeabsichtigt durch zum Beispiel technische Probleme, bricht die gesamte Sitzung ab und muss neu aufgebaut werden. Früh kam daher im Saros-Projekt der Wunsch auf dieses Problem zu lösen und damit die Idee von Nutzer-unabhängigen Sitzungen. Zur Lösung sollte es möglich sein ohne menschlichen Host auszukommen. Ein Server als IDE-losen Sitzungsteilnehmer, der die Rolle des Sitzungshosts übernehmen konnte, sollte entwickelt werden.

Neben der technischen Komponente würde ein Saros-Server als Tool neue Einsatzmöglichkeiten bieten. In Unternehmen könnte ein permanent laufender Sitzungs-Server die Möglichkeit schaffen zu jeder Zeit anderen Entwicklern beizutreten.

Diese Option ist insbesondere für Arbeitnehmer interessant, die beispielsweise von Zuhause oder in Co-Working-Spaces arbeiten.

Auf technischer Seite ergeben sich eine Reihe von Herausforderungen für eine Implementierung, da der Host einer Sitzung in Saros traditionell eine Reihe von exklusiven Rechten besitzt. So ist er als einziger in der Lage Teilnehmer einer Sitzung hinzuzufügen oder Projekte in der Sitzung zu teilen. Außerdem dient er als Mittelpunkt für sämtliche Netzwerkkommunikation. Alle Aktion der Klienten werden zunächst an den Host gesendet, welcher diese anschließend weiterleitet oder verwirft. Die Beibehaltung dieser Rechte funktioniert im Betrieb eines eigenständigen Servers nicht. Es muss die Möglichkeit geben *von außen* - also aus einem Klienten heraus - einer Sitzung beizutreten oder Projekte dem Server hinzuzufügen, was gegeben der aktuellen Netzwerkstruktur einiges an Komplexität hinzufügt.

1.2.1 Erster Server Prototyp (Nils Bussas)

Die erste Umsetzung wurde von Nils Bussas in Form eines Eclipse-basierten Prototypen geschrieben [3]. Zu diesem Zeitpunkt existierte nur die Eclipse-Implementierung, die Entwicklung des Prototypen als Plugin bot sich also an. Während dieser Prototyp viele Schwierigkeiten einer zukünftigen Implementierung bereits gut aufzeigte und als erfolgreiches Experiment anzusehen ist, war es jedoch nicht möglich Projekte zu teilen in den daraus resultierenden Server-Sitzungen. Er war also praktisch nicht einzusetzen, auch wenn dies wegen der Bindung an eine vollwertige Eclipse-Umgebung ohnehin nicht vorgesehen war.

1.2.2 Saros/C

Im Zuge der Portierung des ursprünglichen Eclipse-Plugins auf weitere IDEs wurde die Codebasis aufgeteilt in das genannte Eclipse-Plugin (Saros/E) sowie einen IDE-unabhängigen Kern (Saros/C), der nun ebenfalls von der IntelliJ-Implementierung (Saros/I) verwendet wird. Die Idee war es hierbei möglichst alle Teile des Codes, die unabhängig von der konkreten IDE-Implementierung waren, in eine eigene Bibliothek auszulagern. Die Notwendigkeit ergab sich nach der ersten Version des Saros IntelliJ Plugins, welche große Teile von Saros/E kopierte. Der mehrfach vorhandene Code machte die Wartung und Weiterentwicklung unnötig kompliziert. Die Abkapselung des Saros-Kerns wurde in großen Teilen im Rahmen der Bachelorarbeit von Arndt Lasarzik umgesetzt [4].

1.2.3 Zweiter Prototyp Saros/S (Denis Washington)

Durch diesen Kern konnte dann auch ein echter Sitzung-Server entwickelt werden, ein umfangreiches Projekt dem sich Denis Washington in seiner Masterarbeit annahm [5]. Basierend auf Saros/C wurde ein eigenständiger Kommandozeilen-Klient entwickelt und weiterer Code aus Saros/E nach Saros/C verschoben. Neben den Ergänzungen des Saros-Kerns und der Portierung von Nils Bussas' Code nahm sich Herr Washington dem Problem des Teilens von Projekten mit dem Host an. Es entstand eine erste funktionierende Implementierung, die dem Server als Host nach wie vor die Koordinationsrolle zuwies,

und so stabil ohne die vielfältigen Konsistenz- und Nebenläufigkeitsprobleme der vorhergehenden Implementierungen war [5, S. 11–21].

Leider erwies sich die Umsetzung aller Features als so umfangreich, dass es zwar eine lauffähige Version des Servers gab, alle Bestandteile aber nie vollständig in die Codebasis des Projektes integriert wurden. Dies hängt mit Saros Entwicklungsprozess und seinem Review-System zusammen, welche im folgenden Kapitel detaillierter erklärt werden. Einige für das Starten des Servers wichtige Funktionen und auch die Umsetzung des Teilens von Projekten durch Klienten (Non-Host-Project-Sharing), veralteten so, während sich erst einmal niemand mehr darum kümmerte.

1.2.4 Krummrei

Michael Krummrei versuchte zuletzt dieses Projekt abzuschließen und die ausstehenden Patches des Servers aufzuarbeiten und in die Codebasis einzupflegen [6]. Doch auch er wurde mit dieser Arbeit nicht fertig. Es schienen noch Mängel an der Codequalität der Patches zu geben. Auch der Non-Host-Project-Sharing Patch ist nach Abschluss von Krummreis Arbeit als kompliziertester Teil noch ausstehend.

1.3 Problemstellung

Ziel der Arbeit war es den Saros/S tauglich für den Produktiveinsatz zu machen und damit eine erste veröffentlichte Version für interessierte User bereitstellen zu können. Hierzu musste der aktuellen Stand evaluiert und schlussendlich eine erste Implementierung des Servers in die bestehende Saros-Codebasis integriert werden.

Außerdem sollte eine Liste von noch umzusetzenden Features erarbeitet werden und gegebenenfalls implementiert, die für den Einsatz als unerlässlich angesehen werden.

1.4 Aufbau der Arbeit

Im Folgen ist die Arbeit wie folgt strukturiert.

- Teil 1 ist hiermit abgeschlossen und stellt die Einleitung mit samt Motivation, Vorarbeiten und Problemstellung dar.
- Teil 2 vermittelt notwendige Kenntnisse über die Funktionsweise von Saros und seinen Entwicklungsprozess zum Verständnis der restlichen Arbeit.
- Teil 3 beschäftigt sich dann mit dem Prozess der eigentlich Implementierung. Er geht detailliert auf Probleme und Irrwege während der Arbeit ein.
- Teil 4 beschäftigt sich hingegen mit dem finalen Ergebnis. Hier werden die Erfahrungen während der Entwicklung reflektiert und ein Ausblick auf möglicherweise folgende Arbeiten wird gegeben.
- Teil 5 fasst die wesentlichen Errungenschaften der Arbeit kurz zusammen.
- Teil 6 ist der Anhang.

2 Grundlagen

Zum Verständnis der folgenden Arbeit ist es relevant über einige Interna des Entwicklungsprozesses, als auch über einige Implementierungsdetails von Saros Bescheid zu wissen. Im Folgenden werden diese erklärt bevor es im nächsten Kapitel zu Erklärung der eigentlichen Arbeit geht.

2.1 Entwicklungsprozess

Saros ist ein Projekt das mittels dem Versionskontrollsystem *git* verwaltet wird. Die Softwareentwicklung bei Saros durchläuft auf Grund gegebener Umstände einen klar definierten Prozess. Die Gruppe aktiver Entwickler besteht sowohl aus fest angestellten Mitarbeiten, OpenSource-Entwicklern, sowie Studenten, die im Rahmen einer Arbeit an Saros mitwirken. Die Fluktuation der Studenten ist relativ hoch, da die Studenten zumeist sehr fokussiert darauf sind, ihre gegebene Aufgabe fertigzustellen. Auch wenn Aufräumarbeiten überall am Projekt wie auch sauberer, dokumentierter und getesteter Code ausdrücklich gewünscht sind, fallen diese Aufgaben meist hinten über. Um dem entgegenzuwirken, sind sämtliche Änderungen am Code von anderen Entwicklern einem Review zu unterziehen, bevor sie in der Codebasis übernommen werden können (Merge). Ein positives Review ist ausreichend nach einer Frist von zwei Tagen, um anderen die Möglichkeit zu geben diesem zu widersprechen. Jedes negative Review blockiert einen Patch bis alle offenen Probleme ausgeräumt sind. Sind zwei positive Reviews vorhanden, kann direkt gemergt werden. Gerade die angestellten Entwickler achten meiner Erfahrung nach hierbei vorrangig darauf, dass Code gewissen Qualitätsvorgaben genügt. Zusätzlich besitzt Saros sehr viele Tests. Fehler in einem Testdurchlauf sind ebenfalls ein Ausschlusskriterium für den Merge von Änderungen. Zu Beginn meiner Arbeit wurde technisch dieser Prozess durch das Review-System *Gerrit* unterstützt und die Test-Infrastruktur durch einen *Jenkins*-Server bereitgestellt. Im Laufe der Arbeit wurde dieses System durch ein auf *GitHub* verwaltetes Repository abgelöst. Tests wurden durch *travis-ci* eingebunden.

In Abschnitt 4 komme ich noch einmal auf meine abschließenden Erfahrungen mit diesem Prozess zu sprechen. Er ist durchaus relevant für diese Arbeit, da er mit dafür verantwortlich ist, dass so viele ausstehende Server-Patches es ursprünglich nicht in die Codebasis geschafft haben.

2.2 Context und Lifecycle Management

Saros verwendet zur Verwaltung seiner vielen Komponenten das Konzept von Dependency-Injection. Dependency-Injection beschreibt ein Pattern bei dem ein Objekt die Abhängigkeiten (Dependencies) für andere Objekte bereitstellt. Das Bereitstellen der Abhängigkeit - oft auch gesehen als ein Service - von außen, anstatt dass besagtes Objekt den Service selber erstellt oder auffindet, ist dabei ein zentrales Konzept. Als Ergebnis muss das Objekt weder wissen wie es einen Service instanziiert, noch - im Falle, dass Interfaces benutzt werden - welche konkrete Instanz einen Service bereitstellt, wodurch das Austauschen von Implementierungen sehr einfach wird. Konkret verwendet Saros die Library PicoContainer[1]. PicoContainer stellt *Container* bereit, welche Klassen in Java

[1] http://picocontainer.com

instanziieren und durch Reflektion auch komplexe Abhängigkeitsgraphen programmatisch auflösen können.

Jedes Saros-Plugin startet indem es einen Kontext initialisiert, welcher den Container und damit die einzelnen Bestandteile einer Saros Instanz aufbaut. Die verwendete Implementierung einzelner Interfaces aus Saros/C ist dabei oft spezifisch für das gegebene IDE-Plugin oder im Falle dieser Arbeit des Servers. Dieser Kontext wird normalerweise seinerseits von einer `ContextLifecycle`-Implementierung des entsprechenden Interfaces aus Saros/C verwaltet, welche unter Zuhilfenahme von PicoContainers Lifecycle Features den Aufbau des Containers und Start der Komponenten, sowie das Beenden des Containers verwaltet.

Wer an dieser Stelle genaueres über die technischen Aspekte von PicoContainer erfahren will, dem sei die Einführung der PicoContainer Dokumentation[2] ans Herz gelegt.

2.3 Invitation-Prozess

Sofern eine Saros-Instanz fertig initialisiert ist und mit einem XMPP Server verbunden (das XMPP-Protokol wird intern als Transportmethode und für den Verbindungsaufbau benutzt), kann sie andere Instanzen auf dem selben Server einladen ihrer Sitzung beizutreten. Sofern die Einladung angenommen wird, werden Informationen über die aktuelle Sitzung (wie bereits bestehende andere Teilnehmer) ausgetauscht und danach geteilte Projekte auch mit dem neuen Teilnehmer geteilt (siehe den nächsten Abschnitt Projekt-Austausch).

Für den Server wichtig zu erwähnen ist eine optionale Erweiterung des Prozesses, der von Saros/E und Saros/I aktuell nicht unterstützt wird. Ein Teilnehmer kann den Host einer bestehenden Sitzung bitten eingeladen zu werden. Die Entscheidung dieser Bitte nachzukommen liegt beim Host und kann auf verschiedene Arten implementiert werden. Das Fehlen einer Implementierung kommt einer automatischen Ablehnung gleich.

Der Prozess ist dargestellt in Abb. 1.

2.4 Projekt-Austausch

Zum gemeinsamen Arbeiten an einem Projekt muss sichergestellt werden, dass alle Teilnehmer über den gleichen Stand der Dateien verfügen. Zum Austausch von Projekten dient in Saros die `ProjectNegotiation`. Im ersten Schritt sendet der Host eine Liste von Dateien, die zu dem Projekt gehören, an die Klienten. Diese vergleichen die Liste mit bereits existierenden Dateien auf ihrer Seite - sofern es welche gibt - und senden an den Host eine Liste der noch fehlenden Dateien. Zuletzt werden je nach Übertragungsmethode - traditionell als ZIP-Archiv - diese vom Host und alle Klienten versandt. Im Anschluss sind die Projekte synchronisiert und die `ProjectNegotiation` damit abgeschlossen.

Der Prozess ist dargestellt in Abb. 2

[2] http://picocontainer.com/introduction.html

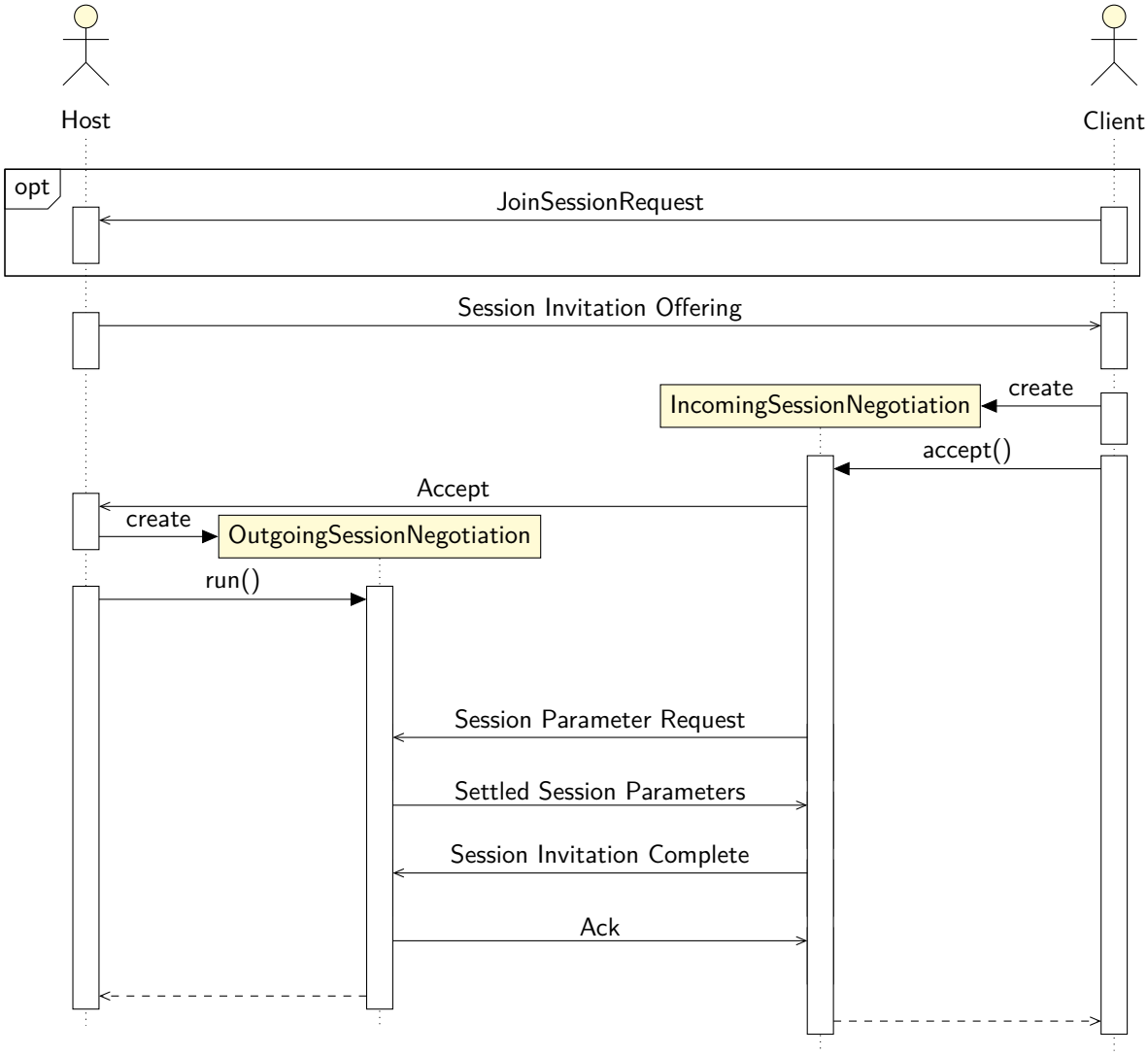


Abbildung 1: Vereinfachte Darstellung des Session-Invitation Prozesses (UML-Sequenzdiagram)

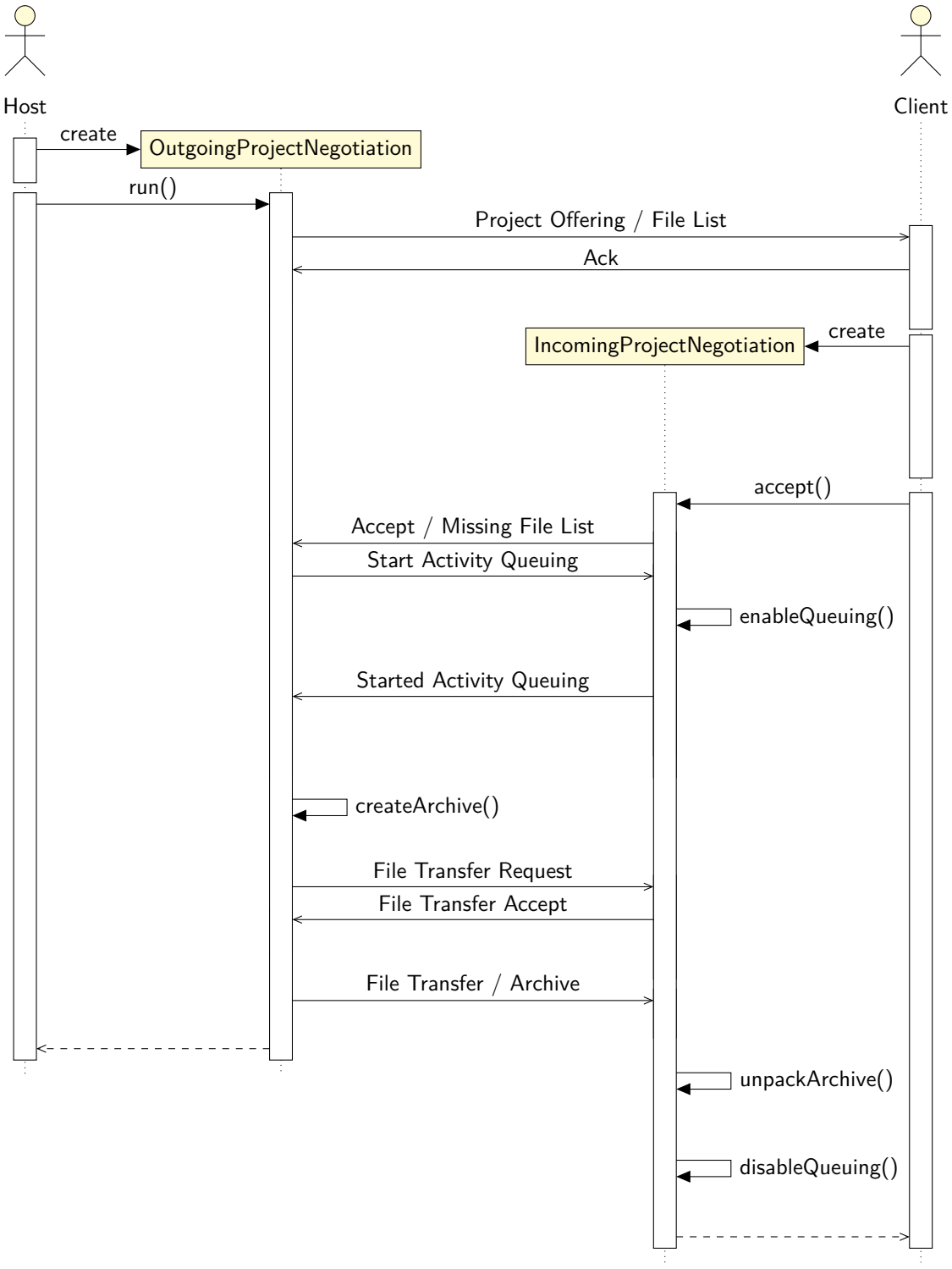


Abbildung 2: Vereinfachte Darstellung des ProjectNegotiation Prozesses

2.5 Aktivitäten

Ist die `ProjectNegotiation` abgeschlossen, werden Änderungen durch die Teilnehmer durch `Activities` oder zu Deutsch Aktivitäten übertragen. Diese können sowohl Statusänderungen wie neue Cursorpositionen in geöffneten Dokumenten, sowie Bearbeitungen von Code enthalten. Konflikte werden automatisch durch den Jupiter-Algorithmus [7] beim Host aufgelöst, welcher als Mittelsmann für alle Aktivitäten dient.

2.6 Editoren

Zur Verwaltung geöffneter Dokumente verwendet Saros sogenannte Editoren. Jedes geöffnete Dokument ist durch einen Editor dargestellt, der normalerweise in den Prozess der Verarbeitung und Erzeugung von Aktivitäten eingebunden ist. Dieses Konzept wird für den Server später relevant, da es hier wie zu erwarten keine geöffneten Dokumente gibt und stattdessen der Editor verantwortlich wird effizient Textoperationen anzuwenden. Eine Aufgabe, die in den Plugins durch die IDE geleistet wird.

2.7 Instant-Session-Start

Im Vorfeld wurde bereits der Prozess des Projektaustauschs (Abschnitt 2.4) erklärt. Ursprünglich endete dieser Prozess im letzten Schritt durch das Packen eines Archives aller Unterschiede des Hosts für jeden weiteren Teilnehmer, welcher dieses nach der Übertragung bei sich wieder extrahierte. Der ganze Prozess kann einiges an Zeit benötigen und blockiert für die Dauer der Übertragung das Interface aller Teilnehmer, um eine De-synchronisation der Dateien während der Übertragung zu verhindern. Letzter Punkt ist die Hauptmotivation für Stefan Molls Arbeit am Instant-Session-Start. Es sollte die Zeit zwischen dem Start der Übertragung eines neuen Projektes und dem Moment an dem alle Teilnehmer daran arbeiten können verkürzt werden. Die Idee war hierbei die Dateien einzeln zu übertragen und während des Austausches bereits die Bearbeitung fertig synchronisierter Dateien zuzulassen.

Die Art der Übertragung war hierbei nicht genauer spezifiziert, es war jedoch angenommen worden, dass eine Übertragung mittels Aktivitäten (Abschnitt 2.5) gut zu realisieren wäre, da die nötige Infrastruktur bereits besteht. Im Verlauf von Stefan Molls Arbeit entschied er sich jedoch dagegen, da eine effiziente Implementierung eine Umstrukturierung der Aktivitätenverarbeitung erfordert hätte [8, S. 13–15].

3 Umsetzung

3.1 Anforderungsbestimmung

Gegeben der Problemstellung (Abschnitt 1.3) sind zur produktiven Nutzung des Servers folgende Anforderungen zu erfüllen.

- Fertigstellung bisheriger Vorarbeiten
- GUI-Integration
- Non-Host-Project-Sharing
- Profiling

Um diese Punkte soll es in den folgenden Abschnitten im Detail gehen.

3.1.1 Fertigstellung vorhandener Funktionalität

Was genau funktionierte war zum Zeitpunkt des Starts meiner Arbeit nicht klar, aber im Laufe der Arbeit ergab sich folgende Liste an ausstehenden Patches die für einen minimalen und funktionsfähigen Server notwendig waren:

- Fertigstellung des `ServerNegotiationHandlers`
- Fertigstellung des `ServerEditorManagers` und `ActivityExecutors`
- Fertigstellung des `JoinSessionRequestHandlers`
- Implementierung des Sitzungsstarts des Servers und eines `ContextLifecycle`
- Fertigstellung/Überarbeitung des Patches zum Non-Host-Project-Sharing

Bei dem `NegotiationHandler` handelt es sich um ein Objekt, um auf eingehende und ausgehende `Session-` und `ProjectNegotiations` zu verarbeiten. Siehe hierzu die Beschreibung der beiden Prozesse der Session-Invitation (Abschnitt 2.3) und des Projekt-Austauschs (Abschnitt 2.4). Bei dem `EditorManager` handelt es sich um ein Objekt, um offene Editoren (Abschnitt 2.6) zu verwalten, während ein `ActivityExecutor` eingehende Aktivitäten (Abschnitt 2.5) verarbeitet. Konkreter geht es bei der Umsetzung darum Aktivitäten, welche Dateien betreffen, auch auf dem Dateisystem des Servers abzubilden. Beim Sitzungsstart geht es darum den sehr unsauberen Start-Code des Servers dahingehend zu erweitern, dass er eine Sitzung initialisiert, und den Code aufzuräumen, sodass er wie auch andere Implementierungen einen `ContextLifecycle` verwendet, um seinen Container zu initialisieren. Siehe hierzu auch den entsprechenden Abschnitt (2.2). Bei dem Non-Host-Project-Sharing handelt es sich um ein derart umfangreiches Unterfangen, dass dieses in einem eigenen Abschnitt beschrieben ist.

3.1.2 GUI-Integration

Die bestehenden IDE-Plugins von Saros behandeln den Server wie einen normalen Teilnehmer, was mindestens zu Verwirrung und teilweise zu Problemen beiträgt. Der Server sollte als solcher im GUI erkennbar gemacht werden, Funktionen die im Kontext des Servers keinen Sinn ergeben, sollten nicht zur Verfügung stehen. Als Beispiel ist hier Saros *Follow Mode* zu nennen, bei dem das Plugin einem anderen Teilnehmer *folgt*, also automatisch den Fokus auf die gleiche Datei und Zeile setzt. Da der Server selber keine Dateien aktiv

bearbeitet, sollte diese Funktion für eine gute User-Expirience im Kontext des Servers nicht aufgerufen werden können. Aktuell kann nur der Server Teilnehmer einladen, da er der Sitzungshost ist. Es gibt aber keine Möglichkeit der Interaktion mit dem Server, um eine Einladung zu veranlassen. Ein Betritt zu einer Server-Sitzung sollte sich aus der GUI von Klienten realisieren lassen. Weitere server-spezifische Funktionen sollten abgebildet werden, sofern realisiert.

Dieses Ziel wurde während der Arbeit jedoch verworfen, da die alte Eclipse-GUI zu dem Zeitpunkt der Arbeit durch eine noch in großen Teilen unfertige HTML-GUI abgelöst werden sollte, welche sowohl in Saros/E wie auch in Saros/I funktionieren sollte. Zur Umsetzung der geplanten Änderungen hätten große Teile der Funktionalität in der neuen GUI erst noch fertiggestellt werden müssen. Deshalb einigte man sich darauf, dass alle nötigen neuen Informationen für die GUI bereitstellt werden sollten und soweit implementiert, dass eine Umsetzung möglichst einfach wird, sobald die Arbeiten an der HTML-Oberfläche weit genug fortgeschritten sind. Außerdem sollte diese Arbeit ausreichend dokumentiert und damit festgehalten werden, welche Änderungen gegenüber der Eclipse-GUI umgesetzt werden müssen. Konkret hieß dies vor allem der Austausch von Saros-Instanzen darüber, ob es sich bei dem Host, um den Server handelt oder eines der IDE-Plugins.

3.1.3 Project-Sharing

Es besteht keine Möglichkeit ein bestehendes Projekt mit dem Server zu teilen. Zwei Ansätze sind hier denkbar. Ein Prototyp für den alten Mechanismus besteht noch aus der Arbeit von Herrn Washington, welcher an die aktuelle Codebasis angepasst und weiterentwickelt werden könnte. Für die zweite Variante braucht man ein grundsätzliches Verständnis eines Features namens Instant-Session-Start. Ich möchte hierzu auf gegeben Abschnitt in Kapitel 2 (2.7) verweisen. Diese neue Übertragungsmethode schien mehrere Probleme des alten Ansatzes zu umgehen und wäre daher bevorzugt zu verwenden. Die Methoden müssen getestet und je nachdem implementiert werden. Bereits Denis Washington ging in seiner Arbeit auf die mögliche Interaktion beider Features ein [5, S. 20].

Der zweite Ansatz war hierbei präferiert worden, da vorherige Versuche den Instant-Session-Start zu implementieren Aktivitäten verwendeten zum Projekt-Austausch. Ein Vorteil der Verwendung von Aktivitäten wäre es gewesen, dass diese bereits von allen Klienten gesendet werden können und Saros auch bereits Konflikte zwischen Aktivitäten auflösen kann (siehe Abschnitt 2.5 - Aktivitäten). Für das Project-Sharing (Abschnitt 3.1.3) hätte eine `ProjectNegotiation` auf Basis von Aktivitäten bedeutet, dass das Senden dieser von Klienten an Stelle des Hosts vermutlich ein relativ triviales Unterfangen gewesen wäre, da die zu Grunde liegende Infrastruktur bereits alle möglichen Konflikte behandelt.

Die Implementierung des Instant-Session-Start sollte als Alternative zur etablierten Implementierung einer `ProjectNegotiation` angeboten werden, was zur Folge hatte, dass eine Abstraktion über verschiedene `ProjectNegotiations` angelegt werden musste, um beide Implementierungen weiterhin verwenden zu können. Um Stefan Molls Arbeit also bestmöglich zu unterstützen, um schnell mit der eigentlich Umsetzung des Non-Host-Project-Sharing anfangen zu können, ergaben sich folgende Aufgaben zur Vorbereitung seiner Implementierung: - Abstraktion des `ProjectNegotiation`-Klassen - Aushandeln der zu verwendeten Übertragungsart zwischen Host und Klient.

Als Alternative stand immer noch die Möglichkeit im Raum die Vorarbeit von Herrn Washington zu dem Thema wieder aufzugreifen. Siehe hierzu die Einführung in den zweiten Prototypen (Abschnitt 1.2.3) und die spätere Umsetzung des Features (Abschnitt 3.2.5).

3.1.4 Jupiter-Speicherleck / Profiling

Es existierte ein bislang unbehandeltes Speicherleck in der Implementierung des Jupiter-Algorithmus von Saros. Dieses wurde bereits in der Arbeit von Herrn Washington entdeckt (siehe [5, S. 6.3.1]) und bislang nicht behandelt, weil es nur im Zusammenhang mit inaktiven Teilnehmer zu bemerken ist. Auch wenn es theoretisch jeden Teilnehmer betreffen kann, erreicht es zumeist nur im Serverbetrieb ein kritisches Ausmaß, da der Server immer einen inaktiver Teilnehmer darstellt und muss deshalb gelöst werden. Es ist nicht auszuschließen, dass dieses Problem in der aktuellen Codebasis noch besteht. Generell wäre erneutes Profiling des Servers gegen Ende der Arbeit wünschenswert, da von deutlich längeren Ausführungszeiten auszugehen ist, als bei einer normalen Sitzung.

3.1.5 Zusammenfassung

Zusammenfassend sind folgende Arbeiten zu erledigen:

- Vorbereitungen zum Instant-Session-Start
- Aufarbeitung vorhandener Patchsets
- Umsetzung einer Lösung zum Non-Host-Project-Sharing
- Vorbereitung einer späteren Umsetzung der GUI-Integration in der HTML-GUI
- Profiling des fertigen Servers

3.2 Implementierungen

Die folgenden Abschnitte beschäftigen sich mit der Implementierung der so eben erarbeiteten Anforderungen (siehe Zusammenfassung - 3.1.5). Sie sind beschrieben in zeitlicher Reihenfolge der Umsetzung. Dies dient der Verdeutlichung einiger der Entscheidungen auf dem Weg sowie zum Verständnis einiger Irrwege.

Zur Einarbeitung in die Codebasis versuchte ich die `ProjectNegotiation` zu abstrahieren, da mir hier das Ziel relativ klar erschien und der Prozess tief in Saros integriert ist, sodass ich hoffte viel über die umstehende Infrastruktur zu lernen. Ein weiterer Vorteil dieser Arbeit ergab sich für mich daraus, dass das Non-Host-Project-Sharing sich mir als vermutlich schwierigste Baustelle darstellte. Um mich dieser also so schnell wie möglich widmen zu können, tat ich mein möglichstes, um die Arbeiten in Bezug auf den Instant-Session-Start als erstes fertigzustellen. Es wird also zuerst um die Abstraktion der `ProjectNegotiation` sowie um das Aushandeln des Übertragungstypen gehen, bevor wir zu der Fertigstellung des Patches meiner Vorarbeiter kommen. Zuletzt beschäftige ih mich mit der HTML-GUI, dem Non-Host-Project-Sharing und dem Profiling.

3.2.1 AbstractProjectNegotiation

3.2.1.1 Motivation

Die Notwendigkeit hierfür ergab sich aus dem Ziel des Project-Sharing (Abschnitt 3.1.3) und der Idee dieses auf einer Instant-Session-Start Implementierung aufzubauen. Um zu Verstehen wie der Instant-Session-Start mit dem Project-Sharing zusammen hängt, ist wichtig zu wissen, dass vorherige Versuche dieses Feature zu implementieren Aktivitäten (siehe 2.5) verwendeten wie erwähnt in der Anforderungsbestimmung zum Thema Project-Sharing. Die bisherige Klassenhierarchie der `ProjectNegotiation` ist abgebildet in Abb. 3.

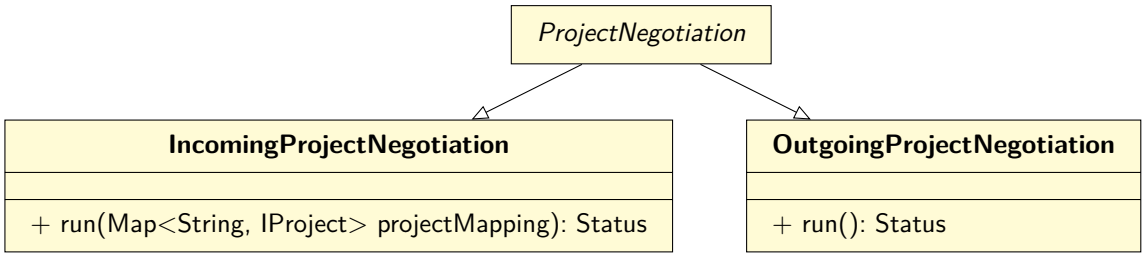


Abbildung 3: Ursprüngliche ProjectNegotiation Klassenhierarchie (zeigt nur als public markierte Operationen). Vergl. Abbildung 4

3.2.1.2 Entwurf

Es existieren zwei vorherige Versuche das Problem zu lösen. Beide sind schlussendlich nicht gemergt worden wegen offener Probleme. Beide abstrahierten damals schon die beiden `ProjectNegotiation`-Klassen `IncomingProjectNegotiation` und `OutgoingProjectNegotiation`. Der erste Patch[3] stammte aus der Arbeit zum Instant-Session-Start von Daniel Theus, welcher zu einer nahezu identischen Klassenhierarchie in seiner Arbeit kam [9, S. 5.3]. Der zweite Patch[4] stammte von Patrick Fehling aus seiner Arbeit zum Instant-Session-Start [10, S. 6.1]. Beide Patches waren relativ groß und sind nie in die Codebasis übernommen worden. Ich entschied mich daher mehrere kleinere Patches zu erstellen, um die Reviews einfacher zu gestalten.

Als ersten Patch wollte ich eine `AbstractIncomingProjectNegotiation` und eine `AbstractOutgoingProjectNegotiation` schaffen auf der die aktuelle ZIP-Archiv-basierte Übertragungsvariante aufbaut ohne weitere Übertragungsmöglichkeiten hinzuzufügen. In einem zweiten Schritt sollte die Art der Übertragung zwischen Host und Klient ausgehandelt werden. Zum Schluss sollte die Implementierung von Stefan Moll nur einen Übertragungstyp und Implementierungen für die `Abstract*ProjectNegotiations` bereitstellen müssen ohne die Infrastruktur stark zu bearbeiten mit Ausnahme der durch die neue Übertragung sicherlich entstehenden Konsistenz- und Nebenläufigkeitsprobleme. Die finale Klassenhierarchie ist zu sehen in Abb. 4.

3.2.1.3 Implementierung

Die eben aufgeführten einzelnen Schritte wurden allerdings nicht so sauber und streng von einander getrennt entwickelt wie sie im Nachhinein eingereicht wurden. Es stelle sich als

[3] https://saros-build.imp.fu-berlin.de/gerrit/#/c/2910/
[4] https://saros-build.imp.fu-berlin.de/gerrit/#/c/3067/

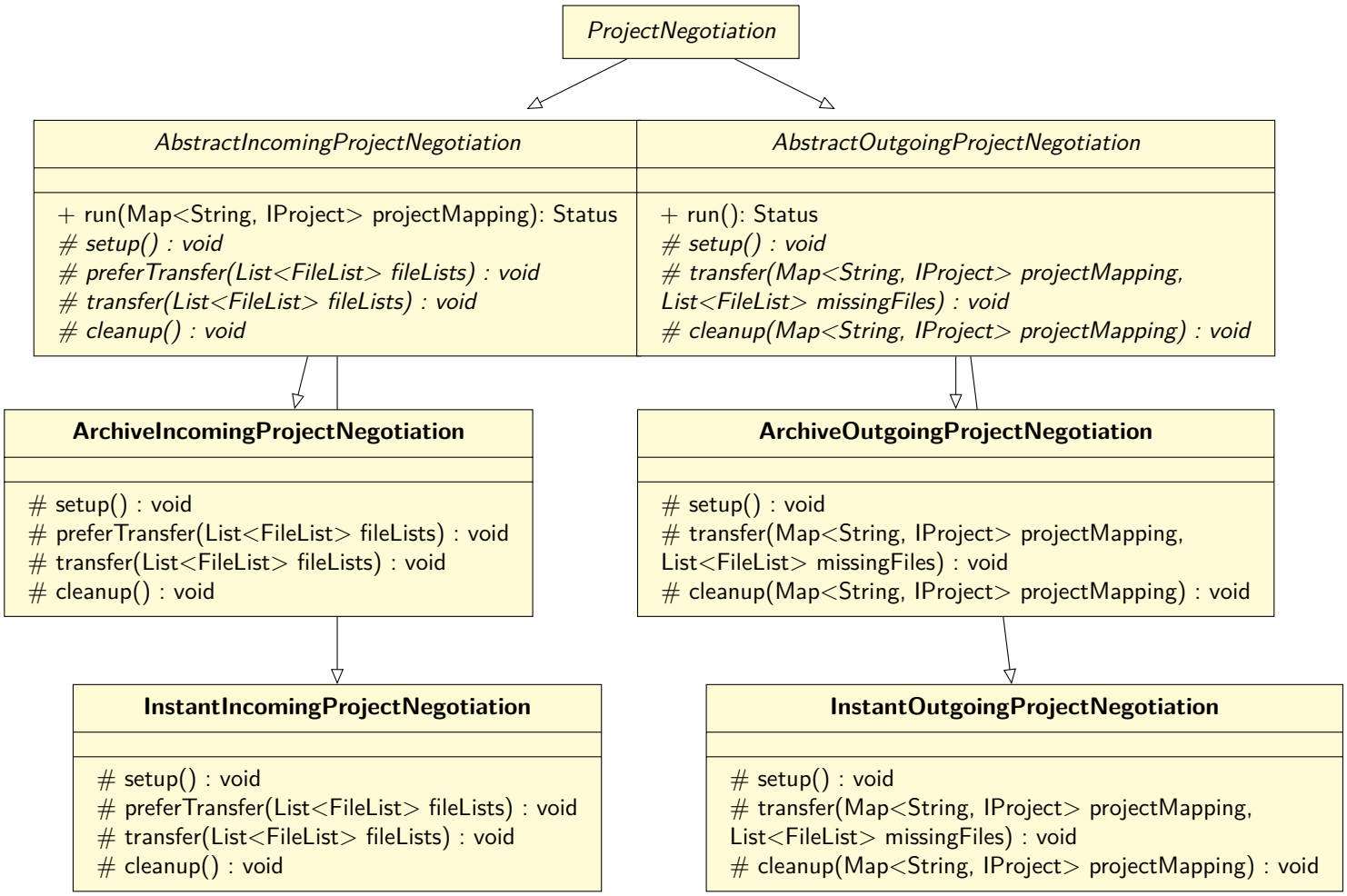


Abbildung 4: Neue ProjectNegotiation Klassenhierarchie mit 'Abstract*ProjectNegotiation' (zeigt nur für den Aufruf / Aufbau wichtige als public und protected markierte Operationen). Vergl. Abbildung 3

sehr schwierig heraus gemeinsame Bereiche der `ProjectNegotiations` zu identifizieren oder auch den Austausch der Übertragungsmethode mit nur einer existierenden Methode zu testen. Als Abfallprodukt entstand daher eine absolut minimalistische und mit vielen Fehlern behaftete Instant-Session-Start Implementierung, die als potentiellen Startpunkt für Stefan Molls Arbeit hätte dienen können, jedoch großteils verworfen wurde. Diese erlaubte es klar die gemeinsamen Teile der (zukünftigen) `ProjectNegotiations` zu identifizieren und in übergeordnete abstrakte Klassen zu verschieben.

3.2.2 Übertragungsmethode

3.2.2.1 Motivation

Im Rahmen der Unterstützung von Stefan Molls Arbeit (siehe Project-Sharing - 3.1.3) musste außerdem ein Weg geschaffen werden, um die zu wählende Übertragungsmethode zwischen den Saros-Instanzen auszuhandeln.

3.2.2.2 Entwurf

Saros kennt viele Arten von Eigenschaften. Diese können entweder klient-spezifisch sein, global für eine Session gesetzt werden oder projekt-spezifisch sein und für all diese Varianten gab es im Quellcode Interfaces, die dazu bestimmt sind, jene Eigenschaft zu verwalten, speichern und falls notwendig zu übertragen. Oft gibt es hierfür keine zentrale Struktur, sondern viel mehr einen etablierten Weg, wo und wann diese Eigenschaften zu senden sind und in welchen Klassen sie gespeichert werden sollten. Die Übertragungsart aber ist spezifisch für eine Verbindung zwischen zwei Saros-Instanzen. Bislang ein Novum in der Saros Codebasis. Es gab nur eine Eigenschaft, die diesem Kriterium entsprach und nach der ich versuchte die Übertragungsmethode zu modellieren. Dabei handelte es sich um die aktuelle und präferierte Farbe eines Klienten. Sie würde übertragen über einen `ISessionNegotiationHook`.

Dieses Interface erlaubt es Klienten präferierte Eigenschaften an den Host beim Aufbau der Sitzung zu senden und bei Abschluss des Sitzungsaufbaus finale Eigenschaften vom Host zu erhalten. Dies bietet die Möglichkeit für den Host die endgültige Entscheidung zu treffen. Der Prozess ist dargestellt in Grundlagen - Invitiation-Prozess (2.3) als *Session Parameter Request* und *Settled Session Parameters*, sowie ausführlicher in Abb. 5. Dieses Interface erschien gut geeignet für den Übertragungstyp, da es ein Aushandeln eben dieses zwischen zwei Instanzen erlaubt. Der Klient kann seinen präferierten Typ angeben, während der Server den tatsächlich verwendeten Typen festlegt und im Zweifel auf die ZIP-Archiv-basierte Übertragung zurückgreifen kann. Da die neue Übertragungsmethode für den Instant-Session-Start zu Anfang ohnehin nur auf Wunsch des Nutzers aktiviert werden sollte, bietet der `ISessionNegotiationHook` genau die nötige Infrastruktur, um zu verifizieren, dass beide Teilnehmer - Host und Klient - die Funktion aktiviert haben.

Leider war kein Auslesen der Eigenschaften eines `ISessionNegotiationHook` auf Hostseite vorgesehen und auch auf Klientseite gab es nur eine Methode namens `applyActualParameters`, um die Einstellungen anzuwenden. Die Idee war auf Klientseite die Eigenschaften entsprechenden Objekten zuzuweisen. Was vielleicht auf Klientseite bei nur einer gleichzeitig aktiven Sitzung noch funktionieren kann, schlägt spätestens auf Hostseite fehl, wenn diese Einstellungen pro Teilnehmer gespeichert werden müssen.

Tatsächlich lasen beide Seiten in einem mit dem Kommentar `HACK` markierten Codeblock die Eigenschaft direkt aus dem `ColorNegotiationHook` aus und übergaben die Werte an den `SarosSessionManager` wodurch sie schlussendlich dem `User` Objekt zugewiesen wurden. Selbst wenn man auf Klienten-Seite hätte versuchen wollen, dieses Verhalten in die dafür vorgesehene `applyActualParameters`-Methode zu verschieben, wäre dies nicht ohne weitere Veränderungen möglich gewesen. Die Farbwerte wurden dem Konstruktor der `SarosSession`-Klasse nach Abschluss der `IncomingSessionNegotiation` übergeben. Das heißt diese Objekte waren noch gar nicht instanziiert zum Zeitpunkt des Aufrufs von `applyActualParameters`. Außerdem schien dieses relativ unwichtige Feature viel zu tief integriert mit einem Kernkonzept wie den Sessions in Saros, wenn es sich sogar im API solch zentraler Klassen niederschlägt. Daher erschein eine Anpassung des `ISessionNegotiationHook` für einen generellen Einsatzweck auf Host- und Klientseite wünschenswert.

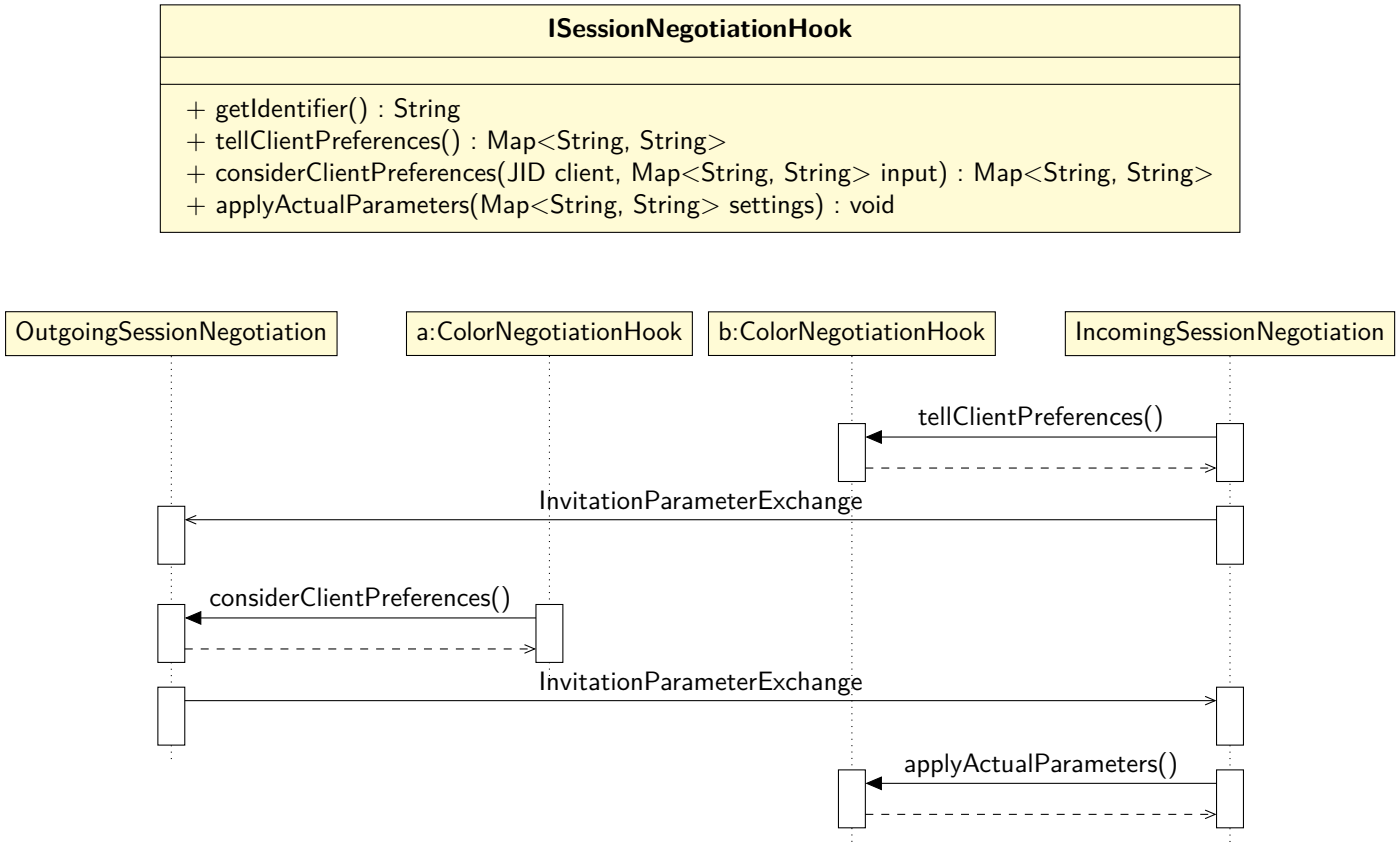


Abbildung 5: Ursprüngliche Struktur des 'ISessionNegotiationHook' und simplifizierte Darstellung des Austausch-Prozesses (vergl. Abbildung 6).

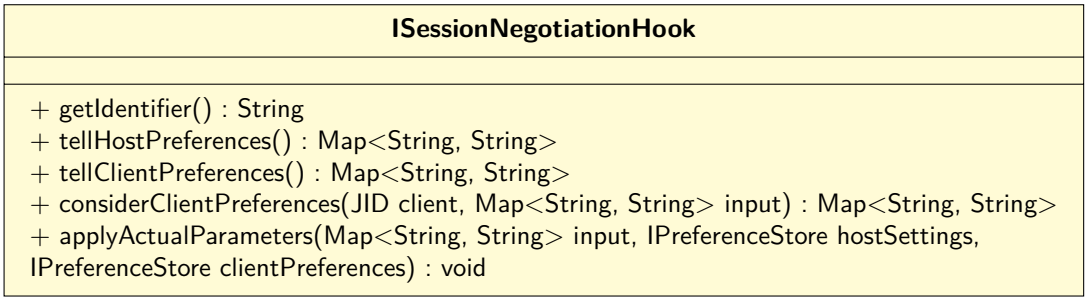
3.2.2.3 Implementierung

Eine erste Implementierung des Austausches eines sogenannten `TransferTypes` verwendete beschriebenen `ColorNegotiationHook`-Hack und speicherte das Ergebnis ebenfalls in der `User`-Klasse. Wenn auch funktionsfähig zeigte die zweite Verwendung eines Hacks in der Codebasis ein konzeptionelles Problem. Nach einiger Diskussion wie das

Problem richtig zu lösen sei mit den anderen Entwicklern, entschied ich mich die Klasse `ISessionNegotiationHook` und die zugehörige Infrastruktur weiter zu entwickeln.

Im Ergebnis besitzt die `SarosSession`-Klasse nun eine Datenstruktur zum Verwalten von Eigenschaften pro User, welche durch die erweitere `ISessionNegotiationHook`-Klasse gefüllt wird. Hierzu wird die `applyActualParameter` Funktion nun sowohl auf Host- als auch auf Klientseite ausgeführt und ihr Objekte zum Speichern der Host-bezogenen und Klient-bezogenen Eigenschaften übergeben. Der Hook kann so das Ergebnis des Austausches global zugänglich machen. Das Format ist hierbei ein Implementationsdetail und nicht näher definiert. Andere Klassen und Funktionen, die auf diese Eigenschaften zurück greifen wollen, müssen den Fall behandeln können, dass diese nicht vorhanden sind - zum Beispiel da in gegebener Saros-Implementierung ein entsprechender Hook fehlt - und entweder einen Fehler werfen oder zum Beispiel auf ein sicheres Standardverhalten zurückfallen. Im Fall des `TransferTypes` ist dies die ZIP-Archiv-basierte Übertragungsmethode.

So wurden für dieses Teilproblem insgesamt drei Patches eingereicht. Einer zur Überarbeitung und Erweiterung des `ISessionNegotiationHooks`, einer um darüber den Übertragungstyp auszuhandeln und einer um gleich die Aushandlung der Farben ebenfalls auf das neue System umzustellen und damit den Hack aus der Codebasis zu entfernen.



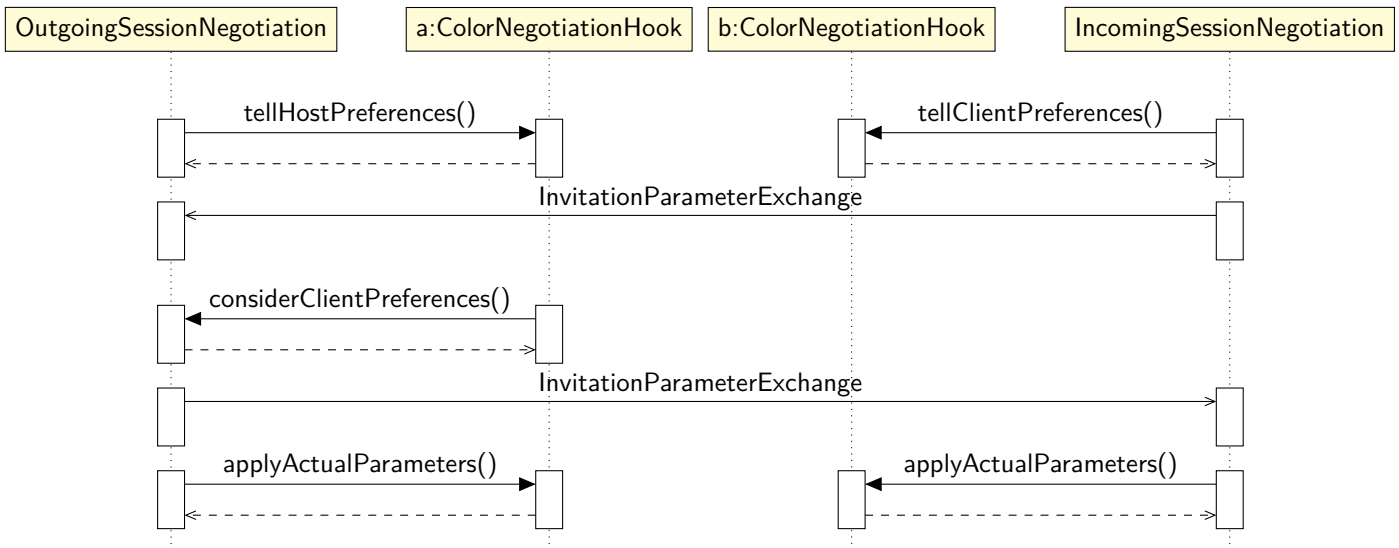


Abbildung 6: Entwurf / Ergebnis der neuen 'ISessionNegotiationHook' Struktur und des angepassten Austausch-Prozesses (vergl. Abbildung 5)

3.2.2.4 Ergebnis

Im Ergebnis hatte sich das Ziel aber erfüllt. Ich hatte ein grundsätzliches Verständnis über den Aufbau und Funktion von Saros/C entwickelt und alle Vorarbeiten zum Instant-Session-Start waren fertiggestellt inklusive einer primitiven Implementierung selbigen. Alle zu Grund liegenden Patchsets wurden zeitnah in die Codebasis übernommen.

3.2.3 Server-Patchsets

Leider konnte ich nach Fertigstellung der Vorbereitungen zum Instant-Session-Start nicht behaupten ein gutes Bild über die Saros/S Codebasis zu haben, was ich im Nachhinein als Fehler ansehe. Ohne meine nun gegeben Grundkenntnisse von Saros/C wäre das Verständnis mir vermutlich deutlich schwerer gefallen, aber ich hätte einiges vielleicht anders priorisiert, wenn mir der Zustand des Servers besser bekannt gewesen wäre. So musste ich feststellen, dass die bisherige Codebasis nicht einmal gestartet werden konnte, da einige für Saros zwingend erforderliche Implementierungen dem PicoContainer (siehe Grundlagen 2.2) nicht hinzugefügt worden waren, da die dazugehörigen Implementierung schlichtweg noch nicht gemergt worden waren. Ein Großteil meiner Arbeit bestand daher darin alte Schwachstellen dieser Implementierungen auszubessern, sie außerdem wieder auf den neusten Stand zu bringen und damit kompatibel mit der aktuellen Codebasis zu machen oder teilweise komplett neu zu schreiben.

Die eigentlichen Implementierungen waren zwar oft sehr simpel, doch hat es bisweilen Zeit gekostet zu verstehen, welchem Zweck sie dienten oder auch welche der verschiedenen ausstehenden Patchsets tatsächlich (noch) relevant für den Betrieb des Servers waren. Auch musste die Entwicklung der Patchsets nachvollzogen werden, da es sich häufig neuere Versionen späterer Arbeiten in dem Review-System fanden, welche die alten Versionen oft genug nicht referenzierten. Die letzte Implementierung, sofern sie denn einwandfrei identifiziert werden konnte, war in einigen Fällen auch nicht die beste Ausgangsbasis für einen neuen Anlauf. Diese waren teilweise mehrfach an neue Interfaces angepasst worden und das ursprünglich Konzept so schwerer nachzuvollziehen. Gerade wenn eine Neuimplementierung angestrebt wurde, waren die früheren Patchsets oft noch klarer in ihrer Struktur und damit die bessere Referenz.

Im Folgenden gehe auch auf jedes dieser Patchsets einzeln ein, bevor es im Anschluss um die verbleibenden Arbeiten geht.

3.2.3.1 EditorManager und ActivityExecutor

Zu den noch ausstehenden Patches zählte eine Implementierung des `EditorManager` Interfaces aus Saros/C, welches in den IDE-Plugins geöffnete Editoren verwaltet (siehe Grundlagen 2.6). Für den Server ist es nicht sinnvoll von *offenen* Editoren zu sprechen, da es keine Benutzeroberfläche gibt. Da wir aber effizient Operationen auf Dateien ausführen wollen, ergibt es Sinn wie ein Editor den Inhalt einer Datei im Speicher vorzuhalten und die Datei nicht direkt wieder zu schließen. Zu diesem Zweck wird eine eingelesene Datei als *Editor* auch auf dem Server im Speicher gehalten und diese geöffneten Dateien mittels einer Map fixer Größe verwaltet. Ist die Map voll wird nach dem Kriterium der zuletzt benutzen Datei die Älteste wieder geschlossen. So sollten Dateien, die gerade auf

Klienten-Seite geöffnet sind und von denen häufige Operationen zu erwarten sind, meistens im Speicher vorliegen.

Eine erste Implementierung von Michael Krummrei [6] benutze Javas `StringBuilder`, um Operationen auf den geöffneten Dateien auszuführen. Diese Implementierung wurde komplett ausgetauscht gegen eine effizientere Gap Buffer Implementierung. Zusätzlich wurde die existierende Implementierung der verschiedenen ActivityExecuter von Herrn Krummrei angepasst, welche die eigentlichen `TextEditActivities` entgegen nehmen und an den Editor weiterleiten. So werden alle Änderungen an Dateien nun auch im Dateisystem des Servers abgebildet.

Ansonsten wurden die alte Patchsets dieser beiden Features großteils wiederverwendet. Der Code ist sehr leicht verständlich und es handelt sich oft um Ein-Zeiler, um die erforderlichen Methoden zu implementieren. Es ist daher uninteressant weitere Details zu erwähnen.

3.2.3.2 NegotiationHandler

Der `NegotiationHandler` ist verantwortlich dafür neue Session- oder ProjectNegotiations zu verschicken und entgegen zu nehmen. Die Server-Implementierung war hier sehr einfach. Jede eingehende Einladung wird vorerst einfach abgelehnt. Jede ausgehende Einladung wird durch einen Thread-Pool nebenläufig ausgeführt. Für die Behandlung eingehender `ProjectNegotiation` musste zuerst das Non-Host-Project-Sharing gelöst werden. Im späteren Verlauf wurde ein Patchset von Herr Washington aufgearbeitet, welches eingehende `ProjectNegotiations` auf Server-Seite unterstützt. Dieses befindet sich aktuell noch in Review. Eingehende `SessionNegotiations` werden gar nicht unterstützt und ersetzt durch die Möglichkeit `JoinSessionRequests` empfangen zu können (siehe JoinSessionRequestHandler). Auch diese Implementierung ist ansonsten technisch sehr uninteressant.

3.2.3.3 JoinSessionRequestHandler

JoinSessionRequests sind ein Implementierungsdetail, dass auf Nils Bussas Arbeit zurück geht (siehe [3, S. 18]) und von Denis Washington später angepasst wurde [5, S. 33]. Es dient dazu eine Anfrage zum Sitzungsbeitritt an einen Host zu versenden. Die `JoinSessionRequestExtension` ist bereits Teil von Saros/C und stellt das Paket bereit, das diese Anfrage darstellt. Bislang behandelt keine Saros-Implementierung dieses Paket.

Ein altes Patchset existiert, das einen `JoinSessionRequestHandler` implementiert. Dieser sehr einfache Handler nimmt auf der Server aktuell jede Request an und versendet darauf hin Sitzungseinladungen. Auch hier waren keine großen Änderungen notwendig. Einzig erwähnenswert ist, dass die Einladungen analog zum `NegotiationHandler` in einer Queue abgelegt und von einem anderen Thread abgearbeitet werden.

3.2.3.4 Sitzungsstart

Erwähnenswert ist außerdem, dass der Server nun automatisch beim Start eine Saros-Sitzung startet und diese mit Beendigung des Prozesses wieder schließt. In Nils Bussas Arbeit diente die `JoinSessionRequestExtension` (siehe vorherigen Abschnitt) noch dazu entweder eine Sitzung zu starten oder einer bestehenden beizutreten. Bei Denis Washingtons Arbeit wurde

die Möglichkeit explizit einen Sitzungsstart zu veranlassen entfernt [5, S. 33] und stattdessen eine Sitzung nach der ersten eingegangen `JoinSessionRequestExtension` transparent für den Klienten gestartet. Diese Arbeit war Teil des "Huge Server Patches"[5] von Denis Washington und war bis zum Ende seiner Arbeit nicht in ein eigenes Patchset extrahiert worden, weshalb mir der Code an dieser Stelle sehr spät auffiel.

Da mir zum Zeitpunkt meiner Implementierung diese Vorarbeit nicht bekannt war und Saros in seiner aktuellen Form ohnehin nicht mehrere parallele Sitzungen unterstützt (ein Problem das im Falle des Servers durch einen Super-Server - Abschnitt 4.4.4 - in Zukunft gelöst werden könnte), entschied ich mich die Sitzung an die Laufzeit des Servers zu koppeln, anstatt diese wie zuvor bei Bedarf zu initialisieren. Es sind keine klaren Vorteile beider Ansätze auszumachen.

Im Zuge dessen wurde eine `ContextLifecycle`-Implementierung hinzugefügt und weitere Aufräumarbeiten an der Initialisierung des Servers vorgenommen, die längst überflüssig waren. So kann der Server nun auch sauber gestoppt werden (die Verbindungen werden beendet und der Context gestoppt anstatt die Programmausführung abzubrechen).

3.2.3.5 Server Konsole

Trotz all dieser Implementierungen war es praktisch nicht möglich den Server zu testen ohne eine Sitzungseinladung in den bestehenden Code hinein zuschreiben, da keine Möglichkeit bestand mit dem Server direkt zu interagieren, noch eine Implementierung die nötigen `JoinSessionRequestExtension` versenden konnte. Zum besseren Testen des Servers entwickelte ich daher parallel eine schlanke Konsole, die es über die Kommandozeile erlaubt neue Teilnehmer einzuladen und Projekte vom Server aus zu teilen. So war es auch möglich bevor das Non-Host-Project-Sharing implementiert war, Implementierungen wie den EditorManager (siehe 3.2.3.1) zu testen, welche ein geteiltes Projekt benötigen.

Sofern der Server mit einer entsprechenden Flag gestartet wird, wird die Konsole beim Start initialisiert und ließt eingegebene Befehle von der Standart-Eingabe. Das erste geparste Wort, definiert durch den ersten gelesenen Whitespace, wird danach verglichen mit einigen eingebauten Befehlen. Wird keiner gefunden, wird im Anschluss mit registrierten Befehlen verglichen. Ein gefundener Befehl wird mit den verbleibenden Argumenten, definiert durch ein gelesenes Line-Break Symbol, aufgerufen. Danach beginnt der Prozess von vorne. Befehle lassen sich durch Implementierungen des `ServerCommand` Interfaces registrieren/hinzufügen. Zwei Befehle sind in Rahmen der Arbeit entstanden. Einer um eine gegebene Liste von JIDs zur Sitzung einzuladen und einer um einen Pfad relativ zum Server-Workspace als Projekt der Sitzung hinzuzufügen. Beide greifen auf gegebene Methoden des `SarosSessionManager` aus Saros/C zurück.

Erwähnenswert ist außerdem, dass eine Implementierung des `IProgressMonitor`, welcher auf Klientseite IDE-spezifisch implementiert ist, um einen Fortschrittsbalken anzuzeigen, für den Server entstanden ist, die - sofern die Konsole aktiviert ist - einen Fortschrittsbalken im Terminal rendern kann.

All diese Funktionen haben sich im Rahmen der Arbeit als sehr hilfreich zum Debuggen des Servers bewährt und wurden dementsprechend später aufgeräumt und in die Codebasis von Saros übernommen.

[5] https://saros-build.imp.fu-berlin.de/gerrit/#/c/2929/

3.2.3.6 Ergebnis

Alle alten Patchsets wurden im Zeitrahmen der Arbeit aufgearbeitet und in die Codebasis gemergt. Die Vorarbeit half die nötigen Schritte zu erkennen, war jedoch nicht immer zu gebrauchen. Die alten Patchsets sind hiermit abgeschlossen, nach den Vorarbeiten zum Instant-Session-Start (bestehend aus den Abschnitten AbstractProjectNegotiation 3.2.1 sowie der Übertragungsmethode 3.2.2) sowie den gerade genannten Patchsets, fehlen noch Anpassungen zur HTML-GUI, das Non-Host-Project-Sharing und das Profiling.

3.2.4 HTML-GUI

Wie oben (siehe 3.1.2) schon erwähnt, wurde der ursprüngliche Plan der GUI-Integration verworfen. Zum Zeitpunkt der Arbeit wurde die gesamte Oberfläche neu gebaut, um als HTML-GUI in verschiedenen IDEs gleicher Maßen einsetzbar zu sein. Dieser Neuimplementierung fehlten allerdings noch zu viele Funktionen, als dass meine Änderungen sinnvoll zu implementieren gewesen wären, ohne mit einem erheblich größeren Mehraufwand verbunden zu sein. Deshalb erstellte ich nur Funktionen, um die Implementierung der GUI-Änderungen mit so wenig zusätzlichem Code wie möglich zu realisieren. Hierzu zählte vor allem die Information, ob es sich bei einem Klient um einen Server handelt oder nicht, durch ein Interface in Saros/C bereitzustellen.

Es gibt bereits eine Möglichkeit festzustellen, ob es sich bei einem Teilnehmer um einen Server handelt. Diese ist realisiert über ein Feature des Saros zugrundeliegenden Netzwerkprotokolls XMPP. Der Server gehört anders als normale Klienten zum `de.fu_berlin.inf.dpp.server` Namespace. Während dieses Feature nett ist, um Server auf einem XMPP-Server zu identifizieren, so ist es keine perfekte Lösung nach Aufbau einer Sitzung. Aufrufe an den XMPP-Stack aus der GUI eines Klienten heraus könnten im Hintergrund Netzwerk-Anfragen auslösen, da sich Namespaces im XMPP-Protokoll zur Laufzeit ändern können, welche damit den GUI-Thread blockieren würden. Andere Fehlerfälle sind denkbar, die Eigenschaft, ob es sich bei einem Teilnehmer um einen Server handelt, ist hingegen statisch und sollte deshalb beim Verbindungsaufbau nur einmalig ausgetauscht werden.

Zum Austausch des Server-Status eines Klients entschied ich mich den bei der Implementierung des Austausches der Übertragungsmethode (siehe 3.2.2) entstandenen Mechanismus des `ISessionNegotiationHook` wieder zu verwenden. Für Details zu dieser Lösung sei auf besagten Abschnitt verwiesen. Die ausgetauschte Eigenschaft (getauft `isServer`) wurde außerdem der `User`-Klasse zugewiesen. Da es sich aktuell beim Server stets um den Host handelt, ist das nicht strikt notwendig. Der einzige Vorteil diese Eigenschaft dem User-Objekt zuzuweisen ist, dass diese Eigenschaft hierdurch auch über den Server zu anderen Teilnehmern propagieren kann, die nicht direkt miteinander verbunden sind. Zwar existiert aktuell kein Grund den Server als normalen Klient einer Sitzung hinzuzufügen, doch solche sind denkbar:

- Hinzufügen eines externen Servers als temporäre Backup-Lösung einer Sitzung.
- Migration einer Sitzung auf einen (anderen) Server.
- Mehrere Server als Fallback in einer Sitzung, falls einer ausfällt.

Diese Implementierung ist bislang nicht Teil der Codebasis. Das Hinzufügen der Eigenschaft zum User-Objekt fügt eine server-spezifische Variable einem Interface von

Saros/C hinzu, was ein Anti-Pattern darstellt. Auch der Vorschlag einer Umbenennung in `isHeadless`, um einen generelleren Use-Case abzudecken, löste das Problem nicht. Den `User`-spezifischen Code wegzulassen - um dem Problem erst einmal aus dem Weg zu gehen - war bis zum Zeitpunkt der Abgabe auch keine Lösung. Die Verwendung von dem `ISessionNegotiationHook`-Interface benötigte relativ viele Zeilen an Code für den Austausch eines simplen Booleans, was auch als Schwäche des Patchsets angesehen wurde. Das rührt von der kürzlichen Veränderung des Interfaces und der deshalb höheren Komplexität, um mehr Use-Cases abzudecken. Auch ein weiteres Patchset, dass eine weitere abstrakte Klasse auf Basis des `ISessionNegotiationHook` hinzufügte, um solche simplen Austausche von Variablen deutlich zu vereinfachen (getauft `PropertyHooks`), stoß auf wenig Gegenliebe, da die `isServer` Variable den einzigen `ISessionNegotiationHook` darstellte, der davon profitieren konnte.

Auf der Suche nach einer besseren Lösung wurde vorgeschlagen das Versions-API zu benutzen, das dazu dient zwischen Saros-Implementierungen die verwendeten Versionen auszutauschen. Das zu Grunde liegende Paket unterstützt bereits zusätzliche Daten einer Implementierung anzufügen, wie zum Beispiel um welche konkrete Saros-Implementierung es sich handelt. Hieraus wäre klar abzulesen, ob eine Saros-Instanz ein Server ist. Die nötigen Anpassungen des APIs sind allerdings gerade durch eine Diskussion des in Zukunft zu verwendeten Versionsschemas blockiert (GitHub Issue #351[6]).

3.2.4.1 Sitzungsbeitritt

Bei der Recherche nach vorherigen Patchsets stolperte ich außerdem auf eines, dass es erlaubt aus Saros/E heraus mittels vorher benannter `JoinSessionRequests` (siehe 3.2.3.3) einer Server-Sitzung beizutreten. Da der Patch sehr klein und ich unzufrieden über den unzureichenden Fortschritt bei der GUI-Integration (siehe 3.1.2 und 3.2.4) war, wurde dieser ebenfalls aktualisiert.

Es wird eine neue `Action` hinzugefügt, welche in Eclipse benutzt werden können, um neue Interaktionen zur GUI hinzuzufügen. Diese testet auf den `de.fu_berlin.inf.dpp.server` XMPP-Namespace (siehe HTML-GUI 3.2.4) und sendet dann eine `JoinSessionRequest` ab. Der Code verändert außerdem den bestehenden GUI-Code so, dass sofern genannter Namespace vorhanden ist ein Eintrag im Kontextmenu eines Kontakts angezeigt wird. Im Ergebnis kann ein Nutzer nun, der einen eingeloggten Server in seiner Kontakt-Liste hat, aus dem Kontextmenu eine `JoinSessionRequest` versenden, die vom Server sofort mit einer Sitzungseinladung quittiert wird.

Der Patch ist aktuell noch nicht in die Codebasis übernommen worden. So wäre zusammen mit dem Non-Host-Project-Sharing-Patch der vollständig kopflose Betrieb des Servers möglich.

3.2.5 Non-Host-Project-Sharing

Die Idee des Instant-Session-Start schien zu Beginn der Arbeit mit dem Problem des Non-Host-Project-Sharing verknüpft zu sein (siehe dazu Project-Sharing 3.1.3). Beim Instant-Session-Start (siehe 2.7) ging es darum, dass anstatt beim Projektaustausch alle zugehörigen

[6] https://github.com/saros-project/saros/issues/351

Dateien in ein Zip-Archiv zu verpacken, diese über Aktivitäten (siehe 2.5) auszutauschen, wie sie auch zur laufenden Sitzung verwendet werden, um neue Dateien und Änderungen zu verteilen. Der Vorteil läge darin einerseits vorhandenen und gut getesteten Code der Aktivitäten wieder zu verwenden, andererseits dass Teilnehmer nicht auf die Zip-Datei warten müssten, sondern direkt mit allen bereits übertragenen Dateien arbeiten könnten. Daher auch der Name des Features.

Die Lösung des Non-Host-Project-Sharing schien bei dieser Implementierung trivial, da Konflikte von Aktivitäten bereits aufgelöst werden können und die Locking-Verfahren der alten `ProjectNegotiation` somit im Konzept überflüssig geworden wären. Während der Umsetzung von Stefan Moll stellte sich allerdings eine alternative Übertragung mittels direkter Netzwerkstreams als die bessere Lösung heraus [8, S. 13–15].

Hierdurch wurde die überflüssig geglaubte Arbeit von Denis Washington zu dem Thema wieder interessant, da beide Vorhaben nicht mehr direkt von einander profitierten. Die umfangreichen Änderungen an der Infrastruktur der ProjectNegotiation machten das Portieren des Patches aufwendiger. Trotzdem wurde das vorhandene Patchset als Grundlage eines neuen Patches genommen, da es große Teile der existierende Infrastruktur der `ProjectNegotiation` wieder verwendet. Um das zu erreichen findet zuerst zwischen Klient an den Host eine `ProjectNegotiation` statt, bevor der Host anschließend wiederum das Projekt mit den anderen Teilnehmern teilt. Dieser Ansatz braucht wenige Änderungen an der bestehende Codebasis, da bloß ein neuer Spezialfall (Klient->Host) eingeführt wird, anstatt auch den Klients zu erlauben untereinander Projekte zu teilen. Dieses Vorgehen lässt außerdem auch alle Kontrolle über geteilte Projekte beim Host.

Trotz des - relativ gesehen - kleinen Patches und einer schlussendlich überschaubaren Zahl von Änderung gegenüber Washingtons ursprünglichen Patchsets war die Entwicklung sehr zeitaufwendig. Viele der Änderungen waren nicht mehr direkt übertragbar, vor allem die Teilung der `ProjectNegotiation` in `InstantProjectNegotiation`-, `ArchiveProjectNegotiation`- und die zugrundeliegenden `AbstractProjectNegotiation`-Klassen im Rahmen von Stefan Molls Änderungen verschob große Teile des Codes und ordnete ihn teilweise auch um. Gerade die Neuordnung machte es nötig alle Änderung präzise nachzuvollziehen, um Nebenläufigkeitsfehler so gut wie möglich auszuschließen. Der Patch musste schließlich garantieren können, dass während der verschiedenen `ProjectNegotiations` kein Teilnehmer die Dateien bereits bearbeiten kann und so Inkonsistenzen erzeugt werden können. Ironischer Weise erschwerten die ursprünglich hilfreich geglaubten Änderungen von Herrn Molls Arbeit somit tatsächlich diesen Patch.

Im Folgenden fasse ich die Unterschiede und Überlegungen auf Grund der neueren Codebasis gegenüber dem alten Patchset zusammen. Für eine ausführlichere Beschreibung des alten Patchsets verweise ich an dieser Stelle auf Herrn Washingtons Arbeit [5, S. 11–21].

3.2.5.1 Queueing

Die erste Änderung betrifft ein Patchset von Stefan Moll, welche parallele `ProjectNegotiations` eines Nutzers in großen Teilen verbietet. Hierzu werden die `ProjectNegotiations` in einer Queue zwischen gespeichert, sollte aktuell eine andere aktiv sein und später ausgeführt. Der *Non-Host-Project-Sharing*-Patch startet eine neue `ProjectNegotiation` an die verbleibenden Klienten nach Abschluss der

initialen Übertragung zum Host. Das neue Patchset ignoriert diesen neuen Queueing-Mechanismus absichtlich und führt wie auch das originale Patchset die anschließende `ProjectNegotiation` in `negotiationTerminated` des `NegotiationListener` des `SarosSessionManagers` aus. Konflikte sind hierbei nicht möglich, da der Code, der die nächste `ProjectNegotiation` startet, im gleichen Listener ist und erst im Anschluss ausgeführt wird. Die Invariante nur einer gleichzeitigen `ProjectNegotiation` wird also aufrecht gehalten.

Alternativ hätte die neue `ProjectNegotiation` der Queue hinzugefügt werden können. Aus Gründen der Nutzerfreundlichkeit habe ich mich dagegen entschieden. Der ganze Prozess betrifft nur ein Projekt und gehört logisch zusammen, so sollte er daher von dem User auch wahrgenommen werden. Das hinzufügen der `ProjectNegotiation` zur Queue hätte bewirken können, dass andere `ProjectNegotiations` zwischendurch ausgeführt werden. Da die Invariante sich auch ohne Nutzen der Queue so einfach aufrecht erhalten ließ, entschied ich mich deshalb dagegen.

3.2.5.2 AbstractProjectNegotiation

Einzelne identische Teile der verschiedenen `ProjectNegotiations` waren zum Zeitpunkt meiner Patches noch nicht in die `AbstractProjectNegotiations` verschoben worden. Dies wurde nachgeholt, da es so möglich war sämtliche logische Änderungen an den `ProjectNegotiations` des alten Patchsets im neuen Patchset in den `Abstract*ProjectNegotiations` umzusetzen. Es sollte also kompatibel sein mit allen darauf aufbauenden `ProjectNegotiations`. Und in der Tat zeigen erste Tests, dass es auch mit der aktuellen Implementierung des Instant-Session-Start funktioniert. Spätere Arbeiten von Stefan Moll ändern besagten verschobenen Teil implementationsspezifisch. Hier ist also noch Arbeit erforderlich, um diese Kompatibilität auch für die Zukunft garantierten zu können.

3.2.5.3 Anpassungen am Instant-Session-Start

Während sämtliche Logik sich in der `AbstractProjectNegotiation` umsetzen ließ, heißt das nicht, dass nicht doch einige Detailveränderungen an den einzelnen Implementierungen vorgenommen werden mussten. An einigen Stellen des Codes wird das `SarosSession`-Objekt (intern zur Repräsentation der aktiven Sitzung) benutzt, um die ID eines Projekts in der `ProjectNegotiation` zu beziehen. Dies schlägt allerdings fehl im Falle der initialen Übertragung des Klienten an den Host, da der Klient das zu teilende Projekt noch nicht als Teil der Sitzung ansieht bis der Host die Übertragung erhalten und bestätigt hat. (Anders als der Host, welche neue Projekt direkt in die Sitzung aufnimmt.)

Folglich besitzt zu diesem Zeitpunkt die `SarosSession` noch keine Informationen über das Projekt. Eine Lösung ist bereits in Denis Washingtons Arbeit entstanden, welche die nötigen Informationen der `ProjectNegotiation` übergibt anstatt sie von der `SarosSession` zu beziehen (vgl. [5], S.17). Diese Lösung wurde auch in der Instant-Session-Start Implementierung umgesetzt, die ebenfalls an einigen Stellen auf die `SarosSession` zugriff, um entsprechende Metadaten zu beziehen.

3.2.5.4 Test

Das alte Patchset ist bisher nur manuell getestet worden. Hier waren zwar keine Fehler auszumachen, aber gerade durch die zahlreichen Anpassungen entschied ich mich dafür den bei Denis Washington bereits gewünschten STF-Test für das Non-Host-Project-Sharing umzusetzen (vgl. [5], S. 19). STF[7] ist Saros eigenes Test-Framework, welches automatisiertes GUI-Testing von Saros/E erlaubt.

Der Test enthält drei Teilnehmer (Alice, Bob und Carl) und läuft wie folgt ab:

- Alice erstellt eine Sitzung mit Bob und Carl und einem im folgenden nicht weiter relevanten Projekt.
- Alice ist also Host der Sitzung.
- Bob erstellt ein neues Projekt mit einer txt Datei.
- Bob startet damit das neue Projekt zu teilen.
- In einem separaten Thread fängt Bob an in besagte Text Datei immer wieder "Bob" zu schreiben.
- Alice (der Host) nimmt Bobs Projekt an, öffnet die Datei und fängt ihrerseits an "Alice" immer wieder in die Datei zu schreiben.
- Carl nimmt nun als letzter das Projekt von Alice an, wartet, dass die Übertragung abgeschlossen ist, öffnet die Datei und schreibt seinerseits "Carl" einmalig in die Datei.
- Alle Teilnehmer hören auf zu schreiben.
- Der Test wartet bis in den Saros-Instanzen alle Aktivitäten verarbeitet wurden und vergleicht den Inhalt der Text-Datei auf allen drei Instanzen, um Inkonsistenzen zu erkennen.

Da zuerst Bob und später Alice, sobald sie Zugang zum dem Projekt haben, konstant in die Textdatei schreiben sollte jeder Nebenläufigkeitsfehler in der Implementierung des Non-Host-Sharing-Patchsets schlussendlich einen anderen Zustand der Datei auf den verschiedenen Instanzen erzeugen.

3.2.5.5 Ergebnis

Im Ergebnis ist eine funktionierende Implementierung entstanden, die zwar nicht von der parallelen Arbeit am Instant-Session-Start profitiert, aber immerhin kompatibel ist. Eine der größten Baustellen der Vorarbeiten ist somit endlich abgeschlossen und auch in die Codebasis integriert worden.

3.2.6 Profiling

Beim Profiling mittels VisualVM konnte der von Herrn Washington entdecke Memory Leak wieder gefunden werden. Dank der detailreichen Beschreibung des Problems waren die entsprechenden Stellen im Code relativ einfach zu finden.

Zum besseren Verständnis möchte ich gegebene Beschreibung zitieren:

> Die Untersuchung eines Abbilds auf Referenzen zu den Jupiter-Aktivitäten ergab, dass diese in Listen gesammelt wurden, die wiederum innerhalb von Objekten der Klasse `Jupiter` als `ackJupiterActivityList` abgelegt waren. Wie eine Recherche des Jupiter-Algorithmus ergab, hing dies damit

[7] http://saros-build.imp.fu-berlin.de/stf/STF_Manual.pdf

> zusammen, dass der Sender einer Textbearbeitungsoperation diese so lange zwischenspeichern muss, bis der Empfänger die Anwendung der Operation bestätigt hat. Andernfalls wäre es bei Erhalt einer Operation unmöglich zu bestimmen, ob – und wenn, wie – diese transformiert werden muss, falls das Gegenüber noch nicht alle an ihn gesendeten Operationen angewendet hat. […] In der Saros-Implementation des Jupiter-Algorithmus können Operationen ausschließlich dadurch bestätigt werden, dass der Empfänger seinerseits eine neue Operation in Form einer `JupiterActivity` versendet. Darin gespeichert ist die Zahl der bisher empfangenen und angewendeten Operationen der Gegenseite (Remote Operation Count); diese kann so darauf schließen, welche Operationen nicht mehr zwischengespeichert werden müssen. Diese Art der Bestätigung ist ausreichend, wenn alle Sitzungsteilnehmer regelmäßig Bearbeitungsoperationen durchführen; sobald das ein Teilnehmer aber nicht tut […], werden die an ihn gesendeten Operationen nie bestätigt und der Sender (sprich, der Host in seiner Rolle als Jupiter-Server) kann sie folglich auch nie löschen, sodass sie sich immer weiter ansammeln. [5, S. 6.2.1]

In der Implementierung des Jupiter-Algorithmus ist das Feld `ackJupiterActivityList` für den Memory Leak verantwortlich. Es wird gefüllt durch Aufrufe der Funktion `generateJupiterActivity()` und geleert durch `discardAcknowledgedOperations()`, welche unter anderem von `receiveJupiterActivity()` aufgerufen wird. Es gilt also auf Serverseite regelmäßig Jupiter-Aktivitäten zu versenden, damit dieser Codepfad aufgerufen wird. Verwaltet werden die einzelnen Instanzen des Algorithmus pro offenen Editor von der Klasse `JupiterClient`.

Das Problem konnte mittels VisualVM klar erkannt werden. Um möglichst schnell das Problem zu reproduzieren, entwickelte mit Hilfe des STF-Frameworks einen kleinen Test, um mit drei Klienten den Server als Sitzungshost mit Aktivitäten zu fluten. Der Code sowie die erzeugten Heap-Dumps finden sich in den beliegenden Dateien. Ein erster Screenshot (Abb. 7) zeigt die Ausgangssituation beim Start des Test ein zweiter (Abb. 8) nach ca. einer Stunde Laufzeit.

3.2.6.1 Entwurf

Mehrere Lösungsansätze sind denkbar.

- Als einfachste Lösung könnten alle Aktivitäten direkt bestätigt werden, anstatt dass dies indirekt durch Bearbeitungsoperationen passiert. Diese Lösung würde den Netzwerkverkehr unnötig stark erhöhen und wurde daher verworfen.
- Etwas eleganter wäre eine Art Heartbeat. Ein Klient versendet hierbei regelmäßig Aktivitäten um für den Fall, dass keine Bearbeitungsoperationen geschehen dem Server eine Möglichkeit einzuräumen trotzdem frühere Operationen zu verwerfen.
- Bereits von Herrn Washington wurde solch eine Inaktivitätsnachricht vorgeschlagen, die jedoch nur nach einer gegebenen Zeit ohne Aktivität versendet werden sollte. Die Lösung würde den nötigen Netzwerkverkehr am wenigsten erhöhen.

Ebenfalls zu berücksichtigen ist, dass - wie den beiliegenden Heap-Dumps und den Grafiken zu entnehmen ist - der Memory Leak nicht besonders gravierend ist. In einer Stunde Laufzeit konnten mit einem unrealistischen Szenario von drei programmatisch spammenden Klienten ca. 10MB geleakt werden. Das heißt eine Inaktivitätsnachricht muss nicht sehr häufig

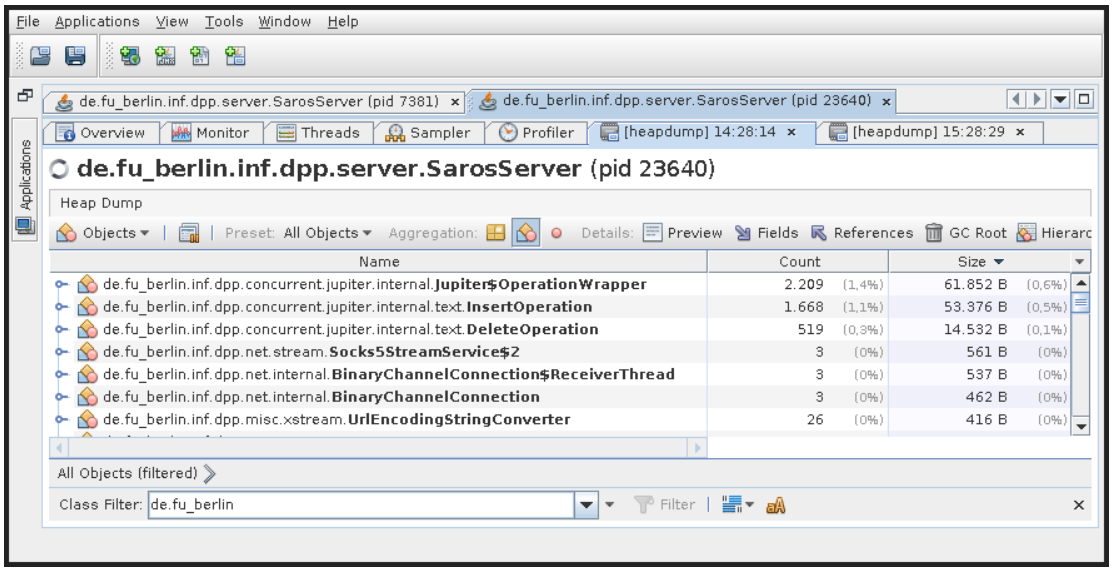


Abbildung 7: VisualVM HeapDump Analysis. Zeigt die meist allokierten Objekte des Servers zum Start des Profilings.

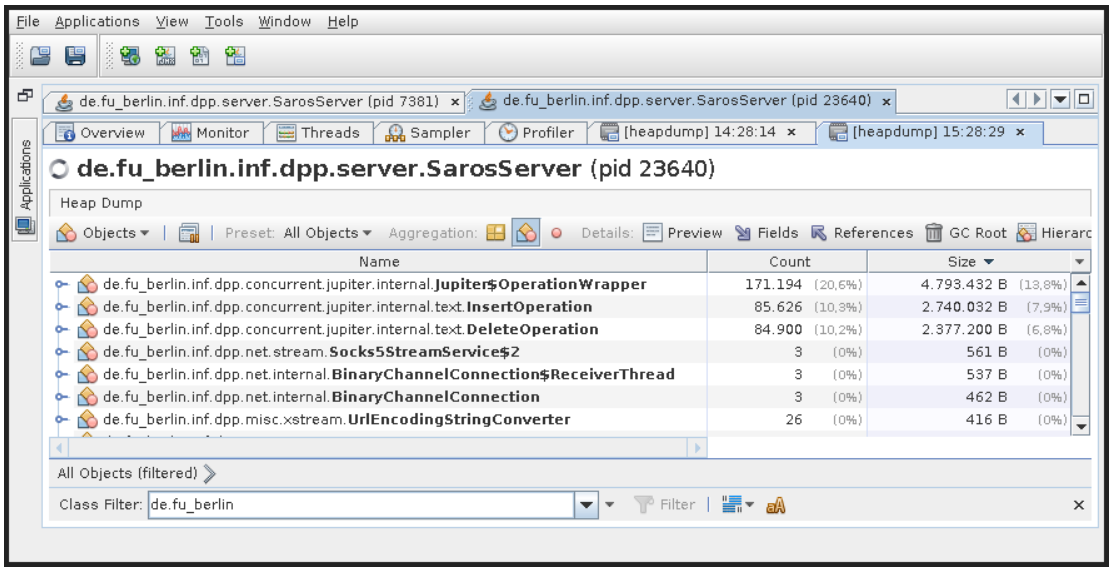


Abbildung 8: VisualVM HeapDump Analysis. Zeigt die meist allokierten Objekte des Servers nach einer Stunde Laufzeit.

gesendet werden. Herr Washington sprach in seiner Arbeit von 10 Sekunden, ich halte eine Nachricht pro Minute immer noch für absolut ausreichend.

Also Gründen der Komplexität des Nachverfolgens jeder letzten Bearbeitung aller offenen Editoren, sowie von Zeitmangel, entschloss ich mich eine Lösung in Form des zweiten Entwurfes (Heartbeat) zu implementieren.

3.2.6.2 Implementierung

Die Lösung startet einen Thread im `JupiterClient` welcher periodisch eine `TimestampOperation` für jede Jupiter-Instanz verschickt. Timestamp-Operationen waren wohl einmal für einen ähnlichen Fall vorgesehen, sind bislang aber in der Codebasis - obwohl vorhanden - nicht zum Einsatz gekommen. Mit einigen leichten Anpassungen erfüllten sie den Nutzen ohne weitere Probleme.

Ansonsten ist die Implementierung nicht sehr interessant, es handelt sich im wesentlich um zwei Zeilen zum Erstellen und Versenden der Aktivität, sowie kleinere Anpassungen, um den Zugriff aus einem weiteren Thread heraus abzusichern.

3.2.6.3 Ergebnis

Ein weiter Test zeigte, dass das Problem mit dem Patch behoben werden konnte (siehe Abb. 9). Es konnte außerdem kein weiterer Leak entdeckt werden.

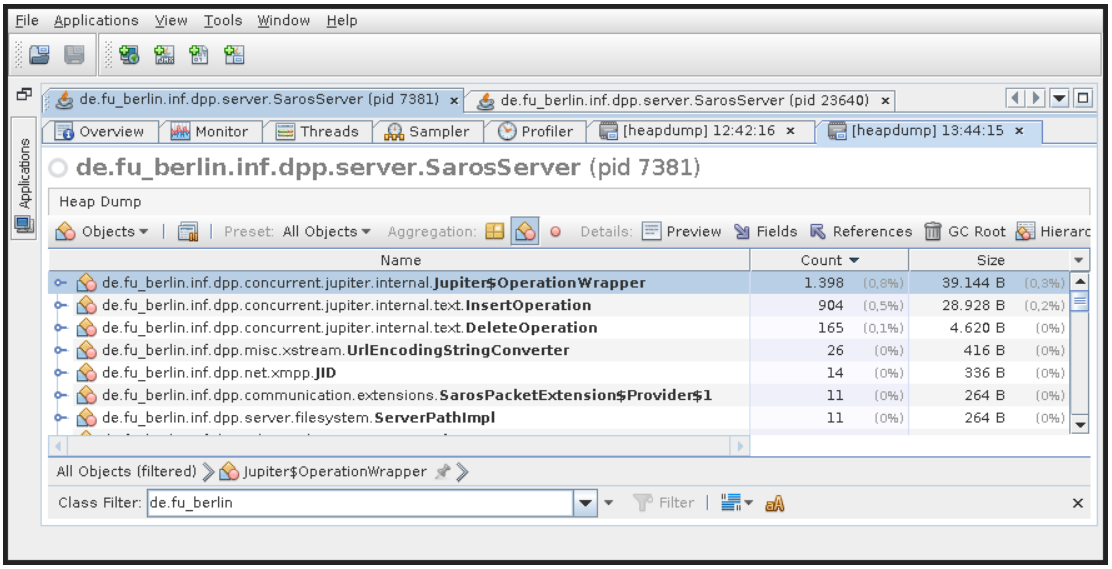


Abbildung 9: VisualVM HeapDump Analysis. Zeigt die meist allokierten Objekte des Servers nach einer Stunde Laufzeit mit dem Patch zur Behebung des Problems (vergl. Abbildung 8).

Der Patch befindet sich aktuell noch in Review.

3.3 Qualitätssicherung

Für das im Nachhinein komplizierteste Feature, dem Non-Host-Project-Sharing ist ein neuer STF Test entstanden, welcher versucht Inkonsistenzen während des Teilen des Projektes zu provozieren und so Fehler durch spätere Änderungen an den `ProjectNegotiations` schnell aufdecken sollte.

Für den Server existiert aktuell keine gute Test-Infrastruktur. Unit-Tests sind möglich, wurden jedoch oft nicht umgesetzt, da es sich zumeist um Minimal-Implementationen von Klassen aus dem Saros-Kern handelt, welche dort oft bereits sehr ausgiebig getestet wurden. Hier ist allerdings noch Verbesserungspotential vorhanden.

3.4 Zusammenfassung

Bis auf einige Rückschläge bei der Umsetzung der Benutzeroberfläche auf Klientseite, ist trotz einiger Irrwege bei der Umsetzung des Non-Host-Project-Sharing (siehe 3.2.5) durch den Instant-Session-Start (siehe 2.7) das Ziel erreicht worden. Alle notwendigen Features für den Einsatz des Servers wurden fertiggestellt. Das betrifft vor allem die Aufarbeitung aller vorher noch ausstehenden Patches meiner Vorarbeiter. Eine erste GUI-Integration für Saros/E zum Betritt der Server-Sitzung wartet aktuell noch auf Review. Das Non-Host-Project-Sharing wurde erfolgreich fertiggestellt und in die Codebasis übernommen. Die minimal notwendige Integration, um den Server verwenden zu können, ist somit im Wesentlichen umgesetzt worden. Das vor Denis Washington vor einiger Zeit gefundene Speicherleck konnte behoben werden und ansonsten zeigt der Server keine weiteren derartigen Fehler.

4 Evaluation

4.1 Einsatz des Servers

Leider ist der Server aktuell noch nicht im Einsatz. Bis zum Schluss sind abschließende Arbeiten vorgenommen wurden, um zu besagtem Ergebnis (siehe 3.4) im Rahmen der gegebenen Zeit zu kommen. Eine erste Alpha-Version für interessierte Tester ist geplant, aber zum aktuellen Zeitpunkt noch nicht veröffentlicht worden.

4.2 Erfahrungen mit dem Entwicklungsprozess

Großteils muss ich den Saros-Entwicklungsprozess loben. Das Review-System resultiert ein einem der bislang am besten strukturierten Java-Projekte, die ich bislang gesehen habe. Die übrig gebliebenen Server-Patches wurden zu Anfang als lästiger Mehraufwand angesehen, jedoch waren an allen Patches die ein oder anderen kleinen Fehler auszubügeln. Die Reviewstruktur ist also vermutlich nötig, um zu dem Bild zu kommen, das ich von Saros habe. Allerdings erhöht dieser Prozess natürlich die Chance, dass solcher liegen gebliebener Code veraltet, also inkompatibel mit aktuellen Codebasis wird. Meist werden die Patches nicht wieder angefasst, bis sich ein nächster Student mit dem Thema beschäftigt, was Monate oder gar Jahre dauern kann. Umfangreiche Änderungen an der restlichen Codebasis erschweren das Aufarbeiten, bereits übernommene Änderungen hätten sich vermutlich mit der Codebasis weiterentwickelt. Bei zum Zeitpunkt dieser Arbeit 141 offenen Fehlern auf GitHub ist es allerdings schwierig einen Überblick über offene Baustellen zu behalten. Bereits übernommener Code würde so vermutlich auch nicht noch einmal angefasst werden, weshalb ich gegeben der schwierigen Umstände den doch etwas bürokratisch erscheinenden Prozess dennoch nachvollziehen kann.

Dokumentation über den Zustand einzelner Teile von Saros (insbesondere unfertiger Teile) ist oft nur in Form der Abschlussarbeiten gegeben, was gerade in diesem Fall mit mehreren vorhergehenden Arbeiten durchaus für Verwirrung sorgen konnte. Code der bereits Teil von Saros ist, ist jedoch sehr oft dokumentiert.

Die Arbeit mit den anderen Entwicklern war in großen Teilen sehr angenehm und das Klima sehr unterstützend, was insbesondere für jemanden, der nicht mit der Codebasis vertraut ist, sehr hilfreich war. Ein paar wenige Ausnahmen gab es jedoch bei denen ich mir wünschen würde, dass diese in Zukunft anders gelöst werden. Ein Beispiel was mir im Kopf geblieben ist und deshalb hier Erwähnung finden muss, ist die PR des Invite-Befehls[8] der Server Konsole (siehe 3.2.3.5), welche gegen Ende eine recht aufgeheizte Diskussion über den Einsatz von Lambdas in Java startete, die sich in anderen Pull Requests sowie in Gesprächen außerhalb des Issue Trackers fortsetze. Auch wenn ich persönlich mit dem finalen Code zufrieden bin und Punkte in der Argumentation meiner Mitentwickler absolut korrekt waren, so hoffe ich doch, dass solche Differenzen in Zukunft auf eine etwas professionellere Art gelöst werden. Diese Kritik trifft auch sicherlich nicht auf alle Entwickler zu, jedoch ist das auch nicht der einzige Fall an dem ich mich unwohl mit dem Gesprächsklima fühlte. Ich kann mir durchaus vorstellen, dass diese Fälle auf andere Studenten eine abschreckendere Wirkung haben können.

[8] https://github.com/saros-project/saros/pull/282

An dieser Stelle muss ich allerdings noch einmal klarstellen, dass ich insgesamt sehr gerne am Saros-Projekt mitgewirkt habe.

4.3 Offene Probleme

4.3.1 GUI

Das größte ungelöste Problem betrifft die Integration des Servers in Saros/E (und Saros/I). Ein Nutzer-Interface zur Interaktion/Steuerung des Servers muss entwickelt werden, insbesondere im Hinblick auf eventuelle zukünftige Features. Als ersten Schritt sollte der Server nach Fertigstellung der HTML-GUI (siehe 3.2.4) als solcher im Interface erkenntlich gemacht werden und Funktionen wie der Follow Mode deaktiviert.

Die weitere Entwicklung ist dann allerdings abhängig vom angestrebten Feature-Set oder sollte zumindest beachten, dass geplante Funktionen sich in Zukunft einfach integrieren lassen. Hierzu müsste man sich allerdings zuerst klar werden, welche Funktionen für Saros/S als wünschenswert angesehen werden. Eine Auflistung möglicher nächster Schritte lässt sich im nächsten Abschnitt finden.

4.4 Ausblick

4.4.1 Server-Rechteverwaltung

Aktuell kann ein Saros-Server nur in einer vertrauenswürdigen Umgebung ausgeführt werden, da jede Saros-Instanz über eine `JoinSessionRequest` beitreten kann und jeder Klient Projekte mit dem Server teilen kann. Eine Art Rechteverwaltung für den Server wäre wünschenswert. Diese kann so einfach sein wie ein Admin-Nutzer, der weitere Nutzer hinzufügen oder entfernen kann, bis hin zu deutlich komplexerem Rechtemanagement. In diesem Kontext ist dann auch die Frage zu lösen, wie das im Klient-Interface funktionieren kann und wie die initiale Konfiguration des Servers erfolgt, je nach Komplexität der gewählten Lösung.

Aktuell erfolgt die Konfiguration des Servers über Kommandozeilen-Argumente. Für eine einfache Implementierung eines Admin-Nutzers wäre es ausreichend einen neuen Parameter hinzuzufügen. Dem Admin-Nutzer könnte vom Server das Recht mittels einer `JoinSessionRequest` beizutreten eingeräumt werden, sowie das Einladen weiterer Sitzungsteilnehmer. Das Einladen durch Teilnehmer, die nicht der Host sind, wird jedoch von Saros aktuell nicht unterstützt, auch wenn einige Überbleibsel in der Codebasis, wie übrig gebliebene aber deaktivierte STF-Tests, zeigen, dass dieses Feature einmal existiert hat. Eine relativ einfache Möglichkeit der Umsetzung wäre es die `JoinSessionRequestExtension` um eine JID des einzuladenden Nutzers zu erweitern. So könnte der Admin-Nutzer andere Teilnehmer durch den Server einladen lassen und der Server müsste nur den Absender verifizieren. Für die Lösung des Problems der Teilung von Projekten könnte entweder nur der Admin berechtigt sein Projekte zu teilen oder alle Teilnehmer. Beides wäre einfach umzusetzen und könnte auch über einen Kommandozeilenparameter konfiguriert werden.

Mächtigere aber kompliziertere Lösungen würden sehr schnell eine Möglichkeit der Konfiguration über die Benutzeroberfläche bedingen, welche erst nach der Fertigstellung einer ersten Version der HTML-GUI umgesetzt werden sollten. Auch würden hier vermutlich eine Menge neuer Pakete dem Netzwerkprotokoll hinzugefügt werden. Es stellt sich dann auch die Frage, wie man diese Protokoll-Erweiterungen im Code gut separieren kann. Sie wären eindeutig Server-spezifisch und sollten demnach nicht Teil von Saros/C sein, allerdings muss jedes GUI-Plugin, das mit dem Server kommunizieren will, diese kennen. Eine Abhängigkeit der GUI-Plugins vom Server-Code erscheint auf den ersten Blick aber nicht sehr sauber.

Eine Aufwandsabschätzung für komplexere Lösung ist daher sehr schwierig, da es viele offene Fragen in Bezug auf die Umsetzung gibt. Die oben erörtere Minimallösung sollte sich hingegen sehr einfach umsetzen lassen und würde eine erhebliche Verbesserung gegenüber dem aktuellen Status darstellen. Ich würde daher auch dieses Problem als relativ dringlich ansehen.

4.4.2 Aufräumen von Projekten auf dem Server

Der Server nimmt alle Projekte an und kann so auf Dauer sehr viele Dateien akquirieren. Aus Gründen der Ordnung und des Speicherplatzes muss es auch möglich sein Projekte wieder zu entfernen und auch tatsächlich die dazugehörigen Dateien wieder aus dem Workspace des Servers zu entfernen. Aktuell ist das nicht vorgesehen. Projekte können nur durch einen Neustart (siehe Wiederherstellen der Sitzung) aus der Sitzung entfernt werden und müssen zusätzlich manuell vom Dateisystem gelöscht werden.

Das Problem lässt sich in zwei Sub-Probleme aufspalten. Das eine ist, dass es im laufenden Betrieb möglich sein sollte Projekte ausgehend von Klienten zu löschen. Hierzu sollte zuerst das Problem der Rechteverwaltung gelöst werden. Danach müsste eine Möglichkeit eingeführt werden, eine Anfrage an den Server zu schicken, um ein Projekt aus der Sitzung zu löschen, was vermutlich relativ einfach umzusetzen wäre. Im Kontext der minimalen Rechteverwaltung (Admin-Nutzer) wäre diese Anfrage vermutlich nur vom Admin aus möglich und eventuell auch von dem Klient der ursprünglich das Projekt geteilt hat.

Sehr viel schwieriger ist die Frage, was mit geteilten Projekten nach einem Neustart oder Crash des Servers passieren soll. Es wäre denkbar, dass der Server als Besitzer seines Workspaces angesehen wird und somit alle existierenden Projekte beim Start wieder löscht. Dies ist aber unter Umständen bei großen Projekten insbesondere nach einem Crash nicht wünschenswert, weil diese wieder komplett mit dem Server geteilt werden müssten. Die Lösung dieses Subproblems ist somit verzahnt mit dem Problem des Wiederherstellens von Sitzungen (siehe nächster Abschnitt Wiederherstellen der Sitzung) und sollte vermutlich in diesem Kontext gelöst werden.

Das erste Problem lässt sich noch mit relativ wenig Aufwand lösen, es müsste eine Erweiterung des Netzwerk-Protokolls ähnlich der `JoinSessionRequest` vorgenommen werden. Das zweite könnte entweder verschoben werden in die Frage nach der Wiederherstellung einer Sitzung oder mittels einer einfachen Implementierung, die zum Start den Workspace löscht, temporär gelöst werden. Auch hier ist der Aufwand sehr überschaubar. Die Lösung dieses Problem ist relativ wichtig, da es zusammen mit einer Rechteverwaltung den unbeaufsichtigten Betrieb von Saros/S ermöglichen würde.

4.4.3 Wiederherstellen des Sitzung-Status nach Neustart

Der Server speichert aktuell keinen Zustand ab. Nach einem Neustart wird eine neue leere Sitzung gestartet ohne Teilnehmer oder die bereits übertragene Projekte. Es werden auch keine Einladungen an vorherige Teilnehmer versendet. Zumindest bei einem Absturz (nicht korrekter Beendigung des Servers) ist die Wiederherstellung des vorherigen Zustandes aber wünschenswert. Dafür müsste der aktuelle Zustand einer Sitzung vom Server regelmäßig auf das Dateisystem gespeichert werden. Hierzu zählen vor allem Teilnehmer und eine Liste der Projekte sowie der geteilten Ressourcen. Die Dateien der Projekte sind ohnehin serverseitig gespeichert.

Solch eine Wiederherstellungsfunktion im Falle von Abstürzen wäre auch außerhalb des Servers interessant, es könnte somit als Teil vom Core mit einem gemeinsamen Format implementiert werden. Gegeben einer Liste von Teilnehmern und Projekten könnte ein Host beim Start alle nötigen Einladungen automatisch versenden und die Projekte wieder der neu aufgebauten Verbindung hinzufügen. Inkonsistenten durch den Verbindungsabbruch würden so durch den letzten Stand des Hosts überschrieben werden.

Sowohl das Speichern des Zustandes als auch das Versenden der Einladungen ist im Prinzip relativ einfach umzusetzen. Da eine generische Umsetzung im Kern jedoch wünschenswert ist und man sich mit einigen Sonderfällen auseinandersetzen muss, wie der Frage, wie mit Teilnehmern umgegangen werden soll, die zum Zeitpunkt der Wiederherstellung nicht online sind, stufe ich dieses Problem doch als etwas schwieriger ein. Ich denke aber dieses Feature könnte durchaus zum Beispiel im Rahmen einer Bachelorarbeit realisiert werden. Ich halte die vorher angesprochenen Probleme allerdings für wichtiger.

4.4.4 Super-Server

Ein Saros-Server steht aktuell für eine Sitzung. Es wäre wünschenswert als übergeordneten Prozess einen Server zu haben, der Sitzungsserver nach Bedarf startet und diese verwaltet. Dieser können außerdem ein Teil der Lösung des vorherigen Punktes darstellen, indem er Crashes erkennen und hinter beendeten Servern aufräumen könnte. Die ursprüngliche Idee geht bereits auf die Arbeit von Denis Washington zurück [5, S. 50]. Auch dieser Super-Server müsste in irgendeiner Form in das Klient-Interface eingebunden werden, hierzu bedarf es noch einiges an weiterer Planung.

Der Aufwand ist somit sehr hoch einzuschätzen, es handelt sich um ein komplettes zusätzliches Projekt, welches tiefer Integration mit den vorhandenen Implementierungen bedarf. Allerdings ist dieses Projekt mehr eine Vision einer “fertigen” Version des Servers. Die bislang benannten Projekte stellen konkrete Probleme mit der aktuellen Version da und sind somit als deutlich dringender zu betrachten.

4.5 Saros/S als STF Klient

Neben aller möglichen Features gilt es außerdem nicht die aktuelle Test-Situation aus den Augen zu verlieren. Teile von Saros/S sind mit Unit-Tests versehen und die Codebasis ist noch sehr schmal, da viel von Saros/C übernommen werden kann, so dass die aktuelle Situation ziemlich gut ist. In Zukunft sollte man jedoch darüber nachdenken Saros/S in

die STF-Infrastruktur als möglichen Teilnehmer hinzuzufügen. Diese Integration ist nicht gerade einfach, da die STF-Tests sehr auf Saros/E ausgerichtet sind und zumeist eine GUI zur Interaktion voraussetzen. Es müssten vermutlich bestehende Tests angepasst werden oder separate Server-Tests im Zusammenspiel mit Klient-Implementationen angelegt werden.

Alternativ könnten STF-Interfaces, die im Kontext des Servers unsinnig sind, gemockt werden. Das könnte eine schnellere Anpassung der Tests ermöglichen. Es würde aber umso schwieriger werden die tatsächliche Abdeckung der Server-Codebasis abzuschätzen, da evaluiert werden müsste welche Interaktionen mit dem Server tatsächlich getestet werden, was das hinzufügen von Tests, die speziell auf den Server zugeschnitten sind, erschweren würde.

All das setzt natürlich das Ziel einer nahezu vollständigen Abdeckung des Codes voraus, was unter Umständen nicht realistisch ist oder ein sehr langfristiges Ziel darstellt. Dadurch wird die schnellere Alternative wieder interessant, da sie die Situation kurzfristig verbessern würde. Genauere Planung der weiteren Schritte sowie eine Aufwandsabschätzung durch jemanden mit mehr Kenntnissen des STF-Codes ist hier erforderlich. Jedoch würde ich das Vorhaben als schwierig betrachten. Auch sind die anderen Probleme mit Ausnahme des Super-Servers in meinen Augen dringender, da wie angemerkt die aktuelle Situation ausreichend ist.

4.6 Zusammenfassung

Insgesamt ist zu Sagen, dass die Arbeit am Server lange noch nicht abgeschlossen ist. Die Grundlagen für ein sehr eigenständiges Produkt mit meiner Meinung nach hohem Potential für neue Einsatzgebiete sind gelegt, doch ist der Server in seiner aktuellen Form zu einfach, um wirklich einsetzbar zu sein. Einige wenige Änderung betreffend einer Rechteverwaltung und automatisiertem Aufräumen von Ressourcen des Server würden einen realen Betrieb bereits ermöglichen. Für komplexe Szenarien sind eine deutlich erweiterte Integration in die Saros-IDE-Plugins sowie eine Verwaltung mehrerer Instanzen wünschenswert. Hier ist aber noch viel Vorarbeit zum Entwurf konkreter Umsetzungen dieser Features zu leisten.

5 Fazit

Das in der Einführung genannte Ziel den Server tauglich für den Produktiveinsatz zu machen ist im Wesentlichen erreicht worden. Der Stand wurde evaluiert und alle zum Betrieb notwendigen Patches meiner Vorarbeiter wurden in die Codebasis integriert einschließlich einer Implementierung des Non-Host-Project-Sharings. Einige komplett neue Funktionalitäten zur Unterstützung der Entwicklung des Servers sind entstanden, wie die Server Konsole. Alte Fehler wie das in vorheriger Arbeit gefundene Speicherleck in Bezug auf der Jupiter-Algorithmus sind gelöst worden.

Der Server kann nun in einer ersten Version für interessierte Nutzer veröffentlicht werden. Für den großflächigen Einsatz dieses sehr jungen Projektes ist jedoch noch einiges an Arbeit zu leisten, die im Ausblick detailliert beschrieben wurde. Vor allem ist hier eine Rechteverwaltung, sowie eine bessere Intergration mit den vorhanden Klienten zu nennen. Trotzdem ist der Server mit einer ersten lauffähigen und vollständig in die Codebasis integrierten Version dem Ziel des Produktiveinsatzes einen großen Schritt näher gekommen.

6 Anhang

6.1 List der eingereichten Codeänderungen

6.1.1 Integrierte Änderungen

- **Gerrit**[9]:
  - **3519** [CORE][REFACTOR] Change RecoveryFileActivity in TargetedFileActivity
  - **3520** [FEATURE][CORE] Introduce project transferType
  - **3537** [INTERNAL][CORE] Remove color negotiation hack
  - **3549** [CORE][FEATURE] Negotiate user properties
  - **3561** [FEATURE][CORE] Abstract ProjectNegotiation
- **GitHub**[10]:
  - **#142** [FEATURE][SERVER] Add a terminal-based progress indicator
  - **#143** [FEATURE][SERVER] Add lightweight console
  - **#144** [FEATURE][SERVER] Add NegotiationHandler
  - **#146** [FEATURE][SERVER] Add EditorManager
  - **#147** [FEATURE][SERVER] Add ActivityExecutors
  - **#281** [FEATURE][SERVER] Start session
  - **#282** [FEATURE][SERVER] Add InviteCommand
  - **#283** [FEATURE][SERVER] Add ShareCommand
  - **#290** [FEATURE][SERVER] Handle JoinSession requests
  - **#306** [S][FIX] Stop context before failing on missing XMPP credentials
  - **#354** [C][BUGFIX] Fix missing export for apache.commons.lang3
  - **#355** [FEATURE] Non-host project sharing

6.1.2 Offene Änderungen

- **GitHub**[11]:
  - **#362** [E/S][FEATURE] Allow Saros/E to join server session
  - **#396** [S][FEATURE] Support incoming ProjectNegotiations
  - **#397** [C][BUGFIX] Add Jupiter-Heartbeats via Timestamp-Operations

6.1.3 Abgelehte Änderungen

- **GitHub**[12]:
  - **#141** [FIX][SERVER] Do not assume projects are at the root of the workspace
  - **#284** [FEATURE][S/C] Expose isServer property
  - **#353** [C][FEATURE] Introduce PropertyHooks

[9] https://saros-build.imp.fu-berlin.de/gerrit/
[10] https://github.com/saros-project/saros
[11] https://github.com/saros-project/saros
[12] https://github.com/saros-project/saros

6.1.4 Entwürfe

- **Gerrit**[13]:
  - **3498** [CORE][FEATURE] Minimal Instant Session Start Implementation

6.2 Bibliographie

[1] L. Williams und R. R. Kessler, *Pair Programming Illuminated*. Addison-Wesley Professional, 2002.

[2] J. Schenk, L. Prechelt, und S. Salinger, „Distributed-pair Programming Can Work Well and is Not Just Distributed Pair-programming“, in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, S. 74–83.

[3] N. H. Bussas, „Entwicklung eines Server-Prototypen für Saros“, 2014.

[4] A. Lasarzik, „Refaktorisierung des Eclipse-Plugins Saros für die Portierung auf andere IDEs“, 2015.

[5] D. Washington, „Entwicklung und Evaluation eines unabhängigen Sitzungsservers für das Saros-Projekt“, 2016.

[6] M. Krummrei, „Überführung des Saros-Server in die Codebasis“, 2017.

[7] D. A. Nichols, P. Curtis, M. Dixon, und J. Lamping, „High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System“, in *ACM Symposium on User Interface Software and Technology*, 1995.

[8] S. Moll, „Verbesserung des Session Start im Saros-Projekt“, 2018.

[9] D. Theus, „Entwicklung einer Infrastruktur für wechselbare Projektübertragungsformen und Weiterentwicklung Bestehender in Saros“, 2015.

[10] P. Fehling, „Entwurf, Implementation und Evaluation des Instant Session Starts für das Open Source IDE-Plugin "Saros"“, 2016.

[13] https://saros-build.imp.fu-berlin.de/gerrit/