Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Software Engineering

Bestandsaufnahme und Arbeit an einer Alpha-Version des Saros-Plugins für die IntelliJ-Plattform

Tobias Bouschen
Matrikelnummer: 4676966
tobias.bouschen@fu-berlin.de

Betreuer: Franz Zieris
Eingereicht bei: Prof. Dr. Lutz Prechelt
Zweitgutachten bei: Prof. Dr. Margarita Esponda-Argüero

Berlin, 26. Mai 2017

Zusammenfassung

Saros ist eine Erweiterung für Softwareentwicklungsumgebungen, welche verteilte Paar- und Gruppenprogrammierung unterstützt. Sie wurde ursprünglich für Eclipse entwickelt, seit einiger Zeit ist aber auch eine IntelliJ-Version in Arbeit. Dieser Version fehlen jedoch noch einige grundlegende Funktionalitäten und Features des Eclipse-Gegenstücks.

Diese Arbeit sollte nun eine funktionsfähige Alpha-Version des Saros-Plugin für IntelliJ erstellen. Dazu sollten Versagen in der bereits bestehenden Implementierung erfasst und behoben, sowie noch für die Alpha-Version nötige Funktionalitäten implementiert werden.

Bei der Bearbeitung der bestehenden Versagen wurden jedoch tiefgreifende Defekte in der Implementierung gefunden, die eine Reimplementierung der betroffenen Komponenten erforderten. Da dies mit einem großem Zeitaufwand verbunden war, wurde die Erstellung der Alpha-Version (und subsequente Veröffentlichung) nicht vollendet, jedoch wurde eine wesentliche Basis für das weitere Vorgehen geschaffen.

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.

26. Mai 2017

Tobias Bouschen

Inhaltsverzeichnis

1 Einführung

In der Welt der Softwareentwicklung ist die Paarprogrammierung (PP[1]) eine einfache Methode, um bei einem geringem Mehraufwand die Qualität der entwickelten Software, sowie die technischen Fähigkeiten der Teilnehmer zu erhöhen [1].

Paarprogrammierung bedeutet hierbei, dass zwei Personen an einem Rechner zusammen an ein und derselben Aufgabe arbeiten. Dabei nimmt eine Person die Rolle des *Drivers* ein, welche für die eigentliche Umsetzung der Aufgabe zuständig ist. Die andere Person übernimmt die Rolle des *Observers/Navigators*, welche die erzeugten Artefakte begutachtet und bei der Bearbeitung den Gesamtzusammenhang der Aufgabe betrachtet. Die Rollen werden während des Vorgehens regelmäßig gewechselt [2].

Da in der heutigen Arbeitswelt eine örtliche Nähe der verschiedenen Mitarbeiter nicht immer gegeben ist, wird versucht, dieses Konzept auf die verteilte Paarprogrammierung (DPP[2]) zu erweitern [3][4]. Dabei wird die direkte Kommunikation durch verschiedene technische Hilfsmittel, wie geteilte Dokumente und den Austausch von Text- oder Sprachnachrichten, ersetzt. Bei der verteilten Paarprogrammierung können dieselben positiven Effekte wie bei der lokalen Paarprogrammierung erzeugt werden [5][6][7].

Um ein Werkzeug zur Durchführung von verteilter Paarprogrammierung zu schaffen, wurde an der Freien Universität das Saros-Projekt[3] gegründet, in welchem eine gleichnamige Erweiterung (Plugin) zur verteilten Paarprogrammierung für verschiedene Entwicklungsumgebungen (IDEs[4]) entwickelt wird.

Die erste Version des Saros-Plugins wurde 2006 in einer Diplomarbeit für die Eclipse-IDE erstellt [8][9] (Saros/E). Aufgrund von Nachfragen nach anderen IDEs wurde später zusätzlich die Arbeit an Saros-Versionen für Netbeans (Saros/N) und IntelliJ IDEA (Saros/I) begonnen.

Für die Entwicklung des IntelliJ-Plugins wurde der bereits bestehende Code angepasst. Die dabei entstandene Implementierung wies jedoch nicht die gewünschte Codequalität auf. Sie enthielt viele Codeduplikate und Defekte, war nicht ausreichend dokumentiert und hatte nur eine geringe Testabdeckung. An diesen Problemen wurden bereits durch verschiedene Bachelorarbeiten (siehe Abschnitt 2.3) und Mitarbeiter des Saros-Teams gearbeitet. Saros/I befand sich zu Beginn dieser Arbeit jedoch immer noch in einem nicht-funktionellen Zustand.

[1] **P**air **P**rogramming
[2] **D**istributed **P**air **P**rogramming
[3] http://www.saros-project.org/
[4] **I**ntegrated **D**evelopment **E**nvironments

1.1 Ziel dieser Arbeit

Das Ziel dieser Bachelorarbeit bestand darin, eine funktionierende Alpha-Version von Saros/I zu erstellen und zu veröffentlichen. Hierzu sollte eine lauffähige Version von Saros/I hergestellt werden, die eine grundlegende Funktionalität anbietet. Diese Version sollte dann über die Saros-Webseite der Öffentlichkeit zugänglich gemacht werden.

Die Anforderungen an diese Alpha-Version sollten als Teil dieser Arbeit bestimmt werden. Eine genauere Beschreibung der im Verlauf dieser Arbeit aufgestellten Ziele kann in Abschnitt 5 auf Seite 14 eingesehen werden.

1.2 Struktur dieser Arbeit

Um einen kurzen Überblick über diese Ausarbeitung zu schaffen, wird im Folgenden die Struktur erläutert:

In Abschnitt 2 werden zunächst einige Grundlagen über das Saros-Plugin, sowie das Saros-Projekt erläutert, mit dem Verweis auf verwandte Arbeiten.

In Abschnitt 3 werden mögliche Vorgehensweisen für diese Arbeit vorgestellt, sowie die Auswahl der genutzten Vorgehensweise begründet und genauer beschrieben.

In Abschnitt 4 wird die Einarbeitung in das Saros-Projekt bzw. Saros/I genauer beschrieben.

In Abschnitt 5 werden die Meilensteine erläutert, welche bei der Erstellung einer Saros/I-Alpha-Version erreicht werden sollten.

In Abschnitt 6 wird die minimal nötige Funktionalität dargestellt, sowie eine Abgrenzung zu in dieser Arbeit nicht betrachteten Funktionalitäten vorgenommen.

In Abschnitt 7 werden die in Saros/I erkannten Versagen beschrieben und den jeweiligen Funktionalitäten zugeordnet.

In Abschnitt 8 wird das iterative Vorgehen zur Behebung der erfassten Defekte beschrieben. Dabei wird genauer auf die Analyse der Versagen, die unterliegenden Defekte, sowie deren Behebung eingegangen.

In Abschnitt 9 werden die Probleme mit der initialen Version der Ressourcenverwaltung sowie die Entscheidungen bei der Erstellung der Reimplementierung beschrieben.

In Abschnitt 10 werden die Ergebnisse dieser Arbeit zusammengefasst.

In Abschnitt 11 wird ein Ausblick auf die weitere Arbeit an Saros/I gegeben.

In Abschnitt 12 werden einige Probleme thematisiert, welche während dieser Arbeit erkannt wurden, deren Lösung jedoch nicht Teil dieser Arbeit sind.

2 Grundlagen

2.1 Saros-Plugin

Saros ist, wie in der Einleitung erwähnt, ein Plugin zur verteilten Paarprogrammierung. Es erlaubt die synchrone Arbeit auf geteilten Dateien, wobei die Eingaben anderer Teilnehmer zur besseren Erkennbarkeit jeweils unterschiedlich farblich markiert werden. Zusätzlich gibt es den sogenannten „follower mode“, über welchen einem Teilnehmer der Sitzung gefolgt werden kann. Dies erlaubt es dem folgenden Teilnehmer, permanent die Dateien bzw. Ausschnitte zu betrachten, an welchen der gefolgte Teilnehmer gerade arbeitet. Durch diesen Modus kann das Verhalten des *Observers/Navigators* bei der Paarprogrammierung repliziert werden. Der folgende Teilnehmer hat dabei jederzeit die Möglichkeit auch selbst Änderungen vorzunehmen, was einen fließenden Wechsel der Rollen, sowie ein paralleles Arbeiten ermöglicht.

Beim Aufbau einer Saros-Sitzung wird zudem der Zustand der geteilten Dateien bei allen teilnehmenden Saros-Instanzen (Clients) an den der initialisierenden Saros-Instanz (Host) angepasst. Hierzu werden nicht vorhandene oder veränderte Dateien durch die Version des Hosts ersetzt und beim Host nicht-vorhandene Dateien verworfen. Dies erlaubt ein einfaches Übertragen der zu bearbeitenden Dateien und garantiert einen konsistenten Zustand nach der Sitzungsinitialisierung.

Saros wurde ursprünglich nur als Eclipse-Plugin konzipiert. Um eine Ausweitung auf andere IDEs zu vereinfachen, wurde die bestehende Saros-Implementierung in zwei Komponenten gespalten:

- Saros/E, der Eclipse-abhängige Teil der Implementierung
- Saros-Kern (Saros Core)

Der Kern enthält nur die generelle Funktionslogik und bietet die Grundlage für alle Saros-Versionen. Er ist alleinstehend jedoch nicht funktionsfähig und hat daher nicht den Charakter einer Bibliothek, sondern agiert eher als ein Framework für die Implementierung des eigentlichen IDE-Plugins. Hierzu stellt er eine Reihe von Schnittstellen[5] zur Verfügung, welche jeweils passend zu der unterliegenden IDE implementiert werden müssen. Die jeweilige Saros/X-Komponente[6] stellt damit die Anbindung an die unterliegende IDE und deren Funktionalität dar.

Um nach dieser Aufspaltung von der jeweiligen Implementierung der IDE-Oberfläche zu abstrahieren, wurde eine HTML-GUI konzipiert, die über ein

[5] Wenn in dieser Arbeit fortlaufend von Saros-Schnittstellen gesprochen wird, sind damit die zu implementierenden Schnittstellen des Saros-Kerns gemeint.

[6] „X“ ist hierbei ein Platzhalter und verweist nicht auf eine existierende Saros-Version

Browserfenster in die IDE eingebunden und über JavaScript an das jeweilige Saros-Plugin angebunden wird. Weitere Details zu der HTML-GUI können im Saros-Wiki [10] eingesehen werden.

Bei der Umsetzung von Saros wurden viele verschiedene Technologien und Komponenten verwendet, von denen nachfolgend auf einige für ein besseres Verständnis dieser Arbeit eingegangen wird.

2.1.1 Consistency Watchdog

Um während einer Saros-Sitzung durchgehend einen konsistenten Zustand aller geteilten Dateien bei allen teilnehmenden Saros-Instanzen zu garantieren, wurde in Saros der sogenannte *Consistency Watchdog* eingeführt. Dieser prüft in regelmäßigen Intervallen, ob die Dateien bei allen teilnehmenden Saros-Instanzen den selben Inhalt haben.

Hierzu wird als angestrebter Grundzustand der Inhalt der geteilten Dateien bei der Saros-Instanz, welche die Sitzung erstellt hat, verwendet. Für die Dateien des Hosts werden Hash-Prüfsummen erstellt, welche an alle teilnehmenden Saros-Instanzen gesendet werden. Die Clients vergleichen diese Prüfsummen mit ihren lokalen Dateien. Sollte dabei eine Inkonsistenz erkannt werden, wird sie dem Nutzer als eine *Desynchronisation* gemeldet. Dieser hat nun die Möglichkeit, eine *Resynchronisation* vorzunehmen, bei welcher der lokale Inhalt der inkonsistenten Datei durch den Inhalts der Datei des Hosts ersetzt wird.

2.1.2 XMPP (Smack)

Der Austausch von Daten zwischen verschiedenen Saros-Instanzen wird über das *Extensible Messaging and Presence Protocol*[7] (XMPP) durchgeführt [11]. XMPP ist ein von der *Internet Engineering Task Force*[8] (IETF) standardisiertes Protokoll zum Austausch von XML-basierten Nachrichten in Echtzeit [12].

In Saros wird als Umsetzung dieses Protokolls die Java-Bibliothek Smack[9] verwendet [11]. Der Nachrichtenaustausch findet dabei indirekt über einen XMPP-Server statt, auf welchem sich die Teilnehmer einer Sitzung zuvor mit einem Account anmelden müssen. Dieser Account wird in Saros als Identifizierungsmerkmal eines Nutzers verwendet. Zusätzlich hält jeder Account eine Liste an bekannten anderen Accounts, mit welchen über Saros eine Sitzung aufgebaut werden kann.

[7] https://xmpp.org/about/
[8] http://www.ietf.org/
[9] https://www.igniterealtime.org/projects/smack/

2.1.3 Dependency Injection (PicoContainer)

Dependency Injection ist ein Entwurfsmuster aus dem Bereich der *Inversion of Control* [13]. Es erlaubt die Abstraktion von den tatsächlichen Implementierungen verwendeter Schnittstellen. Hierbei werden während der Umsetzung nur Schnittstellen angegeben, welche das benötigte Verhalten festlegen. Die zu verwendende Implementierung dieser Schnittstellen wird dann zur Laufzeit von einer dafür zuständigen Komponente bereitgestellt. Dieses Entwurfsmuster erlaubt es, Abhängigkeiten zwischen dem Saros-Kern und spezifischen Saros-Versionen aufzulösen, da die zu der Version passenden Implementierungen zur Laufzeit bestimmt werden können.

Für die Nutzung von *Dependency Injection* wird die Java-Bibliothek PicoContainer [14] verwendet. Über diese wird ein Container erstellt, in welchem Singleton-Objekte[10] der zu verwaltenden Klassen instantiiert werden. In Saros existieren dabei zwei dieser Container, welche an unterschiedliche Lebenszyklen gebunden sind: Der *Plugin Context*, welcher an das Plugin gebunden ist, und der *Session Context*, welcher für jede erstellte Sitzung erzeugt und bei Beendigung der Sitzung auch wieder verworfen wird.

Nähere Details zu der strukturellen Einbindung des PicoContainers in die Saros-Struktur können im Saros-Wiki im Abschnitt „Internal Dependency Management“ der technischen Dokumentation [15] eingesehen werden.

Als Form der *Dependency Injection* wird hauptsächlich *Constructor Injection* verwendet. Dies bedeutet, dass im Konstruktor der verwalteten Klassen nur Schnittstellen angenommen werden, welche die benötigte Funktionalität angeben. Die Logik des PicoContainers wählt bei der Instantiierung der Klasse passende Implementierungen der Schnittstellen, die dem Konstruktor übergeben werden.

Nähere Details zu der Verwendung von *Dependency Injection* in Saros können in den Coderegeln im Saros-Wiki [16] eingesehen werden.

2.1.4 Logging (Log4J)

In Saros wird Logging verwendet, um bestimmte, bei der Nutzung von Saros auftretende Ereignisse (wie z. B. Versagen) nach Abschluss der Nutzung besser analysieren zu können. Dabei werden protokollierte Informationen sowohl in der Konsole ausgegeben, als auch in Log-Dateien geschrieben.

Dafür wird die Java-Bibliothek Log4J verwendet [17]. Diese erlaubt das Einteilen der protokollierten Informationen in unterschiedliche Stufen:

[10] Singleton ist ein grundlegendes Entwurfsmuster in der Softwaretechnik. Es soll garantieren, dass zur Laufzeit von der beschriebenen Klasse maximal genau eine Objekt existieren kann.

ERROR sollte verwendet werden, wenn es ein unerwartetes internes Verhalten von Saros gab, welches die normale Nutzung von Saros einschränkt.

WARNING sollte verwendet werden, wenn es ein unerwartetes internes Verhalten von Saros gab, dieses jedoch keinen Einfluss auf die Nutzung von Saros haben sollte.

INFO sollte verwendet werden, wenn für den Nutzer interessante Informationen ausgegeben werden sollen.

DEBUG sollte verwendet werden, wenn für Entwickler interessante Informationen, die den genaueren Ablauf der Anwendung beschreiben, ausgegeben werden sollen. Hierbei sollte die Menge der ausgegebenen Informationen nicht die Leistung der Anwendung beeinträchtigen.

TRACE sollte verwendet werden, wenn für Entwickler interessante, sehr detaillierte Informationen über den Ablauf der Anwendung ausgegeben werden sollen. Bei der Ausgabe einer großen Menge an Informationen kann es zu einer Beeinträchtigung der Leistung der Anwendung kommen.

2.2 Saros-Projekt

Saros ist ein Open-Source-Projekt. Der gesamte Sourcecode liegt im GitHub des Projekts[11] offen vor. Die Hauptaufgabe des Saros-Projekts besteht in der Instandhaltung und Weiterentwicklung des Saros-Plugins. Hieran können sich bei Interesse auch externe Entwickler beteiligen. Momentan beschränkt es sich jedoch hauptsächlich auf Studenten der Freien Universität Berlin, welche als Teil von Abschlussarbeiten an Saros arbeiten.

Im Saros-Projekt werden verschiedene Infrastrukturen verwendet, von denen nachfolgend einige für ein besseres Verständnis dieser Arbeit erklärt werden.

2.2.1 Peer-Code-Review (Gerrit)

Im Saros-Projekt muss jede Änderung zuerst eine Durchsicht (Code-Review) durchlaufen, mit dem Ziel, die Anzahl der enthaltenen Defekte so gering wie möglich und die Codequalität so hoch wie möglich zu halten. Des Weiteren bietet dies eine Kontrollinstanz, um sicherzustellen, dass die im Patch gemachte Änderung auch wirklich so erwünscht ist. Bei einem solchen Code-Review wird die Änderung von anderen Mitgliedern des Saros-Teams begutachtet und, falls nötig, an die Anmerkungen dieser angepasst.

Details zu diesem Prozess können im Saros-Wiki im Abschnitt „Review“ [18] eingesehen werden.

[11] https://github.com/saros-project/saros

Um diese Peer-Code-Reviews zu erleichtern wird Gerrit[12] verwendet. Gerrit ist eine Erweiterung eines normalen Git-Repositories, auf welchem Commits als sogenannte Changes hochgeladen werden. Änderungen an Changes können dann in Form von sogenannten Patchsets durchgeführt werden, die dem ursprünglichen Change hinzugefügt werden.

Code-Reviews können über eine Browser-Schnittstelle von Gerrit[13] durchgeführt werden. Ist ein Code-Review zufriedenstellend abgeschlossen, kann der Change in den Master-Branch des Git-Repositories integriert werden.

2.2.2 Feature-Requests und Bugtracker (SourceForge)

Um Nutzern die Möglichkeit einer Rückmeldung zu dem aktuellen Stand des Saros-Plugins zu geben, wird sowohl eine Liste von Anfragen auf neue Funktionalitäten (Feature-Requests) als auch eine Liste an bekannten Versagen (Bugtracker) geführt. Diese Listen werden durch das Saros-Team verwaltet. Es können jedoch auch von Nutzern Einträge angelegt werden.

Die Listen werden über den Eintrag des Saros-Projekts auf der Webseite SourceForge[14] bereitgestellt.

2.2.3 Erleichterung der Einarbeitung (jTourBus)

Um die Einarbeitung in die Codebasis des Saros-Plugins zu erleichtern wird das Eclipse-Plugin jTourBus[15] verwendet. Es erlaubt die Betrachtung des Saros-Quellcodes in sogenannten Touren, in denen der Nutzer durch speziell markierte Teile der Dokumentation des Codes geführt wird.

2.3 Verwandte Arbeiten

Aufbauend auf der initialen Diplomarbeit [8], wurde Saros/E über viele Abschlussarbeiten hinweg erweitert. Diese Erweiterungen werden hier nicht betrachtet, können bei Interesse jedoch in der Liste der absolvierten Abschlussarbeiten[16] nachgeschlagen werden.

Als Grundlage dieser Arbeit kann die 2015 geschriebene Bachelorarbeit „Refaktorisierung des Eclipse-Plugins Saros für die Portierung auf andere IDEs“ [19] von A. Lasarzik betrachtet werden. Als Teil dieser Arbeit wurde die Trennung des Saros-Plugins in Saros/E und den Saros-Kern vorgenommen.

Die Arbeit an der Aufspaltung wurde als Teil der 2016 geschriebenen Masterarbeit „Entwicklung und Evaluation eines unabhängigen Sitzungsservers

[12] https://www.gerritcodereview.com/
[13] http://saros-build.imp.fu-berlin.de/gerrit/
[14] https://sourceforge.net/p/dpp/_list/tickets
[15] http://www.saros-project.org/jtourbus
[16] http://www.inf.fu-berlin.de/w/SE/ThesesHome#Abgeschlossene_Arbeiten

für das Saros-Projekt“ [20] von D. Washington fortgesetzt. Die eigentliche Aufgabe dieser Arbeit war die weiterführende Arbeit an einem eigenständigen Saros-Server, welcher die Rolle des Hosts in einer Saros-Sitzung übernimmt. Im Rahmen dieser Arbeit wurden jedoch auch die Funktionalität des Saros-Kerns ausgebaut und Codeduplikate entfernt.

Hiernach wurde die Arbeit an der Aufspaltung als Teil der 2016 geschriebenen Bachelorarbeit „Verbesserung und Erweiterung der Core-Bestandteile von Saros“ [21] von D. Sungaila fortgesetzt. Diese Arbeit beschäftigte sich erneut mit dem Ausbau der Funktionalität des Saros-Kerns, sowie dem Entfernen von Codeduplikaten.

Des Weiteren gab es einige Abschlussarbeiten, die sich mit der Saros-HTML-GUI befasst haben:

2015 „Evaluating the Use of a Web Browser to Unify GUI Development for IDE Plug-ins“, C. Cikryt [22]

2015 „Entwicklung einer IDE-unabhängigen Benutzeroberfläche für Saros“, M. Bohnstedt [23]

2015 „User-Centered Development of a JavaScript and HTML-based GUI for Saros“, B. Sieker [24]

2016 „Einstiegserleichterung für die Weiterentwicklung und Erweiterung der JavaScript- und HTML-GUI von Saros“, N. Weber [25]

3 Vorgehen

Zu Beginn habe ich die Arbeitsgrundlagen für das spätere Vorgehen geschaffen. Hierzu musste ich eine Grundlage an Wissen über die Arbeitsweise und genutzte Infrastruktur schaffen.

3.1 Einarbeitung

Für die Einarbeitung gibt es zwei mögliche Vorgehensweisen.

- Es kann sich theoretisch eingearbeitet werden.

  Dies beschränkt sich hauptsächlich auf das Ansammeln von Wissen über die zu bearbeitende Materie, sowie ihre angrenzenden Bereiche.

- Es kann sich praktisch eingearbeitet werden.

  Hierbei wird zuerst eine theoretische Grundlage geschaffen. Der Hauptteil der Einarbeitung geschieht jedoch durch das praktische Arbeiten in dem jeweiligen Bereich. Dabei ist es wichtig, dass kleinere bzw. einfachere Aufgaben gewählt werden, da sie keine weitere Einarbeitung erfordern und nicht zu viel Zeit in Anspruch nehmen sollten.

Beide Ansätze wurden in dieser Arbeit verwendet, jeweils für unterschiedliche Bereiche von Saros.

Für eine generelle Einarbeitung in die Kernkomponenten von Saros habe ich einen rein theoretischen Ansatz gewählt. Hierbei habe ich mich hauptsächlich auf das Studieren des Saros-Wiki und Fachliteratur zu benutzten Komponenten, sowie eine grobe Durchsicht des Quellcodes beschränkt.

Diesen Ansatz habe ich über andere Alternativen, wie die komplette Erforschung des Quellcodes nach dem Tiefen- oder Breitensuchverfahrens, gewählt, da die Benutzung der vorhandenen Infrastruktur (jTourBus) den Zeitaufwand erheblich verringerte. Zusätzlich ist nur ein grundlegendes Wissen über die Implementierung des Saros-Kerns sowie Saros/E für diese Arbeit notwendig, da sich der Hauptaufwand auf den Saros/I-Teil beschränken sollte.

Für eine genauere Einarbeitung in Saros/I habe ich mich für einen praxisorientierten Ansatz entschieden. Dabei habe ich zur Einarbeitung in die Codebasis und zum Sammeln von Erfahrung mit der Arbeitsweise im Saros-Projekt einen angefangenen Patch vollendet.

Da hier ein praktischer Zugang für die zukünftige Arbeit von Vorteil war, habe ich diesen Ansatz anstatt einer rein theoretischen Einarbeitung, wie sie für den Saros-Kern und Saros/E vorgenommen wurde, gewählt.

3.2 Anforderungsanalyse

Anschließend habe ich eine Anforderungsanalyse für eine Saros/I-Alpha-Version durchgeführt, um die Ziele dieser Arbeit genauer festlegen zu können. Dies ermöglichte auch eine bessere Planung des Vorgehens.

Hierbei hätte ich eine komplett neue Anforderungsanalyse für eine allgemeine Saros-Alpha-Version anfertigen können. Da es mit Saros/E jedoch schon eine funktionierende Version gab, von welcher eine Erfüllung der Anforderungen an eine Saros-Version angenommen werden kann, wurde diese als Grundlage für die Anforderungsanalyse gewählt. Zu erfassen war damit, welcher minimale Satz an Funktionalitäten aus Saros/E vorhanden sein muss, um eine Saros-Version grundlegend benutzbar zu machen. Hierbei wurden sowohl funktionale als auch nicht-funktionale Anforderungen betrachtet. Da es sich um eine Alpha-Version handeln sollte, wurden die funktionalen Anforderungen jedoch in den Vordergrund gestellt.

Die Ergebnisse dieser Analyse sind in Abschnitt 5 auf Seite 14 und Abschnitt 6 auf Seite 16 aufgeführt.

3.3 Analyse des Ist-Zustandes

Aufbauend auf dem minimalen Funktionsumfang habe ich den aktuellen Zustand der für die Saros/I-Alpha-Version nötigen Funktionalitäten erfasst. Dazu habe ich manuell die einzelnen Funktionalitäten geprüft und das auftretende Verhalten notiert. Zusätzlich habe ich die jeweils anfallenden Log-Dateien betrachtet, um nicht offensichtliches Fehlverhalten zu erkennen und mehr über die Natur der auftretenden Versagen zu erfahren. Für auftretende Versagen habe ich jeweils einen Bugtracker-Eintrag erstellt.

Die erfassten Versagen sind in Abschnitt 7 auf Seite 18 aufgeführt.

3.4 Bearbeitung der erkannten Versagen

Für die Bearbeitung der erkannten Versagen musste ich mich für eine Vorgehensweise entscheiden. Zur Auswahl wurden mögliche Vorgehensweisen in der Softwareentwicklung herangezogen, welche in drei grobe Kategorien eingeteilt werden können [26]. Wirkliche Umsetzungen dieser Vorgehensweisen befinden sich dabei meistens auf einem Spektrum zwischen den hier angegebenen Vorgehen.

Statische Vorgehen sind plan- bzw. dokumentgetrieben und versuchen den Umfang der entwickelten Software sowie das dafür nötige Vorgehen möglichst präzise, strikt und vor allem vollständig im Voraus zu planen und festzuhalten. Dies erlaubt eine klare Arbeitsgrundlage

und eine gute Übersicht über die Durchführung sowie die zu erzeugenden Artefakte. Statische Vorgehen sind jedoch in der Planung sehr aufwändig und Planänderungen sind schwierig umzusetzen.

Iterative Vorgehen versuchen so weit wie aktuell sinnvoll zu planen. Die Umsetzung des Plans wird dabei in Iterationen durchgeführt, welche jeweils eine festgelegte Aufgabe haben und nach welchen jeweils ein dem Kunden präsentierbares Produkt vorliegt. Während der Durchführung der Iterationen kann der aufgestellte Plan angepasst oder erweitert werden. Dies erlaubt immer noch ein strukturiertes Vorgehen und einen guten Ausblick auf die nächsten Schritte. Es ist jedoch wesentlich einfacher mit neuen Erkenntnissen und daraus folgenden Anpassungen des Plans umzugehen.

Agile Vorgehen versuchen so wenig wie möglich zu planen. Sie können als eine Erweiterung des iterativen Vorgehens verstanden werden, bei dem sich an Zielen, und nicht an einem aufgestellten Plan, orientiert wird. Hierfür muss nach jeder Iteration der momentane Zustand sowie das weitere Vorgehen bestimmt werden.

Da der aktuelle Stand von Saros/I nicht komplett erfasst werden konnte, es also noch nicht erfasste Versagen geben könnte, welche auch im Rahmen dieser Arbeit bearbeitet werden müssten, wurde für diese Arbeit ein iteratives Vorgehen nach dem Spiralmodell gewählt. Dabei wurde jeweils das schwerwiegendste bekannte Problem zuerst bearbeitet und nach jeder Iteration eine Analyse des momentanen Zustands durchgeführt. Dies ließ sich auch gut mit dem Patch-orientierten Entwicklungsmodell im Saros-Projekt vereinbaren.

Zur Auswahl des Ziels der nächsten Iteration habe ich jeweils das schwerwiegendste Versagen aus der in Abschnitt 7 aufgestellten Liste der erfassten Versagen gewählt. Falls während einer Iteration neue Versagen erkannt wurden, habe ich diese entweder als Teil derselben Iteration behoben oder zu der Liste der bekannten Versagen hinzugefügt.

In jeder Iteration habe ich die Ursache des zu bearbeitenden Versagens genauer analysiert und, aufbauend auf den in der Analyse gefundenen Defekten, Patches erstellt, welche diese Defekte beheben sollen. Diese Patches wurden dann über Gerrit von anderen Mitgliedern des Saros-Projekts begutachtet. Zum Abschluss einer jeden Iteration erfolgte ein Test der gewählten Funktionalität, um sicherzugehen, dass die Iteration erfolgreich abgeschlossen werden kann.

Nach diesem Schema wurden drei Iterationen durchgeführt, von denen die letzte aufgrund eines neu erkannten, schwerwiegenderen Problems mit der Ressourcenverwaltung in Saros/I abgebrochen wurde. Näheres hierzu wird in Abschnitt 9 auf Seite 28 aufgeführt.

3.5 Anpassung des Vorgehens

Aufbauend auf dem entdeckten Problem mit der Ressourcenverwaltung in Saros/I musste das weitere Vorgehen entschieden werden. Dabei gab es folgende Alternativen:

- Die momentane Implementierung hätte repariert bzw. angepasst werden können.

  Diese Option wäre vorteilhaft, wenn angenommen werden könnte, dass die umliegenden Komponenten korrekt funktionieren und somit nach der Reparatur nicht angepasst werden müssten.

- Die momentane Implementierung hätte verworfen und die Funktionalität neu implementiert werden können.

  Diese Option wäre von Vorteil, wenn die Reparatur länger als die Reimplementierung dauern würde. Dies kann bei mangelnder Codequalität oder fälschlich getroffenen grundlegenden Designentscheidungen der Fall sein.

Aufgrund der entdeckten Fehler im Design der Implementierung, sowie einer hohen Anzahl an Defekten und einer mangelhaften Codequalität, welche die weitere Arbeit erheblich erschweren würde, habe ich die Entscheidung getroffen, dass die Reparatur der vorliegenden Implementierung zu aufwändig wäre und eine komplette Reimplementierung für das weitere Vorgehen einen höheren Nutzen erzielen würde.

4 Einarbeitung in das Saros-Projekt

Allgemeine Informationen zu der Arbeitsweise im Saros-Projekt habe ich den Saros-Wiki-Einträgen zu den Projekt-Richtlinien [27] entnommen.

4.1 Herstellung einer funktionsfähigen Arbeitsumgebung

Um produktiv arbeiten zu können, musste ich eine funktionsfähige Arbeitsumgebung einrichten. Hierzu habe ich Eclipse und IntelliJ installiert, sowie eine virtuelle Maschine aufgesetzt. Über diese virtuelle Maschine wurde so eine unabhängige Umgebung geschaffen, auf welcher eine zweite Saros-Instanz zu Testzwecken laufen kann. Danach habe ich das Saros-Projekt geladen und eingerichtet[17], sowie einen Gerrit- und SourceForge-Account angelegt.

4.2 Einarbeitung in die Saros-Gesamtstruktur

Um ein besseres Verständnis von Saros und möglichen Fehlerquellen zu erhalten, habe ich verschiedene Dokumentationen über Saros [28] und benutzte Komponenten (wie den PicoContainer [13][14]) gelesen.

Zusätzlich habe ich mich in den Saros-Quellcode mithilfe der jTourBus-Touren eingearbeitet.

4.3 Einarbeitung in das Saros/I-Projekt

Um ein besseres Verständnis des Saros/I-Quellcodes und der generellen Arbeitsweise mit Gerrit zu erlangen und den Einstieg in das Saros/I-Projekt zu erleichtern, habe ich einen bereits bestehenden, fast vollständigen Patch vollendet, der im Rahmen eines Softwareprojekts an der Freien Universität Berlin angefangen, aber nicht vollendet wurde.

Die Arbeit an dem Patch beschränkte sich größtenteils auf die Anpassung an eine Änderung der Saros-Schnittstellen, sowie eine Umstrukturierung, um die Lesbarkeit zu verbessern. Dennoch erlaubte das Vollenden des Patches einen guten Einblick in mir vorher unbekannte Aspekte des Saros/I-Projekts (den Umgang mit Modulen, dem PicoContainer und Lese- und Schreibaktionen) und einen einfachen Einstieg in das Code-Review-Verfahren. Details zu der Umsetzung können im Gerrit-Patch #3177 eingesehen werden.

[17] http://www.saros-project.org/saros-for-intellij/dev-environment

5 Aufgestellte Meilensteine

Bei der Durchführung einer Anforderungsanalyse für eine Alpha-Version hat sich gezeigt, dass die Herstellung der Grundfunktionalität für eine Alpha-Version nicht ausreicht.

Zusätzlich muss die Saros/I-GUI erweitert werden, da über diese im initialen Zustand nicht alle Funktionalitäten erreicht werden konnten. Es konnten nur XMPP-Accounts hinzugefügt und Sitzungen gestartet bzw. akzeptiert werden. Es war jedoch nicht möglich, die hinzugefügten XMPP-Accounts sowie ihre Einstellungen zu verwalten.

Da die HMTL-GUI schon seit geraumer Zeit in Arbeit war und sich somit eine Erweiterung der bereits bestehenden Swing-GUI[18] aufgrund der kommenden Umstellung nicht lohnen würde, sollte die HTML-GUI in Saros/I integriert werden. Diese vermeidet zusätzlich einen Mehraufwand der Nutzer und Implementierer bei einer späteren Umstellung der Oberfläche. Hierfür muss jedoch die HTML-GUI ebenfalls erweitert werden, da sie im initialen Zustand auch nicht für alle gewünschten Funktionalitäten eine Anbindung bot.

Aufbauend auf diesen Erkenntnissen habe ich drei Meilensteine für die Erstellung einer Saros/I-Alpha-Version definiert:

Meilenstein 1:
Die grundlegende Funktionalität von Saros/I ist vorhanden.

Dieser Meilenstein ist erfüllt, wenn die minimal nötigen Funktionalitäten von Saros/I vorhanden sind, sodass die Nutzung von Saros/I in einem eingeschränkten bzw. grundlegenden Rahmen möglich ist. Hierzu sollte es möglich sein, einfache Sitzungen zu erstellen und auf den geteilten Ressourcen synchron zu arbeiten.

Meilenstein 2:
Die grundlegende Funktionalität der HTML-GUI ist vorhanden.

Dieser Meilenstein ist erfüllt, wenn die HTML-GUI in Saros/I integriert und an alle nötigen Funktionalität angebunden ist. Diese Funktionalitäten beschränken sich vorerst auf das Erstellen und Verwalten von XMPP-Accounts, sowie das Erstellen und Beitreten von Saros-Sitzungen.

[18] Swing ist eine Standard-Java-Bibliothek zum erstellen von GUIs. Da diese Bibliothek für die IntelliJ-GUI verwendet wird, wurde die Saros/I-GUI auch mithilfe dieser Bibliothek erstellt.

Meilenstein 3:
Die erste Alpha-Version ist vollständig und veröffentlicht.
Dieser Meilenstein ist erfüllt, wenn die Alpha-Version veröffentlicht und ihre Benutzung dokumentiert wurde. Eine Veröffentlichung bedeutet hierbei, dass eine gepackte Saros/I-Alpha-Version auf der Saros-Webseite zum Download angeboten wird. Diese Version kann dann von Nutzern heruntergeladen und in IntelliJ importiert werden.

Von diesen Meilensteinen wurde im Rahmen der Bachelorarbeit keiner erreicht, es wurden jedoch Fortschritte in der Bearbeitung des ersten Meilensteins gemacht, sowie eine gute Grundlage für das weitere Vorgehen geschaffen. Gründe hierfür werden im Abschnitt 9 auf Seite 28 und Abschnitt 10 auf Seite 34 erläutert.

6 Minimaler Funktionsumfang

Um die ursprüngliche, zum Start dieser Bachelorarbeit vorhandene Funktionalität von Saros/I besser einschätzen zu können, habe ich parallel zu der Einarbeitung den angestrebten Zustand festgelegt. Hierzu habe ich einen Vergleich zwischen den dokumentierten Funktionalitäten von Saros/E und Saros/I angestellt.

Als Grundlage habe ich das Saros-Wiki [29] sowie Beschreibungen des derzeitigen Stands des Saros/I-Plugins [30][31] verwendet.

6.1 Minimal nötige Funktionalitäten

Aus diesem Vergleich ergab sich eine Liste an minimal nötigen Funktionalitäten:

Ein Nutzer sollte in der Lage sein...

1. *XMPP-Accounts...*
   (a) *zu erstellen,*
   (b) *zu löschen,*
   (c) *zu verwalten,*
2. *eine Saros-Sitzung zu erstellen,*
3. *bei der Erstellung einer Sitzung ein Modul*[19] *zu teilen,*
4. *einer Saros/I-Sitzung*[20] *beizutreten,*
5. *auf dem Inhalt von geteilten Dateien zu arbeiten und*
6. *geteilte Dateien während einer Sitzung...*
   (a) *zu erstellen,*
   (b) *zu löschen,*
   (c) *zu verschieben*[21] *und*
   (d) *umzubenennen.*[21]

[19] IntelliJ-Module werden in Saros als das Äquivalent zu Eclipse-Projekten gehandhabt

[20] Saros/I-Sitzung meint eine Sitzung, die von einer Saros/I-Instanz erstellt wurde.

[21] Diese Funktionalität wurde im Verlauf dieser Arbeit erst nachträglich zu der Liste der minimal nötigen Funktionalität hinzugefügt, weshalb hierfür während der Analyse des Ist-Zustands keine Tests durchgeführt wurden.

Da für die Nutzung von Saros XMPP-Accounts benötigt werden, sollte die Erstellung und Verwaltung solcher Accounts über Saros/I möglich sein.

Um mit Saros/I wenigstens eingeschränkt arbeiten zu können, sollte es zudem möglich sein, Sitzungen zwischen zwei Saros/I-Instanzen zu erstellen und hierbei ein IntelliJ-Modul[19] zu teilen. Auf den Dateien dieses Moduls soll von beiden Saros/I-Instanzen synchron gearbeitet werden können und es soll möglich sein, geteilte Dateien während einer Sitzung zu erstellen, zu löschen, zu verschieben und umzubenennen.

6.2 Nicht betrachtete Funktionalität

Zusätzlich ergab sich eine Reihe an Funktionalitäten, die bewusst nicht Teil der Alpha-Version sein sollten. Sie wären für eine vollständige Saros/I-Version erforderlich, sind jedoch für eine minimale Funktionalität nicht von Bedeutung und wurden deshalb von mir im Rahmen dieser Arbeit nicht näher betrachtet:

Es muss nicht möglich sein...

- eine Sitzung zwischen Saros-Instanzen basierend auf unterschiedlichen IDEs (z. B. Saros/I und Saros/E) aufzubauen,
- komplette Projekte[22] zu teilen,
- während einer Sitzung neue Module zu teilen,
- Module mit mehreren Content-Roots[23] zu teilen,
- nur bestimmte Teile eines Moduls zu teilen oder
- mit anderen Teilnehmern über die Saros-Sitzung zu kommunizieren (z. B. über eine Chatfunktionalität)

[22] Ein IntelliJ-Projekt ist eine Ansammlung aus einem oder mehreren Modulen. Es wird durch eine eigene Instanz der IntelliJ-IDE, ein eigenes IntelliJ-Fenster, repräsentiert.

[23] Eine Content-Root ist ein Ordner, in welchem sich die zum Modul zugehörigen Dateien und Ordner befinden. Ein Modul kann dabei beliebig viele Content-Roots haben. Siehe https://www.jetbrains.com/help/idea/2017.1/content-root.html

7 Beobachtete Versagen

Aufbauend auf dem im Abschnitt 6.1 aufgestellten minimalen Funktionsumfang habe ich den aktuellen Zustand der für die Saros/I-Alpha-Version nötigen Funktionalitäten erfasst.

Beim Testen der im Punkt 2 beschriebenen Funktionalität, dem Erstellen einer Sitzung, wurde folgendes Versagen beobachtet:

1. *Das Ändern von geteilten Dateien während des Sitzungsaufbaus führt zu einem nicht vorhersagbaren Fehlverhalten der Sitzung* (Bug #891)

   Dieses Versagen tritt auf, wenn der Host einer Sitzung während der Initialisierung den Inhalt von Dateien, die Teil der Sitzung sind, verändert. Dies führt nach Vollendung der Sitzungsinitialisierung zu einer Desynchronisation der teilnehmenden Saros-Instanz. Eine Resynchronisation dieser Dateien scheint in diesem Zustand nicht möglich zu sein.

   Ein ähnliches Versagen konnte ursprünglich auch in Saros/E beobachtet werden. Dort wurde es jedoch gelöst, indem während der Sitzungsinitialisierung alle Editor-Fenster[24] geschlossen und Aktionen auf der IDE gesperrt werden. In IntelliJ wurde dieses Versagen jedoch noch nicht offiziell festgehalten, weshalb im Rahmen dieser Arbeit hierfür ein Bugtracker-Eintrag erstellt wurde.

Beim Testen der im Punkt 3 beschriebenen Funktionalität, dem Teilen eines Moduls, wurde folgendes Versagen beobachtet:

2. *IntelliJ-Module werden nach der Übertragung nicht als Module registriert* (Bug #868)

   Dieses Versagen tritt auf, wenn als Teil der Sitzungsinitialisierung das zu teilende Modul zu der teilnehmenden Saros-Instanz übertragen werden soll. Die Dateien und Ordner des Moduls werden dabei zwar übertragen, das Modul wird jedoch nicht bei der IntelliJ-Instanz des Empfängers registriert. Dies führt dazu, dass IntelliJ die Ordner zwar in der Projektstruktur aufführt, sie jedoch nicht als Modul anerkennt. Des Weiteren führt es dazu, dass Aktionen des Empfängers auf Dateien des Moduls nicht über Saros übertragen werden können.

   Der Bugtracker-Eintrag sowie der angefangene Patch (Gerrit-Patch #3177) wurden von B. Sahre erstellt.

[24] Editor-Fenster meint hier eine in der IDE offene Datei. Hierbei sind keine wirklichen Fenster der IDE gemeint. Da es in einer IDE mehrere offene Dateien geben kann und die momentan fokussierte/aktive Datei in der folgenden Ausführung eine Rolle spielt, wird die Metapher der „Fenster“ verwendet.

Beim Testen der im Punkt 5 beschriebenen Funktionalität, dem synchronen Arbeiten auf geteilten Dateien, wurden folgende Versagen beobachtet:

3. *Die Synchronisation funktioniert nicht bei Dateien, die beim Start der Sitzung bereits offen sind* (Bug #872)

   Dieses Versagen tritt auf, wenn der Host einer Sitzung während der Initialisierung Editor-Fenster für geteilte Dateien geöffnet hat. Das hat zur Folge, dass Änderungen auf den geöffneten Editoren nicht an die teilnehmende Saros-Instanz übertragen werden.

   Das Versagen, sowie ein grobe Beschreibung der unterliegenden Defekte, wurde in dem von E. Girard verfassten Überblick über den derzeitigen Stand von Saros/I [30] festgehalten. Da in diesem Dokument jedoch nicht deutlich gemacht wird, welche beschriebenen Defekte bereits bearbeitet wurden und der von B. Sahre angelegte Bugtracker-Eintrag zu diesem Versagen keinerlei Hinweise auf einen ähnlichen Ursprung enthielt, wurde der beschriebene Defekt fälschlicherweise als behoben angenommen.

4. *Das Empfangen von Änderungen an einer Datei, für die kein Editor-Fenster offen ist, führt dazu, dass der gesamte Inhalt der Datei als Aktivität zurückgesendet wird* (Bug #890)

   Dieses Versagen tritt auf, wenn während einer Saros-Sitzung Änderungen an geteilten Dateien, für die lokal kein Editor-Fenster offen ist, empfangen werden. Dies führt dazu, dass der gesamte Inhalt der veränderten Datei als Saros-Aktivität zurückgesendet wird, was bei dem Sender der ursprünglichen Aktivität zu einem Duplizieren des Dateiinhalts führt eine Desynchronisation auslöst.

   Das Versagen wurde von mir während dieser Arbeit entdeckt und in einem Bugtracker-Eintrag festgehalten.

5. *Die Synchronisation von geteilten Dateien funktioniert nicht, wenn als Teil der Sitzungserstellung Dateien übertragen werden müssen* (Bug #894)

   Dieses Versagen tritt auf, wenn als Teil der Sitzungserstellung Dateien übertragen werden müssen. Das ist der Fall, wenn ein neues Modul geteilt wird, oder wenn der Inhalt einiger geteilter Dateien nicht bei allen Clients übereinstimmt. Dies führt dazu, dass teilnehmende Saros-Instanzen, die Dateien empfangen haben, Änderungen auf diesen Dateien nicht übertragen.

   Das Versagen wurde von mir während dieser Arbeit entdeckt und in einem Bugtracker-Eintrag festgehalten.

6. *Offene Editor-Fenster für Dateien, die nicht zu dem geteilten Modul gehören, führen bei dem Start einer Sitzung zu Fehlern* (Bug #896)

   Dieses Versagen tritt auf, wenn der Host einer Sitzung während der Initialisierung Editor-Fenster für Dateien, die nicht zu dem geteilten Modul gehören, geöffnet hat. Daraus folgt, dass Änderungen auf manchen der geöffneten Editoren nicht an die partizipierende Saros-Instanz übertragen werden.

   Zuerst wurde angenommen, dass dieses Versagen mit dem im Punkt 3 beschriebenen Versagen übereinstimmt. Eine Unterscheidung konnte erst nach der Behebung des Bug#872 vorgenommen werden, wonach hierfür ein separater Bugtracker-Eintrag erstellt wurde.

Beim Testen der im Punkt 6a beschriebenen Funktionalität, dem Erstellen neuer Dateien während einer Sitzung, wurde folgendes Versagen beobachtet:

7. *Dateien, die während einer Sitzung über eine Vorlage erstellt wurden, werden leer übertragen* (Bug #889)

   Dieses Versagen tritt auf, wenn während einer Sitzung Dateien mithilfe einer Vorlage[25] erstellt werden. Dies führt dazu, dass die Datei ohne den Inhalt der Vorlage übertragen wird. Wurde die Datei von dem Host der Sitzung erstellt, kann der Inhalt durch eine Resynchronisation nachgeladen werden. Wurde die Datei jedoch von einem der Clients erstellt, führt eine Resynchronisation zum Löschen des Inhalts auf Seiten des Clients, da bei einer Resynchronisation immer an den Zustand des Hosts angepasst wird.

   Das Versagen wurde von mir während dieser Arbeit entdeckt und in einem Bugtracker-Eintrag festgehalten. In dem von E. Girard verfassten Überblick über den derzeitigen Stand von Saros/I [30], sowie einigen Anmerkungen im Saros-Quellcode [32] wird jedoch angedeutet, dass sich mit diesem Versagen bereits zuvor beschäftigt wurde. Des Weiteren bestand bereits ein Bugtracker-Eintrag für eine eingeschränkte Version dieses Versagens (Bug #863), welche jedoch auf der Annahme beruht, dass die damals vorgenommene Reparatur dieses Versagens erfolgreich war.

Beim Testen der im Punkt 6b beschriebenen Funktionalität, dem Löschen geteilter Dateien während einer Sitzung, wurde folgendes Versagen beobachtet:

[25] Ein Beispiel für eine solche Vorlage ist die leere Klasse, welche den Namen der Datei trägt, die bei der Erstellung einer Java Datei automatisch von der IDE eingefügt wird.

8. *Das Löschen einer geteilten Datei während einer Sitzung führt zu einer Desynchronisation der teilnehmenden Saros-Instanzen* (Bug #897)

   Dieses Versagen tritt auf, wenn während einer Sitzung geteilte Dateien gelöscht werden. Dies führt zu einer Desynchronisation der teilnehmenden Saros-Instanzen. Eine Resynchronisation scheint in diesem Zustand nicht möglich zu sein.

   Das Versagen wurde in dem von E. Girard verfassten Überblick über den derzeitigen Stand von Saros/I [30] erwähnt, es wurde jedoch kein Bugtracker-Eintrag angelegt. Dies wurde während dieser Arbeit von mir nachgeholt.

Während der Arbeit an den erkannten Versagen wurde bei dem Testen der im Punkt 6d beschriebenen Funktionalität, dem Umbenennen von Dateien, folgendes Versagen beobachtet:

9. *Das Umbenennen einer geteilten Datei während einer Sitzung führt zu einer Desynchronisation der teilnehmenden Saros-Instanzen*

   Dieses Versagen tritt auf, wenn während einer Sitzung geteilte Dateien umbenannt werden. Dies führt zu einer Desynchronisation der teilnehmenden Saros-Instanzen, bei welcher Aktionen auf den umbenannten Dateien nicht von Saros übertragen werden. Eine Resynchronisation scheint in diesem zustand nicht möglich zu sein. Wenn die Umbenennung als Teil einer Refaktorisierung vorgenommen wird, wird eine Entsprechende Anpassung des Dateiinhalts an die Namensänderung auch nicht von Saros übertragen.

   Für dieses Versagen existiert kein Eintrag im Bugtracker, da es von mir erst bei der Behebung einer der anderen erfassten Versagen erkannt und direkt im Anschluss bearbeitet wurde.

8 Iterative Bearbeitung der erkannten Versagen

Zum Beheben der erkannten Versagen habe ich nach dem in Abschnitt 3.4 beschriebenen Schema drei Iterationen ausgeführt. Da es in der dritten Iteration zu Problemen mit der derzeitigen Implementierung von Teilen der Saros-Schnittstellen kam, habe ich das iterative Vorgehen abgebrochen und mich entschieden, eine größere Reimplementierung dieser Schnittstellen vorzunehmen. Das Vorgehen hierbei wird im Abschnitt 9 auf Seite 28 beschrieben.

8.1 Versagen bei der Synchronisation von Aktionen

In der ersten Iteration sollte die korrekte Synchronisation der geteilten Dateien sichergestellt werden (Punkt 3 der beobachteten Versagen).

Da die Synchronisation von Aktionen auf geteilten Dateien das Kern-Feature von Saros ist und, wie in Abschnitt 7 ersichtlich, die meisten erfassten Versagen entweder direkt Synchronisationsprobleme sind, oder das Versagen dieser als Folge haben, erschien mir das Beheben dieses Versagen als ein grundlegender Schritt, welchen ich vor der Bearbeitung der restlichen Versagen machen musste. Andernfalls hätte ich keine Möglichkeit gehabt, während der Tests einen funktionierenden Ausgangszustand herzustellen, in welchem ich dann gezielt die zu bearbeitenden Versagen beobachten kann. Dies hätte es sehr schwierig gemacht die verschiedenen Versagen voneinander zu differenzieren. Aufgrund dessen habe ich mich entschieden dieses Versagen als Erstes zu bearbeiten.

8.1.1 Suche nach möglichen Ursachen

Um das Versagen beheben zu können, musste ich als Erstes die Ursache erkennen. Diese habe ich zuerst beim Empfangen und Versenden der Aktivitäten über das Netzwerk vermutet. Um diese Vermutung zu überprüfen, habe ich den Smack-Debugger verwendet, um die ausgetauschten Pakete einer Saros/I-Sitzung (eine Sitzung zwischen zwei Saros/I-Instanzen) mit denen einer Saros/E-Sitzung zu vergleichen.

Da Saros/E als komplett funktionell angesehen werden kann, wurde das in Saros/E beobachtete Verhalten als korrekt angenommen und als Grundlage des Vergleichs verwendet. Größere Abweichungen im Protokoll der beiden Tests hätten auf einen möglichen Defekt beim Umgang mit Datenpaketen in Saros/I hinweisen können.

Es gab jedoch nur geringfügige Abweichungen, die auf den erweiterten Funktionssatz von Saros/E zurückzuführen sind, weshalb der Datenaustausch zwischen zwei Saros/I-Instanzen als funktionell angenommen wurde. Dies bedeutet, dass der Defekt in der internen Verarbeitung der ausgetauschten Daten liegen muss.

Um die innere Verarbeitung der ausgetauschten Daten zu überprüfen, habe ich das Versagen erneut reproduziert und die anfallenden Log-Dateien nochmals genauer betrachtet. Hierbei habe ich Warnungen[26] gefunden, die auf den Ursprung des Versagens hinwiesen. Diese Warnungen habe ich während der Analyse des aktuellen Zustandes nicht erkannt, da ich mit Fehlern[27] oder Exceptions gerechnet hatte, und sie somit in der Menge der Log-Einträge untergegangen sind.

8.1.2 Beschreibung der Ursachen

Die Ursache des Versagens war ein Defekt in der Erstellung des internen Zustandes der Saros-Sitzung. Um nur Änderungen an geteilten Dateien zu übertragen, hält sich die Saros-Sitzung einen Verweis auf alle offenen Dateien, die Teil der Sitzung sind. Dieser Verweis wird in Form einer Map der Pfade der Dateien auf eine Objekt-Repräsentation der jeweiligen Editoren gehalten.

Der vorliegende Defekt bestand darin, dass beim Start einer Sitzung für die Erstellung dieser Map immer das Editor-Objekt des momentan aktiven Editors benutzt wurde. War beim Start der Sitzung höchstens ein Editor-Fenster offen, führte dies zu keinen Problemen. Bei nur einem offenen Editor-Fenster musste dies auch immer das aktive Fenster sein, womit immer ein korrekter Eintrag in der Map vorgenommen wurde. Wenn keine Editor-Fenster offen waren, mussten der Map auch kein Einträge hinzugefügt werden. Waren jedoch mehrere Editor-Fenster offen, führte es dazu, dass Aktivitäten auf Editor-Fenstern, die beim Start der Sitzung nicht aktiv waren, nicht übertragen werden konnten, da für sie kein Eintrag im internen Zustand der Sitzung bestand.

Dies kann an einem Beispiel verdeutlicht werden:

- Alice möchte gerne eine Sitzung mit Bob aufbauen. Dafür existiert bei Beiden schon das Modul M mit den Dateien X und Y.
- Alice hat nun für beide Dateien Editor-Fenster offen, wobei das Fenster für die Datei X aktiv ist.
- Bob hat nur für die Datei X ein Editor-Fenster offen.
- Alice erstellt nun eine Sitzung mit Bob.
- Alice und Bob können nun problemlos beide synchron auf der Datei X arbeiten.
- Bob öffnet nun ein Editor-Fenster für die Datei Y.

[26] Log4J Einträge der Stufe *warning*
[27] Log4J Einträge der Stufe *error*

- Bobs Aktionen auf der Datei Y werden nun an Alice übertragen.
- Alice kann zwar lokal auf der Datei Y arbeiten, ihre Änderungen werden jedoch nicht an Bob übertragen. Dies liegt daran, dass in der Map der Pfade für den Pfad der Datei Y ein Editor-Objekt für die Datei X eingetragen wurde. Da Saros das Editor-Objekt für die Datei Y somit nicht finden kann, werden die Änderungen von Alice nicht übertragen.
- Alice kann nun das Editor-Fenster für die Datei Y schließen und erneut öffnen, um eine normale Funktionalität wiederherzustellen.

Eine mögliche Begründung für den Fehler, welcher zu der Entstehung dieses Defekts geführt hat, kann in den von der IntelliJ-API bereitgestellten Methoden gefunden werden. Es gibt nur zwei Methoden, welche ein Editor-Objekt vom benötigten Typ zurückgeben:

- Eine dieser Methoden erwartet andere IntelliJ-Objekte und gibt dann das passende Editor-Objekt zurück.
- Die andere Methode erwartet keine Parameter und gibt ein Editor-Objekt für den momentan aktiven Editor zurück.

Da die Erstellung und Verwendung der für die Nutzung der ersten Methode zu übergebenden Objekte nicht direkt offensichtlich ist und für die Nutzung der zweiten Methode keine Parameter nötig sind, schien es wahrscheinlich einfacher die zweite Methode zu nutzen. Um hierbei die korrekte Funktionalität zu replizieren, wurde versucht, diese Methode in Verbindung mit einem Aufruf, welcher den gewünschten Editor aktivieren sollte, zu verwenden. Dieser Aufruf scheint jedoch nicht korrekt zu funktionieren, was zu dem hier beschriebenen Defekt führte.

8.1.3 Behebung der Defekte

Dieser Defekt konnte einfach behoben werden, indem das jeweils richtige Editor-Objekt beim Start der Sitzung in die Map eingetragen wurde. Details zu der Umsetzung können im Gerrit-Patch #3240 eingesehen werden.

Bei der Arbeit an diesem Patch habe ich zudem festgestellt, dass die reparierte Funktionalität als Reaktion auf das falsche Ereignis ausgeführt wird. Auf das Starten einer Sitzung wird über einen Event-Listener reagiert. Die Funktionalität wurde für ein falsches Ereignis eingetragen, was einen ähnlichen Namen hatte. Dieses Problem habe ich auch noch als Teil der Iteration behoben. Details zu der Umsetzung können im Gerrit-Patch #3242 eingesehen werden.

Des Weiteren habe ich die Funktionalität leicht erweitert, um alle Möglichkeiten der intern genutzten API nach außen weiterzugeben. Bei der Erstellung eines Editor-Objekts über die IntelliJ-API kann angegeben werden,

ob dabei das zugehörige Editor-Fenster aktiviert werden soll. Diese Option sollte auch in dem in Saros genutzten Adapter dieser Funktion vorhanden sein. Details zu der Umsetzung können im Gerrit-Patch #3241 eingesehen werden.

Während der Behebung dieses Defekts wurde zusätzlich erkannt, dass hier ein weiteres Versagen vorlag, welches bei der Analyse des Ist-Zustands zu diesem Versagen gezählt wurde. Dieses wird in Punkt 6 der beobachteten Versagen aufgeführt und näher beschrieben.

Nach Abschluss dieser Iteration war es in Saros/I möglich, auf geteilten Dateien synchron zu arbeiten, unabhängig davon, ob für sie während der Initialisierung der Sitzung bereits Editor-Fenster offen waren. Dies funktionierte jedoch weiterhin nur unter der Einschränkung, dass bei der Initialisierung der Sitzung keine Dateien übertragen werden mussten, und dass alle Dateien, auf denen gearbeitet wird, bei allen Saros/I-Instanzen offen sind.

8.2 Versagen bei der Umbenennung von Dateien

In der zweiten Iteration sollte die korrekte Synchronisation der Umbenennungen von geteilten Dateien ermöglicht werden (Punkt 9 der beobachteten Versagen).

Dieses Versagen habe ich als Aufgabe dieser Iteration gewählt, da es dem in der letzten Iteration behandelten Versagen sehr ähnlich schien, und ich deshalb eine ähnliche Ursache vermutete.

8.2.1 Suche nach möglichen Ursachen

Für die Suche nach der Ursache des Versagens habe ich wieder die bei der Durchführung des Tests anfallenden Log-Dateien betrachtet. Hierbei konnte ich jedoch keine Hinweise auf eine mögliche Ursache finden.

Als nächsten Schritt habe ich den Java-Debugger verwendet, um den internen Zustand der Sitzung begutachten zu können, da ich die Ursache wieder in diesem Bereich vermutete. Hierzu habe ich Haltepunkte in der auszuführenden Funktionalität platziert, über welche ich den Kontrollfluss und jeweiligen internen Zustand nach jedem Aufruf begutachten konnte.
In dem Saros-Quellcode war dabei nicht direkt ersichtlich, welche Methoden für die Verarbeitung von Umbenennungen von geteilten Dateien zuständig waren. Durch die Arbeit in der vorherigen Iteration waren jedoch bereits die Methoden bekannt, welche eingehende und ausgehende Aktionen für die Veränderung der lokalen Dateistruktur verwalten. Deshalb wurden am Anfang dieser Methoden Haltepunkte platziert. Durch den Kontrollfluss konnten dann die genauen Methoden bestimmt werden, welche sich in Saros mit der Umbenennung von Dateien befassen.

Die Vermutung, dass die Ursache des Versagens wieder im internen Zustand der Sitzung lag, erwies sich als korrekt.

8.2.2 Beschreibung der Ursachen

Die Ursache des Versagens war ein Defekt in der Aktualisierung des internen Zustands der Sitzung bei Änderungen an der geteilten Dateistruktur. Bei der Initialisierung einer Sitzung erstellt Saros eine Liste an Objekten, die alle geteilten Module repräsentiert. Jedes Modul-Objekt enthält dabei eine Liste aller zugehörigen Dateien. Dadurch kann aus der Liste an Modul-Objekten eine Liste aller geteilten Dateien erstellt werden, welche die Grundlage für die in Abschnitt 8.1.2 beschriebenen Map der offenen Dateien darstellt.
Der vorliegende Defekt bestand nun darin, dass bei einer Umbenennung der Eintrag der Datei in dem zugehörigen Modul-Objekt nicht aktualisiert wurde. Dies führte dazu, dass umbenannte Dateien beim Öffnen nicht in die Map der offenen Dateien eingetragen werden konnten, weshalb auf die Umbenennung folgende Aktionen nicht von Saros übertragen werden konnten.

Die Anpassung des Pfads im zugehörigen Modul-Objekt wurde wahrscheinlich bei der Implementierung dieser Funktionalität schlichtweg vergessen.

8.2.3 Behebung der Defekte

Dieser Defekt konnte behoben werden, indem das Modul-Objekt der geteilten Dateien und die Map der offenen Editoren korrekt an die Umbenennung angepasst wurden. Zusätzlich musste ich auch noch den Umgang mit empfangenen Umbenennungs-Aktionen, welche durch eine Refaktorisierung ausgelöst wurden, anpassen, da es hier zu Problemen der Ablaufreihenfolge kam. Der Eintrag der Datei im Modul-Objekt wurde angepasst, bevor die durch die Refaktorisierung erzeugte Namensanpassung in der Datei durchgeführt wurde. Dies hatte zur Folge, dass die mit der Namensänderung einhergehende Anpassung des Dateiinhalts nicht angewendet werden konnte. Details zu der Umsetzung können im Gerrit Patch #3252 eingesehen werden.

Zusätzlich musste ich die Anpassung des internen Zustandes auf der Empfänger-Seite korrigieren, da hier jeweils ein neues Modul-Objekt erstellt und angepasst wurde, anstatt das Modul-Objekt der Sitzung anzupassen. Dies konnte durch eine Funktion, welche das Modul-Objekt der Sitzung für eine gegebene Datei zurückliefert, ermöglicht werden. Details zu der Umsetzung können im Gerrit-Patch #3251 eingesehen werden.

Nach dieser Iteration war es, neben dem synchronen Arbeiten auf geteilten Dateien, nun auch möglich, geteilte Dateien umzubenennen.

8.3 Versagen bei der Erstellung von Dateien

In der dritten Iteration sollte die korrekte Übertragung des initialen Inhalts erstellter Dateien, sowie deren Synchronisation sichergestellt werden (Punkt 7 der beobachteten Versagen).

Dieses Versagen habe ich als Aufgabe dieser Iteration gewählt, da es dem in den letzten beiden Iterationen behandelten Versagen sehr ähnlich schien, und ich deshalb eine ähnliche Ursache vermutete.

8.3.1 Suche nach möglichen Ursachen

Hier habe ich, wie in Abschnitt 8.2.1, die Log-Dateien, sowie den Kontrollfluss genauer betrachtet. Die Suche in den Log-Dateien ergab wieder keinerlei Hinweise auf die Ursache des Versagens.

8.3.2 Beschreibung der Ursachen

Die Ursache des Versagens konnte auf zwei Defekte zurückgeführt werden:

1. Der initiale Inhalt der neu erstellten Dateien wurde nicht übertragen. Dies kann darauf zurückgeführt werden, dass der initiale Inhalt (wie z. B. Vorlagen der IDE für verschiedene Dateitypen) eingefügt wurde, bevor der Listener für die Datei mit der Sitzung registriert werden konnte. Dies führt zu einer sofortigen Desynchronisation der Sitzung.

   Für dieses Problem wurde bereits in der initialen Implementierung [33] ein Lösungsversuch in Saros/I integriert. Dieser funktioniert jedoch nur eingeschränkt und führt zu einem Versagen, wenn Dateien gespeichert werden (Bug #890).

2. Die Synchronisation von Änderungen an der neu erstellten Datei funktionierte nicht, da der interne Zustand, wie in Abschnitt 8.2.2 beschrieben, nicht aktualisiert wurde.

8.3.3 Behebung der Defekte

Die Bearbeitung dieser Iteration wurde vor dem Erstellen einer Lösung ausgesetzt. Eine Erklärung hierfür wird im nächsten Abschnitt gegeben.

8.4 Aussetzen des iterativen Vorgehens

Während der Analyse des Versagens bei der Erstellung von Dateien habe ich tiefgreifende Defekte in der Umsetzung der Ressourcenverwaltung erkannt. Da diese Defekte behoben werden mussten, bevor das eigentliche Versagen bearbeitet werden konnte, habe ich die Iteration abgebrochen und die iterative Abarbeitung der erkannten Versagen ausgesetzt. Zu diesem Zeitpunkt war ungefähr ein Drittel der Bearbeitungszeit dieser Arbeit verstrichen.

9 Arbeit an der Saros/I-Ressourcenverwaltung

Nach der zweiten Iteration konnte (unter den am Ende von Abschnitt 8.1.3 erwähnten Einschränkungen) über Saros/I synchron auf geteilten Dateien gearbeitet, geteilte Dateien umbenannt und danach auf ihnen synchron gearbeitet werden. Wie bereits in Abschnitt 8.3 angedeutet, habe ich während der dritten Iteration tiefgreifende Defekte im Umgang mit Ressourcen in Saros/I erkannt, welche das Beheben der betreffenden Versagen nur schwer möglich machten.

9.1 Saros/I-Ressourcenverwaltung

Der Saros-Kern bietet ein Framework zur Ressourcenverwaltung an. Dies besteht aus verschiedenen Schnittstellen, welche von der jeweiligen Saros-Version passend zu der unterliegenden IDE implementiert werden müssen. Ein UML-Diagramm dieser Schnittstellen sowie ihrer Implementierungen kann in der Abbildung 1, eine größere Version in Abschnitt A.5 in Abbildung 3a betrachtet werden.

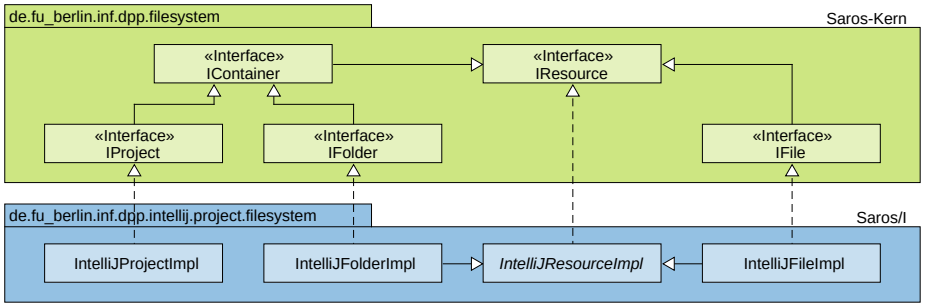


Abbildung 1: Aufbau der Saros/I-Ressourcenverwaltung.

IntelliJProjectImpl implementiert die IProject-Schnittstelle.

Objekte dieser Klasse repräsentieren ein IDE-„Projekt“, was bei Saros/I auf ein IntelliJ-Modul abgebildet wird.

IntelliJResourceImpl implementiert die IResource-Schnittstelle und repräsentiert eine allgemeine Ressource im lokalen Dateisystem.

IntelliJResourceImpl ist eine abstrakte Klasse, welche eine Basis für die Implementierung von IFile und IFolder bietet.

IntelliJFolderImpl erbt von der IntelliJResourceImpl-Klasse und implementiert die IFolder-Schnittstelle.

Objekte dieser Klasse repräsentieren einen Ordner im lokalen Dateisystem.

IntelliJFileImpl erbt von der IntelliJResourceImpl-Klasse und implementiert die IFile-Schnittstelle.

Objekte dieser Klasse repräsentieren eine Datei im lokalen Dateisystem.

9.2 Erkannte Defekte

Die Schnittstellen zur Ressourcenverwaltung wurden, wie bereits in Abschnitt 2.1 erwähnt, aufbauend auf den gleichnamigen Eclipse-Schnittstellen zur Ressourcenverwaltung erstellt. Die Eclipse-API basiert auf dem standardmäßigen Eclipse-Verhalten, in welchem komplett auf virtuellen Verweisen gearbeitet wird. Somit ist es möglich, Ressourcen-Objekte zu erstellen, welche nicht im lokalen Dateisystem vorhanden sind. Um auf Ressourcen arbeiten zu können, muss daher immer erst die Existenz der Ressource geprüft werden.

Da dieses Verhalten vom Saros-Kern erwartet wird, wurde in Saros/I versucht, es zu imitieren. Dies führte bei der Implementierung von Saros/I zu Problemen, da die Dateiverwaltung in IntelliJ keine Verweise auf nichtexistente Ressourcen zulässt, diese jedoch beim Verarbeiten der als Teil der Sitzungsinitialisierung übertragenen Ressourcen benötigt werden. Bei der Verarbeitung werden Objekte für alle übertragenen Ressourcen erstellt. Über einen Aufruf auf diesen Ressourcen-Objekten wird die jeweilige Existenz im lokalen Dateisystem geprüft. Existieren die Ressourcen nicht, werden sie über einen Aufruf auf dem jeweiligen Ressourcen-Objekt im lokalen Dateisystem erstellt. Andernfalls wird nur ihr Inhalt angepasst.

IntelliJ agiert vollständig auf einer Cache-Version des lokalen Dateisystems [34], in welcher genutzte Ressourcen gespeichert werden. Bei der initialen Implementierung wurde versucht, diese Differenz im Design zu umgehen, indem die IntelliJ-Anbindung an das Dateisystem nicht verwendet wurde. Stattdessen wurde, wie in Saros/E, ausschließlich mit Pfaden gearbeitet. Als Resultat dieser Entscheidung wurde die Implementierung nur sehr schwach an die IntelliJ-API angebunden. Aufgrund dieser fehlenden Anbindung mussten der Zustand, sowie die gebrauchte Funktionslogik des Dateisystems in der Implementierung gehalten werden.

Dies führte jedoch zu anderen Problemen, da erneut eine Annahmen der Eclipse-Implementierung fälschlicherweise auf IntelliJ übertragen wurden: In Eclipse liegen alle Ressourcen eines Projektes in einem Ordner, der „project root“. Bei IntelliJ können Module jedoch mehrere solche, von dem Pfad des Moduls unabhängige, Ordner haben. Um die Saros-Schnittstellen verwenden zu können, wurde jedoch ein eindeutiger Pfad für jedes Modul benötigt, über welchen enthaltene Ressourcen angesprochen werden konnten. Hierfür wurde fälschlicherweise der Pfad des IntelliJ-Projekts konkateniert mit dem

Modulnamen verwendet. Wenn das Projekt also z. B. im Pfad `/Foo/Bar/Project/` und mein Modul im Pfad `/My/Module/` liegt, wird als Pfad des Moduls `/Foo/Bar/Project/Module/` angenommen. Dies führte zu Fehlern der Saros-Logik, da mit falschen Pfaden gearbeitet wurde.

9.3 Mögliche Vorgehensweisen

Für die Anpassung der Saros/I-Ressourcenverwaltung gab es zwei mögliche Vorgehensweisen.

- Das vom Saros-Kern bereitgestellte Framework zur Ressourcenverwaltung, sowie das davon abhängige Verhalten des Kerns könnte angepasst werden, um standardmäßig auf dem lokalen Dateisystem und nicht auf virtuellen Verweisen zu arbeiten. Dies würde zusätzliche eine Abstraktion der Ressourcenverwaltung in der Saros/E-Implementierung erfordern.
- Das vom Saros-Kern erwartete bzw. in Saros/E umgesetzte Verhalten der Implementierung des Frameworks könnte für die Ressourcenverwaltung in Saros/I umgesetzt werden.

Im Rahmen dieser Arbeit habe ich mich gegen eine Anpassung des Frameworks entschieden. Anstatt dessen habe ich mich während der Reimplementierung an dem im Saros-Kern erwarteten Verhalten, sowie dem in der Eclipse-Dokumentation [35] aufgeführten, äußeren Verhalten[28] orientiert.

Diese Entscheidung basierte hauptsächlich auf der von mir fälschlicherweise getroffenen Annahme, dass das Framework eine Menge an festgelegten Schnittstellen ist, welche umgesetzt werden sollen. Diese Schnittstellen haben momentan nur einen Platzhalter-Charakter und sollten angepasst werden, wenn eine genauere, von der IDE unabhängige Definition der in Saros benötigten Funktionalität vorliegt. Dies war mir zu dieser Zeit jedoch nicht bewusst.

Des Weiteren wurde von mir die Anpassung des Kerns und die damit verbundene, erneute Einarbeitung als zu Zeitaufwändig angesehen. Diese Einschätzung wurde dadurch verstärkt, dass eine solche Änderung eines Saros-Kern-Frameworks einen erheblichen Aufwand in allen Saros-Implementierungen (Saros/E, Saros/I, Saros/N und Saros-Server) erzeugen würde.

Ansätze des ersten Vorgehens sind jedoch trotzdem in die Reimplementierung eingeflossen. Es wurden einige überflüssige Methoden, welche entweder gar nicht, oder nur noch von der vorherigen Saros/I-Ressourcenverwaltung verwendet wurden, aus den Schnittstellen entfernt (Patch #3290, #3292 und #3296).

[28] Äußeres Verhalten bedeutet hierbei das Verhalten, das von dem Aufrufer der Funktion beobachtet werden kann.

Während der Reimplementierung wurde im Saros-Team der Entschluss gefasst, dass eine Anpassung der Saros-Schnittstellen und eine Veränderung des Umgangs mir Ressourcen in Saros nötig ist. Die Grundlage dieser Entscheidung war die zu hohe Bindung der Saros-Schnittstellen an die ursprüngliche Eclipse-Implementierung. Durch diese hohe Bindung konnte keine gute Abstraktion der Saros-Logik geschaffen werden, was bei der Implementierung von Saros für andere IDEs zu Problemen führte, da die Schnittstellen oft nicht auf die lokalen Strukturen anwendbar war.

Dieser Entschluss beeinflusste ebenfalls das Vorgehen, da ich die Grundlagen der Reimplementierung mit dem Gedanken der zukünftigen Erweiterbarkeit bzw. Aufhebung der Einschränkungen der Alpha-Version geschaffen hatte. Auf diese Erweiterbarkeit habe ich nach dieser Entscheidung einen geringeren Wert gelegt.

Zusätzlich habe ich keine Tests für die neuen Klassen geschrieben, da...

- hierfür die notwendige Zeit im Rahmen der Bachelorarbeit nicht gegeben war,
- ich durch den Charakter als Adapter und die damit erzeugte, starke Abhängigkeit von Kernkomponenten der IntelliJ-Umgebung Probleme bei der Erstellung von Testobjekten hatte und
- die Notwendigkeit von Testfällen durch die absehbare Lebensdauer der Implementierung gemindert wurde.

9.4 Umsetzen der Reimplementierung

Um das unnötige Neu-Erstellen von bereits bestehenden Funktionalitäten zu vermeiden, habe ich die Reimplementierung stark auf der IntelliJ OpenAPI[29] basiert. Dies führte dazu, dass die neue Implementierung fast ausschließlich den Charakter eines Adapters hat, da sie keinen eigenen Zustand hält und größtenteils nach leichten Anpassungen mit der Eingabe andere Funktionen der IntelliJ-API aufruft.

Durch diesen Wechsel des grundlegenden Designcharakters der Implementierungen wurde neben der Reimplementierung der IProject-Schnittstelle auch eine Reimplementierung der restlichen Ressourcen-Schnittstellen (IRessource, IFolder und IFile) nötig. All dies zog auch eine Anpassung vieler umliegender Saros/I-Komponenten nach sich.

Der Wechsel auf ein Adapter-Modell macht die Implementierungen um einiges stabiler und reduziert das Potential für mögliche Versagen. Er erzeugt jedoch auch einige Probleme, wenn die zu implementierende Funktionalität nicht auf die unterliegende IntelliJ-Struktur anwendbar ist. Ein Beispiel

[29] http://www.jetbrains.org/intellij/sdk/docs/index.html

hierfür ist der bereits angesprochene Umgang mit im lokalen Dateisystem nicht-existenten Ressourcen.

Um dieses Problem der Verweise auf nicht-existente Ressourcen zu beheben, habe ich zwei Implementierungen der Schnittstellen IRessource, IFile und IFolder erstellt. Ein UML-Diagramm der beteiligten Schnittstellen sowie ihrer Implementierungen kann in der Abbildung 2, eine größere Version in Abschnitt A.5 in Abbildung 3b betrachtet werden.

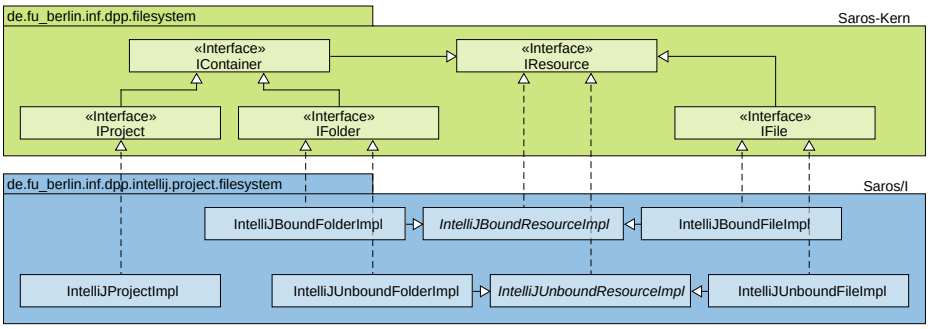


Abbildung 2: Aufbau der angepassten Saros/I-Ressourcenverwaltung.

Ein Satz an Implementierungen ist „gebunden“, was bedeutet, dass Objekte dieses Typs nur erstellt werden können, wenn es eine zugehörige Ressource im lokalen Dateisystem gibt. In Saros/I habe ich fortlaufend nur diese Implementierungen verwendet, da sie das Verhalten der IntelliJ-Dateisystemanbindung widerspiegelt.

Zusätzlich habe ich noch „ungebundene“ Implementierungen erstellt, um die vom Saros-Kern benötigte Funktionalität bereitstellen zu können. Diese Implementierungen sind unabhängig von dem Zustand des lokalen Dateisystems und damit äquivalent zu den Saros/E-Implementierungen.

Von dieser Unterscheidung kann jedoch in der normalen Verwendung der Schnittellen abstrahiert werden, da Ressourcen-Objekte über Methodenaufrufe von der IntelliJProjectImpl abgefragt werden können. Diese prüft, ob die Ressource im lokalen Dateisystem vorhanden ist und gibt entsprechend ein gebundenes oder ungebundenes Ressourcen-Objekt zurück.

Details zu der Umsetzung können im Gerrit Patch #3310 eingesehen werden.

9.5 Abschluss der Reimplementierung

Bei der auf die Reimplementierung folgenden Anpassung der umliegenden Saros/I-Komponenten wurde vorerst auf eine Anpassung des Umgangs mit

Änderungen der geteilten Dateistruktur (also dem Erstellen, Löschen, Verschieben und Umbenennen von Dateien) verzichtet, da diese Funktionalitäten in der vorherigen Version größtenteils nicht gegeben waren und es somit gut möglich ist diese Änderungen in einem separaten Patch anzufertigen, ohne dabei in der Zwischenzeit an Funktionalität zu verlieren.

Die Reimplementierung wurde von mir im Rahmen dieser Arbeit vollendet und über Gerrit von anderen Mitgliedern des Saros-Teams begutachtet. Bei der Integration in die bestehende Codebasis gab es jedoch Probleme, da der Patch durch seine weitreichende Auswirkung sehr groß und schlecht teilbar wurde.

Durch die Reimplementierung wurde die Ressourcenverwaltung an die Unterliegende IntelliJ-Funktionalität angepasst, was eine zukünftige Arbeit an dem Saros/I-Plugin erleichtern sollte. Der genaue Funktionsumfang von Saros/I nach dieser Reimplementierung wird in Abschnitt 10.1 zusammengefasst.

10 Evaluation

Während dieser Arbeit habe ich aufgrund von unvorhergesehenen Komplikationen und dem daraus resultierenden Zeitmangel keinen der in Abschnitt 5 aufgestellten Meilensteine erreicht. Ich habe jedoch einen erheblichen Anteil des ersten Meilensteins bewältigt und, durch die Schaffung einer besseren Codebasis, sowie der Dokumentation der zu behebenden Versagen, eine gute Grundlage für die Bearbeitung der noch offenen Versagen geschaffen. Dies sollte den Arbeitsaufwand bei der Behebung der noch offenen Versagen deutlich verringern und die zukünftige Arbeit an Saros/I erleichtern.

Die Abweichung von dem ursprünglichen Zeitplan kann auf die unvorhergesehenen Mängel zurückgeführt werden. Die Arbeit lief, nach anfänglichen Schwierigkeiten bei der Einarbeitung, bis zu dem Umbruch im Vorgehen größtenteils planmäßig ab. Die Reimplementierung der Ressourcenverwaltung war jedoch komplexer als ursprünglich angenommen und nahm erheblich mehr Zeit in Anspruch als zu Beginn der Reimplementierung geplant. Es wurde mehr als ein Drittel der gesamten Bearbeitungszeit hierfür verwendet.

10.1 Behobene Versagen

Durch die Einarbeitung sowie das iterative Vorgehen habe ich folgende Versagen behoben:

#868 IntelliJ Modules are not shared correctly
https://sourceforge.net/p/dpp/bugs/868/

#872 Intellij Synchronisation
https://sourceforge.net/p/dpp/bugs/872/

Hierdurch wurde das synchrone Arbeiten an geteilten Dateien ermöglicht, solange bei der Initialisierung der Sitzung keine Dateien übertragen werden mussten und nur auf Dateien gearbeitet wurde, die bei allen Teilnehmern geöffnet waren. Das Umbenennen von Dateien wurde auch kurzzeitig ermöglicht, diese Funktionalität wurde jedoch nach der Reimplementierung der Ressourcenverwaltung noch nicht wieder hergestellt.

Durch die Reimplementierung der Ressourcenverwaltung habe ich folgende erkannte Versagen behoben:

#890 Synchronization of activities does not work on closed files
https://sourceforge.net/p/dpp/bugs/890/

#894 Saros/I sessions do not work if files were transmitted during initialization
https://sourceforge.net/p/dpp/bugs/894/

#896 Open files not belonging to the shared module cause IllegalArgumentExceptions
https://sourceforge.net/p/dpp/bugs/896/

Hierdurch wurde das uneingeschränkte synchrone Arbeiten an geteilten Dateien ermöglicht. Es wurden die Einschränkungen bei Sitzungen, welche als Teil der Sitzungserstellung Dateien übertragen haben, aufgehoben. Das Arbeiten auf geteilten Dateien, welche nicht bei allen Teilnehmern geöffnet sind, wurde ermöglicht. Zusätzlich wurde das Öffnen von Dateien, die nicht zum geteilten Modul gehören, erlaubt. Es ist jedoch immer noch nicht möglich, geteilte Dateien während einer Sitzung zu Erstellen, zu Löschen oder zu Verschieben. Des Weiteren wurde im Rahmen der Reimplementierung die Funktionalität des Umbenennens von geteilten Dateien während einer Sitzung, welche durch die zweite Iteration ermöglicht wurde, nicht wieder hergestellt.

10.2 Nächste Schritte

Da im Rahmen dieser Arbeit die Integration der Reimplementierung der Ressourcenverwaltung nicht abgeschlossen wurde, wäre dies der nächste Schritt in der Arbeit an Saros/I. Anschließend sollten die verbliebenen Versagen behoben werden, da hiernach der erste Meilenstein erreicht wäre.

Die noch zu bearbeitenden Versagen sind:

#889 Files created during a session are transmitted empty
https://sourceforge.net/p/dpp/bugs/889/

#891 Changes to shared files during session initialization lead to wrong session behavior
https://sourceforge.net/p/dpp/bugs/891/

#897 Deleting a file during a session leads to the client becoming unsynchroized
https://sourceforge.net/p/dpp/bugs/897/

Des Weiteren müssten auch noch die restlichen Anpassungsarten der geteilten Dateistruktur (das Erstellen, Löschen, Umbenennen und Verschieben von Dateien und Ordnern) betrachtet und an die Reimplementierung der Ressourcenverwaltung angepasst werden.

11 Ausblick

Unabhängig von den in absehbarer Zukunft stattfindenden Änderungen an der Saros-Ressourcen-Struktur und einer damit einhergehenden Reimplementierung bzw. Anpassung von Teilen des Codes ist es sinnvoll, die Arbeit an der momentanen Version von Saros/I fortzusetzen.

Es sollte weiterhin daran gearbeitet werden eine Alpha-Version zu erstellen, da mithilfe dieser eine Nutzerstudie durchgeführt werden kann, um die Anforderungen an Saros/I zu präzisieren. Dieses Wissen kann dann in die während der Anpassung der Saros-Struktur getroffenen Entscheidungen mit einfließen.

Da in dem Zeitraum dieser Bachelorarbeit eine weitere Arbeit begonnen wurde, die sich ausschließlich mit der Saros-HTML-GUI beschäftigt, wird der zweite Meilenstein, welcher sich mit eben dieser auseinander setzte, hierdurch erledigt. Falls dies nicht im Rahmen der Arbeit geschieht, muss noch die Integration der HTML-GUI in Saros/I vorgenommen werden.

Danach kann der dritte Meilenstein, die Veröffentlichung der Alpha-Version, angegangen werden.

11.1 Mögliche Erweiterungen

Da die Alpha-Version nur einen eingeschränkten Satz an Funktionen besitzen wird, könnte dieser noch erweitert werden, um ein Äquivalent zu der in Saros/E enthaltenen Funktionalität zu schaffen. Hierzu müssten die in Abschnitt 6 aufgelisteten „nicht betrachteten Funktionalitäten“ implementiert werden.

Zusätzlich müssten einige fehlende Funktionalitäten zur Erhöhung der Nutzbarkeit, wie z. B. die Auswahl der zu verwendenden Farbe für die Änderungen bzw. Textmarkierungen anderer Sitzungsteilnehmer, implementiert werden.

12 Verbesserungsvorschlag zum Saros-Projekt

Die Einarbeitung in das Saros-Projekt ist sehr aufwändig. Um effektiv arbeiten zu können, wird ein Vorwissen aus vielen unterschiedlichen Bereichen benötigt. Das Erlangen dieses Wissens im Bezug auf Aspekte des Saros-Projekts ist verbesserungswürdig. Hierfür gibt es verschiedene Faktoren:

- Das Saros-Wiki ist nicht vollständig und die Richtigkeit sowie Aktualität der Inhalte kann nicht garantiert werden. Deshalb ist es schwer einzuschätzen, wie verlässlich die aufgeführten Informationen sind.
- Die Codebasis ist nicht durchgehend bzw. ausreichend dokumentiert. Dies ist vor allem problematisch bei Annahmen, die von dem Saros-Kern oder der jeweils zugrundeliegenden IDE getroffen, jedoch nicht ausreichend dokumentiert wurden. Das Übersehen solcher Annahmen des Kerns und der IntelliJ-IDE führte zu den Defekten, welche die in Abschnitt 9 beschriebene Reimplementierung der Ressourcenveraltung nötig machten.

Ein Großteil dieses Problems würde durch die Aktualisierung und bessere Instandhaltung des Saros-Wiki, sowie der Dokumentation der Codebasis gelöst werden. Eine generelle Verbesserung dieser Dokumentationen würde auch deren Instandhaltung bei zukünftigen Änderungen erleichtern, da eine gute Dokumentation dazu motiviert, diesen Zustand auch aufrecht zu erhalten.

Eine generelle Überarbeitung sowie Neugestaltung des Saros-Wikis könnte hierbei als Teil einer Abschlussarbeit an der Freien Universität Berlin vorgenommen werden. Um dieses Wiki danach instand zu halten, wäre es vorteilhaft, wenn die Anpassung des Wikis in den normalen Arbeitsablauf integriert werden würde. Dies würde bedeuten, dass als Teil eines jeden Patches vom Autor, wenn nötig, eine Anpassung des Saros-Wikis vorgenommen wird.

Ein ähnlicher Ansatz könnte auch für die Dokumentation des Quellcodes gewählt werden: Die Dokumentation jeder nicht-trivialen, angepassten Klasse oder Methode muss für den erfolgreichen Abschluss eines Code-Reviews vorhanden sein.

Eine solche Vorgabe existiert zwar schon in den Saros-Coderegeln [36], sie wird jedoch nicht umgesetzt. Wie dies zeigt, ist das Erzwingen dieser Regeln, sowohl bei der Code-Dokumentation wie auch der Anpassung des Saros-Wiki bei jedem Patch, ein sehr idealisierter Ansatz und keine wirkliche Lösung. Dies wird vor allem bei externen Entwicklern deutlich, da hier keinerlei Abhängigkeiten, wie es sie z. B. bei einer Abschlussarbeit gibt, bestehen, welche einen Grundsatz für das Erzwingen dieser Regeln bieten würden.

Der Einzige für mich ersichtliche Weg ist, die Mitarbeiter bzw. Studenten davon zu überzeugen, dass eine gute Dokumentation auch für sie wichtig ist und einen enormen Nutzen haben kann, und sie somit den Mehraufwand wert ist. Dies sollte gut vermittelbar sein, da alle neuen Mitarbeiter bzw. Studenten auch den Prozess der Einarbeitung durchlaufen müssen und somit, jedenfalls bei dem momentanen Stand, selbst auf einige Probleme einer mangelhaften Dokumentation gestoßen sein sollten.

A Anhang

A.1 Vollendete Patches

#3177 [FIX][I] #868 Modules were shared as Directories
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3177/

#3215 [FIX][I] NullPointerException at the start of a Saros/I-Session
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3215/

#3216 [NOP][I] Amend .gitignore and clean up codeStyleSettings.xml
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3216/

#3217 [API] Remove getRemotelyOpenEditors from IEditorManager API
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3217/

#3240 [FIX][I] #872 Fix synchronization issues
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3240/

#3241 [INTERNAL][I] Change implementation of ProjectAPI#openEditor()
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3241/

#3242 [INTERNAL][I] Move initialization of open editors to #resourcesAdded(..)
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3242/

#3251 [INTERNAL][I] Use project objects held by session
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3251/

#3252 [FIX][I] Fix renaming of shared files
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3252/

#3253 [FIX][I] Add document to editorPool during replaceAll(...)
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3253/

#3290 [API] removes IResourceAttributes and its usage
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3290/

#3292 [API] remove IResource#isAccessible()
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3292/

#3296 [API] remove IResource#refreshLocal()
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3296/

#3297 [INTERNAL][I] remove IntelliJProjectImpl#create() and its usage
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3297/

A.2 Offene Patches

#3310 [INTERNAL][I] Overhaul of the handling of resources and modules
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3310/

A.3 Verworfene Patches

#3289 [API] remove unused method IResource#getAdapter(...)
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3289/

#3295 [INTERNAL][CORE] add SarosPluginContext#getComponent(...)
http://saros-build.imp.fu-berlin.de/gerrit/#/c/3295/

A.4 Erstellte Bugtracker-Einträge

#868 IntelliJ Modules are not shared correctly
https://sourceforge.net/p/dpp/bugs/868/

#872 Intellij Synchronisation
https://sourceforge.net/p/dpp/bugs/872/

#885 Building the Saros/I plugin as a zip fails
https://sourceforge.net/p/dpp/bugs/885/

#888 FileSystemChangeListener creates new IntelliJProjectImpl Objects
https://sourceforge.net/p/dpp/bugs/888/

#889 Files created during a session are transmitted empty
https://sourceforge.net/p/dpp/bugs/889/

#890 Synchronization of activities does not work on closed files
https://sourceforge.net/p/dpp/bugs/890/

#891 Changes to shared files during session initialization lead to wrong session behavior
https://sourceforge.net/p/dpp/bugs/891/

#892 EditorManager#addProject() does not work during session initialization
https://sourceforge.net/p/dpp/bugs/892/

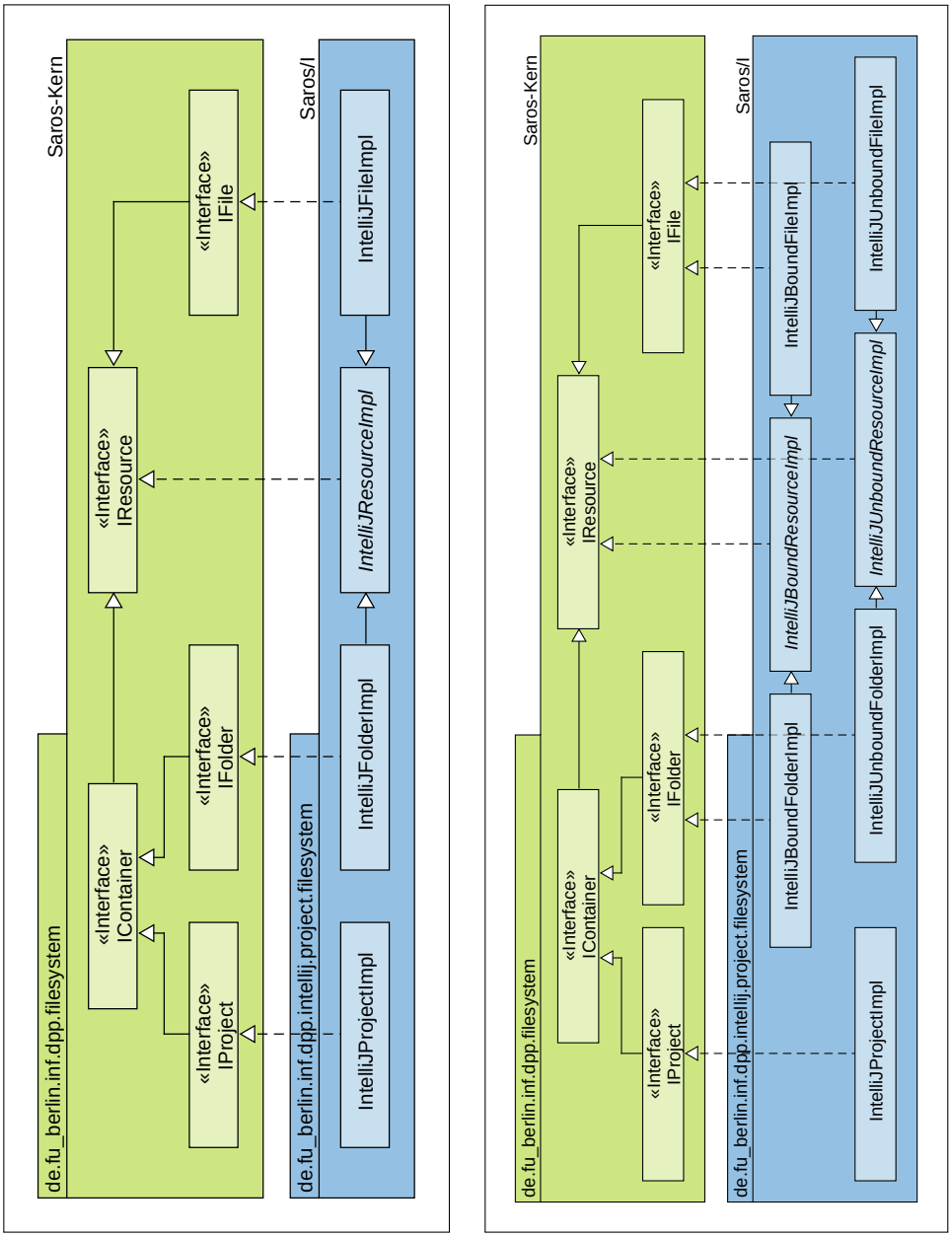
#893 Saros/ I reads keyboard input in menus
https://sourceforge.net/p/dpp/bugs/893/

#894 Saros/I sessions do not work if files were transmitted during initialization
https://sourceforge.net/p/dpp/bugs/894/

#896 Open files not belonging to the shared module cause IllegalArgumentExceptions
https://sourceforge.net/p/dpp/bugs/896/

#897 Deleting a file during a session leads to the client becoming unsynchroized
https://sourceforge.net/p/dpp/bugs/897/

A.5 UML-Diagramme



(a) vor der Reimplementierung (b) nach der Reimplementierung

Abbildung 3: Aufbau der Saros/I-Ressourcenverwaltung.

Literatur

[1] A. Cockburn and L. Williams, "The Costs and Benefits of Pair Programming," in *eXtreme Programming and Flexible Processes in Software Engineering XP2000*. Addison-Wesley, 2000, pp. 223–247. [Online]. Available: https://collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF

[2] L. Williams. (2006) Pair Programming. [Online]. Available: https://collaboration.csc.ncsu.edu/laurie/pair.html

[3] D. Stotts, L. Williams, N. Nagappan, P. Baheti, D. Jen, and A. Jackson, *Virtual Teaming: Experiments and Experiences with Distributed Pair Programming*. Springer, 2003, pp. 129–141. [Online]. Available: https://collaboration.csc.ncsu.edu/laurie/Papers/XPAUDistributedP.pdf

[4] C.-W. Ho, S. Raha, E. Gehringer, and L. Williams, "Sangam: A Distributed Pair Programming Plug-in for Eclipse," in *Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange*, ser. eclipse '04. New York, NY, USA: ACM, 2004, pp. 73–77. [Online]. Available: https://collaboration.csc.ncsu.edu/laurie/Papers/Sangam.pdf

[5] J. Schenk, L. Prechelt, and 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*, ser. ICSE Companion 2014. New York, NY, USA: ACM, 2014, pp. 74–83. [Online]. Available: http://doi.acm.org/10.1145/2591062.2591188

[6] E. Rosen, "Bewertung verteilter Paarprogrammierung im betrieblichen Umfeld," Diplomarbeit, Freie Universität Berlin, 2009. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-se/theses/Rosen09-DPP-management.pdf

[7] P. Baheti, L. Williams, E. Gehringer, D. Stotts, and J. M. Smith, "Distributed Pair Programming: Empirical Studies and Supporting Environments," Department of Computer Science North Carolina State University and Department of Computer Science University of North Carolina, Tech. Rep., 2002. [Online]. Available: https://collaboration.csc.ncsu.edu/laurie/Papers/TR02-010.pdf

[8] R. Djemili, "Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierungverteilter Paarprogrammierung," Diplomarbeit, Freie Universität Berlin, Inst. für Informatik, 2006. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-se/theses/Djemili06-eclipse-erweiterung-PP.pdf

[9] R. Djemili, C. Oezbek, and S. Salinger, "Saros: Eine Eclipse-Erweiterung zur verteilten Paarprogrammierung." in *Software Engineering (Workshops)*, ser. LNI, W.-G. Bleek, H. Schwentner, and H. Züllighoven, Eds., vol. 106. GI, 2007, pp. 317–320. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-se/pubs/saros-2007.pdf

[10] (2017) Saros Wiki - HTML-GUI. [Online]. Available: http://www.saros-project.org/html-gui

[11] (2014) Saros Wiki - Network Layer. [Online]. Available: http://www.saros-project.org/networklayer

[12] XMPP Overview. [Online]. Available: https://xmpp.org/about/technology-overview.html

[13] M. Fowler, "Inversion of Control Containers and the Dependency Injection pattern," 2004. [Online]. Available: http://www.martinfowler.com/articles/injection.html

[14] (2011) PicoContainer - Introduction. [Online]. Available: http://picocontainer.com/introduction.html

[15] (2017) Saros Wiki - Software Architecture. [Online]. Available: http://www.saros-project.org/specoverview#Software_Architecture

[16] (2016) Saros Wiki - Software Design Rules. [Online]. Available: http://www.saros-project.org/coderules#software_design_rules

[17] (2016) Saros Wiki - Error Reporting, Exception Hadling and Logging. [Online]. Available: http://www.saros-project.org/coderules#error_reporting_exception_handling_logging

[18] (2016) Saros Wiki - Reviews. [Online]. Available: http://www.saros-project.org/review

[19] A. Lasarzik, "Refaktorisierung des Eclipse-Plugins Saros für die Portierung auf andere IDEs," Bachelorarbeit, Freie Universität Berlin, 2015. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-se/theses/Lasarzik15-refaktorisierung.pdf

[20] D. Washington, "Entwicklung und Evaluation eines unabhängigen Sitzungsservers für das Saros-Projekt," Masterarbeit, Freie Universität Berlin, 2016. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-se/theses/Washington16-saros-server.pdf

[21] D. Sungaila, "Verbesserung und Erweiterung der Core-Bestandteile von Saros," Bachelorarbeit, Freie Universität Berlin, 2016. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-se/theses/Sungaila16-saros-core-verbesserung.pdf

[22] C. Cikryt, "Evaluating the Use of a Web Browser to Unify GUI Development for IDE Plug-ins," Masterarbeit, Freie Universität Berlin, 2015. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-se/theses/Cikryt15-saros-browser-as-ide-gui.pdf

[23] M. Bohnstedt, "Entwicklung einer IDE-unabhängigen Benutzeroberfläche fur Saros," Masterarbeit, Freie Universität Berlin, 2015. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-se/theses/Bohnstedt15-IDE-unabhaeng-benutzeroberflaeche.pdf

[24] B. Sieker, "User-Centered Development of a JavaScript and HTML-based GUI for Saros," Masterarbeit, Universität Paderborn, 2015. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-se/theses/Sieker15-HTML-GUI-Saros.pdf

[25] N. Weber, "Einstiegserleichterung für die Weiterentwicklung und Erweiterung der JavaScript- und HTML-GUI von Saros," Bachelorarbeit, Freie Universität Berlin, 2016. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-se/theses/Weber16-saros-html-gui-einstiegserleichterung.pdf

[26] L. Prechelt. (2017) Veranstaltung Softwaretechnik an der Freien Universität Berlin, Foliensatz 17 - Process Models. [Online]. Available: http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-SWT-2017/44_Prozessmodelle.pdf

[27] (2014) Saros Wiki - Guidelines. [Online]. Available: http://www.saros-project.org/guidelines

[28] (2017) Saros Wiki - Techdoc. [Online]. Available: http://www.saros-project.org/techdoc

[29] (2017) Saros Wiki - Arbeit an Saros/I. [Online]. Available: http://www.saros-project.org/saros-for-intellij/plan

[30] E. Girard, "Saros/I Backlog," 2016. [Online]. Available: https://docs.google.com/document/d/1Zg6Es9rlj4oROqpABKs-5D0IGmBpVWCaLCTlj2C___1c

[31] A. Jakobi, M. Borzechowski, and B. Sahre, "Saros/I Session Initiation Issues," 2016. [Online]. Available: https://docs.google.com/document/d/1ILqZ6LqDM9OJSNw4J0IFU9iB1r1EZFmPJODXInKGLlU

[32] (2017) Saros GitHub - IntelliJ - FileSystemChangeListener - Zeile 170. [Online]. Available: https://github.com/saros-project/saros/blob/master@{2017-05-15}/de.fu_berlin.inf.dpp.intellij/src/de/fu_berlin/inf/dpp/intellij/project/FileSystemChangeListener.java#L170

[33] H. Schmeisky. (2014) Gerrit Patch 1916 - FileSystemChangeListener - Zeile 189. [Online]. Available: http://saros-build.imp.fu-berlin.de/gerrit/#/c/1916/5/de.fu_berlin.inf.dpp.intellij/src/de/fu_berlin/inf/dpp/intellij/project/FileSystemChangeListener.java

[34] (2017) IntelliJ Platform SDK DevGuide - Virtual File System. [Online]. Available: http://www.jetbrains.org/intellij/sdk/docs/basics/virtual_file_system.html

[35] Eclipse Documentation - Package org.eclipse.core.resources. [Online]. Available: https://help.eclipse.org/kepler/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fcore%2Fresources%2Fpackage-summary.html

[36] (2016) Saros Wiki - Dokumentation. [Online]. Available: https://www.saros-project.org/coderules#documentation