Masterarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Software Engineering

Entwicklung einer IDE-unabhängigen Benutzeroberfläche für Saros

Matthias Bohnstedt
Matrikelnummer: 4202202
matthias.bohnstedt@gmail.com

Betreuer: Franz Zieris
Eingereicht bei: Prof. Dr. Prechelt

Berlin, 06.07.2015

Zusammenfassung

Das Saros-Plug-In wird sowohl für Eclipse als auch für IntelliJ entwickelt. Während beide Plug-Ins zwar auf eine gemeinsame Geschäftslogik zurückgreifen, wird ihr GUI jedoch noch getrennt voneinander entwickelt und gepflegt. Diese Arbeit beschreibt die Entwicklung einer IDE unabhängigen Benutzeroberfläche. Dabei wurden die negativen Implikation für die Evolvierbarkeit von Saros, die eine solch getrennte Oberflächenentwicklung mit sich bringt untersucht. Auf Basis des von Christian Cikryt begonnen Prototypen wurde die Benutzeroberfläche weiter entwickelt und den gegebenen Herausforderungen angepasst. Hierbei wurde die Anzeige des Session-Einladungsprozess von Saros IDE-unabhängig realisiert. In der neuen Benutzeroberfläche wurde die Schnittstelle zwischen Darstellung und Geschäftslogik so konzipiert, dass beide Bereiche unabhängiger voneinander entwickelt werden können. Dies ermöglicht das leichte Austauschen der Oberflächentechnologie. Da die Zusammenarbeit mit anderen Entwicklern hierbei ein wichtiger Faktor war, enthält diese Arbeit gewonnen Erkenntnisse und Handlungsempfehlungen für die effektive Arbeit in einem Team-Software-Projekt.

Inhaltsverzeichnis

1 Einleitung und Motivation

1.1 Saros

Das Saros-Plug-In wurde 2006 erstellt und seitdem kontinuierlich, meist im Rahmen von Abschlussarbeiten, weiterentwickelt. Zum besseren Verständnis und für tiefgreifende Informationen über das Saros-Plug-In empfiehlt sich das Lesen dieser Vorarbeiten, wobei Folgendes nur ein kleiner Auszug der wichtigsten Werke ist.

- Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierung verteilter Paarprogrammierung [1]
- Saros: Eine Eclipse-Erweiterung zur verteilten Paarprogrammierung. [2]
- Weiterentwicklung des Eclipse-Plug-Ins Saros zur verteilten Paarprogrammierung
- einen schnellen Einstieg bietet ebenfalls die Homepage des Projektes: http://www.saros-project.org/

1.2 Motivation: Saros für verschiedene IDEs

Saros, ursprünglich als reines Eclipse-Plug-In entworfen und konzipiert, soll im Zuge der Weiterentwicklung auch auf anderen IDEs [3] verfügbar werden. Es wird derzeit eine IntelliJ IDEA Version (Saros/I) erarbeitet. Saros auf mehreren Plattformen zu entwickeln stellt dessen Softwarearchitektur vor eine Reihe neuer Herausforderungen und Probleme. So müssen z.B. Aspekte der Software, die von Eclipse oder anderen IDEs abhängig sind, von Kernfunktionalitäten des Saros-Plug-Ins, wie der Netzwerkkommunikation, getrennt werden. Momentan existieren sowohl für Eclipse Version von Saros (Saros/E) als auch für Saros/I eine eigenständige Benutzeroberflächen. Damit ist gemeint, dass Saros/E und Saros/I im Bereich der Benutzeroberfläche keine gemeinsame Codebasis besitzen.
So werden beide GUIs momentan getrennt von einander entwickelt. Diese Implementierungen unterscheiden sich je nach IDE und eingesetzter Technologie. In Eclipse wird die UI standardmäßig mit SWT[1] umgesetzt, während in IDEA IntelliJ AWT[2] und SWING[3] in der GUI Entwicklung üblich sind.
"Wiederhole dich nicht" („Don't Repeat Yourself" - DRY) ist eine anerkannte Programmierpraxis. Das DRY-Prinzip wird hierbei sehr oft falsch verstanden, wie Dave Thomas und Andy Hunt in einem Interview erläutern.[4] So

[1] Standard Widget Toolkit https://www.eclipse.org/swt/
[2] Abstract Window Toolkit https://docs.oracle.com/javase/8/docs/technotes/guides/awt/index.html
[3] https://docs.oracle.com/javase/8/docs/technotes/guides/swing/index.html

wird '*das Vermeiden von Wiederholungen*' fälschlicherweise oft nur auf direkte Dopplung von Code angewendet. Tatsächlich verlangt das DRY ein breiteres Verständnis von *Wiederholung*: Jede Systemlogik und jedes Systemwissen sollte eine verbindliche, eindeutige und genau eine einzige Repräsentation haben. Dies umschließt dabei nicht nur Code, sondern bezieht sich auch auf Datenbankschemas, Testpläne, das Build-System und selbst auf die Dokumentation einer Software.

Warum ist die Einhaltung des DRY-Prinzip wichtig für die Entwicklung einer Software? Mehrere Repräsentationen des gleichen Wissens beherbergen die große Gefahr, dass eine dieser verschiedenen Repräsentationen im Laufe der Entwicklung nicht mehr mit den anderen übereinstimmt. Selbst wenn es gelingt, alle doppelten Repräsentationen von Wissen in einer Software synchron zu halten, müssen diese parallel gewartet werden. DRY ist daher wichtiges Mittel um seine Software flexible und leicht wartbare zu halten.[4] Da die Benutzeroberflächen von Saros/E und Saros/I große Gemeinsamkeiten besitzten, führt eine parallele Entwicklung von IDE-spezifischen GUIs dazu, dass Programmlogik zum Anzeigen dieser GUIs mehrfach vorhanden sein muss. Diese Stellen müssen gesondert gewartet, weiterentwickelt und angepasst werden. Neben dem offensichtlichen Mehraufwand entsteht hierdurch noch ein weiteres Problem, und zwar eine unnötige Verteilung des gleichen Systemwissens. Diese Verteilung der Anzeigelogik für die Benutzeroberflächen von Saros ist daher eine Verletzung des DRY-Prinzips. Wie ich in Abschnitt 2.3 darlegen werde, erzeugt dies verschiedenen negativen Auswirkungen für die Evolvierbarkeit des Saros-Plug-Ins.

Saros/E und Saros/I bestehen aus IDE-spezifischen und -unspezifischen Code. IDE-unspezifisch meint hierbei, dass es Teile im Aufbau von Saros gibt, welche sich auf allen IDEs gleich verhalten und somit von diesen unabhängig sind. Die Netzwerkschicht von Saros ist hierfür ein klassisches Beispiel. Während es offenkundig verschiedene Bereiche der Saros Architektur gibt, die von einer einheitlichen Codebasis profitieren würden, fokussiert sich diese Arbeit auf den Bereich der Präsentation, also der GUI von Saros. Ein möglicher Ansatz ist es, die GUI auf allen Systemen mit derselben Technologie zu verwirklichen, sodass möglichst viel Code gemeinsam genutzt werden kann. Als Basis für diese Vereinheitlichung dient dabei die Evaluation von Christian Cikryt [5].

1.3 Aufbau dieser Arbeit

Im folgenden Kapitel 2 Vorgehen und Umsetzung wird zunächst auf die vorhandenen Probleme von getrennt entwickelten GUIs innerhalb des Saros Projektes eingegangen. Basierend auf den angesprochenen Problemen und dem erwarteten Nutzen werden in diesem Abschnitt die genauen Ziele dieser Arbeit formuliert. Abschließend wird im letzten Abschnitt eine Abgrenzung und der Zusammenhang zu begleitenden Abschlussarbeiten in dem Bereich der GUT-Entwicklung erörtert.

Im Anschluss wird in Kapitel 3 Vorgehen und Umsetzung der gewählte Lösungsansatz und das Vorgehen zum Erreichen dieser Ziele erläutert. Ergänzend enthält dieser Abschnitt einen kurzen Überblick über Probleme, die nicht technischer Natur waren. Diese machten in der Summe jedoch einen erheblichen Teil der Arbeit aus.

In Kapitel 4 Erweiterung der HTML GUI wird auf die Implementationen eingegangen.

Abschnitt 5 Ergebnisse stellt in der Folge zusammenhängend die Ergebnisse dar. Dabei wird besprochen und evaluiert, inwiefern die gestellten Ziele erreicht wurden.

Abschließend enthält Abschnitt 5.6 Fazit und zukünftige Arbeiten die Schlussfolgerungen sowie einen Ausblick auf mögliche zukünftige Arbeiten.

2 Problemstellung, Zielsetzung und Abgrenzung

2.1 Plug-In-Entwicklung auf mehreren IDEs

Wie in der Einleitung beschrieben, wird Saros auf mehreren IDE Plattformen entwickelt. Eine Alphaversion des Saros-Plug-In ist für IntelliJ mittlerweile verfügbar[6]. Da diese sich zurzeit noch in der Entwicklung befindet, ist der Funktionsumfang und damit die Codebasis nicht gleich groß wie in Eclipse. Diese Arbeit geht von der Annahme aus, dass Saros/I und Saros/E mittelfristig den gleichen Funktionsumfang besitzen und damit auch gleichwertig nebeneinander entwickelt werden. Das Entwickeln von Saros für zwei (oder mehr) IDEs stellt das Projekt vor eine Reihe neuer Herausforderungen. Grob können diese eingeteilt werden in

- **Architekturanpassungen**
  Dies sind alle Änderungen an der Codebasis, der Code-Struktur sowie die Wahl der verwendeten Technologien, welche das Ziel haben, die Software effektiver auf verschiedene IDEs zu entwickeln.

- **Prozessanpassungen**
  Dies sind alle Änderungen am (Software-)Entwicklungsprozess, die der Entwicklung einer Software auf verschiedenen IDEs Rechnung tragen (z.B. Änderungen der Codingconventions, des Reviewprozesses, des Releaseprozesses und so weiter).

Da die Arbeit sich nur indirekt mit den Prozessanpassungen beschäftigt, wird nicht weiter auf diesen Aspekt eingegangen. Es sei jedoch erwähnt, dass er wichtige Probleme für das Saros Projekt und seine Entwicklung beinhaltet. Die Probleme umfassen eine breite Palette von Themen. Sie erstrecken sich vom Build-Prozess über Qualitätssicherungs- und Testprozesse bis hin zu Marketingfragen[4]. Daher ist dieser Bereich eine eigene Untersuchung wert.

2.2 Architekturanpassungen in Saros

Saros/I und Saros/E sollen zukünftig den gleichen Funktionsumfang aufweisen.[9] Grundsätzlich besitzen die verschiedenen Saros-Varianten[5] viel Ähnlichkeit. Hierdurch erscheint eine vollständig getrennte Entwicklung dieser Plug-Ins nicht sinnvoll. Wie eingangs erwähnt, gibt es trotz der Gemeinsamkeiten in der jeweiligen Codebasis der Plug-Ins IDE-spezifische Bereiche, während andere Bereiche von Saros in jeder IDE gleich umsetzbar sind. Mit 'gleich' ist hierbei tatsächlich gemeinsam genutzter Code gemeint. Dies gilt auch für die Nutzung von gemeinsamen Interfaces.

[4] Saros betreibt als nicht kommerzielle Software keine Gewinnabsichten. Die Verbreitung der Nutzung von Saros ist jedoch Bestandteil des Projektes[7][8]

[5] Mit Varianten sind in dieser Arbeit die verschiedenen IDE Plug-Ins von Saros gemeint

Eine Herausforderung besteht darin, herauszufinden, welche Softwareartefakte sinnvoll wiederverwertet und einheitlich genutzt werden können und welche IDE-spezifisch bleiben müssen oder sollten. Die ursprünglich nur für Eclipse konzipierte Saros-Architektur muss an dieser neuen Herausforderungen angepasst werden.
In Saros wurde daher ein CORE-Modul (im folgenden kurz Kern) eingefügt, welches die Aufgabe hat, eine gemeinsame Basis der Geschäftslogik für alle IDEs bereit zu stellen. Zum Zeitpunkt dieser Arbeit wird aktiv an diesem Modul gearbeitet. Mehrere Arbeiten beschäftigen sich direkt oder indirekt mit der Trennung von IDE-spezifischer und IDE-unspezifischer Logik bzw. der möglichen Konvertierung in IDE unabhängige Logik[10, 11, 12]. Was für die Geschäftslogik schon betrieben wird, steht für die GUI von Saros noch aus, welche vollständig IDE-abhängig ist. Die These dieser Arbeit ist, dass eine einheitliche GUI verschiedene Vorteile für die Entwicklung von Saros bringt, bzw. dass das Nicht-Vereinheitlichen der GUI ernste Probleme verursacht. Diese Arbeit hat das Ziel, eine einheitliche Codebasis für die verschieden IDE-GUIs zu entwickeln, um damit den zukünftigen IDE-übergreifenden Entwicklungsprozess von Saros zu erleichtern. Im Folgenden wird auf die Probleme der getrennten Entwicklung eingegangen, sowie der zu erwartenden Nutzen einer gemeinsamen GUI für diese Probleme besprochen. Daraus spezifizieren sich die genauen Zielsetzungen dieser Arbeit. Die in diesem Abschnitt erfolgten Überlegungen basieren auf den beiden schon existierenden Saros-Varianten für IntelliJ und Eclipse. Es sei jedoch darauf hingewiesen, dass es zukünftig weitere Plug-Ins für andere IDEs, wie Netbeans, geben kann. Die Vorteile einer gemeinsamen UI-Architektur bzw. die Probleme beim Fehlen einer solchen wären dabei dementsprechend größer. Wie wahrscheinlich eine solche Erweiterung z.B. für Netbeans ist, kann zu diesem Zeitpunkt schwer abgeschätzt werden. Es gibt jedoch bereits eine Untersuchung, die sich mit der Umsetzung von Saros für Netbeans beschäftigt.[13]

2.3 Problemanalyse und Zielstellungen

2.3.1 Problem der Dopplung von Logik (Codeduplikaten)

Das Hauptproblem bei der Verwendung von verschiedenen GUIs für Saros/I und -/E ist ihre unterschiedliche Codebasis bei zeitgleich sehr ähnlichem und (wie im Weiteren aufgezeigt wird) untereinander gekoppeltem Funktionsumfang. Daher entstehen ohne eine vereinheitliche GUI eine Menge Codeduplikate. In der Folge müssen Änderungen und Erweiterungen innerhalb einer GUI auch an ihrem gekoppelten Konterpart der anderen GUI(s) vorgenommen werden, was einen deutlichen Mehraufwand erzeugt. "Codeduplikation"wird in dieser Arbeit in einem breiteren Sinne verstanden, da damit nicht nur 1:1-Wiederholungen von Codefragmenten gemeint sind.

Codeduplikation erweiterter Begriff (Semantische Klone — Gleiches Verhalten)
*Als Codeduplikation soll jede **vermeidbare** Verteilung von Wissen und Logik in einer Software verstanden werden. Als Duplikat ist auch Code zu begreifen, der denselben Zweck verschiedenartig erfüllt. So wird nicht nur die direkte 1:1-Codewiederholung als Codeduplikation verstanden, sondern auch die Wiederholung von Wissen und Verantwortlichkeit innerhalb der Softwarearchitektur. 1:1-Duplikationen werden hierbei als „Klone“ bezeichnet, während diese Codeduplikation in dieser Arbeit auch semantische Klone und Code mit ähnlichem Verhalten beinhaltet.[14, S. 2] Solche Codeduplikate sind oft gekoppelt: Die (semantische) Änderung eines Exemplars forciert die Anpassung aller anderen Exemplare.*

So sind semantische Klone oft gefährlicher und schädlicher für eine Codebasis als einfache 1:1-Duplikate, da sie schwerer als solche zu erkennen sind und auch nur selten von IDE-Werkzeugen erfasst werden können. So zeigt die Analyse von Elmar Juergens, Florian Deissenboeck und Benjamin Hummel, dass solche Codeduplikationen in nur knapp 10 Prozent der Fälle richtig erkannt werden[14, S. 5]. Somit ist effektiv nur durch manuelle Durchsichten möglich, solche Dopplungen zu erkennen. Dies ist vergleichsweise aufwendig. Um die Evolvierbarkeit einer Software zu gewährleisten, ist es nötig, diese Duplikation von Wissen zu vermeiden.

graphical user interface - Element
Ein Graphical User Interface Element (kurz GUI-E) ist ein einzelnes Teilstück einer Benutzeroberfläche, wie z.B. eine Box, ein Text, eine Schaltfläche, etc.

Ein GUI-E kann aus anderen Elementen zusammengesetzt sein. So wird ein Wizard, welcher aus verschiedenen kleineren Elementen zusammengesetzt wird, der Einfachheit halber auch als GUI-E bezeichnet.

Ein Beispiel für solch eine Duplikation ist leicht in Saros zu finden: Ein neues GUI-E, beispielsweise eine Hinweisbox, soll in Saros eingeführt werden. Offensichtlich muss dieses GUI-E einmal in Saros/E und Saros/I implementiert werden. Diese Implementationen sind syntaktisch unterschiedlich, da unter anderem verschiedene Technologien zum Einsatz kommen. Insofern sind diese beiden neuen Codestellen nicht unbedingt direkt als Duplikate zu erkennen. Diese Codefragmente haben die gleiche Verantwortlichkeit: Das Darstellen der Hinweisbox. Damit handelt es sich hierbei um semantische Klone, welches nach Definition als Codeduplikationen zu verstehen sind.

Relevanz von Codeduplikation im Bereich der UI-Komponente
Das Java UI-Paket von Saros umfasst über 200 Klassen, verteilt auf 37 Unterpakete. Insgesamt beschäftigen sich ca. 22.000 von 81.000 LoC direkt oder indirekt mit der UI von Saros/E.[6]. So fallen 27 Prozent der gesamten Codebasis nur auf die UI. Dabei sind Projekte und Komponenten wie das Whiteboard von Saros[7] mit eingerechnet. Dies ist insofern besonders interessant, als dass die Saros/I Version in vielen Bereichen noch unfertig ist und das UI dort erst noch gebaut werden müsste. Dabei ist anzunehmen, dass die IntelliJ UI-Komponente kaum eine geringere Komplexität als das Saros/E Pendant aufweisen wird. Deshalb ist die Vermeidung eines solchen Aufwands sehr wünschenswert. Es zeigt sich deutlich, dass es lohnenswert ist, die UI mit fast einem Drittel Anteil an der Codebasis nicht für Saros/E und Saros/I gesondert pflegen und erweitern zu müssen. Diese Verringerung und Vermeidung von Codeduplikation ist somit Schwerpunkt dieser Arbeit.

2.3.2 Problem der Fehleranfälligkeit durch Codeduplikation

Eine direkte Folge mehrerer parallel entwickelter Oberflächen und der daraus resultierenden Codeduplikation in Saros ist eine Erhöhung der Fehlergefahr. Dadurch kann die Defektdichte zunehmen. Leider liegen keine Daten zu der Defektdichte in den aktuellen benutzten GUI vor. Eine Erhebung dieser Daten hat sich als schwer durchführbar erwiesen.[8] Daher will ich diese Behauptung argumentativ begründen und an einem Beispiel demonstrieren.
Der vermutete Fehleranstieg liegt darin begründet, dass Saros/I und Saors/E GUIs gleiche GUI-Es besitzen. Damit sind diese GUI-E faktisch Codeduplikate nach 2.3.1 was eine unnötige Verteilung von Wissen und Verantwortlichkeit bedeutet. Dadurch ergibt sich eine Verletzung des Ein-Verantwortlichkeits-Prinzip. Dieses sagt aus, dass "*Änderungen oder Erweiterungen der Funktionalität einer Anwendung sich auf wenige Klassen beschränken sollen. Je mehr Klassen angepasst werden müssen, desto größer ist das Risiko, dass sich durch die erforderlichen Änderungen Probleme an Stellen ergeben, die im Kern nichts mit der Erweiterung zu tun haben. Eine Verletzung des Single Responsibility Principle führt zu Kopplung und damit zu erhöhter Komplexität, es wird schwieriger den Code zu verstehen.*"[16] Verschlimmernd kommt hinzu, dass diese Kopplungen *versteckte* sind. Versteckt bedeutet hierbei, dass sie nicht im Code als solche direkt erkennbar sind. Ich will im einem Beispiel darstellen, dass gerade diese versteckten Code-Kopplungen fehleranfällig sind.

[6] Daten erhoben mit eclipsemetrics [15] Stand: 14.07.2015
[7] Diese Funktionalität ist in ein eigenes Modul ausgelagert https://github.com/saros-project/saros/tree/master/de.fu_berlin.inf.dpp.whiteboard
[8] Siehe hierfür Abschnitt 3.3

Beispiel Fehlergefahr in getrennten GUIs. Betrachtet wird hier das Szenario, die GUI würden getrennt entwickelt worden sein und Saros/I und Saros/E hätten den gleichen Funktionsumfang. Ein Sarosentwickler macht nun eine Änderung im Saros/E *SessionInvitationManager*. Er verändert die angezeigten Meldungen, und benennt vorhandene Schaltflächen um.
Nun ist der Eclipse-*SessionInvitationManager* mit seinem IntelliJ Pendant gekoppelt. Verschiedene Bezeichner für dieselbe Funktion in den Benutzeroberflächen sind unerwünscht, da sie verwirren. Neben dem Mehraufwand, die GUIs beide ändern zu müssen erhöht sich die Fehlergefahr dadurch, dass diese Kopplung nicht anhand des Codes sichtbar ist. Auch können Refactor-Tools diese Abhängigkeit nicht aufspüren. Für den Entwickler ist es ohne genaue Kenntnis der beiden UI-Komponenten daher völlig unklar, ob es dieses Element in einer ähnlichen Form in IntelliJ gibt. IntelliJ und Eclipse benutzten verschiedene Technologien für die Anzeige. So ergibt sich die Gefahr, dass ein Entwickler versäumt, das GUI-E Pendant in Saros/I ebenso anzupassen.

Während das Beispiel 'nur' in inkonsistente Bezeichner resultiert (aus dem in der Folge weitere Fehler resultieren können), existiert das gleiche Problem für die darunter gelagerte Schicht ebenfalls. Selten wird aus der Benutzeroberfläche Kernlogik direkt aufgerufen. In der Regel gibt es eine Art Zwischenschicht, welche Benutzereingaben auswertet, eventuell validiert und passend an die jeweiligen Kernfunktionalitäten weiterreicht. Die Benutzeroberflächen sind in den Saros-Varianten auch verschiedenartig und an unterschiedlichen Stellen mit der Geschäftslogik verbunden.
Das Verhalten dieser Zwischenschichten in allen Saros-Varianten konsistent zu halten, wird damit erheblich erschwert. So ist es leicht vorstellbar, dass in folge solcher Fehler das gleiche Eingabefeld in Saros/I eine Eingabe akzeptiert, während in Saros/E dem Benutzer ein Validierungsfehler angezeigt wird. Solche Konsistenzprobleme treffen in der Tat auf alle Bereiche der Saros Architektur zu. Abgrenzend beschäftigt sich diese Arbeit jedoch mit der Benutzeroberfläche und der Kommunikations- und Validierungsschicht zur Geschäftslogik. Die Vermeidung von Fehlerpotenzial, welche eine getrennte Entwicklung mit sich bringen würde, ist somit ein weiteres Ziel dieser Arbeit.

Evolvierbarkeit
ist ein Kriterium bei der Entwicklung von Software. Sie gibt an, mit welchem Aufwand und mit welchem Erfolg neue Features eingebracht werden können. Eng damit verbunden ist die Wartung und Wartbarkeit der Software. Eine hohe Evolvierbarkeit beschreibt dabei einen geringen Aufwand und höheren Erfolg für neue Features, während eine geringe Evolvierbarkeit eine große Anstrengung und weniger Erfolgsaussichten für neue Features anzeigt.[17]

2.3.3 Ziel: Verbesserung der Codetransparenz

Bei der Konzeption der gemeinsamen UI muss darauf geachtet werden, dass Fehler besser ersichtlich und somit in der Entwicklung vermieden werden können. Ein Mittel, Fehler und Defekte zu vermeiden, ist es, die Effekte von Änderungen am Code einfacher und direkter sichtbar zu machen. Dadurch können unerwünschte Auswirkungen von Änderungen besser erkannt werden. Diese Erhöhung der Evolvierbarkeit soll durch eine Verbesserung der Codetransparenz realisiert werden.

Transparenz in der UI Architektur (Codetransparenz)
Änderungen an einem UI-spezifischem Codefragment sollen sichtbare, direkte und einfach zu findende Auswirkungen auf das Resultat in der GUI haben. Insbesondere gilt die Architektur eines Moduls als transparent, wenn der Zusammenhang zwischen einem GUI-E und seinem korrespondierenden Codefragment klar ersichtlich, nachvollziehbar und daher intuitiv ist.

Diese Definition ist nicht zu verwechseln mit dem Begriff der Softwaretransparenz, welche einen Benutzerblickwinkel einnimmt und gemeinhin genau das Gegenteil meint: Transparenz wird hierbei nicht als “einfach zu durchschauen“ verstanden, sondern als unsichtbar bzw. nicht auffindbar.[18] Diese Arbeit beschäftigt sich mit Saros vornehmlich mit ihrer Evolvierbarkeit der GUI aus Entwicklerperspektive.

2.3.4 Ziel: Verringerung der Fehleranfälligkeit durch Fail Fast

Ein weiteres Mittel und Ziel ist es, die UI nach dem ’Fail-Fast’ Paradigma zu konzipieren. Dies sagt unter anderem, dass Schnittstellen so zu konzipieren sind, dass Fehler oder Zustände, die zu Fehlern führen, frühzeitig erkannt und aufzeigt werden.[19]

2.3.5 Analyse der GUI im Hinblick auf das Zentralisierungspotenzial

Aus den vorher genannten Zielen ergibt sich direkt eine weitere Aufgabe der Arbeit: Die Analyse, welche Teile der UI zusammengefasst werden können und sollten. Eine Machbarkeitsanalyse mit welchen Technologien dies umsetzbar ist, wurde Christian Cikryt angefertigt.[5, S. 9-27] Ich werde darauf aufbauend untersuchen:

- Welche Bereiche der Sarosoberfläche können über alle verschiedenen IDEs zusammengefasst werden?

- In welchen von diesen Bereichen ist eine Vereinheitlichung sinnvoll?

So können bestimmte Bereiche nicht vereinheitlicht werden. Dies schließt z.B. die Integration in IDE-spezifische Strukturen wie das IDE-Menü ein. Andere Bereiche lassen sich nicht zusammenfassen, da sie nur in einer Version verfügbar sind. Dies trifft beispielsweise auf das Whiteboard zu.[9]
Um zu entscheiden, wie sinnvoll eine Vereinheitlichung ist, bedarf es einer qualitativen Abwägung verschiedener Faktoren. Die Zentralisierung einer bereits vorhanden GUI Komponente wird als sinnvoll betrachtet, wenn:

1. sie allgemein genug ist, um in allen IDEs zum Einsatz zu kommen.

2. sie so umzusetzen ist, dass die Anwenderfreundlichkeit mindestens erhalten bleibt, gegebenenfalls jedoch verbessert werden kann.

3. der Aufwand in einem vertretbaren Rahmen zu dem Nutzen steht, wobei sich dieser im Allgemeinen direkt aus den genannten Zielen in diesem Abschnitt ergibt. Damit ist die Umsetzung sinnvoller je zentraler bzw. wichtiger das GUI-E in Saros ist. Der Aufwand hängt hierbei unter anderem oft, wie von Christian Cikryt auch richtig erwähnt, stark mit der Verfügbarkeit der Funktionalität im Kern ab. Fehlt diese, ist der Aufwand entsprechend höher.

2.3.6 Ziel: Schärfere Trennung von Geschäftslogik und GUI

Da die Evaluation der HTML-UI und die daraus folgenden Technologieentscheidungen für das Frontend zu Beginn dieser Bearbeitung noch nicht final sind, ist es für die gemeinsame UI wichtig, dass sie so unabhängig wie möglich gestaltet wird. So soll bei der Weiterentwicklung der UI-Komponente darauf geachtet werden, dass Abhängigkeiten bestmöglich vermieden werden. Die Zerlegung erfolgt hier, um Entscheidungen in den einzelnen Modulen voreinander zu verbergen. Sie folgt damit dem Geheimhaltungsprinzip nach Parnas.[20]Ein Vorteil dieser Trennung ist es, dass Module ohne großen Aufwand ausgetauscht werden können. Es ist absehbar, dass sich die Wahl der Technologie für das Frontend in Zukunft noch ändern wird. So schreibt Christian Cikryt :"The main problem is that **JavaFX would be the clear choice in maybe a year from now** but currently does not meet the runtime requirements for Saros"[5]. Ich teile hierbei seine Einschätzung. Als sinnvoll erscheint daher die Einführung eines Adapters zwischen der eigentlichen Anzeigelogik und der Geschäftslogik um diese Änderungen in der weiteren Entwicklung leichter durchführen zu können.

[9] Anm. Eine Untersuchung, ob das Whiteboard-Feature IDE-unabhängig realisiert werden kann steht noch aus siehe 3.2.6. Zum Zeitpunkt dieser Arbeit ist es nur in Saros/E verfügbar

2.4 Zusammenhang mit anderen Arbeiten

Es besteht ein Zusammenhang zu anderen, teilweise zeitgleich entstandenen Arbeiten, welche sich mit der Zentralisierung der GUI von Saros beschäftigen, da ich an diesem mitgewirkt habe und sie direkt mit dem Themenbereich der vorliegenen Arbeit verwoben sind. Durch gegenseitige Abhängigkeiten hat die Zusammenarbeit und das Abstimmen mit anderen Sarosteam-Mitgliedern einen beträchtlichen Teil der Bearbeitungszeit eingenommen. Diese Zusammenarbeit verlief erwartungsgemäß nicht reibungslos. Im Folgenden wird auf das Zusammenwirken und die direkten Berührungspunkten eingegangen, wobei im Abschnitt 5.5 Erkenntnisse aus dieser Situation besprochen werden.

2.4.1 Technische Grundlage dieser Arbeit

Zum besseren Verständnis der technischen Aspekte meiner Arbeit empfehle ich dringend, zumindest den Abschnitt 'The SWT browser Component' [5, S. 12-15] sowie 'Implementation'[5, S. 39-41] der Arbeit von Christian Cikryt zu lesen, da sie die Basis für die Entwicklung der GUI Komponente enthält. Ich werde kurz den Zustand der GUI-Architektur vor Beginn der Erweiterung der GUI darstellen und die wichtigsten Begriffe und Konzepte aus der Arbeit von Christian Cikryt zusammenfassen.

(SWT)-Browser
Der von Christian Cikryt entworfene Prototyp nutzt eine von Björn Kahlert erweiterten SWT-Browser zum Laden und Anzeigen von HTML Dateien innerhalb von Saros. Dieser Browser wird in Saros/E direkt in eine SWT-Shell eingebunden, während sie in Saros/I mittels einer ThirdPerson-Bibilothek (SWT-AWT-Brigde) in die dort übliche AWT Komponenten eingebunden wird. Dieser SWT-Browser benutzt native Systembibliotheken und greift dabei auf den Standardbrowser des jeweiligen Betriebssystem zurück. Um den SWT-Browser in die jeweiligen IDEs einzubinden, sind wenige IDE-spezifische Implementationen wie z.B. den BrowserCreator oder den Dialog-Manager nötig.

Browserfunktionen und Java<->Javascript-Kommunikation
Die in HTML realisierten Anzeigen verwenden Javascript zum Aufrufen von Geschäftslogik. Diese Javascriptfunktionen werden mittels sogenannter Browserfunktionen in die Browser injiziert. Durch diese Browserfunktionen ist es innerhalb der HTML-Seite die dieser Browser geladen hat, möglich für Javascript auf durch diese Browserfunktionen gekapselten Java-Funktionen zuzugreifen (Javascript->Java).
Um die Anzeige in den HTML-Seiten von Java-Seite aus zu verändern, werden in der HTML Seite geladene Javascriptfunktionen aus Java aufgerufen.

Diese sind über die Java-Browser-Instanz über verschiedene *run(Script)* Methoden verfügbar. Welche Scripte innerhalb eines Browser verfügbar sind ,also innerhalb der HTML-Seite geladen wurden, muss auf Javascriptseite definiert werden (Java->Javascript).

Weitere Implementierungsdetails, wie die Renderer- und Datenmodellklassen (siehe 4.1) werden soweit nötig an den entsprechenden Abschnitten im Kapitel 4 Erweiterung der HTML GUI erläutert.

2.4.2 Mitwirken im Prototyp und Evaluation der HTML GUI

Wie bereits erwähnt, baut meine Arbeit auf den Ergebnissen von Christian Cikryt auf. Zu der Entwicklung einer gemeinsamen GUI gehört die Evaluation der benutzbaren Technologien. Diese Evaluation führte zur Nutzung von in SWT integrierten Browserkomponenten. Eine ausführliche Beschreibung der Vor- und Nachteile, sowie der möglichen Alternativen zu der gewählten SWT-Browser-Lösung findet sich in Kapitel „Evaluation of an HTML-GUI vs. Java toolkit“ der Arbeit von Christian Cikryt.[5, S. 20-28] Neben der Evaluation war die Umsetzung eines Prototyps Bestandteil seiner Arbeit. Fast alle Teile dieser entstanden GUI-Architektur wurden von mir durch Code-Durchsichten und gemeinsame Absprachen begleitet. Eine Liste Commits an denen ich mitgewirkt habe, kann im Anhang A.2 eingesehen werden. Abgesehen davon habe ich aktiv an der Verwirklichung des Prototypen mitgearbeitet. Die Saros/E spezifische Implementationen des Prototypen wurden von mir entwickelt. Dies beinhaltet im Einzelnen

- SarosViewBrowserVersion – Die Haupt-SarosView der HTML GUI in Saros/E
- EclipseDialogManager – Implementierung des DialogManager
- EclipseBrowserDialog - Implementierung des BrowserDialog
- EclipseResourceLocator – Implementierung **IWebResourceLocator**

Aufgabe des IWebResouceLocator ist es, URL-Pfade der Webressourcen (wie z.B. HTML Dateien, CSS, Bilder, etc.) zur Laufzeit zu bestimmen, welche z.B. beim Erstellen des Browser benötigt werden, um die Webseite laden zu können. Ein größeres Problem stellte dabei das Auffinden der Web-Ressourcen in Saros/E dar.

Während die Ressourcen in Saros/I durch einen einfachen Aufruf des *Class-Loaders* erfolgen konnte, war dies durch die Art, wie das Eclipse OSGI Framework[10] arbeitet, nicht ohne Weiteres möglich. So ist die Laufzeitbestimmung der URLs von nicht Javadateien in Saros/E um einiges aufwendiger

[10] http://www.eclipse.org/equinox/ Stand: 25.05.2015

als im Saros/J Pendant. Der EclipseResourceLocator extrahiert Bundle Ressourcen und stellt die URLs dieser Ressourcen zur Laufzeit von Saros zur Verfügung. Beim erstmaligen Zugriff auf eine Webressource wird die URL von allen Dateien innerhalb eines gegebenen Wurzelordners erstellt. Da diese URLs sich während der Laufzeit von Saros nicht mehr ändern, wird dabei ein Mapping zwischen der Ressource und ihrer URL erzeugt. Der Algorithmus arbeitet auf Grund von Abwärtskompatibilität der Eclipseversionen verhältnismäßig komplex und beinhaltet zudem verschiedene Fallbacks, falls das Extrahieren nicht funktioniert. Hierdurch ergibt sich eine quadratische Laufzeit. In einem Test zur Qualitätssicherung auf einem durchschnittlichen Rechner[11] entsteht ab ca. 5.000 Dateien innerhalb des Wurzelordners eine spürbaren Wartezeit von mehr als drei Sekunden für den Anwender bis die Saros-View geladen wurde. Diese Laufzeit stellt zurzeit kein akutes Problem dar, da nur wenige [12], CSS, HTML und Javascript Dateien zur Laufzeit benötigt werden. Dieses Problem ist jedoch in Zukunft mit wachsendem Umfang der Frontendressource zu beachten. So ist es wichtig darauf zu achten, dass ausschließlich Dateien im Wurzelordner liegen, welche auch tatsächlich von der HTML-GUI zur Laufzeit geladen werden.

Christian benutzt in seiner Arbeit verschiedene Kriterien, um zu entscheiden, mit welchen Technologien die gemeinsame UI umgesetzt wird. Zwei Hauptkriterien, die zu der Entscheidung für einer Browser/HTML Ansatz führten, waren dabei der '*Einmalige Entwicklungsaufwand*' und der '*Laufende Entwicklungsaufwand*' sobald die UI fertiggestellt ist.[5, S. 21-22] Auf das Kriterium des "Laufenden Entwicklungsaufwands"möchte ich kurz eingehen. Christian führt als Nachteil für die Browser/HTML-Variante den zusätzlich nötigen Code für die Kommunikation zwischen Java und Javascript auf. Ich halte dies für keinen wirklichen Nachteil im Vergleich zu anderen Ansätzen. Die Kommunikation zwischen Backend und Frontend erfordert in jedem Fall eine gewisse Menge an Code, unabhängig von der Wahl der Technologien. In dem konkreten Fall ist der benötigte Code für die Kommunikation zwischen Java<->JS nicht groß, da sich dieser Aspekt durch eine geeignet gewählte Fassade minimieren lässt, wie in 4.2.2 Einführung einer zentralen JavaScriptAPI-Klasse gezeigt wird.

[11] Test auf i5-2410M 2 Kern CPU @ 2.30GHz mit 4 GB RAM

[12] Zur Zeit benötigt das Frontend weniger als 20 Dateien/Ressourcen zum Anzeigen der GUI

Ein weiter Minuspunkt der Browser-Variante ist nach Christian der Mix von Technologien (Java und JS) innerhalb der UI. Ein Mix von Technologien ist jedoch per se kein Problem oder Nachteil für eine Software. Ein Technologiemix kann sogar förderlich und wünschenswert sein, wenn:

- es eine klare Abgrenzung zwischen den eingesetzten Technologien gibt. Dies bedeutet, dass für Entwickler eindeutig ersichtlich ist, welche Aufgabe einer Software mit welcher Technologie umgesetzt wird und warum. Hierbei gilt es, den konkurrierende Einsatz von Technologie zu vermeiden: Ein Werkzeug für eine Aufgabe.[13]
- die gewählten Technologien effektiv für ihren jeweiligen Aufgabenbereich sind.
- die Technologien gut ineinander integrierbar sind.

All diese Faktoren können auf die Konzeption der HTML-GUI zutreffen: Hierbei würde Javascript ausschließlich für das Verändern der Anzeige und der Validierung von Benutzereingaben benutzt werden während Java die Geschäftslogik von Saros umsetzt. Diese technologische Aufgabenteilung habe ich zusammen mit Bastian Sieker 2.4.4 konsequent bei der Weiterentwicklung der HTML-GUI durchgesetzt. Dies führte unter anderem zu einer Teilung des Moduls in einen UI-Frontend- und UI-Backend-Bereich was auch der formulierten Zielstellung einer effektiveren Trennung von Anzeige und Geschäftslogik (siehe 2.3.6) Rechnung trägt. Der genaue Aufbau wird in Kapitel 4 erläutert, hier soll es nur demonstrieren, dass der Java-JS-Technologiemix nicht hinderlich, sondern sogar praktikabel für die Entwicklung sein kann.

2.4.3 Abhängigkeit zum Saros Kern Modul

Wie auch schon Christian Cikryt in seiner Arbeit beschreibt, ist die Entwicklung der gemeinsamen GUI entscheidend von dem Stand des Kern-Moduls abhängig. Die gemeinsame GUI ist ausschließlich abhängig vom Kern. Dies ist eine Architekturentscheidung, um die angestrebte Unabhängigkeit der gemeinsamen UI-Komponente zu garantieren. Diese soll per Definition unabhängig von den konkreten IDE-Implementierungen sein und darf somit nur gemeinsame, von allen Versionen unterstützte Interfaces und Implementationen benutzten. Damit kann sie auch nur auf gemeinsam benutzte Interfaces und Methoden aus dem Kern zurückgreifen. In der Konsequenz ist der UI-Komponente die Existenz von Saros/J und Saros/E nicht bekannt. Die Planung der Umsetzung meiner und Christians Arbeit basierte auf der Annahme, dass dieses Kern Modul bereits weit fortgeschritten ist und allenfalls wenige Änderungen und Ergänzungen nötig sein würden.

[13] Technologie kann in diesem Fall als Werkzeug für eine Aufgabe verstanden werden

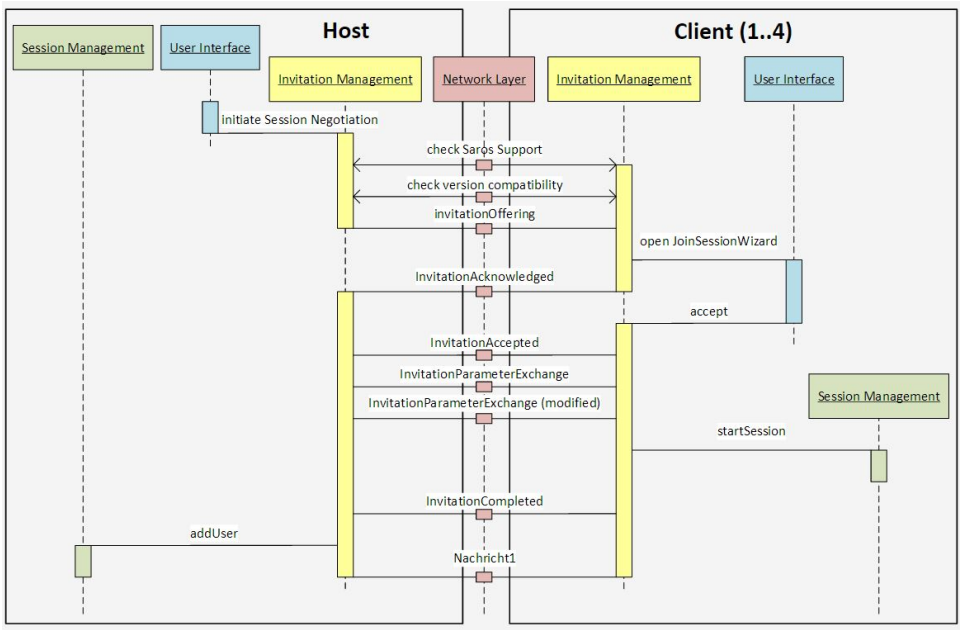


Abbildung 1: Session Negotiation in Saros. Quelle:[21, S. 73]

Eine wichtige Abhängigkeit für die Umsetzung der GUI sind die *Negotiation*-Klassen in Saros. Diese werden benutzt, um den Aufbau einer Session zu verwalten. Wie in Abb. 1 zu sehen, reagiert das Userinterface auf diese Negotiation (z.B. durch das Öffnen eines JoinSessionWizard). Teile dieser Architektur sind noch nicht im Kern verfügbar. So fehlen die Projekt bezogenen Negotiations, sowie ein einheitlicher Handler zum Reagieren auf eingehende und ausgehende Nachrichten. Da sich zum Zeitpunkt der Anfertigung meiner Arbeit, Arndt Lasarzik sich in seiner Abschlussarbeit mit der Integration von Saros/E Funktionalität in den Kern beschäftigte und der Negotiation-Bereich Teil seiner Arbeit war, wurde davon ausgegangen, dass diese für die GUI nötigen Komponenten bald verfügbar sein würden.[10] Zum Zeitpunkt der Abgabe der Arbeit stehen diese Erweiterungen im Kern noch nicht zur Verfügung. Dies führte zu einer Verzögerungen bei der Umsetzung der Session bezogenen GUI-Elemente. [14]

2.4.4 Entwicklung des Frontends

Weiterhin wichtig für die Erstellung dieser Arbeit ist die Masterarbeit von Bastian Sieker, welche sich mit der Umsetzung des HTML-Frontend beschäftigt.[22] Das UI-Modul ist die direkte Schnittstelle zwischen der Anzeigeschicht (UI-Frontend) und dem Kern-Modul. Da unsere Arbeitsbereiche so-

[14] Siehe hierfür auch 3.2.3 Join-Session-Wizards und 4.4 Implementation des JoinSessionWizards

mit direkt voneinander abhingen, wurden viele Schnittstellen-, Modell- und Designentscheidungen (siehe Abschnitt 4.3.1) in Abstimmung mit ihm erarbeitet. So wurde entwickelter Code gegenseitig im Saros-Reviewprozess aktiv gegenseitig begleitet. Abgrenzend beschäftigt sich die Arbeit von Bastian Sieker vornehmlich mit der (Neu-)Konzeption der Benutzeroberfläche mithilfe von HTML Technologien, während ich in meiner Arbeit die Schnittstelle bzw. den Adapter vom und zum Kern entwickelte.

3 Vorgehen und Umsetzung

Wie in Abschnitt 2.3 beschrieben, ist eine Neukonzeption der UI erforderlich gewesen. Aufgrund der gewählten Technologie war es nicht möglich, größere Teile der GUI wieder zu verwenden. Dieser Abschnitt beschreibt die Umsetzung der in Kapitel 2 genannten Ziele. Dabei wird auf Implementationsdetails gesondert im Kapitel 4 eingegangen.

Das Vorgehen kann in 4 Phasen eingeteilt werden:

1. Analyse der vorhanden GUI in Bezug auf das Zentralisierungspotenzial.
2. Konzeption und Durchführung eines Entwicklertests, um Schwachstellen im vorhandenen Prototypen zu identifizieren und Verbesserungspotenzial im Bereich der Codetransparenz zu ermitteln.
3. Die Analyse und Umsetzung einer HTML-GUI für ein kleines Projekt wie JTourBus, um daraus generelle Methodiken und Erkenntnisse für eine Cross-IDE-GUI-Entwicklung in Saros abzuleiten.
4. Die Implementation der noch fehlenden UI-Komponenten sowie der dafür benötigten Kernlogik.

3.1 Verwandte Arbeiten (Cross-IDE-Plugins)

Die Suche nach verwandten Arbeiten oder Quellen, die sich mit der Frage einer einheitlichen GUI-Entwicklung über verschiedene IDEs beschäftigt, war wenig ergiebig. Dabei habe ich nach Plug-Ins gesucht, die simultan für Eclipse und für IntelliJ angeboten werden. In einer Untersuchung der gängigsten Plug-Ins für beide IDEs wurde keins gefunden, das in beiden verfügbar wäre [23, 24]. Dabei wurden Plug-Ins gefunden, die den gleichen Zweck besitzen, jedoch nicht von dem selben Projektteam entwickelt und gewartet werden. Die untersuchten Plug-Ins, spezialisierten sich jeweils auf eine IDE. Plug-Ins welche einen Browser für die Anzeige innerhalb von Eclipse oder IntelliJ benutzten, wurden nicht gefunden. Dies deckt sich auch mit den Recherchen von Christian Cikryt über den Einsatz von Browsern für die Anzeige von Plug-Ins.[5, S. 6-7].

3.2 Eclipse GUI Analyse

Am Anfang dieser Arbeit stand die Analyse, welche Teile der GUI sich vereinheitlichen lassen. Da es früh absehbar war, dass sich nicht die vollständige GUI zentralisieren lässt, war eine Priorisierung nötig. Christian Cikryt hat bereits eine grobe Abschätzung erarbeitet, welche Bereiche der alten Saros/E GUI sich mit dem gewählten SWT-Browser/HTML-Ansatz umsetzen

lassen. Die GUI von Saros/I wird für die Analyse ignoriert, da sich diese selbst noch in der Entwicklung befindet und keine gute Grundlage für eine Abschätzung bietet. Christian Cikryt spricht von ca. 10 Wizards unterschiedlicher Größe, die mithilfe von HTML dargestellt werden können. In Abschnitt 2.3.5 wurde besprochen, dass es nicht sinnvoll bzw. möglich ist, alle Teile der GUI zu zentralisieren. Die einzelnen GUI-Elemente von Saros wurden daher anhand der dort gewählten Kriterien analysiert. Im Folgenden werden die Ergebnisse der Untersuchung für jede Komponente besprochen.

HTML-UI — Begriffsbenutzung
In dieser Arbeit wird der Begriff *HTML-UI* benutzt. Mit HTML-UI ist die Summe aller im Frontend genutzten Webtechnologien gemeint. Das beinhaltet sowohl Javascript (inklusive möglicher Frameworks), Web-Template-Engine[25], CSS[26] als auch HTML selbst.
Wenn daher HTML-UI im Kontext mit anderen Technologien zum Anzeigen von Benutzeroberflächen benutzt wird, ist die Summe aller genutzten Technologien in diesem Vergleich gemeint.

3.2.1 Saros-Hauptansicht

Diese Ansicht enthält den gerade aktiven XMPP-Account[15], den Status der Kontakte sowie den Status der Session. Des Weiteren sind hier die Kernfunktionen von Saros verfügbar:

- Erstellen, Wählen und Verändern des genutzten Accounts
- Verbinden und Trennen des activen XMPP-Accounts
- Hinzufügen, Löschen und Umbenennen von Kontakten
- Aufruf des Einstellungsfensters(siehe 3.2.5)
- Der Saros Chat (siehe 3.2.7)
- Öffnen des Einladungswizards(siehe 3.2.2)
- Aktiveren des Follower-Modus [16]
- Beenden der Session
- Reparieren von Inkonsistenzen
- Anzeige von Awareness-Informationen (siehe 3.2.8)

[15] Die Kommunikation von Saros basiert auf XMPP[27]
[16] http://www.saros-project.org/GettingStarted

Die Saros Hauptansicht wird in allen IDEs zum Einsatz kommen. Damit ist [1] erfüllt. Ihre Anwenderfreundlichkeit kann durch die HTML-UI verbessert werden. Somit trifft [2] auch zu. Im Bezug auf Kriterium [3] ist der Nutzen für eine gemeinsame Codebasis groß, der Aufwand allerdings hoch, da die Hauptansicht indirekt mit allen andern GUI-E interagiert. Somit hat die Umsetzung des Hauptfensters Priorität. Es ist dabei nicht notwendig, alle vorhandenen GUI-E, die in der Saros/E Hauptansicht vorkommen, auch genauso in der HTML GUI umzusetzen oder anzubieten. Bei der Analyse fällt auf, dass das Chatfenster nicht in das Hauptfenster integriert werden muss. Der Platz, den das Chatfenster einnimmt, kann auch anderweitig verwendet werden. Eine mögliche Nutzung für diese Fläche wäre das Feature der '*Verbesserten Action Awareness*', welches von Damla Durmaz entwickelt wurde. Die Anzeige von weiteren zusätzlichen Awareness-Informationen mittels SWT erzeugte größere Leistungseinbußen. Durmaz argumentiert, dass sich diese Informationen mutmaßlich performanter in HTML darstellen lassen[28].[17]

3.2.2 Session-Invitation-Wizards

Für den Einladungsprozess gibt es mehrere Wizards. Der **Hauptwizard ist der SessionInvitationWizard**. In diesem kann aus einer Liste von Projekten und Kontakten gewählt werden. Dadurch wird eine Session mit den ausgesuchten Ressourcen und Kontakten gestartet. In Saros/E gibt es jeweils einen Quicklink für den Fall, dass man eine Session mit genau einem Projekt und genau einem Teilnehmer starten will. Diese sind über ein Context Menü verfügbar. Je nachdem, ob man den Kontakt oder den Projektcontext gewählt hat, braucht man in diesen nur die zu teilenden Projekte bzw. die Kontakte zu wählen. Beides lässt sich auch in dem Hauptwizard bewerkstelligen, in dem die Projekte oder Kontakte bereits vorausgewählt sind, wenn dieser über die Contextmenüs Quicklinks aufgerufen wird.

Nach der Einladung wird der Hauptwizard geschlossen und eine Fortschrittsanzeige eingeblendet. Danach wird ein weiterer Wizard aufgerufen, der über das erfolgreiche Initialisieren der Session informiert.

Auf beide Wizard-Teile treffen die Kriterien [1] und [2] aus Abschnitt 2.3.5 zu. In Bezug auf Kriterium [3] ist der Aufwand für diese Anzeigen recht hoch, da es keine gemeinsame Art gibt, wie Projekte in den IDEs organisiert werden (siehe hierfür auch Tabelle 1). Dieser Dialog benötigt eine einheitliche Darstellung der wählbaren Projekte und Workspace-Ressourcen[18]. Ein weiteres Ziel besteht darin, die Voraussetzungen für solch einen Wizard zu

[17] Der Versuch diese Informationen mit einer akzeptablen Laufzeit mittels SWT darzustellen ist bis zum Zeitpunkt der Abgabe nicht umgesetzt worden.[29]

[18] Definition Ressource: in Kapitel 4.3

Eclips	IntelliJ	Saros Kern	Bedeutung
Workspace	Project	IWorkspace	Höchstes Level der Organisationsherachie innerhalb einer IDE
Project	Module	IProject	Eine diskrete Einheit von Funktionalität, welche unabhängig kompiliert, ausgeführt und getestet werden kann. Diese Einheiten können untereinander abhängig sein.

Tabelle 1: Terminologie der verschiedenen IDEs und des Saros Kern Moduls

schaffen. Auf die Umsetzung und Hürden wird in 4.3.1 eingegangen.

3.2.3 Join-Session-Wizards

Für diesen gilt dasselbe, wie für den SessionInvitationWizard. Er setzt sich aus zwei Teilen zusammen. Zuerst wird der Nutzer über den Host der Session und die geteilten Ressourcen informiert [19]. Nimmt dieser an, wird in einem zweiten Fenster ein Projekterstellungs-Wizard[20] angezeigt, der dem Nutzer die Möglichkeit bietet, ein neues Projekt anzulegen oder ein schon vorhandenes zu verwenden. Während der erste Wizard recht einfach umgesetzt werden kann, da dieser nur Textinformationen enthält, stellt der zweite Wizard zum Erstellen der Projekte aus den empfangenen Dateien ein größeres Problem dar. Sollten diese Probleme gelöst werden, könnten beide kombiniert werden, um wie beim SessionInvitationWizard eine bessere Nutzererfahrung zu bieten.

Das Problem liegt vor allem darin begründet, dass Intellij IDEA und Eclipse ein unterschiedliches Konzept darüber haben, wie Ressourcen gekapselt und organisiert werden. Wie in Tabelle 1 zu sehen ist, verwendet das Kerninterface eine zu Eclipse fast identische Begriffswelt. Dies ist historisch bedingt, da Saros als Eclipse-Plug-In konzipiert wurde und somit auf dessen Konzeption und Organisationsstruktur zugeschnitten ist. Ob diese Übernahme der Organisationseinheiten von Eclipse für Saros weiterhin effektiv ist, müsste beim Voranschreiten der Saros/I Version nochmals evaluiert werden. Das einfache Verpacken von Eclipse-Interfaces erscheint für die Nutzung auf verschiedenen IDEs als nicht zweckdienlich, da bestimmte Funktionen, die in Eclipse existieren, in anderen IDEs nicht vorkommen müssen. Dies führt zum umständlichen Nachahmen von Eclipse- Funktionalität in IntelliJ. Auf daraus resultierende Probleme für die Cross-Plattform-UI wird im Bereich **??** eingegangen.

[19] In Saros/E *DescriptionPage*
[20] In Saros/E *EnterProjectNamePage*

Unabhängig davon, wie das Backend in Bezug auf Organisationseinheiten und das Filesystem konzipiert ist, zeigt sich für die GUI ein Terminologieproblem. So ist ein *Projekt* für einen Intellij IDEA Nutzer etwas ganz anderes als für einen Eclipse Nutzer.

Beispiel Terminologieproblem
Gegeben sei ein Sessionaufbau mit einem Saros/E Nutzer **BobE**, der ein Eclipse Java Projekt **JavaP** mit einem Saros/I Nutzer **AliceI** teilen will.

BobE wählt JavaP in dem SessionInvitationWizard JavaP zum gemeinsamen Bearbeiten aus. AliceJ wird nun im JoinSessionWizard angezeigt, dass BobE mit ihr ein *Projekt* JavaP teilen will. Aus der Sicht von AliceJ ist diese Bezeichnung irreführend. In der Begriffswelt von IDEA gesprochen, handelt es sich bei JavaP tatsächlich um ein *Modul*. Ein *Projekt* würde in IntelliJ implizieren, dass nun ein Container mit mehreren Modulen geteilt werden soll.

Für AliceJ ist es im SessionInvitationWizard aus demselben Grund unverständlich, warum sie ein *Projekt* wählen soll, während ihr der Wizard nur *Module* anzeigt.

An diesem Beispiel erkennt man deutlich, dass es notwendig ist, die Begrifflichkeiten innerhalb der Saros-GUI über alle Saros-Varianten der verschiedene IDEs für Nutzer transparent zu machen. Generalisiert gesprochen, ist es auch für eine gemeinsame GUI-Komponente nicht zu vermeiden, dass Anzeigen IDE-spezifische Inhalte benötigen. Dies sollte sich in den Sarosanzeigen allerdings auf wenige Terminologien beschränken.
Wie auch beim SessionInvitationWizard treffen die Kriterien [1,2] auch auf den JoinSessionWizard zu. Der Aufwand ist verhältnismäßig hoch, die ProjektNegation-Klassen, welche für den JoinSessionWizard benötigt werden noch nicht in den Kern integriert sind. Da der Session-Aufbau ein zentraler, da zwingend notwendiger, Bestandteil von Saros ist, steht diesem Aufwand ein ebenso großer Nutzen gegenüber.

3.2.4 Fortschrittsanzeige

Diese verhält sich in allen IDEs gleich und sollte damit ebenso in die GUI integriert werden. Der Aufwand hierfür ist schwer abzuschätzen, da verschiedene Progressmonitore in Saros zum Einsatz kommen. Ein größeres Problem ist das Abschalten der alten „Fortschrittsanzeigen".
Der zu erwartende Nutzen ist, dass in der neuen GUI der Fortschritt des Session-Einladungsprozesses nicht mehr in gesonderten Fenstern angezeigt werden müsste, sondern sich innerhalb eines Wizards darstellen lässt. Auf

diese Weise einer kann den Nutzer von Anfang bis Ende ohne mediale Umbrüche begleiten werden.

3.2.5 Einstellungsfenster

Das Einstellungsfenster ist nicht für alle IDEs gemeinschaftlich nutzbar. Der Nutzen eines gemeinsamen Einstellungsfensters ist gering. So können nicht alle Einstellungen in einem gemeinschaftlichen Fenster dargestellt werden. Zwar existiert eine Schnittmenge von Einstellungen, die IDE unabhängig zum Einsatz kommen können (z.B. Netzwerk und Accounteinstellungen), aber sie verbleiben IDE spezifisch. Es ist nur möglich, eine Teilmenge von Einstellungen abzubilden. Dies führt dazu, dass einige nur über die IDE spezifischen Einstellungsfenster erreichbar sind und andere über die gemeinsame HTML GUI. Es ist fragwürdig, ob eine praktikable Trennung dieser Einstellungsmöglichkeiten über die verschiedenen Fenster realisierbar ist, damit es für den Nutzer nicht verwirrend ist, wo welche Einstellungen verfügbar sind.

Ein weiterer entscheidender Punkt ist, dass Einstellungen für Plug-Ins in Saros/E und Saros/J jeweils an einer zentralen Stelle vorgenommen werden. Diese IDE-typischen Stellen nicht zu nutzen, wäre höchst kritisch, da es IDE-Konventionen bricht und damit die Anwenderfreundlichkeit stark verringert. So bieten Eclipse „Saros Prefernces“-Fenster. Ein Eclipse Nutzer erwartet, dass Einstellungen für ein Plug-In in diesem Fenster verfügbar sind. Selbst wenn *alle Einstellungsmöglichkeiten* in einem gemeinschaftlichen HTML Fenster verfügbar wären, könnten die IDE typischen Einstellungsfenster nicht verschwinden, ohne die Konvention der Eclipse-Plugin-Verwaltung zu verletzten. So kann hier Codeduplikation nicht vermieden werden, ohne die Anwenderfreundlichkeit zu gefährden.

Das Einstellungsfenster wird aus diesen genannten Gründen nicht Bestandteil der HTML GUI werden. Angemerkt sei, dass Einstellungsmöglichkeiten durchaus über eine gemeinsame HTML erreichbar sein können. Das ist allerdings eine zusätzliche Möglichkeit und kein Ersatz für das IDE typische Einstellungsfenster. Es empfiehlt sich Einstellungen, welche bereits über Saros-Wizards verfügbar sind, zu zentralisieren, so z.B. die Farbwahl oder andere Einstellungsmöglichkeiten des Konfigurations-Wizards. Diese Funktionalität sollte mit der Umsetzung der jeweiligen Wizards eingeführt werden.

3.2.6 Whiteboard

Dieses Funktionalität existiert momentan nur für Saros/E.[30] Es ist noch unklar ob dieser in Saros/I mit ähnlichen Funktionsumfang umgesetzt werden kann. Pläne hierfür existieren allerdings bereits, siehe: `http://www.saros-project.org/gsoc/2015/ideas#html5-whiteboard`. Ob diese in ei-

ner einheitlichen HTML GUI darstellbar ist, und welcher Aufwand dafür betrieben werden muss, lässt sich erst abschätzen sobald dieser in IntelliJ umgesetzt wurde.
Da er nicht Bestandteil der Hauptfunktionalität von Saros ist und wegen des unbekannten Aufwands wird das Whiteboard im Rahmen dieser Arbeit nicht in die HTML GUI integriert werden. Sobald das Whiteboard-Feature für Saros/I umgesetzt wird sollte über eine zentralisierte Darstellung per HTML in Erwägung gezogen werden.

3.2.7 Saros-Chat

Diese können IDE unabhängig implementiert werden. Allerdings ist der Nutzen der Umsetzung eher fragwürdig. Dies betrifft nicht die Codebasis und Entwicklung von Saros, die von einer einheitlichen Chat GUI profitieren würden, sondern eher das Feature an sich. In fast allen Fällen finden Saros-Sessions in Kombination mit einer Telefonkonferenz (z.B. Skype) statt. Ohne diese ist ein effektives kollaboratives Arbeiten am gemeinsamen Code selten möglich. Da die Teilnehmer miteinander reden, ist der Nutzen des Chats eher gering. Text und Codefragmente können auch in dem normalen Editorfenster geteilt und Informationen schneller verbal vermittelt werden. Es gibt Überlegungen im Sarosteam, den Support für dieses Feature vollständig fallen zu lassen.

Aus diesen Gründen ist die Implementierung einer gemeinsamen Chat-GUI nach hinten gestellt worden. Bis abschließend geklärt ist, ob dieses Feature weiterhin angeboten wird, erscheint der Aufwand für die Umsetzung unnötig.

3.2.8 Anzeige von Awarnesse-Informationen

In Saros sind Informationen über die Aktivitäten anderer Nutzer verfügbar. Diese beinhalten z.B. die Informationen an welcher Codestelle ein Sessionteilnehmer gerade arbeitet. Das wird unter anderem im Hauptfenster dargestellt. Während die Anzeige nur den Dateinamen anzeigt, enthält sie auch die Informationen über die genaue Codestelle. So kann der Nutzer beim Klick auf den angezeigten Dateinamen direkt zur Codezeile springen, an welcher der Kontakt arbeitet. Saros/E benutzt diese Informationen auch, um andere Fenster innerhalb des Editors zu verändern. Da die HTML GUI nur Anzeigen innerhalb des Browsers manipulieren kann, ist es nicht ohne weitgreifende Anpassungen möglich, diese Veränderungen an anderen, nicht vom Saros-Plug-In erzeugten IDE Fenstern, zu vereinheitlichen. Unabhängig von dieser Einschränkung können diese Informationen in allen IDEs angezeigt werden. Daher soll das Auswerten von Awarness-Informationen durch die UI möglich sein und an den Browser weitergegeben werden können. Da dieses

Feature erst nutzbar ist, nachdem der Sessionaufbau mit der HTML-GUI umsetztbar ist, wird dieser Bereich der GUI dementsprechend priorisiert.
Der Aufwand für die Umsetzung hängt wiederum von der Verfügbarkeit der entsprechenden Aktivität im Kern-Modul ab. Diese Aktivitäten müssten von der UI ausgewertet, ein Modell für die Informationen gebildet und an das HTML-Frontend weitergereicht werden.

3.2.9 Konfigurations-Wizard

Dieser beinhaltet die wichtigsten Einstellungen, die für eine Sarosnutzung notwendig sind. Daher sind alle Teile, die benutzt werden, nicht IDE spezifisch und erfüllen somit Kriterium [1] und [2]. Da alle Einstellungen von diesen Wizard auch über andere GUI-E erreichbar sind (z.B. durch das Einstellungsfenster, welches alle Einstellungen enthält) ist der Nutzen als etwas kleiner einzuordnen. Die Umsetzung wird hinter den Awarnesse priorisiert.

3.2.10 Weiter Wizards und Dialoge

Nicht genauer besprochen werden weitere kleinere Teile der Saros GUI, wie der Feedbackdialog, Warnungsanzeigen und Ähnliches. Auch diese sollten vereinheitlicht werden. Jedoch macht es erst Sinn, diese Teile zu implementieren, nachdem die größeren Wizards in die HTML GUI integriert worden sind. Die HTML Warnungsanzeige wird nicht benötigt, solange noch keine Funktionalität da ist, die diese Warnung auslösen kann.
Eine Ausnahme besteht hierbei in dem Fehlerdialog. Ein Fehler während der Ausführung von Saros muss dem Nutzer kenntlich gemacht werden. Anstatt dafür jeweils neue kleine Fenster anzeigen zu lassen, ist es für die HTML GUI sinnvoller, diese innerhalb der gerade geöffneten Anzeige darzustellen. So soll für die Fehlermeldung eine Möglichkeit geschaffen werden, sie an den Browser weiter zu leiten, damit der Nutzer informiert werden kann. Im Prototyp ist dies durch ein einfaches “alert(FEHLER)“ gelöst. Das ist in den vorhandenen HTML-GUI zu verbessern.

3.2.11 Zusammenfassung

Die Tabelle 3.2.11 stellt die Ergebnisse der Untersuchung dar. Zu beachten ist bei dieser Tabelle, dass die meisten positiven Effekte und der Großteil des Nutzens erst mit der vollständigen Umsetzung der neuen gemeinsamen GUI zum Tragen kommen. Dies liegt vor allem daran, weil die IDE spezifischen GUIs und die HTML GUI nicht ohne größere Codeänderung zeitgleich nebeneinander laufen können. So sollten die alten Codefragmente erst dann gelöscht werden, sobald die neuen stabil laufen. Jede Aufsplitterung verringert den Nutzen einer einheitlichen GUI bzw. beherbergt oder verschlimmert die in Abschnitt 2.3 genannten Probleme. Werden neue GUI verändernde

GUI-E	Nutzen	Aufwand	Umsetzung
Saros-Hauptansicht	+++	+++	1.
Session-Invitation-Wizards	+++	+++	2.
Join-Session-Wizard	++	+++	3.
Fortschrittsanzeige	++	+	4.
Einstellungsfenster	-	+	Nein
Konfigurations-Wizard	+	+	5.
Awareness-Informationen	++	++	6.
Whiteboard	+	-	zu evaluieren
Saroschat	+	-	zu entscheiden
Weiteres	+	+	7.

Tabelle 2: Übersicht Analyse der zu vereinheitlichen GUI-E

- kein Nutzen — unbekannter Aufwand
\+ geringer Nutzen — geringer Aufwand
++ mittlere Nutzen — mittlerer Aufwand
+++großer Nutzen — hoher Aufwand

Features hinzugefügt, sollte immer eine Kosten/Nutzen- Abwägung stattfinden, ob es machbar und effektiv ist, diese IDE spezifisch oder durch eine gemeinsame GUI zu realisieren.

Die Reihenfolge der Implementierung basiert nicht ausschließlich auf dieser Analyse. Sie wurde maßgeblich durch die Bedürfnisse des zeitgleich entwickelten Frontends bestimmt. So war früh für das neue Frontend ein Usibility-Test geplant.[21] Dort sollte ein Sitzungsaufbau mit der neuen HTML GUI mit Probanden erprobt werden. Um diesen Test durchführen zu können, war es nötig, die Session bezogenen Wizards zuerst fertigzustellen.

3.3 Entwicklertest

Nach der Fertigstellung der Analyse der GUI-Elemente, untersuchte ich den aktuellen Status des Prototypen im Hinblick auf die Evolvierbarkeit des Systems. Während dieser Zeit, befand sich wurde dieser noch von Christian Cikryt entwickelt. So ist, wie in Abschnitt 2.3.2 beschrieben, ein Ziel des gemeinsamen UI die Verringerung von Fehlerpotenzial u.a. durch die Verbesserung der Codetransparenz. Dabei ist eine These, dass eine einheitliche und auf HTML Technologien basierte UI-Architektur die Entwicklung und Erweiterung vereinfacht, da diese eine höhere Codetransparenz haben als SWT. Zur Erinnerung sollen zudem Effekte von Änderungen leicht zu erkennen und Stellen die bearbeitet werden müssen für Entwickler einfa-

[21] [22]

cher auffindbar sein. In wie weit die Architektur des Prototypen diesen Anforderungen genügte sollte nach der Analyse der vorhanden GUI evaluiert werden. Angedacht war dafür ein Vergleich zum ehmaligen SWT/AWT basierten Ansatz. Die Suche nach Patches und Codeanpassungen in der alten GUI war jedoch wenig ergiebig. Es zeige sich, dass ein Entwicklungsprozessvergleich nur schwer zu bewerkstelligen ist. Gründe dafür sind unter andrem die geringe Anzahl an hauptsächlich GUI bezogenen Patches. Eine Rekonstruierung wie viele Patches tatsächlich die UI änderten, und wie groß diese Änderungen waren erwies sich als schwer da es keine geeignete Datengrundlage gab. Selbst wenn klar war, dass eine Änderung mit der GUI-Entwicklung zu tun hatte, war Aufgrund fehlender Prozessdaten nicht nachvollziehbar wie groß der Aufwand in der Entwicklung war.

Die verfügbaren Daten (wie Anzahl der GUI bezogenen Patches, Lines of Codechange) sind leider nur begrenzt aussagekräftig für die Frage wie fehleranfällig die Architektur der GUI war bzw. ist. Für diese Frage interessanter sind Kenngrößen des GUI-Entwicklunsprozesses, wie Entwicklungszeit, Arbeitsaufwand oder Anteil am Gesamtumfang der Entwicklung. Da diese aus genannten Gründen nicht verfügbar sind, müssten diese über einen längeren Zeitraum gemessen werden. Bei einer Grobdurchsicht der Patch-Historie von Saros[22] fällt auf, dass die GUI in Saros nicht gleichförmig entwickelt wird. Der Entwicklungsprozess der GUI kann als „ad hoc“ charakterisiert werden. So wurde die GUI meist nur angepasst, wenn neue Funktionen hinzugefügt oder entfernt wurden. Es scheint auch, dass einmal erstelltet GUI-E nur sehr selten später nochmal bearbeite werden. Meist nur in Verbindung mit Bug-Fixes. Eine Stichprobe einger GUI Änderungen der Vergangenheit zeigte, dass es teilweise zu einer Vermischung, bzw. ungenügende Kapselung von Anzeige- und Geschäftslogik kam. Eine genaue Analyse, welche für einen sinnvollen Vergleich nötig wäre, erschien recht Aufwendig, der Nutzen und die erwartete Aussagekraft eher fragwürdig. Daher verzichtete ich auf einen Vergleich der alten und neuen GUI. Für die Zielstellung der Fehleranfälligkeit (siehe 2.3.2), ist dieser Vergleich auch nicht zwingend erforderlich. Entscheiden ist die Frage, wie sich der aktuelle Prototyp verhält. Dafür wurde mit dem vorhanden Prototypen ein Entwicklertest gemacht.

3.3.1 Konzeption und Testaufbau

Hauptziel des Tests war es Einstiegshürden für neue Entwickler sowie Architekturprobleme für die Evolvierbarkeit aufzudecken. Ursprünglich geplant war ein iteratives Vorgehen. Es sollte nach dem ersten Testdurchlauf die erkannten Schwachstellen und Probleme behoben und in weiteren Iterationen validiert werden, um zu erkennen ob die Schwierigkeiten gelöst wurden.

[22] Es wurde ausschließlich das Datum, die Commit-Tags und die Überschriften der letzten 500 Commits ausgewertet. Zeitpunkt 15.03.2015

Getestet wurde auf dem Stand des Commits https://github.com/saros-project/saros/commit/1029060. Der Entwicklertest wurde so konzipiert, dass für die Umsetzung der Aufgabe keine Anpassungen an der Architektur des UI-Moduls oder des Frontendmoduls nötig war. So sollte anhand des Prototypen ein weiterer Dialog in das Hauptfenster eingeführt werden, mit deren Hilfe man Accounts Anlegen, Editieren und Löschen kann. Die genaue Aufgabenbeschreibung kann im Anhang A.5 eingesehen werden. Die dafür nötigen BrowserFunctions-Klassen waren vorhanden und das UI-Modul enthielt bereits einen ähnlichen Wizard. Notwendig zur Erledigung dieser Aufgabe war das Hinzufügen der entsprechenden BrowserFunktionen und ein Einbinden dieser in den neuen Dialog. Des weiter musste eine HTML Seite mit den entsprechenden Javascript-Aufrufen angelegt werden. Die Bearbeitungszeit betrug 120 Minuten und wurde an dem Rechner des Probanden durchgeführt, da die Arbeitsumgebung so natürlich wie möglich sein sollte. Beim Test wurde der Bildschirm des Probanden aufgezeichnet um später Auswertungen durchzuführen, außerdem wurde mit einen Mikrophone die Kommentare mitgeschnitten. Ich befand mich im selben Raum um bei technischen Schwierigkeiten oder Unklarheiten zu der Aufgabenstellung klärend eingreifen zu können. Es wurde bewusst vermieden, Tipps oder Hilfe zu der Lösung der Aufgabe zu geben, da diese vom Probanden selbst erarbeitet werden sollten.

Durchgeführt wurde der Test mit Bastian Sieker. Wie in 2.4.4 erwähnt war er zu dieser Zeit in der Vorbereitungsphase seiner Masterarbeit. Dadurch besaß Bastian ein größeres Vorwissen über die eingesetzten Technologien im HTML-UI-Bereich. Außerdem besitzt er Erfahrungen mit Javascript und Webentwicklung.

3.3.2 Ergebnisse des Tests

Ich will im Folgenden eine Liste aller erkannten Probleme aufführen:

Boilerplate-Code Viele Codefragmente im Javascriptbereich mussten kopiert und wiederholt werden. So entstand eine Menge Boilerplate Code[23]. Dies war zum Teil dem eingesetzten Web-Framework **Angular** geschuldet.[31]

Browserfunktion und Renderer Das Wiederverwenden von Browserfunktionen in neuen Dialogen war durch die Art, wie die Renderer konzipiert waren, nicht möglich.

Vorwissen Angular Ohne sehr gute Kenntnisse des Angluar-Frameworks ist die Funktionsweise des Frontends völlig unklar. Eine passende Template-

[23] Definition Boilerplate Code: https://en.wikipedia.org/wiki/Boilerplate_code

Engine erscheint nötig, um die Entwicklung zu vereinfachen.

3.3.3 Abgrenzungsproblematik nach Entwicklertest

In der Einleitung zu diesen Kapitel 3 ist das zeitliche Vorgehen zu Umsetzung der in Abschnitt Ziele genannten Ziele aufgeführt. Der Punkt ***3. Analyse und Umsetzung einer HTML-GUI für JTourBus*** war ursprünglich nicht geplant. Die Entscheidung für dieses Vorgehen fiel in einen gemeinsamen Diskussionsrunde aufgrund einer Abgrenzungsproblematik während der Bearbeitung. Diese Problematik will ich kurz erörtern.

Anlass war die Analyse des Entwicklertest. Die meisten der erkannten Probleme aus diesem fielen nicht in den Abgrenzungsbereich meiner Arbeit. Das Evaluieren eines passenden Javascirpt-Frameworks war zu diesen Zeitpunkt bereits ein geplanter Kernarbeitsbereich von Bastian Sieker. Damit konnten die Problematik des sich wiederholenden Javascript-Codes und eine Verbesserung der Benutzung des HTML-Frontends, welcher für eine weitere Iteration dringend Notwendig gewesen wäre, nicht von mir bearbeitet werden, ohne in den Arbeitsbereich von Bastian Sieker einzugreifen. Die erkannten Probleme im UI-Schnittstellenbereich waren zu dem Zeitpunkt noch im Hauptverantwortungsbereich von Christian Cikryt der in der Endphase seiner Bearbeitung bereits verschiedene Refactorings dafür plante. Das Anpassen und Verändern des Prototypen und hätte eine sehr viel engere Abstimmung mit Christian erfordert. Das zeitgleiche verändern der UI-Architektur ohne diese zeitintensive Abstimmung erschien mir nicht sinnvoll. Massive Änderungen und Erweiterungen an dem bestehenden Prototypen während der Bearbeitungszeit von Christian beherbergte zudem die Gefahr, eine klare Trennung, welche Designentscheidungen durch Christian und welche durch mich getroffen wurden, zu erschweren. Dies war ein Hauptunterschied zum "*normalen*" Arbeiten in einem Entwicklerteam. Normalerweise ist in diesem Fall nicht wichtig wer, wie stark an welchen Problemlösung beteiligt war, sondern allein das Ergebnis der gemeinsamen Arbeit. Im Rahmen einer Abschlussarbeit ist dies jedoch ein Argument, wenn es um die Abgrenzung der Arbeiten geht.

Da nicht klar war, wie groß die Änderungen innerhalb der Architektur des UI-Moduls noch sein würden, erschien uns das Verbreitern der Schnittstelle durch weitere BrowserFunktionen als wenig produktiv während Christian noch die Architektur des Prototypen finalisierte. Nach Analyse der verschiedenen GUI-E in 3.2.11 war von mir geplant zuerst die Session-Bezogenen Wizards zu implementieren. Für diese kam erschwerend hinzu, dass einige der nötige Komponenten für die Umsetzung Session-Invitation zu diesem Zeitpunk noch nicht im Kern integriert waren.[24] Nach einer größeren Diskus-

[24] Siehe 2.4.3

sionsrunde mit allen beteiligten Entwicklern und dem Sarosteam entschied ich mich die Erweiterungen, bis zur Fertigstellung der Arbeiten von Christian Cikryt und Arndt Lasarzik, nach hinten zu stellen. Diese Zeit nutzte ich stattdessen mit der Analyse und Umsetzung einer HTML-GUI für das JTourBus-Plug-In.

3.4 Entwicklung einer HMTL-GUI für JTourBus

Das JTourBus Plug-In bietet eine einfache Möglichkeit für das Verständnis eine bestehenden Codebasis. Saros verwendet JTourBus um Codetouren anzubieten, welche neue Entwickler zu bestimmten Themen direkt durch den Code begleitet.[32] JTourBus ist nur für Eclipse erhältlich. Ziel der Analyse war es die schmale Benutzeroberfläche des Plugins in eine HTML basierte zu überführen, um auf Grundlage dieser das Plug-In auch für IntelliJ anbieten zu können. Prämisse dabei war, dass JTourBus als wesentlich kleineres Projekt mit überschaubaren Rahmen sehr viel schneller und einfacher in HTML-GUI überführbar ist als Saros. Dabei sollten Erkenntnisse bei der Umsetzung später auf den Saros überführt werden.

Diese Prämisse stellte sich jedoch als falsch heraus. Zwar ist die Codebasis des JTourBus, im Folgenden mit JTB abgekürzt, wesentlich kleiner als die Saroscodebasis, jedoch ist der Code schlecht dokumentiert und unübersichtlich gekapselt. In einem ersten Schritt sollte Kern-Funktionalität und Anzeigelogik getrennt werden. In einer prototypischen Implementierung zeigte sich, dass JTourBus stark an die Nutzung mit SWT angepasst war. So ist die gesamte verwendete Datenstruktur und Geschäftslogik auf das Zusammenarbeiten mit SWT-Bäumen[25] ausgelegt. Während dies für ein Eclipse-Plugin sinnvoll und nützlich ist, erschwert es jedoch die Überführung in eine SWT unabhängige Version. Das Bearbeiten wurde zusätzlich durch das alter der Codebasis erschwert. So wurde JTourBus für Java 1.5 entwickelt und viele Möglichkeiten die neue Java Versionen bieten, konnten bei der Entwicklung des JTourBus so nicht genutzt werden. Trotz dieser Schwierigkeiten wurde im Prototyp des HTML-JTourBus die grundlegenden Voraussetzungen für eine Anzeige in HTML geschaffen:

Datenmodelle Es wurden Modelle für die genutzten Datensturkturen geschaffen, welche zu JSON serialisiert werden können. [26]

Java-JavaScript-Adapter Alle vom Benutzter möglichen Interaktionen des Plug-Ins wurden erfasst. Wie auch in der Saros-GUI werden diese Java-Methoden über, durch Javascript aufgerufene *BrowserFunctions*, implementiert. Diese BrowserFunctionen stellen die Schnittstelle zwischen UI und

[25] SWT-Tree http://goo.gl/zs0Fm3
[26] JSON = JavaScript Object Notation http://json.org/

Geschäftslogik da. Alle benötigten Funktionen sind in *HtmlJavaScriptAdapter*-Klasse gekapselt. So ist die Schnittstelle zwischen Javascript und Java an einen zentralen Ort möglich. Diese Schnittstelle wurde gut dokumentiert und die noch erforderlichen Arbeiten in ihr festgehalten. Siehe dafür Anhang A.4

Browser Komponente Ein IDE unabhängiger *BrowserCreator* wurde implementiert. Dieser ermöglicht, ebenso wie im Saros Prototypen von Christian Cikryt, das Erzeugen neue Browser-Instanzen innerhalb einer SWT-Shell[5, S. 47].

HTML-Ressourcen Eine prototypische HTML Seite, sowie Bootstrap und jQuery sind ebenfalls in das Projekt integriert worden.

Der Code zu dieser Entwicklung sind zurzeit in einen lokalen Repository gespeichert. Ich plane jedoch die gemachten Änderungen noch auf den Masterbranch[27] des Projektes zu integrieren, damit auf dieser Basis daran weiter entwickelt werden kann.

In der Summe stellte sich die Umsetzung des Cross-GUI-Ansatzes für den JTourBus als viel schwieriger heraus, als ursprünglich angenommen. Durch diese Probleme war ein Abschluss des Vorhabens in der verbleibenden Zeit nicht mehr möglich. Da die Arbeiten von Christian und Arndt zu diesem Zeitpunkt beendet waren, und Bastian die Fertigstellung des UI-Backends für seine Arbeit benötigte setzte begann ich mit der Erweiterung des HTML-Prototypen. Die dort gemachten fortschritte sind im folgenden Kapitel 4 beschrieben.

Ich empfehle auf Basis des entwickelten JTB-Prototypen die vollständige Umsetzung für IntelliJ in zukünftigen Arbeiten. Die Möglichkeit der Nutzung des JTB in IntelliJ erscheint lohnenswert für das Sarosprojekt, da die Entwicklung von Saros mit dem voranschreiten der Saros/I Version zunehmend auch in IntelliJ stattfinden wird. Dabei kann JTB die Einstiegshürden für neue Saros/I Entwickler, durch gut Konzipierten Code-Touren, enorm senken.
Für die Umsetzung des JTourBus in Eclipse und IntelliJ mit einer gemeinsamen HTML-GUI sehe ich noch folgende verbleibende Arbeitsschritte:

1. Neuimplementation der Geschäftslogik. Dies beinhaltet die neu Konzeption der genutzten Datenstruktur und zum speichern der Informationen gefunden JTourBus-Annotationen.

[27] http://saros-build.imp.fu-berlin.de/gerrit/#/q/project:jtourbus

2. Da zu erwarten ist, dass die Eclipse-Implementation nicht genauso in IntelliJ umgesetzt werden kann, müsste für den *HtmlJavaScriptAdapter* ein Kern-Interface geschaffen werden, welche IDE unabhängige Methoden bereitstellt.

Der Aufwand dafür lässt sich nur schwer abschätzen. Während die Eclipse Logik zumindest teilweise wiederverwendet werden kann, muss zumindest die Kernlogik für IntelliJ neu entwickelt werden. Es empfiehlt sich dringend zuerst die Kern-Komponente fertig zu stellen, bevor die HTML-GUI finalisiert oder die IntelliJ Version entwickelt wird. Dies erleichtert die Entwicklung der HTML-GUI deutlich. Für das Frontend erscheint mir der Einsatz größerer Javascript-Framworks Aufgrund der geringen Komplexität der JTB-Anzeige als unnötige Overhead.

4 Erweiterung der HTML GUI

Um die in 3.2 analysierten GUI-E in der HTML-GUI verfügbar zu machen, wurde der von Christian Cikryt erstellte Prototyp angepasst und erweitert. In diesem Kapitel werden diese vorgenommenen Anpassungen und Erweiterungen an dem UI Prototypen besprochen.
Ergänzend zu den Ausführungen von Christian Cikryt über die Kapselung des neuen UI als gesondertes Modul möchte ich mich kurz zu den Gründen für diese Aufteilung äußern, da ich an dieser Entscheidung mitgewirkt habe.

Für die Wartung und Kapselung der gemeinsamen UI waren zwei Lösungswege möglich. Die HTML-GUI innerhalb des Coremoduls zu entwickeln oder hierfür ein eigenes Modul einzuführen. Hauptargument für eine Kapselung der UI als extra Modul war die strikte Trennung von Anwendungslogik und Benutzerinterface im Sinne von SoC[28]. Diesem Ansatz ging die Überlegung voraus, was der Inhalt des Saros-Kern-Moduls, im Folgenden als Kern bezeichnet, konkret sein sollte und was nicht. So kann argumentiert werden, dass die gemeinsame UI offensichtlich ein zentraler Bestandteil der Software ist und somit direkt in den Kern, z.B. als Paket, in die Saros-Codebasis integriert werden sollte. Soll das Kern-Modul als eine Gemeinschaftsbasis von in jeder der verschiedenen IDE Versionen von Saros benutzten Funktionalität sein, ergäbe sich tatsächlich diese Implikation. Der Kern sollte jedoch nicht als ein 'Sammelbecken' für alles 'Gemeinsame' der verschiedenen Saros Plug-Ins verstanden werden, sondern als wirklichen *Kern* der Saros Logik. Zu dieser gehört die Benutzeroberfläche (offensichtlich) nicht. Eine Verwendung des Kern-Moduls ist eine reine Server-Version von Saros, welche sich zur Bearbeitungszeit bereits in der Entwicklung befand[12]. Diese braucht keine Anzeigelogik und basiert auf dem Kern. So wird ersichtlich dass GUI-Code in diesem Kern-Modul deplatziert ist. Aus diesen Gründen wurde sich dagegen entschieden, UI Komponenten im Kern zu entwickeln.

Mit dieser Arbeit wurde die strikte Trennung von Anzeigenlogik und Geschäfts-

logik fortgesetzt und erweitert. Dafür wurde ein UI-Frontend-Modul als zusätzliche Abstraktionsschicht hinzugefügt. Dieses Frontend-Projekt wird in der Arbeit von Bastian Sieker entwickelt und ausführlich besprochen [22]. Zu beachten ist, dass dieses Modul bei Fertigstellung dieser Arbeit noch nicht vorlag. Daher können sich hier genannte Informationen zum Inhalt dieses Projekts noch ändern. Im Folgenden wird dieser Part kurz Frontend genant, das Backend beschreibt dabei vornehmlich das UI-Modul und alles, was dahinter gelagert ist.

[28] Separation of Concerns [33]

Diese Trennung wurde zum einen durch den wachsenden Umfang des Webteils notwendig. Zum anderen wird dadurch konsequent das Geheimhaltungsprinzip umgesetzt [**ProgramPragmatics**], da für das UI Modul die Kenntnis der konkreten Umsetzung der angezeigten Webseiten nicht notwendig ist. Die einzigen Informationen, die bekannt sein müssen, sind der Speicherort der HTML Seiten sowie die verfügbaren JavaScript Methoden des Frontends. In Abb. 2 sind die Abhängigkeiten der verschiedenen Saros-Module untereinander dargestellt. In diesem Abschnitt wird öfter von den in der Abb. 2 gezeigten Begrifflichkeiten gesprochen. Daher möchte ich eine kurze Abgrenzung und Definition vornehmen:

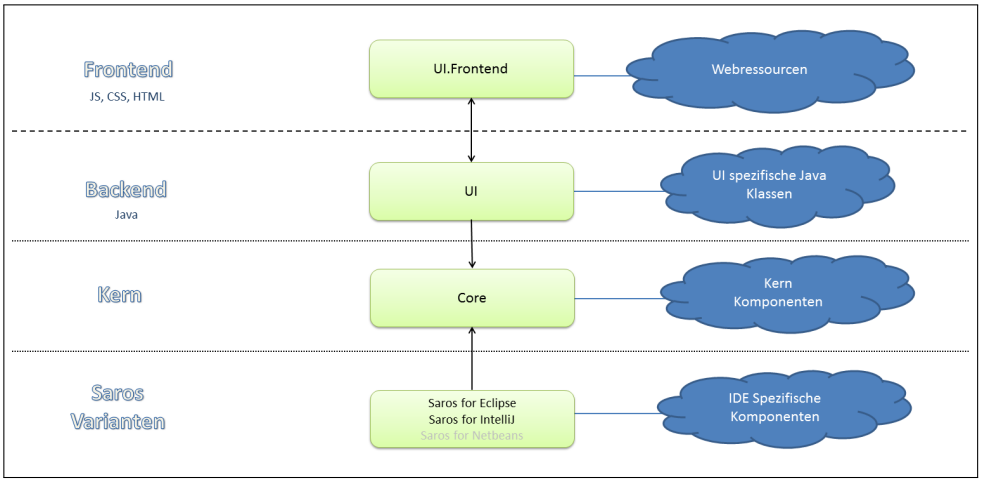


Abbildung 2: Begriffswelt, Abhängigkeiten und Inhalt der Sarosmodule

Frontend kapselt alle Webelemente der GUI. Dies umfasst Webressourcen wie CSS-, HTML-, Template- und JavaScriptdateien. Des Weiteren enthält es webbasierte Frameworks, Bibliotheken und eine Testumgebung. Dieser Bereich ist nicht Teil der Arbeit. Anforderungen vom Frontend haben jedoch Einfluss auf das UI-Modul gehabt.

Backend kapselt die Aufrufe des Frontends (Benutzereingaben) und wandelt sie in Kernaufrufe um. Das Gleiche passiert für Aufrufe aus dem Kern (bzw. den verschieden Saros Versionen) und übersetzt sie an das Frontend. Somit stellt es eine Zwischenschicht zwischen Programmlogik und Benutzeranzeige dar. Das UI Modul muss dabei IDE unabhängig arbeiten, daher ist die einzige Abhängigkeit der Saroskern.

Kern beinhaltet Saros Versions unabhängige Klassen und Methoden. Das Kernmodul war während dieser Arbeit in der Entwicklung: Siehe dazu 2.4.3. Da die UI direkt von dem Funktionsumfang des Kerns abhängig ist, erfolgten hier notwendige Anpassungen.

Saros/E -/I beinhaltet die jeweiligen IDE spezifischen Klassen und Methoden der Plug-Ins. Saros/E ist dabei wesentlich weiter entwickelt als Saros/I. Saros/N (für Netbeans) existiert nur als Proof-of-Concept[13] und wurde für die Entwicklung nicht weiter beachtet.

Aus dieser Kapselung ergibt sich der programmatische Hauptarbeitsbereich dieser Arbeit: Neben den Erweiterungen des UI-Projektes an sich, musste an verschiedenen Stellen sowohl das Kern-Modul als auch das UI-Modul an die Anforderungen des Frontends angepasst werden.

4.1 Anpassungen am Rendering des Programmstatus

Die Darstellung der Information aus den verschiedenen Saros Versionen geschieht in mehreren Schritten: [5, S. 45-46]

1. **Datenbeschaffung**
   Die benötigten Informationen zur Anzeige im Frontend werden durch die vom Kern bereitgestellten Methoden akkumuliert.

2. **Modellerstellung der Daten**
   Diese so akquirierten Daten werden in entsprechenden Modellklassen gespeichert.

3. **Aufruf des renderings in JS**
   Diese Modelle werden serialisiert und an das Frontend über die JS-API weitergereicht. Das eigentliche Anzeigen der Daten (z.B. durch entsprechende Manipulation des DOMs in der geladen HTML Seite) findet dort statt.

Diese Schritte werden durch die sogenannten Rendererklassen realisiert. Ich schließe mich der Argumentation von Christian Cikryt an, dass eine konsequente Einhaltung des Beobachter-Musters [34] in wünschenswert ist, da sie Abhängigkeiten reduziert. Die Rendererklassen halten, anders als er schreibt, allerdings nicht tatsächlich die Informationen über die Daten zum Anzeigen. Die Informationen zum aktuellen Status der Sarosanwendung werden ausschließlich in den jeweiligen Modellklassen vorgehalten. Die Renderer lösen lediglich das Aktualisieren dieser Modelle aus, sobald es erforderlich ist.
Da die Renderer für das Erlangen, Verarbeiten und Weiterleiten der Modellinformationen verantwortlich sind, erscheint mir die Namensgebung hier ungünstig gewählt. Die Funktionsweise eines Renderers entspricht eher dem eines Controllers des MVC-Patterns[35]. Zwar gehört auch das Anstoßen der Anzeige der Daten zu den Aufgaben dieser Klasse, ist jedoch nicht Kern seiner Aufgabe aus programmatischer Sicht. Ich sehe die Gefahr, dass die Namensgebung irreführend sein könnte. So lässt sie vermuten, dass ein Renderer Programm-Logik für die konkrete Anzeige der Daten enthält, z.B. durch die Erzeugung von HTML Code oder die Veränderung des DOMs. Dieser Part ist allerdings in das Frontend ausgelagert und somit nicht Aufgabenbereich dieser Klassen. Nach einer Diskussion über die Namensgebung dieser Klassen auf der Mailingliste des Saros Projekts habe ich mich dafür entschlossen, diese Änderung nach hinten zu schieben, da kein Konsens für eine bessere Namensgebung gefunden wurde. Ich halte dies jedoch für eine angebrachte Änderung mit dem Potenzial, die Codetransparenz für Entwickler zu erhöhen. Ein Vorschlag wäre, hier die Namensgebung dem MVC-Muster anzupassen:

- Rendererklassen sollten von XYZ-Renderer in XYZ-Controller umbenannt werden, damit die Aufgabe der Klasse klarifiziert wird.

- Aus dem gleichen Grund sollten Modelklassen das Suffix - 'Model' erhalten. Dies würde auch eine einheitlicher Namensgebung dienlich sein, so sind alle BrowserPages XYZ-Pages, und alle BrowserFunctions mit XYZ-Functions benannt worden.[29]

Die ursprüngliche Version des Prototypen sah vor, einen Renderer in einem Browser einzusetzen. Dies war zu diesem Zeitpunkt effektiv, da jeder Renderer in genau einem Browser eingesetzt wurde. Mit der Implementation des Session-Invitation-Wizard als neuer Dialog wurde dies jedoch unpraktikabel. Der *ContactListRenderer* bietet die Möglichkeiten, die aktuellen Kontakte und deren Status an einen Browser im Frontend weiterzugeben. Da nun mehrere Browser diesen Renderer nutzen müssen, wurde die Basisklasse dahingehend angepasst, als das sie nun eine Liste von Browsern verwaltet[30].

Ergänzend zu den Ausführungen von Christian möchte ich hinzufügen, dass die Modellklassen innerhalb des UI-Moduls als *Data-Transfer-Object* konzipiert werden.Diese sind vor allem nützlich beim arbeiten mit Remote-Interfaces da sie die Anzahl der erforderlichen Methodenaufrufe reduzieren. [36] Das Javascript-Frontend stellt so ein Interface dar. So erfüllen diese Klassen allgemeinen den Zweck die Zustände des Kerns (Backends) für die Anzeige zu kapseln. Dies ermöglicht den zentralen, von den jeweiligen IDE-Implementierungen unabhängigen, Zugriff auf den Zustand von Saros, welche für die Anzeige benötigt werden. Das dies unabhängig von der Frontendendtechnologie möglich sein soll, wurde bei der Konzeption des ProjektList-Models (4.3.3)beachtet. So werden in der jetzigen Version diese Modelle zwar zur Serialisierung von JSONs für das Javascript-Frontend benutzt, sie könnten jedoch auch von anderen Anzeigetechnologien (z.B. JavaFX) verwendet werden.

4.2 JavaJavascript Schnittstelle

Bevor ich auf die Implementation des SessionInvitationWizards eingehe will ich die wichtigsten vorgenommen Architektur-Veränderungen am UI-Modul in diesem Abschnitt diskutieren. Wie in Abb.2 zu sehen ist, bildet das UI-Modul die Schnittstelle zwischen der Anzeige im Frontend und der Geschäftslogik im Kern. Mit der Aufsplittung der GUI in Frontend und Backend und dem voranschreiten beider Module, wurde auch Anpassung an der Java<->Javascript Schnittstelle nötig.
Die beiden zentralen Konzepte sind dabei die BrowserFunktions und die von mir eingeführte JavaScriptAPI-Klasse.

[29] Diese Änderung befindet sich zur Abgabezeit im Reviewprozess siehe A.1

[30] 'Renderer can be used in multiple browsers' https://github.com/saros-project/saros/commit/bdd1e

4.2.1 Verbesserung der BrowserFunction Beschreibungen

Das Package BrowserFunctions innerhalb des UI-Moduls stellt die Schnittstelle zum Backend für das UI-Frontend dar (siehe auch Abb.7 in Abschnitt5.2.4). D.h. alle Java-Methoden die vom Frontend (=Javascript) innerhalb eines Browsers aufgerufen werden können, sind in solchen BrowserFunction-Klassen gekapselt. Ich möchte in diesem Abschnitt die erfolgten Änderungen an diesem Package erörtern. Diese hatte verschiedene Schwachstellen:

- Eines der Hauptprobleme bei der Verwendung war das Fehlen von Java-Dokumentation für BrowserFunctions. So war es, ohne das lesen der Implementation einer BrowserFunction, unmöglich für Entwickler zu erkennen wie die erzeugte Javascript Funktion heißt, welche Werte sie erwartet und was sie tut. Das hinzufügen dieser JDok. war zudem gar nicht möglich, da diese durch anonyme Methoden bereitgestellt worden sind. Für jede erzeugte Javascript-Funktion existiert nun eine eigene Java-Methode welche durch J-Dok dokumentiert wurde. Auf die Vorteile dieser Änderung werde ich gleich gesondert eingehen.
- Alle BF-Klassen wiesen ein gleiches verhalten auf. Jede implementierte eine "GetJavaScriptFunctions()Methode. Um diese Codeduplikation zu eliminieren wurde eine abstrakte BF-Klasse eingeführt von der alle andern BF-Klassen erben.
- Da alle BrowserFunctions sind als Singeltons realisiert. Da sich die erzeugten JS-Funktionen zur Laufzeit von Saros nicht ändern, sollten diese auch nur einmal erzeugt werden. Statt die JS-Funktionen bei jeden laden einer BrowserPage zu generieren, werden diese nun bei Instanziierung der jeweiligen BrowserFunktions-Klasse erzeugt. So wird das mehrfache Erzeugen der JS-Funktionen, durch das laden neuer BrowserPages jetzt verhindert. 'GetJavaScriptFunctions()' gibt nun eine, in der Klasse gespeicherte Liste zurück, statt diese beim Aufruf zu generieren.

Die Kapselung der Browserfunktionen ist momentan größtenteils Wizard gebunden. So existiert eine BrowserFunktion-Klasse (BF-K) für die MainPage, eine für den StartSessionWizard usw. Ein Problem dieser Kapselung ist, das die BF-Organisation zu eng mit den vorhanden GUI-E verwoben. So konnte einer Browser-Instanz nur ALLE vom ContactBrowserFunctions erzeugten Metoden injiziert werden. Es war daher nicht möglich einem (neuen) GUI-Element nur bestimmte Methoden einer BF-Klasse zu injizieren. Dies schränkt die flexible Nutzung in der Anzeige ein.
Effektiver wäre eine Kapselung der Funktionen nach Themen - so könnte eine SessionBrowserFunktion-Klasse alle Methoden enthalten, die Session bezogene Aktionen bieten. Dies löst jedoch immer noch nicht das Problem,

das wieder alle, oder keine diese Funktionen injiziert werden müssten.
Durch das einführen von dedizierten, statt anonymer, Methoden konnte nicht nur die dringend benötigten Java-Dokumentation zu dieser zentrallen Schnittstelle hinzugefügt werden - sie bietet jetzt zudem die Möglichkeit einzelne Javascript Methoden auszuwählen. Das hinzufügen von BrowserFunktions-Bundle (wie z.b. Account-Specific-Browserfunctions) kann dabei weiterhin durch entsprechende Getter-Methoden und Listen erreicht werden.
Im weiteren Verlauf der Entwicklung, ist es in meinen Augen sinnvoller diese nicht mehr Thematisch, oder Dialogbezogen in verschiedene BrowserFunktions-Klassen zu kapseln. So ist zu evaluieren, ob es nicht generell effektiver und besser für die Evolvierbarkeit wäre eine **zentrale BrowserFunktions-Klasse** einzuführen. Damit wären die Schnittstelle von Javascript->Java in einer UI-Klasse zentralisiert. Die im Prototypen praktizierte Trennung dieser Funktionen über mehrere Java-Klassen, aufgrund einer mutmaßlichen Verwendung durch das Frontend erscheint fragwürdig. Eine Schnittstelle sollte keine Annahmen über ihre zukünftige Verwendung treffen und besser einen flexiblen Zugriff ermöglichen. Diese 'BrowserFunctionsAPI' wäre damit ein Pendant zu der Java->Javascript Schnittstelle, die mit der JavaScriptApi-Klasse eingeführt wurde.(Siehe **??**)
Ein möglicher Nachteil wäre enorme Größe diese zentralen Java-Klasse. Wobei die Größe per se kein Problem sein muss. Ein Logische Aufbau der Klasse,mit entsprechenden Kommentare und vorbereiteten Funktionslisten kann den Überblick erleichtern. Ein weitere Vorteil ist die gewonnene Flexibilität in de Benutzung dieser Schnittstele. So könnte ein neues GUI-E sich die gewünschten Methoden aus den (nun gut dokumentierten Funktionen) an einer zentralen Stelle aussuchen. Auch muss ein Entwickler keine Mutmaßungen mehr darüber treffen, welche BrowserFunktion in welche Klasse zu finden sind.

4.2.2 Einführung einer zentralen JavaScriptAPI-Klasse

Das Frontend basiert auf einem MVC-Muster[22], d.h. die Anzeige im Frontend wird aktualisiert, sobald sich ein Datenmodell verändert. So werden vom Backend keine direkten Aktionen wie SZeige Projekte anän das Frontend ausgelöst sondern lediglich Datenmodelle aktualisiert. Diese Modelle werden vom Backend generiert und zu JSON serialisiert. Das genaue Vorgehen ist bereits in 4.1 beschrieben. Diese JSON werden an im Frontend vorhanden Javascript Methoden, sogenannte 'Events' übergeben, welche die Aktualisierung des entsprechenden Models bewirkt.

Verschiedene Klassen des UI-Moduls müssen auf im Frontend hinterlegte Javascript-Methoden zurückgreifen. Die verfügbaren Methoden dieser Javascript API sind momentan dezentral über verschiedene Javascript-Dateien verteilt. In dem Prototypen waren die Aufrufe direkt in die jeweiligen Klas-

sen implementiert, die diese nutzen. So ergab sich die in Abb. 3 dargestellte Schnittstelle. Wie man sieht, ist es nur schwer nachvollziehbar, welche Klasse auf welche JS-API Methode zugreift. Dieses Problem verschlimmert sich dadurch, dass keine Java-Dokumentation für diese Aufrufe zur Verfügung stand. Es ist für einen Entwickler nicht nachvollziehbar, welche Methoden innerhalb des UI-Moduls zu Verfügung stehen, wie diese zu verwenden sind und was sie genau bewirken. Da es keine sichtbare Abhängigkeit gibt, die von den IDE über die verschiedenen Sprachen erkannt wird, war es aufwendig, heraus zu finden, welche Methode mit welchen Parametern aufgerufen werden musste. Vor allem das völlige Fehlen einer Java-Dokumentation macht es für einen neuen Entwickler unmöglich, die vorhandenen Javascript Methoden zu nutzen. Es erforderte genaue Kenntnis der Interna des Frontendmoduls, um effektiv mit der Schnittstelle entwickeln zu können.

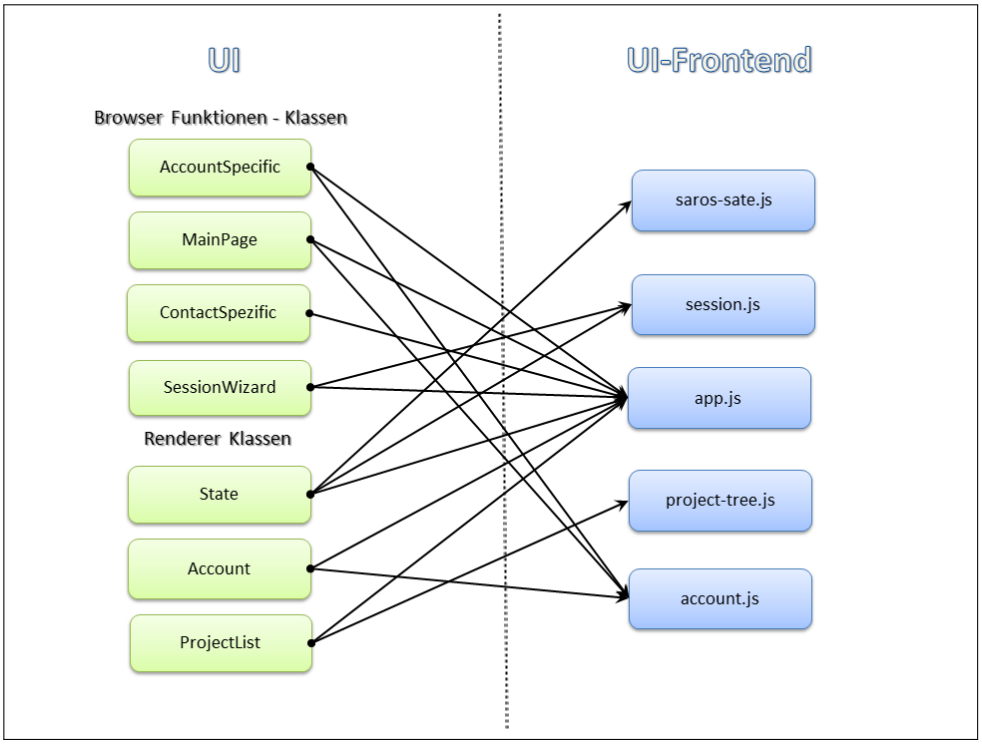


Abbildung 3: Java zu Javascript-Schnittstelle vor Einführung einer Fassade

Dieser Zustand stand im direkten Widerspruch zu dem in Abschnitt 2.3.3 formulierten Ziel, die UI-Architektur evolvierbar und transparent zu gestalten.
Dieses Problem wurde durch das Einführen einer Fassade behoben. Die neue resultierende Schnittstelle wird in Abb. 4 dargestellt. Diese bündelt alle Aufrufe, die an das Frontend möglich sind, an einer zentralen Stelle in der UI.

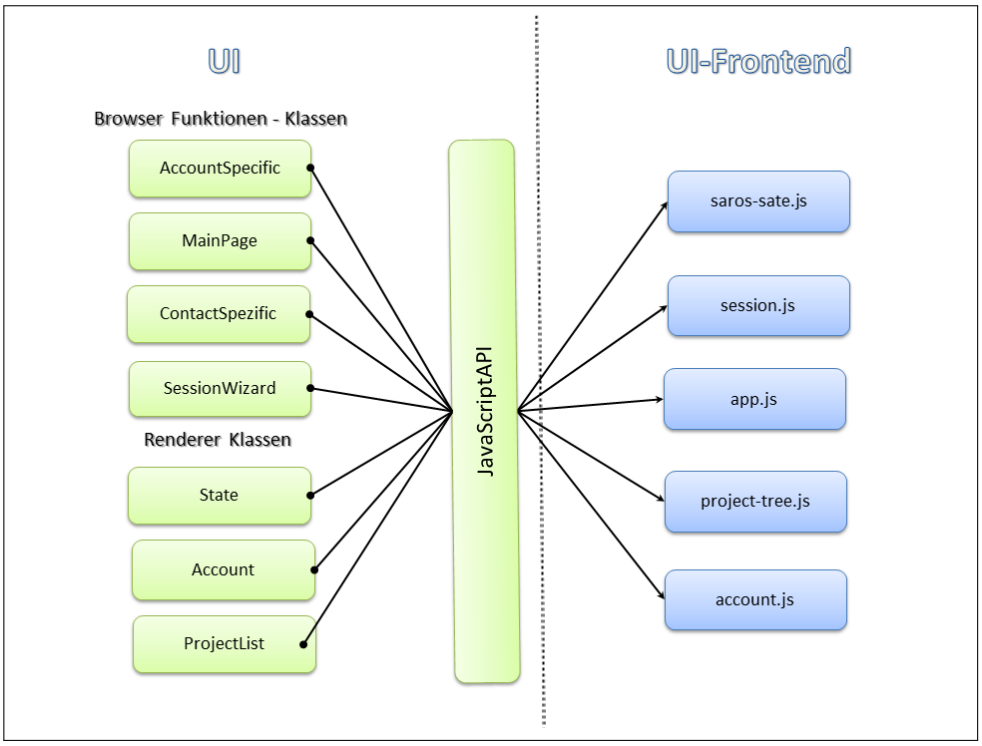


Abbildung 4: Java zu Javascript-Schnittstelle nach Einführung einer Fassade

Außerdem werden die tatsächliche Javascript Aufrufe durch umgebende Javamethoden abstrahiert, wie in Abb. 5 zu sehen ist. So ist es nun möglich, für Javascript Aufrufe im Frontend Java-Dokumentation zu schreiben. Damit gelingt es effektiv, die Wirkung, Verwendung und den Wiedergabewert einer Javascript-Methode des Frontends kenntlich zu machen. So wird eine genaue Kenntnis der Frontend-Interna nicht mehr benötigt.

Diese Anpassung erleichtert auch künftige Entwicklungen. Veränderungen im Frontend wirken sich nur noch auf eine Stelle der Architektur, der JavaScriptAPI Klasse aus. Die Serialisierung der Modellklassen erfolgt nun an einen zentralen Ort. Ehemals waren alle Klassen von der GSON-Bibliothek abhängig. So ist das Austauschen der Datenserialisierung ohne größeren Aufwand möglich.

Im Zuge der Erweiterung der HTML GUI sollten alle zukünftigen *Java zu Javascript* Aufrufe zentral durch diese Klasse verwaltet werden. Javascript Entwickler, sollten vom Backend Aufrufbare Methoden hier bekannt machen und dokumentieren.

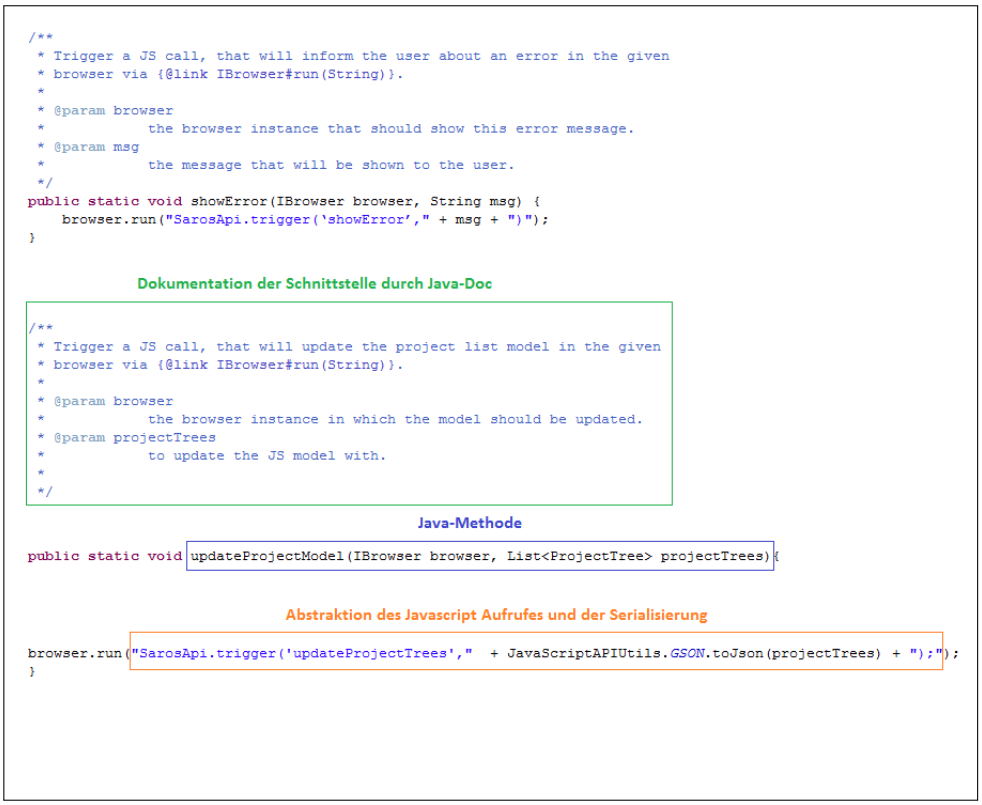


Abbildung 5: Schnittstellenbeschreibung und Abstraktion in der JavaScriptAPI Klasse

4.3 Implementation des SessionWizards

Nachdem Christian Cikryt seine Arbeit am Prototypen und Arndt Lasarzik seine Arbeit an der Erweiterung des Kerns beendet hatten, und damit die Abgrenzungsproblematik[31] nicht mehr vorhanden war, fing ich mit der Implementierung des Session-Invitation-Wizards[32] (im Folgenden SIW) an. Die Arbeiten am JTourBus-Plug-In beendete ich, da es nicht absehbar war ob dieser in der verbleibenden Zeit noch vervollständigt werden könne.

Begriffsdefiniton Ressource
Als Ressource wird jede Datei verstanden, die ein Sarosnutzer innerhalb einer Session teilen kann. Eine Ressource kann Container für weitere Ressourcen sein. So wird ein einzelnes Eclipse-Projekt/IntelliJ-Modul oder ein einzelner Ordner ebenfalls als Ressource bezeichnet.

Wie bereits erwähnt ermöglicht dieser das Auswählen von vollständigen Projekten oder einzelnen Ressourcen im Workspace der IDE. Ich bin im Ab-

[31] Siehe 3.3.3
[32] Siehe 3.2.2

schnitt 3.2.3 bereits kurz auf die verschiedenen Terminologien von IntelliJ und Eclipse eingegangen. Hinter diesen Terminologien stehen unterschiedliche Konzepte wie Ressourcen innerhalb dieser IDEs strukturiert und organisiert sind. Um den SIW in HTML darstellen zu können, müssen die in den IDE verfügbaren Ressourcen in eine einheitliche Datenrepräsentation gebracht werden. Unter anderem hatte ich 4 Teilprobleme zu lösen:

1. Die zum Teilen in einer Session verfügbaren Ressourcen mussten IDE-unabhängig über Methoden des Kern-Moduls verfügbar gemacht werden.

2. Diese Ressourcen mussten in ein für die Anzeige praktikables Datenmodell verpackt werden um sie danach über einen entsprechenden Renderer an das HTML-Frontend weiterzugeben.

3. Ein konsistente Verbindung (Mapping) zwischen dem Datenmodel und den eigentlichen Ressourcen musste bereitgestellt werden.

4. Dem Frontend musste eine SStart-Session-MethodeBrowsermethode zu Verfügung gestellt werden, welche mit dem von Frontend zurückgegeben Informationen an den Kern adaptiert wird.

Auf die Umsetzung und aufgetauchten Schwierigkeiten bei diesen Aufgaben möchte ich in den folgenden Abschnitten einzeln eingehen.

4.3.1 Erweiterung der Kern-Filesystems

Um die benötigten Daten für die HTML-Anzeige zu akquirieren war eine Erweiterung des Filesystems im Kern-Modul nötig, da es bis dato keine IDE-übergreifende Möglichkeit hierfür gab.[33]
Das Kern Filesystem-Package enthält und kapselt alle Interfaces zum Zugriff auf Daten innerhalb einer IDE.[34] Diese Interfaces werden jeweils von Saros/E und Saros/I implementiert. Ein wichtiges Interface für meine Arbeit war hier das IWorkspace, das die gerade geladene IDE-Umgebung repräsentiert, sowie IResources, welches eine Datei innerhalb eines Workspaces repräsentiert. Leider bot das Kern-Filesystem-Interface, das in weiten Teilen eine Kopie des Eclipse-Filesystems-Interface ist, keine Möglichkeit alle vorhanden Ressourcen eines Workspaces zu extrahieren.
Im Eclipse-Filesystem gibt es ein den IWorkspaceRoot, welches die Wurzel-Ressource in der Ressourcen-Hierarchie des Workspaces darstellt. Damit sind über diese Wurzel alle anderen Ressourcen des Workspaces erreichbar. Es darf damit genaue eine Wurzel-Ressource pro IDE-Instanz existieren.[37] Da dies genau die Funktionalität beinhaltet, die von der HTML-GUI

[33] Eingeführt in Commit: https://github.com/saros-project/saros/commit/0898a22
[34] Filesystem package:https://github.com/saros-project/saros/tree/master/de.fu_berlin.inf.dpp.core/src/de/fu_berlin/inf/dpp/filesystem

benötigt wird, macht sich die eingeführte Erweiterung diese Klasse für die Saros/E Implementierung zu Nutze.
Da IntelliJ Ressourcen anders verwaltet, war es nicht möglich, die Implementierung dort genauso einfach umzusetzen. Ein Problem hierbei war, dass das Konzept eines WorkspaceRoots in IntelliJ so nicht existiert. Außerdem ist das Filesystem von Saros/I nicht so weit entwickelt wie in Saros/E. So fehlte eine IContainer-Implementation welche ebenfalls von der UI genutzt wird. Da nach mehrere Tagen kein Durchbruch erzielt wurde, und es absehbar wurde, dass der Aufwand für diese Implementationen nicht genau abgeschätzt werden konnten, beschloss ich dieses Problem nach hinten zu stellen und die weiteren Schritte für die Einführung des SIW anzugehen. (Das sinnvolle Anpassen des Filesystems von Saros/I hätte eine tiefer greifendere Recherche erfordert)

4.3.2 Einführung eines Benennungsmusters für Kern Implementationen

Bevor ich Schritt 2, die Einführung eines Datenmodells für die Ressourcenanzeige beschreibe, möchte ich eine weitere Änderung im Zusammenhang mit dem Filesystem von Saros diskutieren, welche ich eingeführt habe.
Während der Entwicklung des Projektmodells fiel auf, dass die Namensgebung in Saros/J und Saros/E nicht einheitlich war. In Saros/E werden Implementationen des Filesystems *Eclipse*NAMEDESINTERFACES*Impl* benannt. So ist sofort erkennbar, welches Interface in diesen Klassen implementiert ist. In Saros/J existierte keine einheitliche Namensgebung. Des Weiteren wurde das *filesystem package* unnötigerweise mit 'fs' abgekürzt. Solch undeutliche Abkürzungen sind nach der Saros Code-Konvention nicht erwünscht.

Nach einem Refactoring[35] wurde dies vereinheitlicht. Dieses Muster kann auch auf andere Klassen und Packte ausgeweitet werden. Dadurch würde sich innerhalb der verschiedenen Saros-Module Kern-Klassen von IDE spezifischen Klassen deutlicher abheben. Dies ist zum schnelleren Verständnis der Codearchitektur förderlich und vermeidet potenziell Fehler durch das Verwechseln gleichnamiger Klassen. Außerdem hat es den positiven Nebeneffekt, dass die Klassen-Suche über mehrere Module vereinfacht wird. Daher schlage ich vor, das Klassen-Benennungsmuster 1 und 2 in die Code-Konvention von Saros aufzunehmen. Für das Filesystem von Saros/I—E wurden es bereits umgesetzt.

[35] Changes "filesystem"package naming to match common pattern https://github.com/saros-project/saros/commit/c0ff1a3

Klassen-Benennungsmuster 1 (für Kern Interface Implementationen)
Eine Klasse, welche ein Interface des Kernmoduls implementiert, sollte folgendem Namensmuster entsprechen:

[IDE][Implementierte-Kernklasse]Impl

Beispiel: IWorkspace im Core-Modul
Saros/E — EclipseWorkspaceImpl
Saros/J — IntelliJIWorkspaceImpl

Klassen-Benennungsmuster 2 (für multiple Klassen)
Klassen, welche in mehreren Saros-Varianten zum Einsatz kommen, und dort den selben Zweck dienen sollten folgendem Namensmuster entsprechen:

[IDE][Klassenname]

Beispiel: CollaborationUtils in Saros/E und Saros/I
Saros/E — EclipseCollaborationUtils
Saros/J — IntelliJCollaborationUtils

Ebenfalls wurde das Präfix '*Saros*' aus den Klassen des UI-Moduls gestrichen. Das Präfix bietet keinerlei semantische Hilfe bei der Erfassung des Inhalts einer Klasse und bläht oft ohnehin schon lange Bezeichner unnötig auf. Solch ein Präfix würde nur Sinn ergeben, wenn Klassen noch von anderen Plug-Ins außer Saros genutzt würden - was nicht der Fall ist. So ist offensichtlich jede Klasse *Saros* bezogen. Daher wurde z.B. SarosHTMLUIContextFactory in HTMLUIContextFactory umbenannt.[36] Das selbe Argument gilt für alle anderen Klassen der Saros-Module, so ist enthält die Namensgebung innerhalb von Saros/E auch dieses redundante Präfix: SarosCoreContextFactory, SarosEclipseContextFactory. Sie sollten in späteren Refactorings ebenfalls dementsprechend umbenannt werden.

4.3.3 Konzeption des ProjektList-Modells

Als nächstes stand die Entscheidung an, welche Daten zur Anzeige eine Liste von wählbaren Projekten an das Frontend übertragen werden müssen. Während die im Prototypen bereits vorhandenen Modellklassen für Kontakte und Accounts[37] verhältnismäßig einfach umzusetzen waren, musste in Zusammenarbeit mit Bastian Sieker (Der zu dieser Zeit den HTML/JS Part der GUI entwickle) erst die konkreten Anforderung an diese Modell spezifiziert werden. Ein Hauptunterschied zu den vorherigen Modellen ist, dass das HTML-Frontend das ProjektListModell (PLM) direkt verändern und

[36] Die vollständige Liste kann in Anhang A.1-6 eingesehen werden).
[37] Diese wurden als ValueObjects umgesetzt[38]

diese an das UI-Backend zurückgeben muss. In Abb. 6 wird dieser Vorgang dargestellt. Die Abbildung ignoriert die für den SIW ebenso verwendete ContactModel. Sie stellt die Interaktion der einzelnen Komponenten für die Generierung der Projektanzeige innerhalb des SIW dar. So ist zu sehen, das eine eingeführte zentrale ProjekListManagerklassse die Kommunikation zum Kern abstrahiert und verwaltet. Mithilfe des angepassten Kern-Filesystem wird das für den Renderer nötige ProjektListModell erstellt. Dabei wird auch ein Mapping zu den korrespondierenden Ressourcen der Modelle erstellt und vorgehalten. Im Frontend wird dieses JSON-Modell für die Anzeige verwendet. Der User wählt die gewünschten Projekte und Ressourcen aus. Startet der User den Einladungsprozess, wird vom Frontend ein verändertes JSON-Modell an das UI-Backend übertragen. Dieses JSON wird zuerst in sein Java-Pendant umgewandelt und in der Folge die gewählten Ressourcen durch das vom ProjektListManager zur gehaltenen Mapping akquiriert.

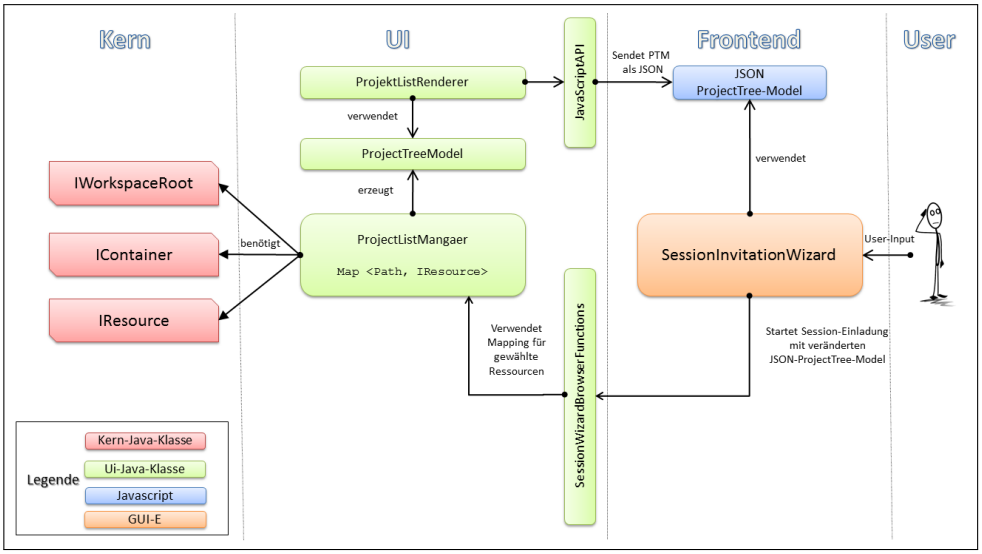


Abbildung 6: Interaktion der Saros-Komponenten zur Anzeige der Ressourcen im SessionInvitationWizard

Angezeigte Projekte werden durch das ProjektTree-Modell repräsentiert. Ein solches besteht aus seinem Namen (welches als ID fungiert) und genau einem (Wurzel)ProjektTreeNode-Modell. Eine ProjektTreeNode (PTN) stellt das Anzeigemodell eine Ressource da. In ihr sind folgende Informationen gekapselt:

- Eine Liste aller Mitglieder (bzw. Kinder) eine Ressource, um die angestrebte Baumhierarchie zu realisieren.
- Der Pfad zur Ressource.

Dieser wird nicht direkt in der Anzeige dargestellt. Da es jedoch nötig ist, das Anzeigemodell(Frontendrepräsentation) mit seiner korrespondierenden Ressource (Backendinstanz) zu verbinden, wird dieser Parameter benötigt. Der Pfad einer Ressource muss zwingend eindeutig sein. Damit eignete er sich am besten für dieses Mapping. Abgesehen davon, wird dieser Parameter zu Generierung des Anzeigenamens benutzt.

- Der Anzeigename für die Darstellung.
  Er generiert sich aus dem letzten Segmentes des Ressourcen-Pfades.[38]

- Der Ressourcentype
  Das Kernfilesystem kennt hierbei die Typen "Project, Folder, File". Das Frontend kann diese Information verwenden, um die Anzeige des Eintrags entsprechend des Types zu verändern. So können PTN vom Type "Project"(z.B. per CSS) anders formatiert werden als eines vom Type "File".

- Ob diese PTN zum Teilen ausgewählt ist.
  Wird beim Erzeugen einer PTN dieser Wert nicht gesetzt ist er standardmäßig als zum Teilen ausgewählt. Das Frontend verändert diesen Wert für eine ProjektTreeNode, je nachdem welche Dateien ein Sarosnutzer im SIW wählt. Anhand dieses Wertes wird bestimmt, welche Dateien letztlich mit den gewählten Kontakten geteilt werden soll. Diese Interaktion ist in in Abb. 6 veranschaulicht. Der Parameter erlaubt auch dem Frontend eine Vorauswahl bestimmter Projekte zu übergeben, wie es auch in der Saros/E Version dieses Wizards möglich ist.

In einer ersten Entwicklung besaß eine PTN nicht nur eine Liste seiner Kinder, sondern auch einen Verweis auf seinen Vaterknoten. Damit ist das Einfügen und Löschen von PTN einfacher zu bewerkstelligen. Das UI-Modul verwendet die von Google entwickelte GSON-Bibliothek zum Serialisieren der Java-Modell-Objekte zu JSON, sowie zum Konvertieren von JSON in Java-Objekten.[39]. Während der Qualitätssicherung mittels eingeführter JUnit-Tests für den ProjektListManager fiel jedoch auf, dass es nicht möglich ist, solch eine Baumstruktur mithilfe der GSON-Bibliothek zu JSON zu serialisieren. Dies ist dem Umstand geschuldet, dass JSON nativ keine Zirkelreferenzen `A->B->A` unterstützt.[40] Daher wurde diese Referenz im Java-Modell wieder entfernt.

[38] *C://MyFooWorkspace/FooProjekt/FooOrdner/foo.Java* erzeugt den Anzeigestring '*foo.java*'

4.3.4 Einführung von Unit-Tests für UI-Modul

Abb. 6 zeigt, dass der ProjektListManager eine zentrale Klasse für den Start des Einladungsprozess ist. Um die einwandfreie Funktionalität des komplexen PLM zu gewährleisten habe ich einen JUnit-Test implementiert. Zum effektiv Testen musste zuerst die Möglichkeit geschaffen werden, die benötigten Kern-Klassen (siehe Abb. 4.3.1) effektiv zu mocken. Mooks imitieren das Verhalten von Klassen, ohne sie tatsächlich vollständig zu implementieren. So werden anstelle der tatsächliche Klassen, die der PLM benötigt, innerhalb des Tests diese Imitate verwendet. Der Sinn solcher Mockups ist, die zu testende Klasse unabhängig von den von ihr verwendeten Klassen testen zu können. In einem Unit-Test sollen Bestandteile einer Software isoliert getestet werden können.[41] Saros/E und Saros/I verwenden für die dortigen JUnit-Tests die Easymock-Bibliothek[39]. Diese wurde auch für das UI-Modul eingeführt. Der Test brachte verschiedene Fehler im PLM zum Vorschein, welche im Reviewprozess nicht aufgefallen waren. Diese wurden daraufhin beseitigt. Für Details siehe Anhang A.1#11.

4.3.5 Performanzproblem im Frontend

Während der Qualitätssicherungsphase wurde in Zusammenarbeit mit Bastian Sieker ein Performanzproblem bei der Darstellung der Ressourcen im SIW entdeckt. Im Gegensatz zu dem bisher verwendeten Datenmodell für Kontakte und Accountinformationen ist das ProjekList-Modell sehr viel komplexer und umfangreicher. Es enthält Informationen zu allen verfügbaren Ressourcen innerhalb einer IDE-Instanz. Der SIW wurde mit dem derzeitigen Saros-Projekt getestet. Dies enthielt ca. 10.000 über 11 Projekte verteilte Dateien. Das Aufrufen des SIW mit solch einer großen Anzahl von Workspace-Ressourcen lies die Anwendung für über 30 Sekunden einfrieren. Dabei wurden die Prozessoren des Testrechners vollständig ausgelastet, und es ergab sich ein Anstieg der RAM-Nutzung um mehr als 1,5 GB.

Ich und Bastian Sieker und ich versuchten den Grund für diesen enormen Leistungsverbrauch zu analysieren. Die Analyse erbrachte 4 Möglichkeiten für dieses Problem:

1. Bei der Erstellung der Java-Model-Instanzen. Da hierfür der gesamte Workspace rekursive durchsucht wurde, bestand die Befürchtung, dass dies zu Leistungseinbußen führen könnte.

2. Bei Serialisierung der erstellten Modelle zu JSON

3. Beim Javascript-Aufruf der das Frontendmodell aktualisiert.

[39] http://easymock.org/

4. Bei der Anzeige im Browser selbst, welche für jeden ProjektTreeNode Eintrag ein Objekt im DoM erzeugen musste.

Punkt 1. und 2. konnte ich schnell ausschließen. Deaktivierte man den Aufruf der Frontend-Javascript Methode traten keinerlei Leistungseinbußen mehr auf. Das Erzeugen und Serialisieren der Daten stellt somit Java seitig kein Problem dar. Da die Javascript Entwicklung im Verantwortungsreich von Bastian lag, habe ich mich nicht weiter mit Punkt 3 und 4 beschäftigt. Seine Untersuchungen konnten nicht genau klären ob dies ein Implementations- oder generelles Problem in der Anzeige ist. Da die Planung noch die Entwicklung des JoinSessionWizards und dessen Test vorsah, wurde die Behebung dieses Problems nach hinten geschoben. Der Bug wird dafür noch im Bugtracker von Saros dokumentiert.
Falls sich herausstellt, dass die Anzeige von solch großen JSONs (Das erzeugte JSON für 10.000 Dateien ist ca. 4MB groß) ein generelles Problem für den Browser ist, erscheint eine Anpassung am Interface der UI angezeigt. So könnte dem Frontend alternativ jeweils nur die gerade gewählte Hierarchieebene übergeben werden. Sobald der User tiefer in den Datenbaum navigiert, müssten über eine entsprechende Browserfunktion dem Frontend die benötigten Kinder für die Darstellung übergeben werden. Dies würde allerdings auch die Rückgabe des SIW an das UI-Backend verändern was zu einer größeren Anpassung führen kann.

4.3.6 Bereitstellung der StartSession Methode

Wie eingangs beschrieben war der letzte Schritt für die Implementation des SessinInvitationWizards die Bereitstellungen der entsprechenden Kern-Funktionen. Dafür wurde eine entsprechende JavaBrowserfunctions-Methode entwickelt. Dieser Methode werden die gewählten Ressourcen und die gewünschten Kontakte für die Session als JSON- Parameter übergeben.

Während die Rückwandlung des JSON zu Java und das Bestimmen der gewählten Ressourcen über das Mapping innerhalb des *ProjektListManager* unproblematisch war, stellte sich der Zugriff auf den eigentlichen Einladungsprozess als schwieriger heraus. Zwar bietet der Kern ein *ISarosSessionManager* (in Folge SSM), welcher Methoden zum Starten und Verwalten einer Session behinhaltet, dieser wird jedoch niemals direkt benutzt. Sowohl Saros/I als auch Saros/E verwenden den SSM nur über eine statische *CollaborationUtils* Klasse.
Da diese Klasse die selben öffentlichen Methoden haben, war der erste Ansatz, diese Klassen im Kern zusammen zu führen, um sie im UI-Modul zu benutzten. Bei einer genauen Analyse des Codes fiel auf, dass diese Klassen, obwohl zu 90 Prozent gleich aufgebaut, spezifische IDE-Abhängigkeiten besitzen, welche ebenfalls in den Kern integriert werden müssten. Insbesondere

wird die Session-Beschreibung, welche Informationen zum Umfang der geteilten Projekte enthält, in den IDEs verschieden akquiriert. Da die Saros/E CollaborationUtils-Klasse zudem nicht das Kern-Filesystem ,sondern das Eclipse-Filesystem für Ressourcen verwendete wäre eine Anpassung an allen Saros/E Klassen nötig gewesen, welche die CollabUtils. benutzten. Mit über 30 Referenzierungen innerhalb von Saros/E ergab sich daraus ein massives Refactoring. Aus diesen Gründen war eine Zentralisierung nicht ohne großen Aufwand möglich. Stattdessen wurde ein Interface in die UI eingeführt[40], welches über 'dependency injection' die jeweiligen IDE-abhängigen Implementationen nutzt und im UI-Modul bereitstellt.[42]
Mittelfristig ist es jedoch wünschenswert, das alle Verwendungen des Eclipse-Filesystem-Interfaces durch das Kern-Filesystem-Interface ausgetauscht werden. Dies würde eine Zentralisierung von Logik vereinfachen und Vorteile für die Evolvierbarkeit von Saros insgesamt bieten.

Mit der Lösung dieser Probleme, ist es nun möglich in der HTML-GUI die vorhanden Ressourcen anzuzeigen, und den Einladungsprozess zu starten.

4.4 Implementation des JoinSessionWizards

Wie in Abschnitt 2.4.3 beschrieben, konnte dieser in der verbleibenden Zeit aufgrund des Fehlens der ProjektNetotiation im Kern nicht abgeschlossen werden. Der erste Teil des Wizards, welcher die Informationen enthält wer zu einer Session einlädt und was geteilt werden soll, konnte mit den Vorhanden SessionNegotiations umgesetzt werden. Dafür wurde eine *NegotiationHandler* in die UI eingeführt, der auf eingehende Session Benachrichtigungen hört und darauf hin den JoinSessionWizard Dialog anzeigt. Für den 2. Teil dieses GUI-E ist die Einführung der ProjektNegotiations in den Kern zwingend erforderlich. Zur Zeit der Abgabe wird der 2. Teil dieses Wizards noch durch die jeweiligen IDE-spezifischen GUI-Implementationen umgesetzt.

[40] Anhang:A.1-17

5 Ergebnisse

In diesem Kapitel werden zuerst die Ergebnisse präsentieren, um zu evaluieren, inwieweit die Ziele dieser Arbeit erfüllt wurden. Dafür werden zuerst die Ziele einzeln besprochen. Ein Ergebnis dieser Arbeit, welches nicht in Kapitel 2.3 geplant war, ist ein Erkenntnisgewinn bei der engen Zusammenarbeiten im Team innerhalb eines Softwareprojekts. Diese werden in Abschnitt 5.5 erörtert.

5.1 Verringerung von Codeduplikation

Eine Zielsetzung war Codeduplikation im Bereich der UI zu vermeiden (siehe 2.3.1). Zum Zeitpunkt der Abgabe beträgt der Umfang des HTML UI Moduls (exklusive des Frontend Moduls) ca. 2500LoC in 50 Klassen. Dazu kommen IDE spezifische Implementationen für diese Module mit jeweils ca. 300LoC. Auch wenn noch GUI-E in der neuen UI fehlen (siehe 5.6 Fazit und zukünftige Arbeiten), wird sich dieses Modul nur noch um wenige Klassen erweitern. Selbst bei einer Verdopplung oder gar Verdreifachung der Codebasis im Zuge der Weiterentwicklung ist das neue Modul mit ungefähr 9000 LoC nicht einmal halb so groß, wie die aktuelle Saros/E Version mit 22.000LoC. Bedenkt man, dass dieser Code für Saors/J und Saros/E nutzbar ist verdoppelt sich diese Ersparnis. Jede weitere Saros IDE-Variante, wie zum Beispiel Saros für Netbeans, erhöht diesen Faktor um 1.

Neben dieser reinen Verringerung von Code, war ein wesentliche Anspruch jede Codeduplikation, im Sinne der Definition in 2.3.1, zu vermeiden. Mit den getroffenen Erweiterung der HTML-GUI ist das Ziel, für jede Anzeige der verschiedenen Saros-Varianten genau EINE Representation in Saros zu haben, ein entscheidendes Stück näher gekommen. So sind jetzt die Saros-SessionWizards zentral über die HTML-GUI umgesetzt. Die Architektur ist geeignet zukünftige diese Codeduplikationen zu vermeiden.

In diesem Sinne sehe ich das Ziel als im erreicht.

5.2 Verbesserungen im Bereich der Evolvierbarkeit

Eine genaue Evaluation ist schwierig, da die UI noch nicht vollständig ist. So gibt es noch keine Erfahrungswerte für die '*tägliche*' Entwicklung des neue UI-Systems. Ich sehe jedoch die Code-Transparenz als verbessert an. Allein durch die Tatsache, dass Änderungen nun zentral in der UI erfolgen, ist es für Entwickler einfacher, die GUIs von Saros/I und Saros/E zu pflegen. Den Ort einer nötigen Änderung sowie ihren Effekt schneller zu finden und zu sehen wird durch die bessere Trennung von Anzeige und Kernlogik erreicht. Im Folgenden will ich detaillierter auf die erreichten Verbesserun-

gen in der GUI-Entwicklung eingehen und hierbei diskutieren inwieweit die formulierten Ziele aus Abschnitt 2.3 im Hinblick auf die Evolvierbarkeit erreicht wurden.

5.2.1 Vermeidung von Speicherlecks

In SWT wie auch AWT/Swing muss darauf geachtet werden, Speicherlecks zu vermeiden. Der Entwickler ist dafür verantwortlich, genutzte GUIe wieder zu „entsorgen“, sobald sie nicht mehr gebraucht werden. Es ist ein ”gern gemachter”Fehler, dies zu vergessen, zu spät oder zu selten zu tun. Da alle GUIe nun durch Browser dargestellt werden, entfällt diese Verantwortlichkeit für den Entwickler. Die Speichernutzung von GUIe werden im Allgemeinen sehr effektiv durch die jeweiligen Browserengine verwaltet und müssen nicht durch den Entwickler gesondert behandelt werden.

5.2.2 Fail Fast

Wie in Kapitel 2.3.2 beschrieben, war ein Ziel die Fehleranfälligkeit des Systems zu minimieren. Dafür wurden Schnittstellen zum Frontend so aufgebaut, dass ein Fehler im Aufruf von Javascript-Funktionen direkt zu einem sichtbaren Versagen führt. Ebenso werden inkonsistente Zustände durch vordefinierte Zustandsmodelle verhindert. Sollte ein Modell nicht die erwartet Form haben, spiegelt die Anzeige des Frontends diese direkt wider. Aufrufe vom Frontend in das Backend sind durch Browserfunktionen definiert. Sie folgen dem Guard-Clause-Muster, welches das Ausführen von Methoden bei fehlerhaften Aufrufen verhindert.[43] In einem solchen Fall wird das Frontend über die falschen Benutzung informiert. Hierdurch werden Fehler während der Entwicklung schnell sichtbar gemacht, was wiederum der Evolvierbarkeit zugute kommt.

5.2.3 Vereinfachter Entwicklungseinstieg

Ein weiteres (in Abschnitt 2.3.3 formuliertes) Ziel war es, die Einstiegshürden durch eine einheitliche Technologie und transparenter Code-Architektur zu verringern. Bemerkt sei hierbei, dass in diesem Bereich persönliche Präferenzen und das individuelle Lernverhalten natürlich eine Rolle spielen. Da CSS, HTML und Javascript gut dokumentiert und weit verbreitete Standardtechnologien sind, lässt sich argumentieren, dass diese für neue Saros-Entwickler eher bekannt sein dürften als SWT/AWT/SWING. Als Open-Source-Projekt ist es für Saros positiv, wenn die eingesetzten Technologien weit verbreitet und gut dokumentiert sind. Dadurch wird die Einstiegshürde für Entwickler gesenkt, Änderungen in Saros vorzunehmen. Die Wahl der Frameworks, welche für sich wiederum Vorwissen und Lernkurven beinhalten, sind nicht Bestandteil dieser Arbeit. Auf die Wahl der JS-Framworks und Technoliguen wird in der Arbeit von Bastian Sieker eingegangen. Sie sind zum Zeitpunkt

der Fertigstellung meiner Arbeit noch nicht final. Daher kann ich an dieser Stelle keine genauere Aussage über den Umfang und das benötigte Wissen zur Entwicklung des Frontends treffen und verweise somit auf die Arbeit von Bastian Sieker für weitere Informationen.[22] Es lässt sich vermuten, dass Webtechnologien und Webentwicklung für Entwickler eher zum Repertoire gehören als SWT/AWT/SWING. Auch wenn dies nur schwache Indikatoren sind, ist eine Verbesserung in diesem Bereich vor allem durch die Reduzierung der eingesetzten Technologien in der GUI Entwicklung gegeben. So ist es für die Sarosentwicklung zukünftig nicht mehr erforderlich, die verschiedenen IDE-Anzeige-Technologien zu erlernen, sondern nur noch das eingesetzte Javascript Framework.

5.2.4 Trennung von Kern-Logik und Benutzerfläche

Wie in 2.3.6 Ziel: Schärfere Trennung von Geschäftslogik und GUI beschrieben, bestand ein Ziel in der effektiveren Trennung von Geschäftslogik und Anzeigelogik. In der vorhanden GUI von Saros sind beide teilweise in Wizard-Klassen vermischt. Dies ist zum einen der Art geschuldet, wie in SWT Dialoge und Widgets erstellt werden, zum anderen Folge von inkonsequenter Kapselung durch vorherige Entwickler. Das UI Modul als Zwischenschicht von Frontend und Kern ist in seinen Funktionen räumlich wie auch logisch getrennt. Das wurde durch das Einführen der JavaScriptApi (4.2.2) und der klaren Aufgabenteilung der einzelnen Module erreicht. Abb. 7 stellt den aktuellen Aufbau und die Kapselung der UI dar. Die Benutzeranzeige verändernde Aufrufe finden ausschließlich über der JavaScriptAPI statt, während Aufrufe, die mit der Geschäftslogik interagieren durch Fassaden und Manager von der eigentlichen Anzeigelogik getrennt sind. Die Darstellung übernimmt das Frontend. Durch entsprechende Namensgebung und Java-Dokumentation sind diese Trennungen schnell für neue Entwickler ersichtlich. Somit wurde neben den Vorteilen der Zentralisierung auch eine effektivere Entkopplung und Kapselung von Kern und Frontend erreicht. Damit wird ebenso ein einfacher Austausch der Frontendtechnologien ermöglicht.
Neben dem einfacheren Austauschen von Technologie ist ein weitere den positiver Effekt, dass diese Kapselung das Evaluieren verschiedener Anzeigen und Dialoge massiv vereinfacht. Die Funktionalität aller GUI-E Komponenten und damit aller vom Nutzer ausführbarer Aktionen werden nun zentral im BrowserFunctions-Package verwaltet und könnenn, durch entsprechende Wahl dieser Browserserfunktionen, unkompliziert für neue Anzeigen und Dialoge verfügbar gemacht werden. Damit ist es schneller möglich Anzeigen auszutauschen und neu zu konzipieren, was direkt der leichteren Verbesserung der Anwenderfreundlichkeit zugute kommt.

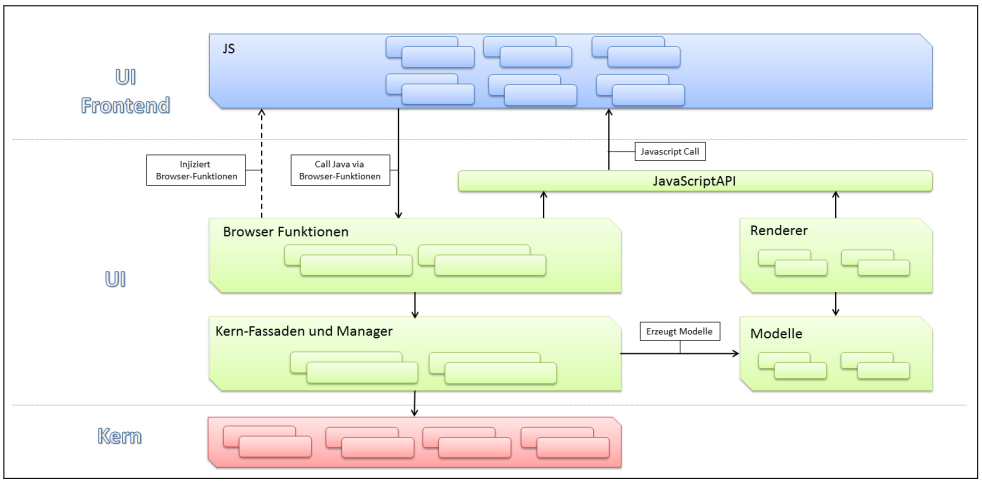


Abbildung 7: Aufbau des UI-Moduls als Adapter zwischen Kern und Frontend

5.2.5 Zusammenfassung

Abschließend kann die Frage, wie hoch die Evolvierbarkeit der HTML GUI ist, nur über das Erheben von Daten über einen längeren Zeitraum nach Inbetriebnahme geklärt werden. Weiterhin kann auch eine Wiederholung des durchgeführten Entwicklertests nützlich sein, um diese Ergebnisse zu verifizieren und noch offene Problem zu finden. Es empfiehlt sich ebenfalls eine Erstellung eines Quick-Guides mit den wesentlichen Informationen und einer Sammlung nützlicher Links zu den eingesetzten Technologien.

5.3 Geplante abschließende Arbeiten

Um die Weiterentwicklung für weitere Saros-Entwickler zu vereinfachen möchte ich eine englischsprachige Dokumentation zur aktuellen Architektur erstellen welche auf der Projektseite veröffentlicht wird. Außerdem plane ich die Schnittstelle des UI-Modul für das Frontend zu vervollständigen, indem eine Liste aller Browserfunktionen angelegt wird. Diese können als Einstiegspunkt dienen und sollen zukünftigen Entwicklern das schnelle Erfassen der noch ausstehenden Implementierungsarbeiten ermöglichen. Gleiches gilt für die benötigten Modellklassen.
Aufgrund der fehlenden Funktionalität in Saros/I innerhalb des Filesystem sind GUI-Komponenten welche auf den *ProjektTreeNode*-Modelle aufbauen im Moment nicht mit IntelliJ benutzbar. Diese fehlenden Funktionen will ich noch umsetzten, um die UI für Saros/E und Saros/I uniform zu halten.

5.4 Mitwirken im Saros-Team

Neben den Hauptaufgaben und Zielen dieser Arbeit wurde ein Teil der Zeit mit Aufgaben im und um das Saros Projekt selbst erledigt. Dazu gehörten allgemeines Feedback auf Fragen aus den Mailinglisten, Code-Durchsichten für andere Entwickler, sowie Anpassung und Weiterentwicklung von Dokumentation. Ich war als Dokumentationsmanager im Softwarerealse-Prozess für die Aktualisierung dieser zuständig. In dieser Tätigkeit habe ich verschiedene Seiten auf der Homepage aktualisiert und Dokumentationsprobleme behoben. Eine Liste der gefixten Bugs und Änderungen an der Homepage findet sich im Anhang A.3. Desweiterem wurde im Zuge der gemeinsamen Entwicklung der UI an verschiedenen Codepatches anderer Saros-Team-Mitglieder mitgewirkt. Ein Auswahl der Commits, an denen ich mitgewirkt habe, findet sich im Anhang A.2.

Einführung neuer Tags für Commits Die Coding Conventions[41] des Saros-Projektes sehen eine Reihe von möglichen *Commit Tags* vor. Diese sollen das schnelle Erfassen des Inhalts eines Commits für Entwickler erleichtern. Außerdem sind sie hilfreich beim Durchsuchen der GIT-Historie.

Die vorhanden Committags sind dafür konzipiert, auszudrücken 'was für eine *Art von Änderung* in einem Commit vorgenommen wurde. Es fehlten jedoch Tags für die Lokalisierung der Änderung. Solange Saros nur für Eclipse entwickelt wurde, waren die Ziele dieser Tags eindeutig: **[UI]** stand für Änderung an der Saros/E Version der UI. [Internal] stand für interne Veränderungen in Saros/E. Mit der Entwicklung auf mehreren IDEs und der Einführung weiterer Module wie *Core* und *UI* reichten diese Tags für sich genommen nicht mehr aus, um eindeutig zu sagen, welchen Bereich der Codebasis von Saros sie betreffen. Da nun dieselben Arten von Änderungen in verschiedenen Modulen möglich waren, sind Entwickler darauf angewiesen, den Ort der Änderung mit anzugeben, um klarzustellen, um welchen Bereich der Sarosentwicklung es geht. So entstanden unnötig lange Commit-Überschriften mit redundanten Inhalten wie:

- **[UI]** HTML-UI: adds a minimal contact list [42]
- **[INTERNAL]** Saros/I use ConnectionHandler from Core [43]
- **[UI]** Use new HTML-UI in Saros/Eclipse [44]

Ich habe daher einen Vorschlag an die Mailingliste geschrieben, um neue kurz Tags einzuführen, welche in Kombination mit den schon vorhanden

[41] http://www.saros-project.org/coderules
[42] https://github.com/saros-project/saros/commit/083b592
[43] https://github.com/saros-project/saros/commit/24b8724
[44] https://github.com/saros-project/saros/commit/0ed3d13

Tags benutzt werden sollen, um das Ziel[45] genau zu bestimmen. Eine Beispielverwendung dieser Tags wäre
[FIX][I] = Dieser Commit behandelt ein Bug der Saros/I bezogen ist.

Zusätzliche Commit-Tags (Orginal Text)
[E] *This commit ONLY affects the Eclipse version of Saros*
[I] *This commit ONLY affects the IntelliJ version of Saros*
[HTML] *This commit ONLY affects the html ui project.*
[CORE] *This commit ONLY affects the core project.*

5.5 Reflexion, Erkenntnisse und Handlungsempfehlungen

Ich habe im Abschnitt 3.3.3 erörtert, dass ich durch die starke Abhängigkeit zu anderen Arbeiten Probleme in der Abgrenzung meiner Tätigkeit während der Bearbeitung sowie der Planung der weiteren Schritte hatte. Ein Ergebnis dieser Arbeit ist daher nicht technischer Natur, sondern ein persönlicher Erkenntnisgewinn über den Umgang mit solchen Schwierigkeiten. Ich denke, dass viele der Erfahrungen einen typischen Charakter für die Entwicklung von Software im Team haben. Da Software, meist im Team und nur sehr selten außerhalb der natürlichen Welt unter 'Laborbedingungen' entwickelt wird, lassen sich die gemachten Erfahrungen gut generalisieren und auf andere Projekte und Arbeiten übertragen. Auch wenn dies ohne Zweifel individuelle Erfahrungen und Sichtweisen beinhaltet, so denke ich, dass sie für Entwickler in ähnlichen Situationen nützlich sein können. Daher will ich in diesem Abschnitt diese gemachten Beobachtungen diskutieren und mögliche Handlungsempfehlungen zum Umgang mit diesen formulieren.

5.5.1 Problemursachen bestimmen

Um ein Problem richtig anzugehen, muss es verstanden und seine Ursache(n) erkannt werden. Während man sehr schnell merkt, dass *etwas* schiefläuft verlangt es die Fähigkeit an Reflexion und Nachforschung, um die Ursachen dafür benennen zu können. Dieses Phänomen beschreiben auch Coplien und Harrison in ihrer Untersuchung "*Organizational Patterns of Agile Software Development*". In dem Abschnitt 'Human Concerns'[44, S. 47-48] stellen sie fest, dass es in einem Firmenumfeld nicht unüblich ist, dass Menschen nicht "den Finger auf ein bestimmtest Problem legen können, jedoch sehr wohl wissen, *dass* sie Schwierigkeiten haben".

Diese Schwierigkeiten zu erfassen erfordert Zeit und zum Teil auch einen gewissen Abstand zur Problematik. Zudem fällt es leicht, mit der Begründung unter Zeitdruck zu arbeiten, sich nicht die Zeit für diese Reflexion zu nehmen. Dabei ist Zeitdruck für Softwareentwickler ein nahezu tägliches Pro-

[45] Mit Ziel ist hier der Wirkungsbereich einer Codeänderung gemeint

blem. In dem Blog-Eintrag "*If I had more time I would have written less code*"diskutiert David Green die Gründe und zahlreichen negativen Folgen "sich ständig unter Zeitdruck zu wähnen". So ist ein vermeintlicher Zeitdruck kein Grund, wichtige Software-Praktiken nicht umzusetzen oder der Analyse der Anforderungen nicht genügend Platz einzuräumen.[45]

Das tägliche Reflektieren gehört als Praktik für erfolgreiche Softwareentwicklung genauso dazu, wie der Einsatz eines Versionskontrollsystems. Diese Praktiken sind Bestandteil der „Clean-Code-Entwicklung"[46], welche ich sehr empfehlen kann. Ohne Reflexion entwickelt man sich nur langsam und wiederholt Fehler unnötig oft. Neben praktischen Tipps zu den konkreten Problemen will ich zeigen, dass es ein großer Trugschluss ist, Reflexion als verlorene Zeit zu betrachten. Tatsächlich hätte mir, wie ich aus eignen Erfahrungen gelernt habe, ein Zurücknehmen und Reflektieren oft mehr Zeit gespart als gekostet.

Eine schnell durchführbare wie effektive Form der Reflexion ist die Ursachenforschung mit der 5-Warum-Methode. Diese Technik habe ich zum Ende meiner Bearbeitungszeit genutzt, um ohne großen Zeitaufwand diesen Abschnitt zu erstellen. Dabei ist es nicht entscheidend, 5 mal Warum zu fragen. Die Idee der Methode ist, eine tiefgreifende Ursache (Wurzel) zu finden, um sie entsprechend angehen zu können. Es hat sich gezeigt, dass man nach dem 5. „Warum" schon ziemlich tief an den Kern der Ursache stößt.[46] Eine mögliche Vorlage ist in Abb.8 zu sehen.[47]

[46] Beispiel: https://en.wikipedia.org/wiki/5WhysExample

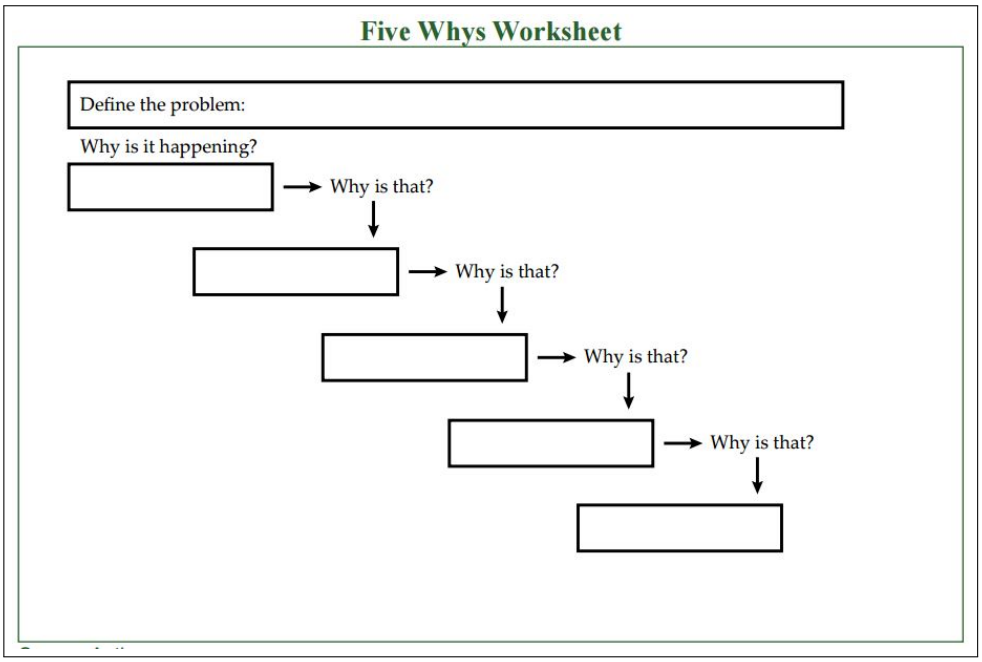


Abbildung 8: Beispiel für Ursachenforschung mittels "5 x Warum? Methode

5.5.2 Vorwärtsscheitern

Bevor ich im Folgenden die genauer auf die Teamarbeit und Zielabgrenzungsproblematik eingehe und mögliche Lösungsansätze diskutiert werden, will ich das Augenmerk auf die Wahrnehmung und den Umgang mit Rückschlägen lenken. Denn jeder der an der Verwirklichung eines größeren Projektes mitgewirkt hat muss lernen, mit Fehlplanungen und 'Scheitern' umzugehen. Ich halte dies auch für die persönliche Entwicklung für sehr wichtig. In dem Buch „*Failing Forward: Turning mistakes into stepping stones for success*" beschreibt John C. Maxwell eine alternative Sicht- und Herangehensweise an das eigene Versagen.[48] Das Buch zeigt eine in meinen Augen sehr nützliche Haltung zum Umgang mit Fehlern und einem möglichen Scheitern auf. Diese basiert auf der Erkenntnis, was Versagen *nicht* ist: (Orginal Zitat)

1. **Failure is not avoidable – humans are bound to fail sooner or later.**
2. **Failure is not an event, but a process. Success is not a destination – it is the journey you take and what you do day to day – success is a process, and so is failure.**
3. **Failure is not objective. You are the only person who can label your actions a failure.**

4. **Failure is not the enemy – it takes adversity to achieve success. It is fertilizer.**
5. **Failure is not irreversible.**
6. **Failure is not a stigma – they are not permanent markers. Make each failure a step to success**
7. **Failure is not final – failure is simply a price we pay to achieve success and if we learn to embrace that new definition of failure, then we can move ahead. It's the price you pay for success**

Dieses Buch bietet einen Ansatz, Fehler anders wahrzunehmen, sie zu nutzen und als nützlich zu begrüßen. Die Verschiebung der Wahrnehmung und des Umgangs ist ein großer Unterschied zum einfachen 'aus Fehlern lernen'.

5.5.3 Sinnvolle Arbeitsteilung im Team

In der engen Zusammenarbeit zwischen den einzelnen Teilbereichen der Arbeit fand ich mich öfter in der Situation wieder, das weitere Vorgehen abstimmen zu müssen. Die Ergebnisse der Arbeit von Christian Cykrit waren die Arbeitsgrundlage meiner Arbeit. Meine wiederum Basos für die Arbeit von Bastian Sieker. Nun fanden diese Arbeiten teilweise simultan statt. So waren ich und meine Team-Kollegen oft nicht frei in der Entscheidung, was die nächsten Schritte bei der Umsetzung der HTML-GUI sind. Vielmehr mussten wir uns auch nach den Anforderungen der anderen Bereiche richten. Zwar waren meine Entscheidungen zu dem jeweiligen Zeitpunkt gut begründet, im Nachgang wäre es jedoch sinnvoll gewesen, mehr Zeit und Energie in eine effektivere Teilung der Arbeitspakete zu stecken. Eine Problematik ergab sich unter anderem dadurch, dass wir es als Sarosteam versäumt haben, eine klare Aufgabenteilung vorzunehmen.
In diesem Zusammenhang möchte ich das Design-Muster 'Aufgaben-Teilung'[47] erwähnen. Unter anderem geht es hier darum, fällige Arbeiten in dringende und weniger dringende Bereiche zu unterteilen [44, S. 82-83]. Da Christian Cykrit zu Beginn meiner Bearbeitungszeit Hauptverantwortlicher für die Entwicklung des Prototypen war, hätte er die dringenden Aufgaben erledigen können, während ich mit den weiteren Aufgaben beginne. So wäre es effektiv gewesen, die Schnittstelle zum Frontend weiter auszuarbeiten, obwohl Christian Cikryt sich noch um die Frage kümmerte, ob der SWT-Browser-Ansatz überhaupt valide ist. Zwar beinhaltet dieses simultane Vorgehen Risiken, jedoch ist es oft so, dass Systemanforderungen oder Schnittstellenbeschreibungen noch nicht vollständig bekannt sind. Das Risiko größere Teile meiner Arbeit nochmal ändern zu müssen, weil die Anforderungen sich

[47] Englisch Original: WORK SPLIT

geändert haben, wäre jedoch durchaus vertretbar gewesen. So beschreiben Coplien und Harrison, dass Firmen erfolgreicher sind deren Entwickler nicht darauf warten, dass 100 Prozent der Spezifikationen und Anforderungen vorhanden sind bevor sie mit der Arbeit beginnen. Sobald man genügend vertrauen in ein Projekt hat, sollte damit anfangen werden dieses umzusetzen. Dabei ist es Sinnvoll mit den Bereichen des Projektes zu beginnen, von denen man annimmt, dass diese in einer ähnlichen Form gebraucht werden. Dies wird in mit dem Muster,'GET ON WITH IT' (Fang damit an!) beschrieben.[44, S. 62-65] So zeigte sich nach Coplien und Harrison, dass es oftmals gar nicht notwendig ist, die vollständigen Spezifikationen zu kennen. Die Anpassungen, die am Ende nötig sind, um den tatsächlichen Anforderungen zu genügend, sind selten so groß wie befürchtet, Wenn mit den Bereichen begonnen wird die am sichersten sind. Wendet man dieses Muster an, wäre es durchaus auch richtig gewesen direkt mit der Erweiterung der Schnittstelle zum Frontend zu beginnen, unabhängig von der Kenntniss dessen was bei der Evaluation von Christan Cykrit konkret herausgekommen wäre. Damit eng verbunden wären auch die Muster 'INFORMAL LABOR PLAN'[44, S. 88-89] sowie 'DEVELOPING IN PAIRS'[44, S. 214-217] nützlich für die bessere Abstimmung gewesen.

5.5.4 Zielstellungen S.M.A.R.T formulieren - Timeboxing

In einem Projekt, dass mehrere Monate beansprucht ist die genaue Zielsetzung und ein entsprechendes Zeitmanagement ein wichtiger Faktor für den Erfolg dieses Projektes.[49] Insofern ist es nötig, für formulierte Ziele einen Zeitrahmen zu definieren. Rückblickend bleibt festzuhalten, dass eine effektive Bearbeitung desto schwieriger ist, je ungenauer das Ziel und die geplante Bearbeitungszeit formuliert wurde. Zum Verständnis will ich eine getroffene Aussage in meiner Planung mit einer möglichen alternativen Formulierung vergleichen und beleuchten wo die Vorteile in der zweiten Variante liegen.

1. *In den nächsten Wochen werde ich die GUI-Analyse abschließen.*
2. *14-Tage plane ich für die Analyse, welche der vorhanden GUI-E in Saros/E in der HTML-GUI umgesetzt werden können.*

Genau betrachtet stellt die Formulierung [1] gar keine richtige Zielplanung dar. Denn Zielstellungen sind nur nützlich, wenn deren Erfolg auch überprüfbar ist. So sollten Ziele S.M.A.R.T formuliert werden. S.M.A.R.T. steht hierbei als Akronym für '**S**pecific **M**easurable **A**ccepted **R**ealistic **T**imely'. [50].
So ist die Formulierung [2] wesentlich genauer und damit hilfreicher. Der Zeitrahmen ist klar definiert, die Aufgabenstellung ist messbar und spezifisch. Ich habe festgestellt, dass vor allem der Aspekt der Terminierung effektiv für die Planung und Durchführung eines Projektes ist. Damit verwandt ist das Prinzip des Time-Boxing (fester Zeitrahmen) ,welches auch

Buchstabe	Bedeutung	Beschreibung
S	Spezifisch	Ziele müssen eindeutig definiert sein (nicht vage, sondern so präzise wie möglich).
M	Messbar	Ziele müssen messbar sein (Messbarkeitskriterien).
A	Akzeptiert	Ziele müssen von den Empfängern akzeptiert werden/sein (auch: angemessen, attraktiv, abgestimmt ausführbar oder anspruchsvoll.
R	Realistisch	Ziele müssen möglich sein.
T	Terminiert	zu jedem Ziel gehört eine klare Terminvorgabe, bis wann das Ziel erreicht sein muss.

Tabelle 3: Formulierung von Zielen nach SMART Methode. Quelle:[51]

in Agilen Prozessen und Scrum Anwendung findet.[52]. Die Idee ist, dass es für eine Sache, beispielsweise in Scrum ein monatlicher Sprint, einen festen Zeitrahmen gibt. Dieser wird, nach dem er einmal formuliert wurde, NICHT angepasst. Wenn, wie in [2] formuliert die Rede von 14-Tagen ist, dann werden, auch nur genau 14 Tage für die Analyse verwendet. Das Verschieben und somit Vergrößern des Zeitrahmens für den Abschluss einer Aufgabe ist nicht erlaubt. Auch und insbesondere dann nicht, wenn das Ziel nur teilweise erfüllt wurde. Nach Ablauf des gesetzten Zeitrahmens wird das Ergebnis festgehalten und mit weiteren Aufgaben begonnen. Es ist hierbei durchaus legitim, nach Ablauf des Zeitrahmens eine neue Zielformulierung zu erstellen, die unerledigte Teilaufgaben des letzten enthält. Allerdings darf dies nicht die selbe Zielsetzung sein wie im Zeitrahmen zuvor, sonst wäre dies nur ein künstlicher Zeitrahmen ohne limitierenden Effekt. Diese Praktik hat den großen Vorteil das Zeitplanung sich nicht so leicht 'dehnen' lassen, wie es die Zielformulierung [1] sehr einfach zulässt. Die Praxis erfordert allerdings einiges an Disziplin von allen Beteiligten, selbst gesetzte Deadlines auch als solche wahrzunehmen, und nicht mehr einfach zu verschieben. Nützlich kann dies vor allem für Personen sein (zu denen ich mich hinzuzähle), welche dazu neigen, erst unter Termindruck effektiv zu arbeiten. Ein andere effektiver Vorteil des Time-Boxing ist, dass es viele Arbeiten gibt, bei denen man gar nicht sagen kann, wenn diese vollständig abgeschlossen sind. So bietet gerade das gewählte Beispiel einer Analyse eine ideale Möglichkeiten sie (zu)lange, (zu)ausführlich und (zu)detailliert zu machen. So schützt ein effektiv genutztes Time-Boxing auch davor Zeit und Ressourcen zu verschwenden.

Time-Boxing ist nützlich um dem **Paretoprinzip**, besser bekannt als "**80-20-Regel**", zu entgehen. Die 80-20-Regel besagt, "*dass 80 Prozent der Ergebnisse mit 20 Prozent des Gesamtaufwandes erreicht werden. Die ver-*

bleibenden 20 Prozent der Ergebnisse benötigen mit 80 Prozent die meiste Arbeit." [53] Diesem Problem erliegen vor allem Personen die zu Perfektionismus neigen, zu denen ich mich hinzuzähle und unter Informatikern keine seltene Eigenschaft ist.[48].

5.6 Fazit und zukünftige Arbeiten

Aufgrund des Umfangs und dem in 3.3.3 genannten Problemen war es leider nicht möglich, die GUI von Saros in der gegebenen Zeit vollständig umzusetzen. So sollten sich zukünftige Arbeiten mit der Finalisierung der neuen HTML-GUI beschäftigen. Im Folgenden werden diese kurz Aufgelistet und die erforderlichen Arbeiten beschrieben.

Fortschrittsanzeige für Session Aufbau. Die Fortschrittsanzeige beim Teilen von Projekten kann von der UI für das Frontend zur Verfügung gestellt werden. Dafür muss ein IProgressMonitor im Kern-Modul eingeführt werden, welche von jeder IDE implementiert werden sollte. Über diesen und einer entsprechenden Javascript aufrufen, können entsprechende Wizards auf diese Informationen zugreifen und anzeigen.

Vervollständigung des Negotiationhandler. Da der *SarosSessionMangager* die Negotiation-Handler verwaltet müsste dieser zuvor in das Kernmodul überführt werden. Das selbe trifft auf die Projekt-Negotiation Klassen zu. Zum Zeitpunkt der Fertigstellung dieser Arbeit beschäftigt sich Denis Washington mit dieser Aufgabe. Sobald diese im Kern sind, müsste der INegotiationHandler[49] und deren Implementation innerhalb des UI-Moduls angepasst werden. Mit der Überführung der genannten Klassen in den Kern wären auch automatisch die noch benutzten IDE-spezifischen GUI-Elemente die während es Sessionaufbaus zum Einsatz kommen leicht zu deaktivieren, da die alten IDE-spezifischen Listener nicht mehr gebraucht werden.

Zugriff auf Saroseinstellungen. Um Wizards den Zugriff auf Einstellungen zu ermöglichen muss eine neue Core-Fassade und entsprechende BrowserFunktions bereitgestellt werden. Diese Fassade kann voraussichtlich ohne größeren Aufwand umgesetzt werden, da er ohne die Erzeugung von komplexeren Datenmodellen auskommt und die benötigten Komponenten bereits im Kern vorhanden sind. So wird hierfür eine Fassade gebraucht die den *IPreferenceStore* des Kern-Moduls benutzt. Eine neue *PreferencesBrowserFunktion-Klasse* scheint angebracht um den Zugriff auf den IPreferenceStore zu kapseln. Zu beachten ist hierbei lediglich, dass nicht alle Einstellungen von allen IDEs unterstützt werden, daher müsste vorher ein Subset an möglichen

[48] Dies ist eine persönliche Einschätzung, aus Erfahrungen meines Informatikstudium und mit Teamkollegen

[49] http://saros-build.imp.fu-berlin.de/gerrit/#/c/2769/

Einstellungen ermittelt werden. Dieses Subset an Einstellungsmöglichkeiten sollte jedoch nahezu deckungsgleich mit den in *EclipsePreferenceConstants* definierten Standardeinstellungen sein.

Bereitstellung der Awarness-Informationen. Um den vollen Funktionsumfang der alten Saros/E-GUI zu gewährleisten, müsste dieses Feature noch implementiert werden. Dafür müssten die Actionen aus dem Kern ausgewertet in ein entsprechend konzipiertes Datenmodell umgewandelt und an der Frontenend übertragen werden. Ein interessanter Aspekt, und neu für die jetzige HTML-GUI, ist der Umgang mit sich sehr schnell und oft ändernden Anzeigeinformationen. Während sich Kontakt-,Account- und Projektdaten nicht oft verändert werden müssen, ändern sich die Awarness-Informationen mehrmals die Sekunde.

Testframework für HTML-GUI. Mit den ersten JUnit-Test für den komplexeren ProjektListManager ist hier schon ein Grundstein gelegt. Mit steigenden Funktionsumfang, oder spätestens mit Inbetriebnahme der HTML-GUI ist eine größere Testabdeckung wünschenswert. Neben der Erweiterung der Testabdeckung der UI-Schicht, sollte das STF für Oberflächentest erweitert werden und für die Verwendung in HTML angepasst werden. Christian Cykrit hat hier bereits Vorarbeit geleistet welche es noch ausbauen gilt.[5, S. 62]

Alles in allen ist die Einführung einer HTML basierten GUI mit einheitlicher Codebasis für verschiedene IDEs ein ambitioniertes Projekt, welches jedoch, großes Potential für die Weiterentwicklung von Saros besitzt.

Literatur

[1] Riad Djemili. „Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierungverteilter Paarprogrammierung“. Diplomarbeit. Freie Univeristät Berlin, 2006.

[2] Riad Djemili, Christopher Oezbek und Stephan Salinger. „Saros: Eine Eclipse-Erweiterung zur verteilten Paarprogrammierung.“ In: *Software Engineering (Workshops)*. Hrsg. von Wolf-Gideon Bleek, Henning Schwentner und Heinz Züllighoven. Bd. 106. LNI. GI, 2007, S. 317–320. ISBN: 978-3-88579-200-0. URL: http://www.inf.fu-berlin.de/inst/ag-se/pubs/saros-2007.pdf.

[3] Wikipedia. *Integrierte Entwicklungsumgebung*. 2015. URL: https://de.wikipedia.org/wiki/Integrierte_Entwicklungsumgebung (besucht am 02. 08. 2015).

[4] Bill Venners. *Orthogonality and the DRY Principle*. 2003. URL: http://www.artima.com/intv/dry.html (besucht am 08. 08. 2015).

[5] Christian Cikryt. „Evaluating the Use of a Web Browser to Unify GUI Development for IDE Plug-ins“. Masterarbeit. Freie Univeristät Berlin, 2015. URL: http://www.inf.fu-berlin.de/w/SE/ThesisTestingSaros.

[6] *Saros-for-Intellij*. 2015. URL: http://www.saros-project.org/saros-for-intellij (besucht am 28. 07. 2015).

[7] AG-Software-Engineering. *Innovationseinführung von Saros zur verteilten kollaborativen Softwareentwicklung in Unternehmen*. FU Berlin. 2015. URL: http://www.inf.fu-berlin.de/w/SE/ThesesDPP#EinfuehrungUnternehmen (besucht am 08. 08. 2015).

[8] AG Software Engineering. *Einführung von Saros in Open-Source-Projekten*. FU Berlin. 2015. URL: http://www.inf.fu-berlin.de/w/SE/ThesesDPP#EinfuerungUnternehmen (besucht am 08. 08. 2015).

[9] AG Software Engineering. *Portierung von Saros auf andere Entwicklungsumgebungen*. FU Berlin. 2015. URL: http://www.inf.fu-berlin.de/w/SE/ThesesDPP#PortierungIDEs (besucht am 08. 08. 2015).

[10] Arndt Lasarzik. „Refaktorisierung des Eclipse Plugins Saros für die Portierung auf andere IDEs“. Bachlorarbeit. Freie Universität Berlin, 30. Apr. 2015. URL: http://www.inf.fu-berlin.de/w/SE/ThesisRefaktorisierungFuerAndereIDEs.

[11] Daniela Benze. „Erweiterung der Tests für Intellij+Saros (Arbeitstitel)“. Bachlorarbeit. Freie Universität Berlin, 2015. URL: http://www.inf.fu-berlin.de/w/SE/ThesisSarosIntellijTest (besucht am 08. 08. 2015).

[12] Denis Washington. „Weiterentwicklung und Evaluation eines dedizierten Servers für das SAROS-System (Arbeitstitel)“. Masterarbeit. Freie Universität Berlin, 2015. URL: https://www.mi.fu-berlin.de/w/SE/ThesisSarosServerStandalone (besucht am 08. 08. 2015).

[13] Sabine Bender. „Prototyp einer Netbeans Version von Saros (Arbeitstitel)“. Masterarbeit. Freie Universität Berlin, 2015.

[14] Florian Deissenboeck Elmar Juergens und Benjamin Hummel. „Code Similarities Beyond Copy & Paste“. In: *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference*. (15. März 2010). IEEE, 2010, S. 78–87. ISBN: 978-1-61284-369-8. URL: https://www.cqse.eu/publications/2010-code-similarities-beyond-copy-paste.pdf (besucht am 08. 08. 2015).

[15] *Eclipse Metrics Plugin*. 2015. URL: http://eclipse-metrics.sourceforge.net/ (besucht am 13. 04. 2015).

[16] Ralf Westphal; Stefan Lieser. *Single-Responsibility-Principle (SRP)*. de. 2015. URL: http://ccd.lieser-online.de/Oranger-Grad.ashx#Single_Responsibility_Principle_SRP_1 (besucht am 13. 04. 2015).

[17] Matthias Riebisch und Stephan Bode. „Software-Evolvability“. In: *Informatik-Spektrum. 32, Nr. 4* (2009). URL: http://swk-www.informatik.uni-hamburg.de/~riebisch/publ/riebischbode_software-evolvability.pdf (besucht am 23. 04. 2015).

[18] Margaret Rouse. *Transparenzbegriff in Software aus Benutzersicht*. 2010. URL: http://whatis.techtarget.com/definition/transparent (besucht am 03. 04. 2015).

[19] Jim Shore. *Fail Fast*. IEEE Computer Society. 2004. URL: http://www.martinfowler.com/ieeeSoftware/failFast.pdf (besucht am 08. 03. 2015).

[20] David L. Parnas. „On the Criteria To Be Used in Decomposing Systems into Modules,“ in: *Communications of the ACM* (1972).

[21] Patrick Schlott. „Analyse und Verbesserung der Architektur eines nebenl¨aufigen und verteilten Softwaresystems“. Masterarbeit. Freie Universität Berlin, 2013.

[22] Bastian Sieker. „User-Centered Development of a JavaScript and HTML-based GUI for Saros. GUI module“. Stand: 2015-07-26. Masterarbeit. Freie Univeristät Berlin, 31. Aug. 2015. URL: http://www.inf.fu-berlin.de/w/SE/ThesisUCDSarosGUI (besucht am 13. 08. 2015).

[23] JetBrains. *Community Edition Plugins for IntelliJ IDEA*. JetBrains. 2015. URL: https://plugins.jetbrains.com/?idea (besucht am 03. 06. 2015).

[24] The Eclipse Foundation. *Home of Eclipse Plugins, Bundles and Products - Eclipse Marketplace*. The Eclipse Foundation. 2015. URL: https://marketplace.eclipse.org/ (besucht am 03. 06. 2015).

[25] Karsten Wendland. *Der Template-Zyklus. Web-Templates im Spannungsfeld von schöpferischem Gestalten und einschränkender Zumutung*. Shaker, 2006. ISBN: 9783832252854.

[26] W3C. *Cascading Style Sheets*. 2015. URL: http://www.w3.org/Style/CSS/ (besucht am 06. 04. 2015).

[27] XMPP. *Jabber Software Foundation Publishes Open VoIP and Multimedia Protocols*. The Jabber Software Foundation. 2005. URL: http://xmpp.org/xsf/press/2005-12-15.shtml (besucht am 14. 08. 2015).

[28] Damla Durmaz. „Verbesserung der Action Awareness im Open Source Plug-in Saros“. Masterarbeit. Masterarbeit am Institut für Informatik der Freien Universität Berlin, Arbeitsgruppe Software Engineering, 2015.

[29] Damla Durmaz. *Code-Review - Improves the performance of the activity log*. 2015. URL: http://goo.gl/0xe9lP (besucht am 14. 08. 2015).

[30] Michael Jurke. *A whiteboard for Saros. To develop a whiteboard functionality for distributed pair programming, iterative by prototypes*. Hrsg. von -. Institut für Informatik Fu Berlin. 2010. URL: http://www.inf.fu-berlin.de/inst/ag-se/teaching/S-BSE/170_jurke_whiteboard-saros-concept.pdf.

[31] Google. *Hauptseite des Angular-Framworks*. Google. 2015. URL: https://angularjs.org/ (besucht am 29. 06. 2015).

[32] C. Oezbek und L. Prechelt. „JTourBus: Simplifying Program Understanding by Documentation that Provides Tours Through the Source Code.“ In: 23th IEEE International Conference on Software Maintenance (ICSM'07). 2007.

[33] Ralf Westphal und Stefan Lieser. *Clean-Code-Developer Guid - Roter Grad*. 2015. URL: http://clean-code-developer.de/die-grade/roter-grad/ (besucht am 21. 07. 2015).

[34] Erich Gamma u. a. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[35] Hanspeter Mössenböck. *Objektorientierte Programmierung*. Springer-Verlag, 1993. ISBN: ISBN 3-540-55690-7.

[36] Martin Fowler. *Data Transfer Object*. 2015. URL: http://martinfowler.com/eaaCatalog/dataTransferObject.html (besucht am 17. 05. 2015).

[37] *Eclipse IWorkspaceRoot Interface*. 2015. URL: http://help.eclipse.org/mars/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/api/org/eclipse/core/resources/IWorkspaceRoot.html (besucht am 12. 07. 2015).

[38] Martin Fowler. *ValueObject*. 2015. URL: http://martinfowler.com/bliki/ValueObject.html (besucht am 13. 04. 2015).

[39] *Gson 2.3.1 API Documentation*. Google Inc. 2015. URL: https://google-gson.googlecode.com/svn/trunk/gson/docs/javadocs/com/google/gson/Gson.html (besucht am 12. 06. 2015).

[40] *Object references with JSON*. 2015. URL: https://en.wikipedia.org/wiki/JSON#Object_references (besucht am 12. 06. 2015).

[41] Martin Fowler. *Mocks Aren't Stubs*. 2007. URL: http://martinfowler.com/articles/mocksArentStubs.html (besucht am 13. 05. 2015).

[42] Martin Fowler. *Inversion of Control Containers and the Dependency Injection pattern*. Jan. 2004. URL: http://www.martinfowler.com/articles/injection.html (besucht am 04. 08. 2015).

[43] *Guard Clause pattern*. 2009. URL: http://c2.com/cgi/wiki?GuardClause (besucht am 19. 05. 2015).

[44] Coplien und Harrison. *Organizational Patterns of Agile Software Development*. Pearson Prentice Hall, 2005. URL: http://www.inf.fu-berlin.de/inst/ag-se/teaching/S-AGILE-2014/lit/CopHar05.pdf.

[45] David Green. *If I had more time I would have written less code*. 9. Feb. 2011. URL: http://blog.activelylazy.co.uk/2011/02/09/if-i-had-more-time-i-would-have-written-less-code/ (besucht am 04. 08. 2015).

[46] Robert C. Martin. *Clean Code: Refactoring, Patterns, Testen und Techniken für sauberen Code*. 1. Aug. 2008.

[47] Olivier D Serrat. *The Five Whys Technique*. 2009. URL: http://www.adb.org/sites/default/files/publication/27641/five-whys-technique.pdf (besucht am 10. 07. 2015).

[48] John C. Maxwell. *Failing Forward: Turning Mistakes into Stepping Stones for Success*. 2012.

[49] Liz Cassidy. *Project Management: Time estimates and planning*. ProjectSmart.co.uk. 2015. URL: https://www.projectsmart.co.uk/project-management-time-estimates-and-planning.php (besucht am 12. 08. 2015).

[50] George T. Doran. „There's a S.M.A.R.T. way to write management's goals and objectives". In: *Management Review, Volume 70*. Issue 11(AMA FORUM). 1981, S. 35–36.

[51] *SMART (Projektmanagement)*. 2015. URL: https://de.wikipedia.org/wiki/SMART_(Projektmanagement) (besucht am 15. 08. 2015).

[52] Ken Schwaber. *Agile Project Management with Scrum*. New York: O'Reilly Media, Inc., 2009. ISBN: 978-0-7356-3790-0.

[53] *Pareto principle in Software*. 2015. URL: https://en.wikipedia.org/wiki/Pareto_principle#In_software (besucht am 12. 08. 2015).

A Anhang

A.1 Liste eingereichter Commits

Hinweis: Die Nummerierung stellt keine zeitliche Reihenfolge dar.

1. **[UI] Deleted the hint about several XMPP provider**
   https://github.com/saros-project/saros/commit/a43c1a4
2. **[INTERNAL][API] changed BrowserPage getWebpage() IFSPEC**
   https://github.com/saros-project/saros/commit/4a64b0e
3. **[UI] Use new HTML-UI in Saros/Eclipse**
   https://github.com/saros-project/saros/commit/0ed3d13
4. **[HTML][UI] Adds StartSessionInvitation to UI**
   https://github.com/saros-project/saros/commit/9c4e3c7
5. **[HTML][INTERNAL] Renderer can be used in multiple browsers**
   https://github.com/saros-project/saros/commit/bdd1edd
6. **[HTML][REFECACTOR] Remove Saros naming from UI Project**
   https://github.com/saros-project/saros/commit/d1b08a2
7. **[REFACTOR][I] Changes "filesystem"package naming to match common pattern**
   https://github.com/saros-project/saros/commit/c0ff1a3
8. **[INTERNAL] Adds WorkspaceRoot implementation to Eclipse**
   https://github.com/saros-project/saros/commit/0898a22
9. **[HTML][INTERNAL] Introduce ProjectTree model for UI**
   https://github.com/saros-project/saros/commit/0385ab9
10. **[HTML][UI] Introduces ProjectListRenderer**
    https://github.com/saros-project/saros/commit/6818ca5
11. **[INTERNAL][HTMl] Fixes for ProjectListManager**
    https://github.com/saros-project/saros/commit/ee888534ee
12. **[HTML][Internal] Changes UI_BUNDLE_ID to ui.frontend**
    https://github.com/saros-project/saros/commit/7614a06
13. **[HTML][INTERNAL] Adds JID property to Contactmodel**
    https://github.com/saros-project/saros/commit/ebd76e7

14. **[REFACTOR][I] Renames Resource- and FileImp to common name pattern**
    https://github.com/saros-project/saros/commit/ea16945
15. **[INTERNAL][HTML] Introduces ProjectListManager**
    https://github.com/saros-project/saros/commit/9b28feb
16. **[JUNIT][HTML] Introduces basic ProjectListManagerTest with JUnit**
    http://saros-build.imp.fu-berlin.de/gerrit/#/c/2749/
17. **[API][HTML] Introduces ICollaborationUtils and IDE implementations**
    https://github.com/saros-project/saros/commit/8d4c43f
18. **[INTERNAL][HTML] Adds session Invitation BrowserFunctions**
    https://github.com/saros-project/saros/commit/862c06c
19. **[REFACTOR][DOC][HTML] Refactors BrowserFunctions**
    http://saros-build.imp.fu-berlin.de/gerrit/#/c/2777/
20. **[INTERNAL]Prepares UI backend for HTML JoinSessionWizard**
    http://saros-build.imp.fu-berlin.de/gerrit/#/c/2769/
21. **[API][HTML] Introduces JavaScriptApi**
    http://saros-build.imp.fu-berlin.de/gerrit/#/c/2780/
22. **[REFACTOR][HTML] Refactors Model package to clarify functionality**
    http://saros-build.imp.fu-berlin.de/gerrit/#/c/2779/

A.2 Liste mitgewirkter Commits

- **[INTERNAL] Reorganized class structure for HTML UI**
  https://github.com/saros-project/saros/commit/c112539
- **[INTERNAL][HTML] Encapsulate application state in designated model**
  https://github.com/saros-project/saros/commit/95d6689
- **[HTML]BrowserPage-getTitle to show title in dialogs**
  https://github.com/saros-project/saros/commit/571719d

A.3 Liste behobener Dokumentationsprobleme

- **Troubleshooting : Eclipse editor technicalities**
  http://sourceforge.net/p/dpp/documentation-issues/129/
- **Setup Own XMPP Server : Suitable Jabber servers**
  http://sourceforge.net/p/dpp/documentation-issues/128/
- **Setup Own XMPP Server : GTalk JID Setup**
  http://sourceforge.net/p/dpp/documentation-issues/127/
- **Saros - TechDoc - Network Layer - Data Connections in Saros**
  http://sourceforge.net/p/dpp/documentation-issues/126/
- **Page on 'Installation Usage' missing**
  http://sourceforge.net/p/dpp/documentation-issues/123/
- **112 Saros - Troubleshooting - Hanging invitation**
  http://sourceforge.net/p/dpp/documentation-issues/112/
- **Updating Sites which mention old features (VoiP, Screensharing, Needbased-Sync)**
  http://sourceforge.net/p/dpp/documentation-issues/131/

A.4 Codeauszug JTourBus-HTML-GUI-Prototyp

```
// HtmlJavaScriptAdapter.java
package de.fu_berlin.inf.jtourbus.view;

import org.eclipse.jdt.core.ISourceRange;
import org.eclipse.swt.browser.Browser;
import de.fu_berlin.inf.jtourbus.TourPlan;
import de.fu_berlin.inf.jtourbus.model.BusStopNew;
import de.fu_berlin.inf.jtourbus.model.TourPlanNew;
/**
* This Class provides the connection between the HTML
    Javascript UI and the
* business logic
*
* @author Matthias
*/
public class HtmlJavaScriptAdapter {

private Browser browser;
private TourPlanNew jTourPlan = new TourPlanNew();

public HtmlJavaScriptAdapter(Browser browser) {
this.browser = browser;
}
```

```
/**
* Trigger the build of the tours. The AST Model will be
    updated.
*
* Notice: This operation will block the UI till it's
   finished.
*
* @return the current TourPlan to render it
*/
public TourPlanNew rebuildTours() {
// Call CORE to rebuild TourPlan
// Return the the TourPlan
return jTourPlan;
}

/**
* Renaming a given BusStop. This don't change the
   structure of the AST, but
* will change the name param of the BusStop.
*
* Notice: This operation will block the UI till it's
   finished.
*
* @param id
*            the ID of the BusStop that should be
   renamed
* @param newName
*            the new Name for the BusStop
* @return the current TourPlan to render it
*/
public TourPlanNew setBusStopDescription(String id,
   String newDescription) {
// Get the right BusStop
BusStopNew busStop = jTourPlan.getBusStop(id);

// Update the Busstop description
busStop.setDescription(newDescription);

// Update the TourPlan
// Return the updated TourPlan to render it
return jTourPlan;
}

/**
* Append a new BusStop after the given BusStop f.e. if
   the parent ID =
* Tour_1.0, the ID of new BusStop will become Tour_1.1.
*
```

```
* Notice: This operation will block the UI till it's
    finished.
*
* @param id
*            the id of the parent BusStop
* @return the current TourPlan to render it
*/
public TourPlanNew appendJTourBus(String id) {
// Create new Busstop
// UpdateTourPlan
// Return Tourplan

return jTourPlan;
}
/**
* Nest a new BusStop after the given BusStop f.e. if
    the parent ID =
* Tour_1.0, the ID of new BusStop will become Tour_1
    .0.1
*
* Notice: This operation will block the UI till it's
    finished.
*
* @param id
*            the id of the parent BusStop
* @return the current TourPlan to render it
*/
public TourPlanNew nestJTourBus(String id) {
// Create new Busstop
BusStopNew busStop = new BusStopNew(null, null, null,
    null) {

@Override
public ISourceRange getSourceRange() {
// TODO Auto-generated method stub
return null;
}
};
// UpdateTourPlan
return jTourPlan;

}
public TourPlanNew removeBusStop(String ID){
//remove BusStop
jTourPlan.routes.remove(ID);
return jTourPlan;
}

/**
```

```
* This updates all BusStops with the given tour and
    changes their ID's
*
* f.e. tourname = "MainTour". All BusStops with the ID
    pattern
* "MainTour_x.x) will be changed to the newTourname.
*
* Notice: This operation will block the UI till it's
    finished.
*
* @param oldTourname
*            the name of the tour which should be
    renamed
* @param newTourname
*            the new name of the tour.
* @return the current TourPlan to render it
*/

public TourPlanNew renameTour(String oldTourname,
    String newTourname) {
// Find all BusStops of the given Tour
// Change the BusStops IDs accordingly
// UpdateTourPlan
// Return Tourplan

return jTourPlan;
}

public void goToBusStop(String ID) {
// Call the CoreLogic to change the editor curser to
    the given BusStop
// Maybe return some "done" notification.
// UI don't need to be blocked
}
}
```

A.5 Aufgabe Entwicklertest

Aufgabe
Füge ein neues Preferences Fenster zur HTML GUI hinzu.

Funktionsumfang des Dialogfensters

1. **View Accounts**
   Zeige alle momentan verfügbaren (vorhandenen) XMPP Accounts an.

2. **Add Account**
   Es soll möglich sein weitere Accounts hinzuzufügen.

3. **Edit Account**
   Es soll möglich sein vorhandene Accounts zu editieren.

4. **Remove Account**
   Es soll möglich sein Accounts zu löschen.

5. **Set default Account**
   Auswählen eines vorhanden Accounts welcher zum auto connect verwendend wird (Aktiver Account) – Der momentan aktive Account sollte visuell hervorgehoben werden.

6. **Confirm remove**
   Der User muss das Löschen von Accounts bestätigen (siehe Hinweis).

Weitere Anforderungen

- Änderungen (Hinzufügen, editieren, löschen von Accounts) sollen die View aktualisieren.

- Die View muss in Saros/E sowie in Saros/J benutzbar sein und in beiden den gleichen Funktionsumfang haben.

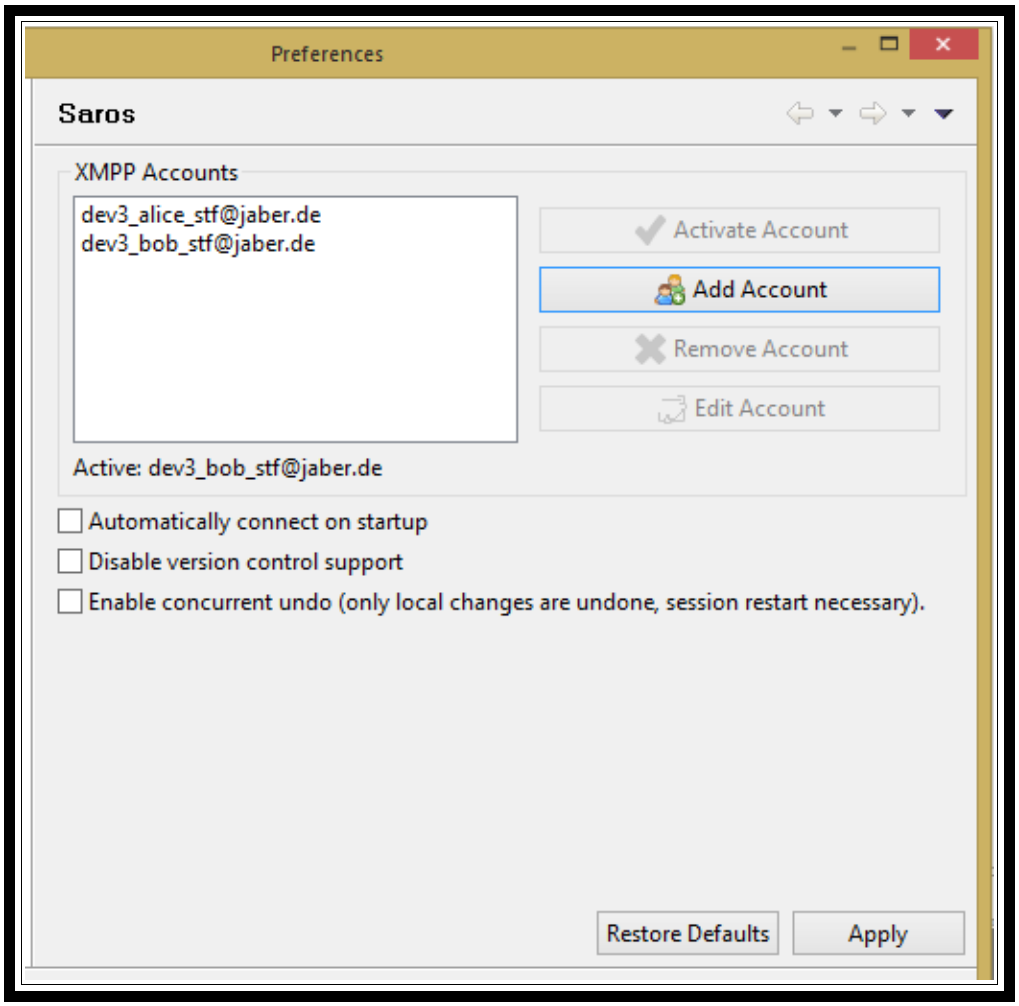
- Das Dialogfenster soll aus der Hauptview zu öffnen sein.

(Erweitert – wenn noch Zeit bleibt)
Verhalten des Dialogfensters ändern

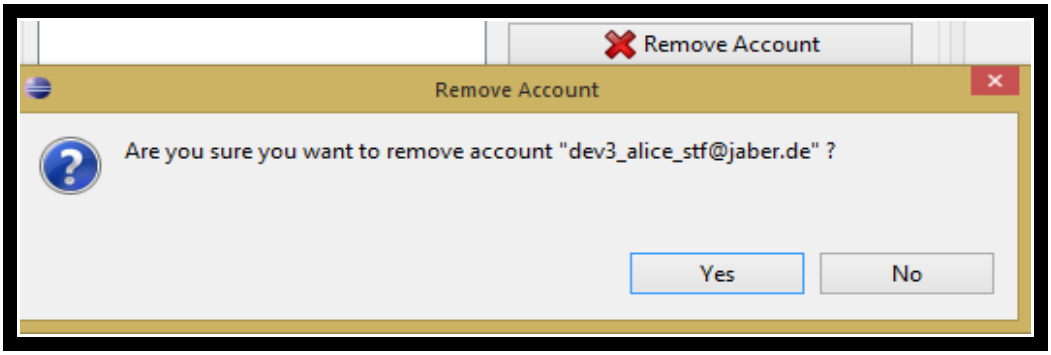
Das Dialogfenster soll im Vordergrund bleiben solange es offenen ist. Die jeweilige IDE soll also nicht benutzbar sein bis der Dialog geschlossen wird. (Klicken auf irgendwas anderes als das Fenster aktiviert es und lässt es in den Vordergrund schieben)

Hinweise

Als Referenz kann die aktuelle SWT UI aus Saros/E dienen:



Die remove Account Bestätigung soll in einem Popup angezeigt werden. (Zusatz: Welches den Dialog „sperrt“ solange aktiv, schließen ist gleich bedeutend mit „no“)



- Die Checkboxen sowie „Restore Defaults“ und „Apply“ werden nicht benötigt.
- Add Account kann den bereits bestehenden AddAccountWizard benutzten sollte dies nützlich erscheinen.
- Schönheit kommt nach Funktionalität – während das Anordnen der GUI Elemente und praktikable anzeigen der Funktionen wichtig ist, muss nicht all zu viel Wert auf wunderschöne CSS Funktionen gesetzt werden. (Bootstrap und ein basic css sind verfügbar).

A.6 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.

Datum

Name