Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,
Arbeitsgruppe Software Engineering

Überführung des Saros-Server in die Codebasis

Michael Krummrei

Matrikelnummer: 4135770

michael-krummrei@fu-berlin.de

Betreuer: Franz Zieris, Prof. Dr. Lutz Prechelt

Eingereicht bei: Prof. Dr. Lutz Prechelt, Prof. Dr. Margarita Esponda-Argüero

Berlin, 01.06.2017

Zusammenfassung

Das freie Entwicklerwerkzeug Saros für verteilte Kollaboration arbeitet nach einem hostbasierten Sitzungsmodell, in welchem die Sitzungen durch einen Benutzer, den Host der Sitzung, initiiert werden. Dieser lädt weitere Teilnehmer ein. In früheren Arbeiten wurden bereits Lösungen entworfen, um einen Betrieb nach einem serverbasierten Modell zu ermöglichen und so langlebigere Sitzungen zu ermöglichen. Diese Arbeit knüpft an diese Lösungen an, erläutert den Stand der Entwicklungen und beschreibt, neben dem allgemeinen Prozess der Integration von Änderungen, Maßnahmen, um den unabhängigen Sitzungsserver in die Codebasis von Saros zu überführen und so den Nutzern zur Verfügung zu stellen.

Eigenständigkeitserklärung

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.

Datum Unterschrift

Inhalt

Typografische Konvention:

`Maschinenschrift`

zur Markierung von Code, Ein- und Ausgabe von Programmen

Kursiv

für Abkürzungen, Fachbegriffe, Markenzeichen

1 Hintergrund und Motivation

1.1 Saros

Saros ist ein an der Freien Universität Berlin von der Arbeitsgruppe Software Engineering entwickeltes quelloffenes Werkzeug zur verteilten Paarprogrammierung und entstand 2006 zunächst als Idee Stephan Salingers und Christopher Özbeks. Riad Djemili griff diese auf und entwickelte während seiner Diplomarbeit [6] im Jahr 2006 das Saros Plug-in für die freie Entwicklungsumgebung Eclipse. Seitdem haben zahlreiche Studenten im Rahmen von Abschluss- und Studienarbeiten dessen Weiterentwicklung vorangetrieben und eine Reihe von Funktionen hinzugefügt.

Heute kann es zum Beispiel direkt in Eclipse über den *Marktplatz* oder über dessen Webseite als Drop-in heruntergeladen und installiert werden. Mit bis heute mehr als 34000 erfolgreichen Installationen belegt es aktuell den 108. Platz im Allzeit-Vergleich der dort verfügbaren Lösungen.[1] Alternativ kann auch über Sourceforge die aktuelle Variante des Drop-Ins als Zip-Archiv gepackt heruntergeladen werden.[2]

Unterstützt werden Sitzungen von bis zu 5 Teilnehmern, wobei sämtliche Änderungen an geteilten Projekten zwischen ihnen synchronisiert werden. Traditionell funktioniert das Plug-in nach einem hostbasierten Modell, d.h. der Initiator einer Sitzung fungiert als Host. Er teilt ein Projekt und lädt weitere Teilnehmer ein. Dabei wird bei jedem Teilnehmer eine lokale Kopie der Projektdaten erzeugt. Im Verlauf der Sitzung werden durchgeführte Änderungen eines Teilnehmers an dem Projekt vom Saros-Plugin erfasst, an die anderen Teilnehmer übertragen und dort reproduziert, wodurch der Projektzustand bei allen Teilnehmern konsistent gehalten wird.

Entwickelt wird Saros in der Programmiersprache *Java* und auch zur Ausführung wird mindestens *Java SE 6* vorausgesetzt.

1 http://marketplace.eclipse.org/metrics/successful_installs/alltime
2 http://www.saros-project.org/installation

1.2 Der unabhängige Sitzungsserver

Im Jahr 2013 begann Nils Bussas im Rahmen seiner Bachelorarbeit [1] damit, einen unabhängigen Sitzungsserver für Saros zu entwickeln und somit einen Betrieb nach einem serverbasierten Modell zu ermöglichen. In solch einem Modell fungiert der Server als Host der Sitzung und übernimmt das Sitzungsmanagement. Das Ergebnis dieser Arbeit war ein erster Prototyp, der die Möglichkeit bot, eine geteilte Sitzung zu erzeugen und Anfragen von Teilnehmer zum Beitritt bereits automatisch akzeptierte. Leider war aus Zeitgründen das Teilen von Projekten in diesem Entwicklungsstadium noch nicht implementiert.

Dies änderte sich in den Jahren 2015/16, als Denis Washington während seiner Masterarbeit [2] in Zusammenarbeit mit Ute Neise eine Möglichkeit für Nicht-Hosts Projekte zu Teilen entwarf und den Bussas-Prototypen um zahlreiche weitere Funktionen ergänzte. Der resultierende Prototyp arbeitet komplett unabhängig von Eclipse, unterstützt unter Anderem das Teilen von Projekten, sowie das kollaborative Bearbeiten von Dateien. Somit stellt er einen praktisch nutzbaren, unabhängigen Sitzungsserver dar, der zu Beginn dieser Arbeit zwar als Change-2929 im projekteigenen Review-System[3] vorliegt, jedoch in großen Teilen noch nicht in die Codebasis von Saros integriert ist.

3 http://saros-build.imp.fu-berlin.de/gerrit/#/c/2929

1.3 Zielsetzung

Im Rahmen dieser Arbeit soll die Integration des unabhängigen Sitzungsservers vorangetrieben werden. Das Ziel ist erreicht, wenn dieser den Reviewprozess erfolgreich absolviert hat, somit Teil der Codebasis von Saros ist und als praktisch nutzbare Funktion den Nutzern zur Verfügung steht.

1.4 Aufbau dieser Arbeit

Zunächst beschreibe ich die Integration von Änderungen betreffende Aspekte des Saros-Projekts, gebe dabei einen Überblick über Motivation und Hintergründe der Code-Durchsichten, des Prinzips der Kontinuierlichen Integration und zeige allgemein die Vorgehenswiese bei der Durchführung von Integrationen im Projekt. Im Anschluss gehe ich auf die konkrete Umsetzung im Fall des Saros-Servers ein. Dazu beleuchte ich die bisherige Geschichte der Entwicklung des Servers, beschreibe die grundsätzliche Funktionsweise des Entwurfs, gehe auf einige bereits bekannte Schwierigkeiten ein und zeige den Stand der Integration zu Beginn meiner Arbeit. Darauf folgen Beschreibungen der von mir durchgeführten Maßnahmen zur weiteren Integration der Komponenten. Den Abschluss bilden die Ergebnisse dieser Maßnahmen in Zahlen und Fakten, die Nennung eventueller erkannter neuer Schwierigkeiten, sowie möglicher Lösungsansätze dazu, und ein Ausblick in die Zukunft.

2 Grundlagen

2.1 Änderungen an Saros

Die Entwicklung von Saros erfolgt mittels kontinuierlicher Integration, einem von Kent Beck im Rahmen von *Extreme Programming* beschriebenen Prinzip der permanenten Integration.[4] Dabei werden neue Komponenten kontinuierlich mit einer bestehenden Anwendung zusammengeführt.

Eine der Grundlagen bei der Implementation dieses Konzepts bildet eine gemeinsame Codebasis, in welcher die Mitglieder einer Gruppe von Entwicklern ihre Änderungen veröffentlichen können. Bei Saros wird diese Basis mit dem freien, quelloffenen Versionskontrollsystem Git[5] verwaltet und über ein entsprechendes Repositorium auf GitHub[6] bereitgestellt. Durch diese Form der Bereitstellung wird, einem weiteren Grundsatz Fowlers folgend, auch Nicht-Entwicklern einfacher Zugriff auf die Ergebnisse der Entwicklung ermöglicht.

Zum Zwecke der analytischen, sowie konstruktiven Qualitätssicherung darf jedoch keine Änderung direkt in diese Basis übertragen werden, sondern erfordert zunächst manuelle Durchsichten durch andere Entwickler des Teams.[7] Diese Durchsichten werden *Reviews* genannt und basieren lose auf den 1976 von Michael Fangan [8] definierten „Code Inspektionen", einem strikten, formellen Prozess, bei dem der Code Zeile für Zeile betrachtet wird. Dadurch sollen unter anderem eventuelle Defekte gefunden werden. Diese können struktureller Natur, also Fehler bei der Implementation eines Entwurfs, sein oder auch auf konzeptionellen Fehlern beruhen, wenn also der Entwickler von falschen Annahmen bezüglich der Funktion einer zu ändernden bzw. der beabsichtigten Wirkungsweise einer neuen Komponente ausgegangen ist. Heutige Code Reviews werden auch „Modern Code Reviews" genannt, weil sie lockerer, also eher informell, durchgeführt werden.

Zur Durchführung dieser Durchsichten kommt im Saros-Projekt das quelloffene Review-System Gerrit zum Einsatz.[8]

4 https://www.martinfowler.com/articles/continuousIntegration.html
5 https://git-scm.com
6 https://github.com/saros-project/saros
7 http://www.saros-project.org/review
8 https://www.gerritcodereview.com/

2.2 Reviews und Gerrit

Ursprünglich während der Entwicklung des Android Betriebssystems durch Google begonnen und in Python geschrieben, ist Gerrit ab Version 2.0 in *Java EE* verfasst. Es wird zusammmen mit dem Git-Versionskontrollsystem benutzt und setzt auf diesem auf. Der Name selbst leitet sich vom dem niederländischen Architekten Gerrit Rietveld ab.

Statt, wie bei Git üblich, Änderungen mittels eines *Push*-Befehls direkt in die Codebasis zu schreiben, werden diese zunächst in Form sogenannter *„Changes"* an die Gerrit-Instanz[9] geschickt. Über eine Web-Oberfläche haben Entwickler dort die Möglichkeit, diese zu begutachten, zu Testzwecken herunterzuladen, Kommentare zu schreiben und schließlich zu bewerten. Die Bewertung erfolgt in Gerrit durch die Vergabe von Punkten zwischen „-1" wenn sie den Change ablehnen und „+1" wenn sie die Integration des Changes befürworten. Identifiziert werden solche Changes über fortlaufende Nummern. Im weiteren Verlauf dieser Arbeit wird an zahlreichen Stellen Bezug zu solchen *Changes* genommen.

Prinzipiell sind die Saros-Entwickler angehalten, sich bei der Benutzung des Reviewsystems je nach der Perspektive an bestimmte Konventionen zu halten.

Für Autoren von *Changes* stellt sich der Ablauf wie folgt dar:

1. Erstellung eines lokalen *Commit*, der die beabsichtigten Änderungen enthält
2. Initiierung des Reviewprozesses durch Senden (auch *„Push" genannt*) an die Saros Gerrit-Instanz
3. Abwarten der Ergebnisse der automatisierten Builds und Tests durch Jenkins
4. Einladung anderer Entwickler, den Change zu begutachten
5. Verständnis der durch die Reviewer gemeldeten Probleme
6. Umsetzung von Vorschlägen der Entwickler und Veröffentlichung in Form eines neuen Patches
7. Integration des Changes in die Basis, wenn mindestens zwei positive (+1 Punkte) und insbesondere keine negative Bewertung (-1 Punkt) vorliegt

Vor der Erstellung sollte der aktuelle Stand des master-branch abgefragt werden. Arbeitet man auf einer älteren Version, so haben sich eventuell relevante Teile der Basis geändert. Der Commit würde dann zunächst gut aussehen und könnte begutachtet werden, wäre allerdings nicht auf die Basis anwendbar. Dadurch müsste er auch bei positivem Ergebnis der Durchsicht abermals editiert werden um mit der Basis zusammengeführt werden zu können. Diese Änderung muss dann als neuer Patch in Gerrit veröffentlicht und abermals von den Reviewern

9 http://saros-build.imp.fu-berlin.de/gerrit/

bewertet werden, womit die ersten Durchsichten einen unnötigen Arbeitsaufwand darstellen und schlimmstenfalls die Bereitschaft der anderen Entwickler senken, sich Änderungsvorschläge anzuschauen.

Ein weiterer wichtiger Aspekt, welcher auch im Kontext dieser Arbeit eine hohe Bedeutung hat, ist der Umfang einzelner Änderungsvorschläge. Dieser sollte so gering wie möglich gehalten werden. Dadurch erfordert die Durchsicht weniger Zeit und es gibt weniger potentielle Fehlerquellen die eine erfolgreiche Integration verhindern können. Dies unterstützt den ebenfalls Martin Fowler definierten Grundsatz kontinuierlicher Integration, Integrationsintervalle so kurz wie möglich zu halten und so den gemeinsamen Arbeitsfortschritt der Entwickler in der Codebasis zu sichern. Auch sollten durch die Autoren möglichst zeitgleich zu jeder Änderung geeignete Tests zur Verfügung gestellt werden. Im Saros-Projekt findet dabei das JUnit-Framework[10], sowie das projekteigene Saros-Test-Framework[11] Verwendung. Letzteres wurde von Sandor Szuecs im Rahmen seiner Diplomarbeit eingeführt und durch Lin Chen 2011 während ihrer Diplomarbeit weiterentwickelt. STF-Tests ermöglichen die Ausführung diverser Anwendungsfälle des Saros-Plugins indem Benutzer, sowie deren Interaktionen mit der GUI der Anwendung simuliert werden. JUnit dient im Gegensatz dazu der automatisierten Überprüfung der Methoden einer Klasse selbst. In beiden Fällen werden für jeden Test geeignete Vor- und Nachbedingungen definiert und das jeweilige Testszenario dann unter Benutzung seiner Vorbedingung ausgeführt. Erfüllt das Resultat der Ausführung die Nachbedingungen, so gilt der Test als erfolgreich. Die Ergebnisse dieser Tests werden über einer grafische Oberfläche bereitgestellt, welche im Fall einer nicht erfolgreichen Ausführung auch eine Analyse der Ursachen von Fehlern ermöglichen.

Auch für die Reviewer gibt es im Saros-Projekt diverse Vorgaben, die es bei der Durchführung von Durchsichten zu beachten gilt.[12] Bezüglich des Änderungscodes im Detail sollen sie sicherstellen, dass

- er gut dokumentiert ist und die Dokumentation auch tatsächlich den Inhalt des Codes widerspiegelt
- er thread-sicher geschrieben ist
- sämtliche Ausnahmen durch Verweise auf nicht-existente Objekte (auch „NullPointerExceptions“) korrekt abgefangen werden
- alle Bezeichner einfach zu lesen und verständlich sind
- er entsprechend der im Projekt geltenden Coding-Konventionen[13] wohlformatiert ist
- sich implementierte Methoden gut in die Struktur der umgebenden Klasse einfügen

Aus einem etwas allgemeineren Blickwinkel gilt es außerdem zu überprüfen

10 http://www.junit.org
11 Im Folgenden „STF“ abgekürzt
12 http://www.saros-projekt.org/review
13 http://www.saros-project.org/coderules

- ob die beabsichtigte Änderung Sinn ergibt und der in der Commit-Nachricht angegebene Zweck auch erreicht wird
- ob eventuell andere Vorgaben des Saros-Projekts verletzt werden
- das behandelte Problem auch eleganter hätte gelöst werden können

Werden während der Durchsicht Probleme gefunden, so sollen diese nicht nur benannt, sondern in einer Form beschrieben werden, dass der Autor sie im Detail verstehen kann. Bestenfalls sollte ein Reviewer zu einem Problem auch gleich einen möglichen Lösungsansatz präsentieren oder wenigstens die seiner Ansicht nach geltenden Anforderungen an eine Lösung benennen. Generell sind sowohl Autoren als auch Reviewer angehalten im Fall von Verständnisschwierigkeiten oder Unklarheiten Fragen zu stellen bzw. dem Fragenden genügend Information zur Bewältigung dieser zur Verfügung zu stellen. Dies dient, in Anlehnung an den Sinn und Zweck gemeinsamer Code-Durchsichten, dem Wissensaustausch zwischen den Entwicklern, erhöht allseits das Verständnis über die Struktur des Entwicklungscodes, zeigt, besonders im Falle wiederkehrender Fragen, an welchen Stellen zusätzliche Dokumentation oder eine Vereinfachung des bestehenden Codes nötig sein kann und erhöht somit im Ergebnis die allgemeine Qualität der entwickelten Software.

2.3 Kontinuierliche Integration mit Jenkins

Weitere Grundsätze kontinuierlicher Integration wurden im Saros-Projekt durch die Einbindung des quelloffenen Automatisierungsservers Jenkins[14] umgesetzt. Dieser erfüllt eine Reihe von Aufgaben.

- Um Kompilierungsfehler zu vermeiden, erzeugt er aus der Git-Codebasis regelmäßig die gesamte Anwendung
- Es werden eine Reihe automatisierter Tests ausgeführt und dadurch sichergestellt, dass sich das erwartete Programmverhalten nicht geändert hat
- er führt mehrere, die Verteilung von Saros betreffende Aufgaben durch

Die Konfiguration der Saros-Instanz von Jenkins erfolgt über eine Weboberfläche.[15] Dort sind eine ganze Reihe von Projekten definiert. Im Kontext von Jenkins stellen Projekte eine Menge von automatisiert durchzuführenden Aufgaben dar. Die zu einem Jenkins-Projekt gehörenden Informationen werden in einem sogenannten Build gespeichert und sind, neben einer Historie über Builds der Vergangenheit, ebenfalls über die Weboberfläche einsehbar. Zur Funktionsweise von Jenkins im Detail sei auf die entsprechende Online-Dokumentation verwiesen.[16]

14 https://jenkins.io/
15 http://saros-build.imp.fu-berlin.de/jenkins/
16 https://jenkins.io/doc/

Für diese Ausarbeitung besonders interessant sind eine Reihe von Projekten, die nach dem Schema „Saros-*-Hourly“[17] benannt sind. Bei diesen wird jede Stunde die Codebasis von Git abgefragt und, sofern sich diese geändert hat, ein Build ausgelöst. Dieser startet im ersten Schritt ein Ant-Projekt, welches den gesamten Quellcode kompiliert. Danach werden eine Reihe weiterer Befehle ausgeführt, welche z.B. die Untersuchung durch *FindBugs* und *PMD* initiieren. Neben den bereits genannten wird jeden Tag um 6 Uhr morgens das Projekt „Saros-Eclipse-STF-Nightly“ gestartet, welches die STF-Testfälle ausführt.

Kommt es während der Ausführung der genannten Projekte zu Problemen, gilt dieser Build als „unstable“ und entsprechende Berichte werden an die Entwickler-Mailingliste von Saros geschickt.

Auch Gerrit stellt eine Schnittstelle für Jenkins zur Verfügung. Dadurch können einige der Jenkins-Projekte direkt auf Gerrit-Changes angewendet werden, und somit eventuelle Probleme bereits nach deren Veröffentlichung aufgedeckt werden und nicht erst nach erfolgter Integration in die Codebasis. Im Fall der Saros-Instanz von Gerrit löst jeder veröffentlichter Change eine Überprüfung durch Jenkins aus. Während dieser wird durch Jenkins, äquivalent zu den oben beschriebenen Projekten, der zu dem Change gehörende Code von Gerrit abgefragt und kompiliert. Jenkins nimmt dabei die Rolle eines Reviewers ein und führt als solcher auch eine Bewertung des Changes durch. Kann Jenkins die Kompilierung ohne Probleme durchführen, so bewertet es den Change mit +1, anderenfalls mit -1. Das Ergebnis der Überprüfung wird in diesem Fall direkt in Gerrit als Kommentar zu dem jeweiligen Change bereitgestellt und enthält einen Link zu den detailierten Ausgaben der durchgeführten Überprüfungen in der Jenkins-Instanz.

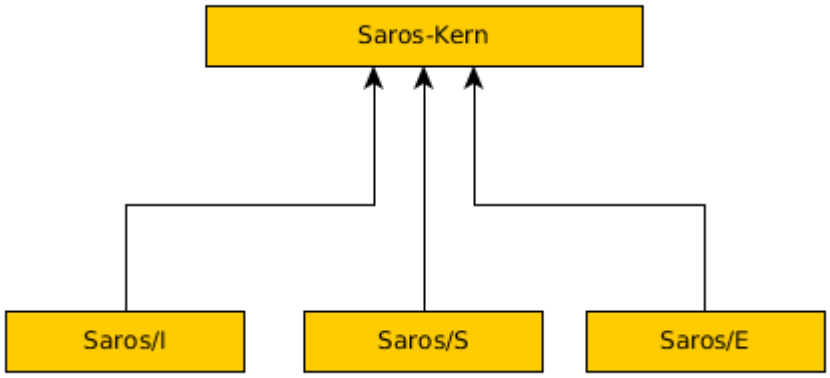
2.4 Organisation des Quellcodes

Während Saros ursprünglich als Plug-in für Eclipse entwickelt wurde, verteilt sich der gesamte Quellcode mittlerweile auf mehrere Teilbereiche. Dies begann 2014 zunächst mit der Bereinigung der Gesamtarchitektur durch die Aufteilung des Codes in den von Eclipse unabhängigen *Saros-Kern*, welcher das Datenmodell, die Funktionen der Netzwerkschicht und die Logik des Konkurrenz-Managements enthält, und einen Eclipse-spezifischen Teil, welcher lediglich die für die Steuerung der Plug-in-GUI nötigen Komponenten und jene zur Anbindung der Kern-Komponenten enthält. Dieser zweite Bereich wurde Saros-Eclipse bzw. kurz *Saros/E* getauft. Heute existieren einige weitere Bereiche die jeweils den Entwicklungscode bestimmter Saros-Features beinhalten. So wurde in *Saros/I* beispielsweise begonnen eine Saros-Variante für die integrierte Entwicklungsumgebung IntelliJ IDEA[18] zu entwickeln. Eine Übersicht über die verschiedenen Saros-Entwicklungszweige bietet die Webseite des Projekts.

Im Verlauf dieser Arbeit wird neben dem Saros-Kern (auch *Saros/C*, *Core* oder einfach *Kern*) insbesondere der Bereich Saros-Server oder kurz *Saros/S* häufige Erwähnung finden. Hier befindet sich der gesamte server-spezifische Quellcode. Den Zusammenhang zwischen den

17 „*“ dient hier als Platzhalter für den jeweiligen Bezeichner der einzelnen Projekte
18 https://www.jetbrains.com/idea/

genannten Bereichen soll der folgende Graph verdeutlichen, wobei die gerichteten Kanten Abhängigkeiten zwischen ihnen darstellen.



Saros/S ist unabhängig von dem in Saros/E oder Saros/I enthaltenen Code, benutzt jedoch die in den Komponenten des Kerns realisierte Logik.

2.5 Das Kommunikationsprotokoll

Für die Kommunikation zwischen den an einer Sitzung beteiligten, verteilten Instanzen von Saros wird auf Netzwerkschicht das Extensible Messaging and Presence Protokoll, oder kurz *XMPP*, verwendet. Dabei handelt es sich um einen, ursprünglich von der Internet Engineering Task Force unter dem Namen *Jabber* entwickelten, offenen Standard eines Kommunikationsprotokolls.

2.6 Sitzungsaufbau

Sobald ein Nutzer einen anderen Nutzer zu einer gemeinsamen Sitzung einlädt, beginnt die als *Session Negotiation* bezeichnete, erste Phase des Sitzungsaufbaus. Dabei sendet der einladende Nutzer, in der Rolle des Hosts, über das User Interface seiner Saros Instanz eine Einladung an den einzuladenden Nutzer, im folgenden Klient genannt. Im Hintergrund überprüft zunächst eine als *InvitationManagement* bezeichnete Komponente, ob der Klient überhaupt eine Saros-Instanz gestartet hat und ob diese mit der des Hosts kompatibel ist. Ist dies der Fall, so wird eine Einladungsnachricht an den Klienten geschickt. Stimmt er zu, so wird eine Bestätigungsnachricht an den Host geschickt. Danach tauschen sich beide Saros Instanzen über die vom Klienten präferierten Sitzungsparameter aus, der Host evaluiert diese, schickt dem Klienten die aktuellen Parameter, welcher lokal eine neue Sitzung erzeugt und mit einer finalen Bestätigungsnachricht an den Host antwortet. Auf Seiten des Hosts wird mit Eintreffen dieser Bestätigung die *SessionManagement*-Komponente über den Beitritt eines neuen Nutzers zur Sitzung informiert. Im zweiten Schritt findet das Teilen von Projekten, die

Project Negotiation, statt.

3 Zur Durchführung

Aus den im vorangegangenen Kapitel beschriebenen, am Saros-Integrationsprozess beteiligten Akteuren und deren Interaktionen ergeben sich somit die folgenden möglichen Anwendungsfälle bezogen auf die noch zu integrierenden Komponenten:

Komponente nicht in Durchsicht und Abhängigkeiten sind erfüllt

In diesem Fall gilt es, die Komponente nebst hinreichender Information in einem neuen Change auf Gerrit zu veröffentlichen und so den Reviewprozess einzuleiten.

Komponente besitzt offene Abhängigkeiten

Nicht erfüllte Abhängigkeiten zu anderen Komponenten müssen aufgelöst werden, bevor die Integration vorangetrieben werden kann. Ist dies nicht gestehen, kann die Komponente zwar dem Reviewprozess zugeführt werden, ein Test der Ausführung ist in diesem Zustand aber nicht möglich. Somit kann hier lediglich das Konzept des Entwurfs beurteilt oder manuelle Code-Durchsichten durchgeführt werden.

Change benötigt weitere Durchsichten

Es sind weitere Entwickler, z.B. durch Anschreiben per Email, einzuladen den Änderungsvorschlag zu begutachten.

Change hat negative Bewertung erhalten

Hier ist entsprechend der Ursache der negativen Bewertung vorzugehen, woraus einige weitere Anwendungsfälle folgen. Dazu gehören:

Verifizierung durch Jenkins fehlgeschlagen
Anhand der von Jenkins bereitgestellten Build-Information ist die genaue Quelle des Fehlschlags zu finden. Diese ist durch Bereitstellung eines Patches zu korrigieren und erfordert im Anschluss erneute Bewertung durch Jenkins und die Entwickler.

Reviewer hat Fragen oder Verständnisprobleme
Hier gilt es, durch Kommunikation miteinander, zunächst Sicherzustellen, dass die Fragen des Reviewers ausreichend Verstanden wurden und dann die entsprechenden, zu deren Klärung nötigen Informationen zu Vermitteln. Außerdem ist es sinnvoll, die Dokumentation des Codes entsprechend zu

ergänzen, um wiederkehrende Fragen im Laufe weiterer Durchsichten zu vermeiden. Gegebenenfalls ist auch hier ein neuer Patch zu erstellen und erneute Review durchzuführen.

ein konzeptioneller Fehler liegt vor
In diesem Fall sollen gemeinsam mögliche Lösungswege gesucht und in einem Patch umgesetzt werden.

die Implementierung weist Mängel/Fehler auf
Durch Erstellung eines Patches sind Mängel zu Beseitigen bzw. Fehler zu beheben → neue Review

Change bereit zur Überführung

Ein Gerrit-Benutzer, mit dem Recht *Submits* durchzuführen, ist zu informieren.

Anwendung auf die Codebasis nicht möglich

In diesem Fall liegt ein Konflikt durch eine seit Erstellung des Changes erfolgte Änderung an der Basis vor. Dieser kann nur manuell aufgelöst werden, indem die aktuelle Basis von Git angefragt und dann ein Patch zu dem Änderungsvorschlag erstellt wird, welcher die ursprünglich intendierte Änderung unter Benutzung der aktuellen Basis durchführt, aber auch ggf. bereits erhaltene Bewertungen verwirft und somit erneute Durchsichten, sowie eine Verifizierung durch Jenkins erfordert.

Integration erfolgt

Sofern es Komponenten gibt, deren Einführung von der gerade integrierten abhing, so ist für diese zu überprüfen, ob nun alle Abhängigkeiten erfüllt sind und ggf. der Integrationsprozess einzuleiten.

Bevor ich nun auf die Struktur des Servers mit all seinen Komponenten zu Sprechen komme, deren Status zu Beginn meiner Arbeit darlege und, darauf aufbauend die weitere Integration unter Benutzung der eben beschriebenen Anwendungsfälle durchführe, galt es zunächst einmal eine ausführliche Untersuchung des von Denis Washington produzierten Materials hinsichtlich Ausführbarkeit, unterstützter Funktionalität und eventueller Mängel durchzuführen.

4 Der unabhängige Sitzungsserver

Der Servercode liegt zu Beginn in Form des vierzehnten Patch-Sets eines als „Huge Server Patch" bezeichneten Changes in der Gerrit-Instanz von Saros vor.[19]

Den Einstiegspunkt zur Ausführung dieses Codes bildet die Klasse *SarosServer*. Da das Entwicklungsziel der Betrieb eines von den Entwicklungsumgebungen Eclipse und IntelliJ vollständig unabhängigen Servers ist, bietet es sich an, den gesamten Code in ein JAR-Archiv zu verpacken und die Ausführung über die Kommandozeile des Betriebssystems zu initiieren, wobei individuelle Konfigurationen für den Server in Form von Anweisungen an die virtuelle Maschine von Java mittels der Option **-D** übergeben werden können.[20] Ein solcher Aufruf sollte mindestens den zu benutzenden XMPP-Account, sowie das zugehörige Passwort enthalten und sieht wie in dem folgenden Beispiel einer Ausführung unter Linux aus:

```
$ java -Dde.fu_berlin.inf.dpp.server.jid=username@saros-con.imp.fu-berlin.de
-Dde.fu_berlin.inf.dpp.server.password=userpass
-Dde.fu_berlin.inf.dpp.server.workspace=${workspace_loc}/../workspace-server
-Djava.io.tmpdir=/home/username/tmp -jar sarosserver.jar
```

Neben den bereits erwähnten wurde hier zusätzlich das zu benutzende Arbeitsverzeichnis angegeben, wo zur Laufzeit die Daten der geteilten Projekte erzeugt werden. Die vierte Option definiert explizit das Verzeichnis für temporäre Dateioperationen und dient dazu eine Ausnahme zu vermeiden, welche während der ersten Evaluation des Servers aufgetreten ist. Diese ergab sich aus der von Washington beschriebenen Umsetzung der Dateisystem-Abstraktion des Saros-Kerns (siehe [2] Abschnitt 5.4.1) unter Benutzung der Dateisystem-Klassen der Java-Standartbibliothek j*ava.io*. Wird ein UNIX-Dateisystem verwendet, so muss das Arbeitsverzeichnis des Servers unter dem gleichen Einhängepunkt in das Dateisystem eingebunden sein, wie das von der *Java Virtual Machine* zu benutzende Verzeichnis für temporäre Dateioperationen. Ist dies nicht der Fall, so kann das Verschieben von Dateien nicht, wie von den Kern-Schnittstellen spezifiziert, atomar ausgeführt werden und es kommt während der Erstellung der lokalen Kopien von Dateien eines geteilten Projekts zu einer *AtomicMoveNotSupportedException*, was in der Folge zu einem Abbruch des Projektaustauschs führt und effektiv eine Benutzung des Servers zur kollaborativen Entwicklung verhindert.

Angewendet auf den Huge-Server-Patch von Denis Washington produziert der obige Befehl die folgende Ausgabe auf der Kommandozeile:

```
INFO  13:36:53,922 [main] (SarosContext.java:151) creating Saros runtime context...
DEBUG 13:36:54,002 [main] (XMPPAccountStore.java:118) loading accounts from file:
/home/username/.saros/config.dat
DEBUG 13:36:54,696 [main] (XMPPAccountStore.java:177) loaded 1 account(s)
INFO  13:36:55,047 [main] (SarosContext.java:194) successfully created Saros runtime
context
```

19 http://saros-build.imp.fu-berlin.de/gerrit/#/c/2929/14
20 Vergl.: Java MAN-Pages

```
DEBUG 13:36:55,084 [main] (DataTransferManager.java:467) used transport order for
the current XMPP connection: [XMPP-Socks5-Transport, XMPP-IBB-Transport]
Mai 03, 2017 1:36:55 PM org.jivesoftware.smack.provider.UrlProviderFileInitializer
initialize
INFORMATION: Loading providers for file [classpath:META-INF/core.providers]
Mai 03, 2017 1:36:55 PM org.jivesoftware.smack.provider.UrlProviderFileInitializer
initialize
INFORMATION: Loading providers for file [classpath:META-INF/extension.providers]
DEBUG 13:36:55,251 [main] (XMPPConnectionService.java:439) started Socks5 proxy on
port: 45614 [listening on all interfaces]
DEBUG 13:36:55,253 [main] (XMPPConnectionService.java:459) using autodetected
addresses: [192.168.178.32, fe80:0:0:0:1cd2:d0ec:5124:d848%wlp2s0]
DEBUG 13:36:55,259 [main] (XMPPConnectionService.java:398)  new connection state ==
CONNECTING
DEBUG 13:36:56,381 [main] (XMPPConnectionService.java:398) username@saros-
con.imp.fu-berlin.de/Saros new connection state == CONNECTED
DEBUG 13:36:56,382 [main] (DataTransferManager.java:467) used transport order for
the current XMPP connection: [XMPP-Socks5-Transport, XMPP-IBB-Transport]
INFO  13:36:56,387 [main] (ServerFeatureAdvertiser.java:50) Starting to advertise
ourselves as server
```

Zuerst wird der sogenannte *Kontext* erzeugt. Dies folgt einem als „Dependency Injection“[21] bezeichneten Muster der Softwareentwcklung. Im Anschluss dazu wird unter Nutzung des übergebenen Accounts eine XMPP-Verbindung aufgebaut und diesem Account eine Eigenschaft zugeordnet, welche es verknüpften Kontakten ermöglicht, zu erkennen, dass es sich hier um einen Sitzungsserver handelt.

Von diesem Zeitpunkt an ist die Initialisierung abgeschlossen und der Server wartet auf Sitzungsanfragen potentieller Nutzer. Im Anschluss habe ich mittels des Saros Eclipse Plugins und zweier anderer Jabber-Accounts solche Anfragen an die Server-Instanz geschickt. Diese wurden beide bestätigt und somit waren beide Plugin-Instanzen Mitglieder der geteilten Sitzung. Bei der weiteren Evaluation hat sich ergeben, dass der Funktionsumfang den Angaben Washingtons entspricht, d.h. Projektdaten werden zwischen den Sitzungsteilnehmern synchronisiert und auch das gleichzeitige Bearbeiten von Resourcen ist möglich, wobei den Benutzern der *Cursor* der beteiligten anderen Benutzer angezeigt und parallele Änderungen an den Inhalten konsistenzerhaltend synchronisiert werden.

Beendet werden kann die Ausführung lediglich durch ein Unterbrechen des Kommandos oder durch Schließen der Kommandozeile. Eine Methode zum kontrollierten Herunterfahren existiert bisher nicht.

4.1 Stand der Integration

Nachdem die Evaluation des Materials von D.Washington bezüglich der Ausführbarkeit abgeschlossen ist, habe ich mir einen Überblick über die Struktur seines Entwurfs, also der enthaltenen Komponenten sowie deren Abhängigkeiten zu anderen Elementen des Quellcodes, verschafft. Sehr hilfreich war hier die Erstellung eines entsprechenden Graphen, in welchem

21 Siehe [3]

die Knoten die einzelnen Komponenten und gerichtete Kanten die Abhängigkeiten darstellen. Eine Abbildung dieses Graphen befindet sich im Anhang dieses Dokuments.[22] Auch im weiteren Verlauf meiner Arbeit am Server diente dieser als sehr nützliche Orientierungshilfe auf dem Weg zur vollständigen Integration und wurde stets aktuell gehalten. Eine anschließende erste Überprüfung der enthaltenen Komponenten bezüglich des Integrationsstandes hat ergeben, dass 7 von ihnen bereits Teil der Saros Codebasis sind:

- ServerContainerImpl
- ServerFileImpl
- ServerFolderImpl
- ServerPathImpl
- ServerProjectImpl
- ServerResourceImpl
- ServerResourceAttributesImpl

Bei diesen handelt es sich um Implementierungen von Kern-Schnittstellen, welche der Abstraktion vom Dateisystem dienen. Für 6 von ihnen sind außerdem JUnit-Test implementiert und ebenfalls Teil der Codebasis. Der Übersicht halber wurden sie aus dem Graphen entfernt, ein Schritt, der mit jeder erfolgreich integrierten Komponente wiederholt wurde.

Wie man sieht, besitzen viele der enthaltenen Komponenten keine ausgehenden Abhängigkeiten und können gemäß den in Kapitel 3 ermittelten Anwendungsfällen dem Reviewprozess zugeführt werden.

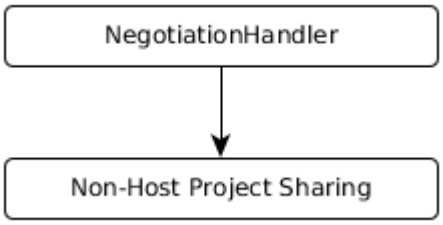
Aufgrund der sehr hohen Zahl an Abhängigkeiten der *ServerContextFactory* [23] erscheint diese zunächst als kritischer Knoten, da ja erst alle Abhängigkeiten aufgelöst werden müssten, bevor ihre Integration überhaupt begonnen werden kann. Das wiederum würde auch die Bereitstellung des Haupteinstiegspunktes des Servers (die Komponente „*Main Entry Point*“) stark verzögern, da dieser wiederum von der ContextFactory abhängt. Aufgrund von deren Funktionsweise kann diese Problematik jedoch elegant gelöst werden, indem sie zunächst in leerer Form integriert wird, also noch keine Anweisungen zur Einbindung von Komponenten enthält (bzw. nur jene der Komponenten, welche bereits Teil der Basis sind), und dann mit jeder Integration einer ihrer Abhängigkeiten die entsprechende Anweisung zur Einbindung der Komponente zur Kontextfabrik hinzugefügt wird. Zur Sicherstellung der Überprüfbarkeit ihrer Funktion, unabhängig von den eingebundenen Komponenten selbst, wurden von mir, einem Hinweis Franz Zieris‘ folgend, entsprechende JUnit-Tests bereitgestellt.

Somit verbleiben lediglich 2 Teile des Graphen, wo kritische Abhängigkeiten zwischen den Komponenten bestehen. Der erste beinhaltet den *NegotiationHandler* und seine Abhängigkeit

22 Siehe 6.3.1 „Komponenten des Huge-Server-Patch“

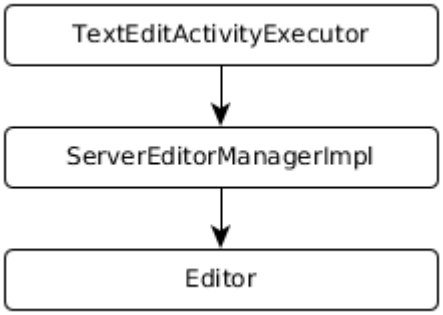
23 Diese wird im *Huge-Server-Patch* noch als *SarosServerContextFactory* aufgeführt, wurde von mir jedoch zusammen mit der *ServerSessionContextFactory*, in Anlehnung an, von Stefan Rossbach seit dessen Veröffentlichung durchgeführte, Refaktorisierungen, ebenfalls umbenannt.

zum *Non-Host Project Sharing.*



Die Wichtigkeit dieses neuen Protokolls zum Projektaustausch wurde durch Denis Washington bereits ausführlich in seiner Arbeit dargelegt.[24]

Der zweite Teilgraph besteht aus den Komponenten *TextEditActivityExecutor*, *EditorManager* und *Editor*.



Hier findet die Verwaltung von zur Bearbeitung geöffneten Textdateien während einer laufenden Sitzung statt.

Der restliche Graph ist somit nur noch eine Liste voneinander unabhängiger Komponenten, die, bezüglich der weiteren Integration, ebenso unabhängig gemäß den Anwendungsfällen in Kapitel 3 behandelt werden können.

4.2 Zeitlich bedingte Anpassungen

Seit der Veröffentlichung des Server-Patches haben eine Vielzahl von Änderungen an der Codebasis stattgefunden. Mein nächster Arbeitsschritt sollte also die Anpassung des Server-Codes an die geänderte Basis sein. Dies erwies sich als unerwartet zeitaufwendig, da es ein detailliertes Verständnis vieler beteiligter Saros-Komponenten, ihrer Funktionsweise, sowie insbesondere der durchgeführten Änderungen und deren Intention selbst voraussetzte.

So wurde die *VCSProviderFactory*-Schnittstelle aus dem Kern entfernt, womit die von Washington bereitgestellte Dummy-Implementierung, die *NullVCSProviderFactoryImpl*, nicht mehr benötigt wird.

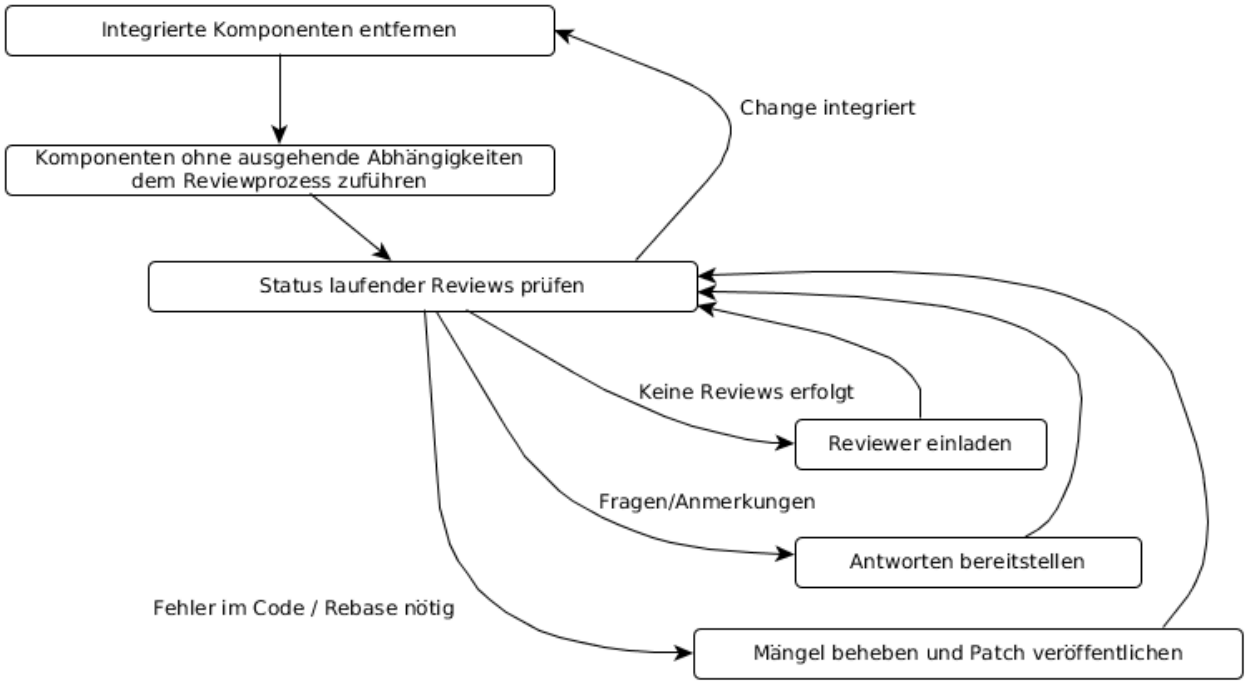
24 Siehe [2] Kapitel 2.

Die *SarosCoreContextFactory* dagegen wurde zwischenzeitlich in den Kern migriert, muss also ebenfalls nicht weiter betrachtet werden. Des Weiteren wurden diese Klasse, *SarosSession*, *SarosSessionManager*, sowie von diesen implementierte Schnittstellen im Kern umbenannt.

Auch der *MemoryPreferenceStore* ist nicht mehr notwendig. Im Kern liegt mittlerweile eine die *IPreferenceStore*-Schnittstelle implementierende Klasse vor, welche ohne Probleme vom Server benutzt werden kann.

4.3 Die weitere Vorgehensweise

Den ersten Schritt bei der weiteren Integration bildet die Entfernung aller bereits integrierter Komponenten aus dem bereits erwähnten Abhängigkeitsgraphen. Anhand des resultierenden Graphen werden im Anschluss aus den verbliebenen Komponenten jene als individuelle Changes dem Reviewprozess zugeführt, welche die Endknoten des Graphen bilden, also von keiner anderen Komponente des Graphen abhängen. Danach wird der Status jeder zum Servercode gehörenden, in Review befindlichen Änderungen überprüft und dementsprechende Maßnahmen durchgeführt. Sobald eine weitere Komponente in die Codebasis übertragen wurde, wiederholen sich die Schritte beginnend beim ersten. Das folgende Diagramm verdeutlicht den beschriebenen Ablauf:



Folgt man diesem Ablauf ist mit jedem Passieren des Schritts „Status laufender Reviews prüfen“ entweder die Zahl der in die Codebasis überführten Komponenten gestiegen oder die der offenen Probleme laufender Reviews gesunken. Enthält der Abhängigkeitsgraph keine

Knoten mehr, ist der Integrationsprozess abgeschlossen.

Bereits im ersten Durchlauf fanden sich so mehrere Komponenten, welche keine ausgehenden Äbhängigkeiten mehr aufwiesen und somit dem Reviewprozess zugeführt werden konnten. Diese seien im folgenden kurz beschrieben:

SubscriptionAuthorizer bestätigt automatisch beim Server eintreffende Anfragen der Nutzer, den Server in ihre Kontaktliste aufnehmen zu dürfen.

ServerFeatureAdvertiser gibt die laufende Saros-Instanz während des XMPP-Verbindungsaufbaus als Server bekannt. Dies erkennen mit dem XMPP-Account des Servers verbundene Instanzen des Saros Plug-ins, sofern die Unterstützung der Serverfunktionen in den Plug-in-Einstellungen aktiviert wurde, sobald sie die Features des Kontakts abfragen und bieten dann per Kontextmenü die Möglichkeit, den Server um eine Einladung zu einer geteilten Sitzung zu bitten.

Editor stellt während einer geteilten Sitzung einen Puffer für zur Bearbeitung geöffnete Textdateien zur Verfügung.

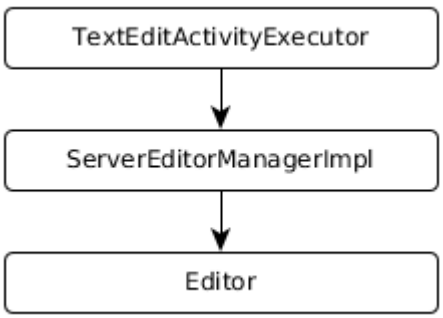
NullRemoteProgressIndicator bildet die Dummy-Implementierung einer weiteren Kern-Schnittstelle. Traditionell zeigen Saros-Instanzen ihrem Benutzer den Status von entfernt laufenden Aktionen anderer Instanzen auch lokal mittels eines Fortschrittsbalkens, sofern sie Teil der gleichen Sitzung sind. Dies ist offensichtlich auf Seiten einer Server-Instanz, mangels eines Benutzers, nicht notwendig.

JoinSessionRequestHandler behandelt eintreffende Beitrittsanfragen zur auf dem Server laufenden Sitzung.

ServerPreferences und *ServerPathFactoryImpl* stellen Implementierungen von Schnittstellen des Kerns bereit.

4.4 Ergebnisse

In Durchsicht, aber noch nicht Teil der Basis ist der aus den Klassen *TextEditActivityExecutor*, *ServerEditorManagerImpl* und *Editor* bestehende Teilgraph des Abhängkeitsgraphen.



Editor, also die Implementierung des Puffers für geteilte und zur Bearbeitung geöffnete Textdateien, bildet die Basis dieses Graphen und ist somit Voraussetzung für die beiden anderen Komponenten. Aktuell weist die Umsetzung dieser Kernschnittstelle jedoch eine schlechte Laufzeit auf. Der Aufwand pro Textänderung beträgt O(n) , weswegen durch die Reviewer eine Umsetzung unter Benutzung einer geeigneteren Datenstruktur gefordert wird.[25] Als Alternative wurde ein *Gap-Puffer* vorgeschlagen. Die Umsetzung dieses Vorschlags befindet sich im Entwurfsstadium, wurde aber noch nicht als Patch veröffentlicht.

Zugriff auf die Editoren einer laufenden Sitzung bietet der *EditorManager*. Dazu verfolgt er die lokal geöffneten, sowie die bei den entfernten Sitzungsteilnehmern geöffneten Editoren und informiert die entfernten bei Änderungen an den lokalen durch das Verschicken entsprechender Aktivitäten. Solche Änderungen sind zum Beispiel das Öffnen oder Schließen eines Editors und Änderungen des Inhalts der geöffneten Textdateien. Spezifiziert wird dieses Verhalten durch die *IEditorManager*-Schnittstelle des Kerns. Die Serverimplementierung dieser Schnittstelle, die Klasse *ServerEditorManagerImpl,* wurde von mir an die aktuelle Codebasis angepasst. Der resultierende Patch kann aktuell jedoch nicht von Jenkins verifiziert werden.[26] Ursächlich dafür sind Kompilierungsfehler des Servers während der Ausführung des Jenkins-Jobs *Saros-Server-Gerrit*. Demnach ist die Klasse nicht abstrakt implementiert und überschreibt die abstrakte Methode `getOpenEditors()` nicht.

In Folge dieses Problems schlägt auch die Verifizierung des *TextEditActivityExecutor*, des Konsumenten für empfangene Textbearbeitungsaktivitäten, fehl. Dieser Umstand sollte sich jedoch mit dessen Lösung und erfolgreicher Überführung des *EditorManagers* ebenfalls auflösen.

Insgesamt stellen die, diesen Teilgraph betreffenden, offenen Probleme keine großen Schwierigkeiten dar und sollten sich relativ leicht beheben lassen. Den größten dabei zu erwartenden Aufwand bildet die Implementierung eines alternativen Text-Puffers, da hier die neu zu schreibenden Methoden zum einen den größten Raum für Fehler bei der Implementierung bieten und andererseits die Praxistauglichkeit der gewählten Alternative bezüglich der Laufzeiten noch zu überprüfen wäre.

Auch die implementierten Konsumenten für Ordner- und Dateiaktivitäten, *FolderActivityExecutor* und *FileActivityExecutor*, sind weiterhin offen. Neben kleineren gewünschten Anpassungen am Quellcode beider Konsumenten ist noch eine Frage eines der anderen Entwickler offen, welche das Zusammenwirken beider Komponenten betrifft.[27] [28]

JoinSessionRequestHandler weist einen logischen Fehler auf. Genauer gesagt ist die Logik inkompatibel zu der des im Kern vorliegenden *SarosSessionManager*. Ein Aufruf von dessen `getSession`-Methode gibt unter Umständen nicht NULL zurück, obwohl die Session noch nicht initialisiert ist. Ausserdem wird für die Methoden `createSession()` und `inviteToExistingSession()` des Handlers empfohlen, diese durch 2 Thread-Pools zu

25 http://saros-build.imp.fu-berlin.de/gerrit/#/c/3262/2
26 http://saros-build.imp.fu-berlin.de/gerrit/#/c/3263/2
27 http://saros-build.imp.fu-berlin.de/gerrit/#/c/3266/1
28 http://saros-build.imp.fu-berlin.de/gerrit/#/c/3265/1

ersetzen.[29]

Ebenfalls noch nicht Teil der Basis ist die für die Realisierung des Serverbetriebs unablässige Umsetzung des *Non-Host Project Sharing*. Unter anderem fiel einem der Entwickler auf, dass der Schritt des Deaktivierens der Aktivitätswarteschlange in der vorhanden Implementierung Konsistenzprobleme aufweist.[30] Dies kommt zustande, da die Methode ursprünglich nicht entworfen wurde, um mit mehreren zeitgleich auftretenden *IncomingProjectNegotiations* umzugehen. Um dieses zu ändern, muss die Methode `disableQueuing()` so angepasst werden, dass sie als Argument ein *IProject* übergeben bekommt, also nicht, wie bisher, die Aktivitätswarteschlange für alle Projekte deaktiviert wird, sondern lediglich für das jeweils übergebene. Des weiteren hat sich während der laufenden Durchsichten abermals die Basis in einer Form geändert, welche zu Konflikten beim Zusammenführen führen würde. Die Lösung beider, dieses Feature betreffender Probleme befindet sich aktuell im Stadium eines Entwurfs und erfordert erneute Begutachtung und Bewertung.

Erst wenn diese erfolgt und mit positivem Ergebnis abgeschlossen sind und es somit auch Nicht-Hosts möglich ist, Projekte zu teilen kann der Integrationsprozess für den *ServerNegotiationHandler* begonnen werden. Dieser verarbeitet einkommende und ausgehende *Session*- und *ProjectNegotiations* indem er sie in jeweils eigenen Threads ausführt. Prinzipiell kann der Quellcode des Handlers zwar auch die aktuell in Saros vorhandene Variante des Projektsharing benutzen und somit durch die Entwickler begutachtet werden. Das Testen unter realen Bedingungen ist jedoch nicht möglich.

Der bereits im Januar 2015 von Ute Neise erstellte Änderungsvorschlag, welcher durch Ergänzungen der Eclipse-GUI den Zugriff auf den Server-Prototypen ermöglicht, wurde von mehreren Personen begutachtet und auch befürwortet. Da seitdem beinahe 2 Jahre vergangen sind, ist er im aktuellen Zustand jedoch nicht auf die Basis anwendbar und muss dementsprechend geändert und dann erneut begutachtet werden.[31]

Erfolgreich in die Codebasis übertragen wurden dagegen die Server-Implementierungen der beiden Kern-Schnittstellen *IWorkspace*[32] und *IPathFactory*[33]. Außerdem wurde die Implementierung der *IResource*-Schnittstelle um 2 Methoden ergänzt, von denen eine der Berechnung eines Hash-Wertes des Resource-Objekts dient und die andere überprüft, ob ein übergebenes Objekt gleich diesem ist.[34] Somit wurde sämtliche Dateisystemabstraktion des Kerns im Server umgesetzt. Mit *ServerPreferences*, *ServerUISynchronizerImpl*, *NullRemoteProgressIndicator* und der *NullRemoteProgressIndicatorFactory* konnten einige weitere Implementierungen von Schnittstellen des Kerns erfolgreich in die Basis übertragen werden.

Ebenfalls enthalten sind nun auch die Komponenten *ServerSessionContextFactory*[35] und

29 http://saros-build.imp.fu-berlin.de/gerrit/#/c/3273/1
30 http://saros-build.imp.fu-berlin.de/gerrit/#/c/2264/
31 http://saros-build.imp.fu-berlin.de/gerrit/#/c/2062/
32 http://saros-build.imp.fu-berlin.de/gerrit/#/c/2921/
33 http://saros-build.imp.fu-berlin.de/gerrit/#/c/3243/5
34 http://saros-build.imp.fu-berlin.de/gerrit/#/c/3261/4
35 http://saros-build.imp.fu-berlin.de/gerrit/#/c/3260/

ServerContextFactory.[36] Beide werden zwar im Rahmen der aktuell noch nicht abgeschlossenen Integration weiterer Komponenten noch Aktualisierungen erfahren müssen, jedoch handelt es sich dabei lediglich um das Ergänzen der Anweisung, die jeweilige Komponente zur Laufzeit dem Kontext hinzuzufügen.

Mit *ServerFeatureAdvertiser* und *SubscriptionAuthorizer* wurden zwei speziell für den Server entworfene Features erfolgreich in die Basis migriert.[37] [38]

Zu guter Letzt konnte die Integration der Komponenten *SarosServer* und *ServerConfig*, sowie einer Properties-Datei für *log4j* zum Abschluss gebracht werden.[39] [40] [41]

5 Zusammenfassung

Im Verlauf dieser Arbeit wurden 14 Komponenten des Saros-Servers als neue Änderungsvorschläge dem Reviewprozess zugeführt werden. Lediglich 1 Komponente befindet sich noch nicht in Durchsicht.

Insgesamt wurden 13 Änderungen der Codebasis erfolgreich abgeschlossen. Dabei wurden 12 Serverkomponenten integriert und 2 bereits vorhandene geändert. Bei 8 weiteren haben Durchsichten Schwierigkeiten aufgezeigt, die aktuell noch nicht gelöst werden konnten und in der Zukunft behandelt werden müssen. Die Anzahl der in diesem Zweig des Quellcodes vorhanden Klassen hat sich von 7 auf 19 erhöht.

Änderungen am Saros-Kern und Saros-Eclipse haben nicht stattgefunden. Für Beide liegt allerdings jeweils Änderungsvorschlag im Reviewsystem vor und auch hier wurden zu behandelnde Probleme aufgezeigt.

Eine Liste sämtlicher von mir behandelter Änderungen befindet sich im Anhang.[42]

Somit wurde die Integration des unabhängigen Sitzungsservers zwar effektiv vorangetrieben, das Ziel der vollständigen Überführung in die Codebasis wurde aber nicht erreicht.

Der nächste Schritt sollte unbedingt die Integration des *Non-Host Project Sharing* sein. Im Anschluss daran sind die den Server-Code selbst betreffenden Änderungen abzuschließen bzw. der noch nicht in Durchsicht befindliche *ServerNegotiationHandler* ist dem Integrationsprozess zuzuführen. Die niedrigste Priorität bei der weiteren Bearbeitung besitzt die Ergänzung der Server-GUI-Features.

36 http://saros-build.imp.fu-berlin.de/gerrit/#/c/3281/
37 http://saros-build.imp.fu-berlin.de/gerrit/#/c/3275/
38 http://saros-build.imp.fu-berlin.de/gerrit/#/c/3274/
39 http://saros-build.imp.fu-berlin.de/gerrit/#/c/3283
40 http://saros-build.imp.fu-berlin.de/gerrit/#/c/3280
41 http://saros-build.imp.fu-berlin.de/gerrit/#/c/3282
42 Siehe Kapitel 6.2 „Liste der Code-Änderungen“

6 Anhang

6.1 Literatur

[1] Nils Hauke Bussas, "Entwicklung eines Server-Prototypen für Saros", Bachelorarbeit, Freie Universität Berlin, 2014
[2] Denis Washington, "Entwicklung und Evaluation eines unabhängigen Sitzungsservers für das Saros-Projekt",Masterarbeit,Freie Universität Berlin,2016
[3] Martin Fowler, „Inversion of Control Containers and Dependency Injection pattern"
[4] Stefan Rossbach, „Einführung einer kontinuierlichen Integrationsumgebung und Verbesserung des Test-Frameworks", Bachelorarbeit, Freie Universität Berlin, 2011
[5] Dennis Ventzke, „Analyse, Entwurf und prototypische Implementierung einer Wissensdatenbank zur Teilautomatisierung des Code-Review-Prozesses im Saros-Projekt", Bachelorarbeit, Freie Universität Berlin, 2016
[6] Riad Djemili, „Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierung verteilter Paarprogrammmierung", Diplomarbeit, Freie Universität Berlin, 2006
[7] Arndt Lasarzik, „Refaktorisierung des Eclipse-Plugins Saros für die Portierung auf andere IDEs", Bachelorarbeit, Freie Universität Berlin, 2015
[8] M.E. Fagan, „Design and code inspections to reduce errors in program development", IBM Systems, 1976

6.2 Liste der Code-Änderungen

6.2.1 Integrierte Änderungen

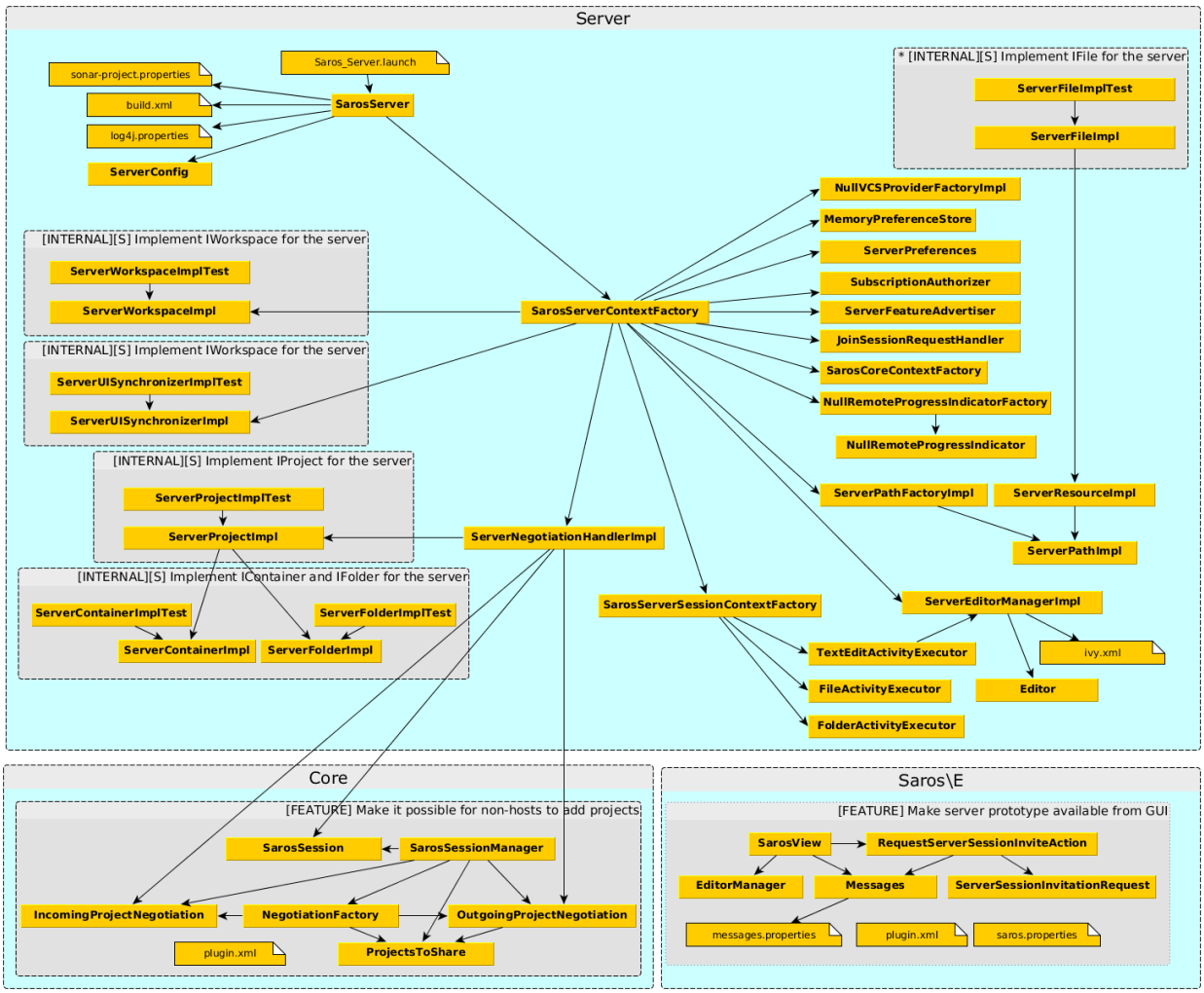
- 3274: [INTERNAL][S] Add SubscriptionAuthorizer
- 3276: [INTERNAL][S] Implementation of Preferences
- 3243: [INTERNAL][S] Implement IPathFactory for the Server
- 3260: [INTERNAL][S] Implement ISarosSessionContextFactory for the Server
- 2921: [INTERNAL][S] Implement IWorkspace for the Server
- 3278: [INTERNAL][S] Implement IRemoteProgressIndicator for the Server
- 3280: [INTERNAL][S] adds Server Configuration
- 3281: [INTERNAL][S] Adds ServerContextFactory
- 3282: [INTERNAL][S] adds log4j property file
- 3283: [INTERNAL][S] adds Server startup/shutdown logic – Main Entry Point
- 3261: [INTERNAL][S] ServerResourceImpl – add equals() and hashcode() methods
- 2940: [INTERNAL][S] Implement UISynchronizer for the Server
- 3275: [INTERNAL][S] Add ServerFeatureAdvertiser

6.2.2 Änderungen in Begutachtung

- 3262: [INTERNAL][S] Add a Buffer for shared Text Files
- 3265: [INTERNAL][S] Add Consumer for for FileActivities
- 3266. [INTERNAL][S] Add Consumer for FolderActivities
- 3264: [INTERNAL][S] Add Consumer for TextEditActivities
- 3263: [INTERNAL][S] Implement IEditorManager for the Server
- 3273: [INTERNAL][S] Add Handler for JoinSessionRequests
- 2264: [FEATURE] Make it possible for non-hosts to add projects
- 2062: [FEATURE] Make server prototype available from GUI

6.3 Abbildungen

6.3.1 Komponenten des Serverpatch



Legende: