Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Software Engineering

Analyse und Evaluation von Verbesserungsmöglichkeiten der User Experience in Saros

Alexander Narweleit
Matrikelnummer: 4206380
anarweleit@hotmail.de

Betreuer: Julia Schenk
Eingereicht bei: Prof. Dr. Lutz Prechelt
Zweitgutachter: Prof. Dr. Elfriede Fehr

Berlin, 17.11.2011

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 verwendeten Abbildungen sind von mir selbst erstellt worden oder mit entsprechenden Quellennachweisen bei der Referenzierung der betroffenen Abbildungen versehen. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

17.11.2011

Alexander Narweleit

Zusammenfassung

Bei der IT-Produktentwicklung wird immer mehr Wert darauf gelegt, dass die Erfahrungen, die ein User beim Benutzen dieses Produktes macht, ihn dazu animieren, die Nutzung des Produktes fortzusetzen. Dieses Nutzerlebnis ist insbesondere durch Erwartungswerte des Benutzers bestimmt, welche im Laufe der Interaktion mit dem Produkt entweder erfüllt werden oder nicht. Dieser und eine Reihe weiterer Aspekte, welche auch im Kontext der Nutzungserfahrung stehen, werden durch den Begriff der „User Experience“ zusammengefasst und geben Aufschluss darüber, inwieweit der Umgang mit dem jeweiligen Produkt aus User-Sicht angenehm ist. Es handelt sich hierbei also um ein Kriterium dafür, ob eine Software weitläufig verbreitet ist und genutzt wird.

Die vorliegende Arbeit beschäftigt sich mit der Frage, wie die User Experience in dem an der Freien Universität Berlin entwickelten Eclipse-Plugin Saros verbessert werden kann. Hierzu werden andere Projekte, welche - wie Saros - die verteilte Zusammenarbeit ermöglichen, analysiert und verglichen. Daraus resultiert eine Menge an Features und Konzepten, welche nach definierten Kriterien ausgewertet und auf ihre potentielle Umsetzbarkeit in Saros untersucht werden. Ziel ist es, Mechanismen zu entdecken, welche die User Experience in Saros dahingehend verbessern, dass der Umgang mit dem Plugin angenehmer wird. Weiterhin werden innerhalb von Saros Aspekte der Nutzer-Interaktion betrachtet und verbessert, welche negative Auswirkungen auf das Nutzungserlebnis haben könnten.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Aufbau dieser Arbeit . . . 1
- 1.2 Notationen . . . 2

2 Definition von User Experience **3**

3 Verschiedene Zustände der User in Saros **4**
- 3.1 Mögliche Zustände der Nutzer . . . 4
- 3.2 Grundlegende Interaktionsmöglichkeiten in Saros . . . 5
- 3.3 Überblick über Userzustände und die damit verbundenen Interaktionsmöglichkeiten . . . 6

4 Arbeitsansatz: Analyse von Saros und Vergleiche mit anderen Projekten **10**

5 Erkannte Probleme in den Kontextmenüs **11**
- 5.1 Buddies einladen, die offline sind . . . 11
- 5.2 Irreführende Nicht-Selektierbarkeit . . . 12
- 5.3 Multiples Sharing und Einladen trotz nicht vorhandener Elemente . . . 13
- 5.4 Keine Skype-Calls innerhalb einer Session . . . 14

6 Vorstellung der Vergleichsprojekte **15**
- 6.1 PeP - Pair Eclipse Programming . . . 16
- 6.2 google docs . . . 18
- 6.3 MoonEdit . . . 19
- 6.4 MeetingWords . . . 21
- 6.5 Adobe Connect . . . 22
- 6.6 SubEthaEdit . . . 24

7 Kriterien zur Evaluation **26**
- 7.1 Technische Umsetzbarkeit . . . 26
- 7.2 Mehrwert für die User Experience . . . 26
- 7.3 Passt das Konzept zu Saros? . . . 27

8 Features und Konzepte der anderen Projekte **28**

9 User Stories zu ausgewählten Features **34**

10 Im Rahmen dieser Arbeit umgesetzte Änderungen **36**
- 10.1 Grundlage: Benutzung der plugin.xml . . . 36
- 10.2 Skype-Button innerhalb der Session . . . 37
- 10.3 Irreführende Nichtselektierbarkeit . . . 39

10.4 „Multiple“-Funktionen erst ab zwei Einträgen verfügbar machen 41
10.5 Einladung zu Interaktionen 44

11 Resümee und Ausblick **47**

12 Zusatz: Konkurrenzfähigkeit von Saros **49**

13 Anhang **54**
13.1 Quellcode 54

Abbildungsverzeichnis **58**

1 Einleitung

Bei Saros handelt es sich um ein Plugin für die Entwicklungsumgebung Eclipse, welches es Nutzern ermöglicht, gleichzeitig von verschiedenen PCs aus in dem gleichen Software-Projekt Änderungen vorzunehmen, also verteilte Paarprogrammierung durchzuführen. Es ist das Ergebnis von Riad Djemilis Diplomarbeit aus dem Jahre 2006 [Dje06] und der Zuarbeit vieler Studenten, die ebenso ihre Abschlussarbeiten im Rahmen dieses Projektes geschrieben haben. Diese hatten größtenteils die Entwicklung von Funktionalitäten zum Inhalt, weswegen ein Fokus auf die Benutzbarkeit aus User-Sicht nicht in vollem Umfang gegeben war.
In der klassischen Paarprogrammierung nimmt jeder der beiden Beteiligten eine Rolle ein - Driver oder Observer. Hierbei ist der Driver derjenige, welcher den Code eintippt, während der Observer auf Fehler hinweist und auch die Codequalität beachtet. Dies ist ebenso nützlich, um Wissen über den Code zu transferieren und Problemlösungen schneller zu finden [CW00]. Saros setzt diese agile Entwicklungsmethode um, erlaubt jedoch die Existenz mehrerer Driver und Observer. Diese finden sich in einer sogenannten „Session“ zusammen, im Rahmen derer die zu bearbeitenden Dateien verteilt und daraufhin gemeinsam editiert werden können. Die Abwicklung findet über das Extensible Messaging and Presence Protocol (XMPP) statt, welches in der Lage ist, Nachrichten und Präsenzinformationen zwischen XMPP-Accounts auszutauschen.
Wenn man nun in Erwägung zieht, dass 'das Internet und einzelne Geschäfts-Modelle [...] sogenannte „Kritische-Masse-Systeme“' [Ger11] sind, dann leitet sich daraus ab, dass der Wert und Nutzen dieses Projektes mit zunehmender Anzahl der User, die Saros verwenden, steigt. Aus diesem Grund ist es von fundamentaler Wichtigkeit, bei den Nutzern von Saros ein positives Benutzungserlebnis hervorzurufen, um das Vorankommen des Projekts und auch die Wert- und Nutzungssteigerung zu unterstützen.

1.1 Aufbau dieser Arbeit

Diese Arbeit lässt sich thematisch in drei Abschnitte unterteilen.
Der erste Abschnitt, welcher die Kapitel 2 bis 5 umfasst, dient der Schaffung eines genaueren Verständnisses für den Begriff der „User Experience“ und veranschaulicht, welche Auswirkungen die verschiedenen Zustände, die ein User während seiner Arbeit mit Saros einnehmen kann, darauf haben. Zu diesem Zweck werden die Interaktionsmöglichkeiten des Users in jedem dieser Zustände betrachtet und Problematiken in den Kontextmenüs von Saros aufgezeigt.
Die Kapitel 6 bis 9 beschäftigen sich mit Vergleichen zu anderen Projekten, welche auch die verteilte Zusammenarbeit ermöglichen. Es werden die Projekte, welche zur Evaluation herangezogen wurden, vorgestellt und die

Kriterien, nach welchen die Untersuchung erfolgte, werden erläutert. Kapitel 8 beinhaltet eine Auswertung aller bei den Vergleichsprojekten entdeckten Features und Konzepte, welche Auswirkungen auf die User Experience haben, und bewertet sie im Kontext einer möglichen Umsetzung in Saros nach den gegebenen Kriterien. Kapitel 9 gibt eine Übersicht über die Features, welche den Kriterien entsprachen.
Der dritte Teil zeigt auf, welche Änderungen im Rahmen dieser Arbeit durchgeführt wurden und diskutiert diese. Kapitel 11 fasst die Ergebnisse der Arbeit zusammen.
Das Zusatzkapitel 12 liefert eine Aussage darüber, inwiefern Saros gegenüber anderen Projekten, welche sich ebenso mit der verteilten Programmierung beschäftigen, konkurrenzfähig ist. Da eine solche Auswertung bisher nicht existiert und es viele Überschneidungen dieser Fragestellung mit meiner Recherche gibt, wurde ich von Prof. Dr. Lutz Prechelt um eine solche Darstellung gebeten.

1.2 Notationen

Quelltextauszüge und coderelevante Begriffe werden in `Maschinenschrift` dargestellt. Ist von einem Package oder einer Klasse `X` die Rede, so bezieht sich dies stets auf den Pfad `de.fu_berlin.inf.dpp.X` im Quellcode von Saros.

Direkte Zitate aus anderen Publikationen werden durch einfache Anführungszeichen (' ') kenntlich gemacht. Handelt es sich um eine indirekte Wiedergabe von Inhalten einer solchen Publikation, so verwende ich den Konjunktiv zu deren Wiedergabe.

2 Definition von User Experience

'I invented the term because I thought human interface and usability were to narrow. I wanted to cover all aspects of the persons experience with the system including industrial design, graphics, the interface, the physical interaction and the manual.' [Jor08]
(frei übersetzt: Ich erfand den Begriff, weil ich dachte, dass Benutzerschnittstelle und Bedienbarkeit [als Kriterium] zu eingeschränkt waren. Ich wollte alle Aspekte der Erfahrungen einer Person mit dem System, das Industrie-Design, die Grafik, das Interface, die physischen Interaktion und das Handbuch eingeschlossen, abdecken.)
Diese Worte aus einer Email von Donald Norman, welcher den Begriff der User Experience in den 90er Jahren prägte und verbreitete, lassen bereits erahnen, dass es sich hierbei um einen sehr vielschichtigen Begriff handelt. Auch Hans-Christian Jetter schrieb, dass die Bedeutung nicht auf 'Effektivität, Effizienz und Zufriedenheit bei der Benutzung' zu reduzieren sei, sondern auch emotionale Reaktionen mit einbeziehe, da diese hauptausschlaggebend dafür seien, ob ein User den Umgang mit dem Produkt als angenehm empfinde [Jet06]. Gleicht man dies mit Definitionen der Agentur „userfriend" ab, so fällt auf, dass sich diese Aufzählung von Qualitätsmaßen mit der Beschreibung von „Usability" deckt, welche als Gebrauchstauglichkeit beschrieben wird. Dies impliziere vor allem, dass das betrachtete System einfach zu nutzen sei [use11].

Somit lässt sich festhalten, dass die User Experience einen großen Bereich an verschiedenen Qualitätsmerkmalen abdeckt, welche in ihrer Gesamtheit die Aufgabe haben, bei dem User ein angenehmes Gefühl bei der Benutzung des jeweiligen Produktes hervorzurufen. Die Frage, ob dieses Produkt seine Aufgabe erfüllt, ist demnach von gleicher Bedeutung wie die Freude, die ein Nutzer bei der Bedienung empfindet, und auch die Auswirkungen von Design-Entscheidungen spielen eine Rolle. Diese Liste lässt sich nach Donald Norman auf alle Aspekte erweitern, welche in Zusammenhang mit dem jeweiligen Produkt stehen.
Durch die Abschlussarbeiten von Meike Johannsen [Joh11] und Björn Kahlert [Kah11] wurden bereits sowohl die Dokumentation als auch die Nutzerfreundlichkeit von Saros enorm verbessert. Ein interessanter Punkt ist dabei allerdings noch offen: Der Nutzen von Saros an sich. Das Ziel dieses Projektes ist es, den Usern das Erstellen von Code in verteilter Paarprogrammierung zu ermöglichen. Sowohl das Einrichten der Session zur Zusammenarbeit als auch der Vorgang des Programmierens selbst ist dabei aktionsorientiert, da jeweils eine Interaktion von Nutzerseite vorausgesetzt wird.

Aus diesem Grund liegt der Fokus der vorliegenden Arbeit auf Aspekten der User Experience im Kontext eben dieser Interaktionen.

3 Verschiedene Zustände der User in Saros

Ein Saros-Nutzer nimmt bei dem Umgang mit dem Plugin verschiedene Zustände ein, welche Auswirkungen auf die Interaktionsmöglichkeiten haben, die ihm zur Verfügung stehen. Dieses Kapitel beschäftigt sich mit der Frage, welche Zustände existieren, wie sie vom User angenommen werden können und welche Aktionen für sie vom Saros-Konzept her vorgesehen sind. Hierzu wird zunächst ein Überblick darüber gegeben, welche verschiedenen Zustände es im Rahmen der Verwendung dieses Projektes gibt. Darauf folgend werden alle grundlegenden Interaktionsmöglichkeiten erläutert, welche in dem Plugin vorhanden sind. In einem dritten Schritt werden diese Informationen miteinander verknüpft, um einen Gesamteindruck darüber zu vermitteln, welcher User in welchem Zustand zu welcher Interaktion befähigt sein soll. Dies dient einem Abgleich der vom Saros-Konzept vorgesehenen Aktionsmöglichkeiten mit dem tatsächlichen Stand.

3.1 Mögliche Zustände der Nutzer

Der Zustand eines Saros-Users setzt sich aus Kombinationen folgender Komponenten zusammen:

- Um eine Saros-Session starten zu können bzw. einer bereits bestehenden beizutreten, muss der Nutzer mit einem XMPP-Account angemeldet sein. Ob dies der Fall ist, wird in den Diagrammen in Abschnitt 3 dieses Kapitels mit dem Stichwort **„connected“** bezeichnet. Sämtliche Interaktionen (mit Ausnahme des Verbindungsaufbaus zum XMPP-Server) sollen erst möglich sein, wenn das Kriterium „connected“ erfüllt ist.

- **„In session“** beschreibt, ob sich der User bereits in einer verteilten Kollaboration befindet oder nicht.

- In Saros gibt es eine Unterscheidung zwischen demjenigen, der eine Sitzung erstellt hat, und demjenigen, welcher zu dieser eingeladen wurde. Dies betrifft insbesonders die Frage, wer in der Lage ist, eine bestehende Session zu schließen. Hier gibt es die Unterscheidung **„host | client | none“**, wobei „none“ bedeutet, dass sich der User nicht in einer Session befindet und somit keinen der beiden anderen Zustände innehaben kann.

- Teilnehmer (clients) einer gemeinsamen Sitzung haben initial die Möglichkeit, selbst Codeänderungen vorzunehmen. Der Eröffner (host) der Session kann diese Berechtigung auf einen lesenden Zugriff beschränken. Dieses Attribut wird im folgenden als **„has write privileges“** bezeichnet.

- Der Follow Mode ist ein Feature von Saros, welches es dem User ermöglicht, die Handlungen eines von ihm bestimmten anderen Nutzers nachzuvollziehen. Hierbei handelt es sich um die wichtigste Funktionalität, was die 'workspace awareness' (das Wissen um die Interaktionen anderer User in der gemeinsam genutzten Umgebung) betrifft - diese ist nach Gutwin und Greenberg [GG04] ausschlaggebend für eine effektive verteilte Zusammenarbeit. Es werden alle Bewegungen dieses Users auf die lokale Entwicklungsumgebung desjenigen, der sich im Follow Mode befindet, übertragen. Ob ein User diesen Modus verwendet, wird in diesem Kapitel durch **„follow mode active“** gekennzeichnet.

3.2 Grundlegende Interaktionsmöglichkeiten in Saros

Es existiert eine Reihe an Aktionen, welche dem User - abhängig von seinem Zustand, welcher sich durch die in Abschnitt 1 dieses Kapitels benannten Komponenten ergibt - nach dem Konzept von Saros zur Verfügung stehen sollen. Diese werden im Folgenden vorgestellt.

- **„create / join session“**: Ein User, welcher „connected“ ist, sich aber noch nicht in einer Session befindet, soll die Möglichkeit haben, eine solche zu erstellen oder einer bestehenden beizutreten.

- **„share project(s)“**: Besteht der Wunsch, eine Session zu erstellen, so muss sichergestellt sein, dass der Nutzer Projekte zur gemeinsamen Bearbeitung an die Teilnehmer vor Beginn der Zusammenarbeit verteilen kann.

- **„leave / stop active session“**: Möchte ein Client aus der Kollaboration austreten, so soll ihm dies möglich sein, ohne dass die Session an sich davon betroffen ist (leave). Handelt es sich bei demjenigen, der die Zusammenarbeit beenden möchte, um den Host, so soll die gesamte Sitzung geschlossen werden (stop) - dies hat die Folge, dass alle Teilnehmer ebenfalls aus der Session entfernt werden.

- **„enter / leave follow mode“**: In einer bestehenden Session soll es möglich sein, einen anderen User durch Einsatz des Follow Modes zu beobachten. Auch ein Verlassen dieses Zustandes muss gewährleistet sein. Ebenso ist es möglich, lediglich zu der Position eines anderen Teilnehmers zu springen.

- **„manage write privileges“**: Der Host einer Session kann bestimmen, welche Teilnehmer Codeänderungen vornehmen können (siehe Attribut „has write privileges“ aus Abschnitt 1)

- **„edit buddy list“**: Jeder User soll, sobald er sich mit seinem XMPP-Account angemeldet hat, in der Lage sein, seine Kontaktliste zu be-

arbeiten. Dies impliziert das Hinzufügen, Entfernen und Umbenennen von Buddies.

- **„connect / disconnect“**: Ein nicht mit dem XMPP-Server verbundener User soll sich anmelden können. Auch soll es für angemeldete Nutzer jederzeit möglich sein, sich auszuloggen.
- **„add buddy / project to a running session“**: In eine bereits laufende Kollaboration sollen neue Teilnehmer und Projekte einbringbar sein. Im Gegensatz zu „share project(s)“ soll dies keine neue Session erzeugen.
- **„use Saros interaction features“**: Befindet sich der User im Zustand „connected“, so soll er Dateien an Mitglieder seiner Kontaktliste übertragen, eine VoIP-Verbindung starten, seine Bildschirmansicht teilen, das Whiteboard einsetzen und den Chat benutzen können.
- **„start Skype call“**: Zur besseren Kommunikation soll es dem Nutzer möglich sein, in angemeldetem Zustand eine Skype-Verbindung zu einem seiner Buddies herzustellen.

3.3 Überblick über Userzustände und die damit verbundenen Interaktionsmöglichkeiten

Im letzten Abschnitt dieses Kapitels soll nun dargestellt werden, welche Interaktionen Usern in betimmten Zuständen nach dem Konzept von Saros zugänglich sein sollen, um dann überprüfen zu können, inwieweit dies mit dem aktuellen Stand des Projekts übereinstimmt. Zu diesem Zweck wird das in Abbildung 1 gezeigte Schema verwendet.

Hierbei behandelt die „Beschreibung des Zustandes“ die in Abschnitt 1 dargelegten Komponenten des Benutzer-Status. Grün markiert sind jene Teil-Zustände, welche auf den User in diesem Moment zutreffen und damit den Gesamtzustand bilden. Zur Identifizierung erhält jeder Gesamtzustand eine „Bezeichnung“. Die „Interaktionsmöglichkeiten“ nehmen Bezug auf Abschnitt 2 - grün hervorgehoben sind hierbei all die Aktionen, die ein User in dem betreffenden Zustand ausführen können soll.
Um das Diagramm nicht unnötig zu verkomplizieren, werden bis auf jeweils einen exemplarischen Eintrag jene Übergänge nicht dargestellt, welche :

- den User nach „disconnect“ wieder in den Urzustand versetzen (Bezeichnung „not connected user“)
- den User infolge der Aktion „leave / stop active session“ in den Zustand „connected user without session“ versetzen

3.3 Überblick über Userzustände und die damit verbundenen Interaktionsmöglichkeiten

Abbildung 1: Schema zur Darstellung des User-Zustandes und der daraus resultierenden Interaktionsmöglichkeiten

Weiterhin wird das Diagramm auf zwei Seiten verteilt, um eine gute Lesbarkeit zu gewährleisten. Der Urzustand, in welchem sich der User nach dem ersten Starten von Saros befindet, ist mit einem roten Stern markiert. Aktionen, welche nicht mit Übergängen zu anderen Zuständen versehen sind, ändern den momentanen Zustand des Nutzers nicht.

3.3 Überblick über Userzustände und die damit verbundenen Interaktionsmöglichkeiten

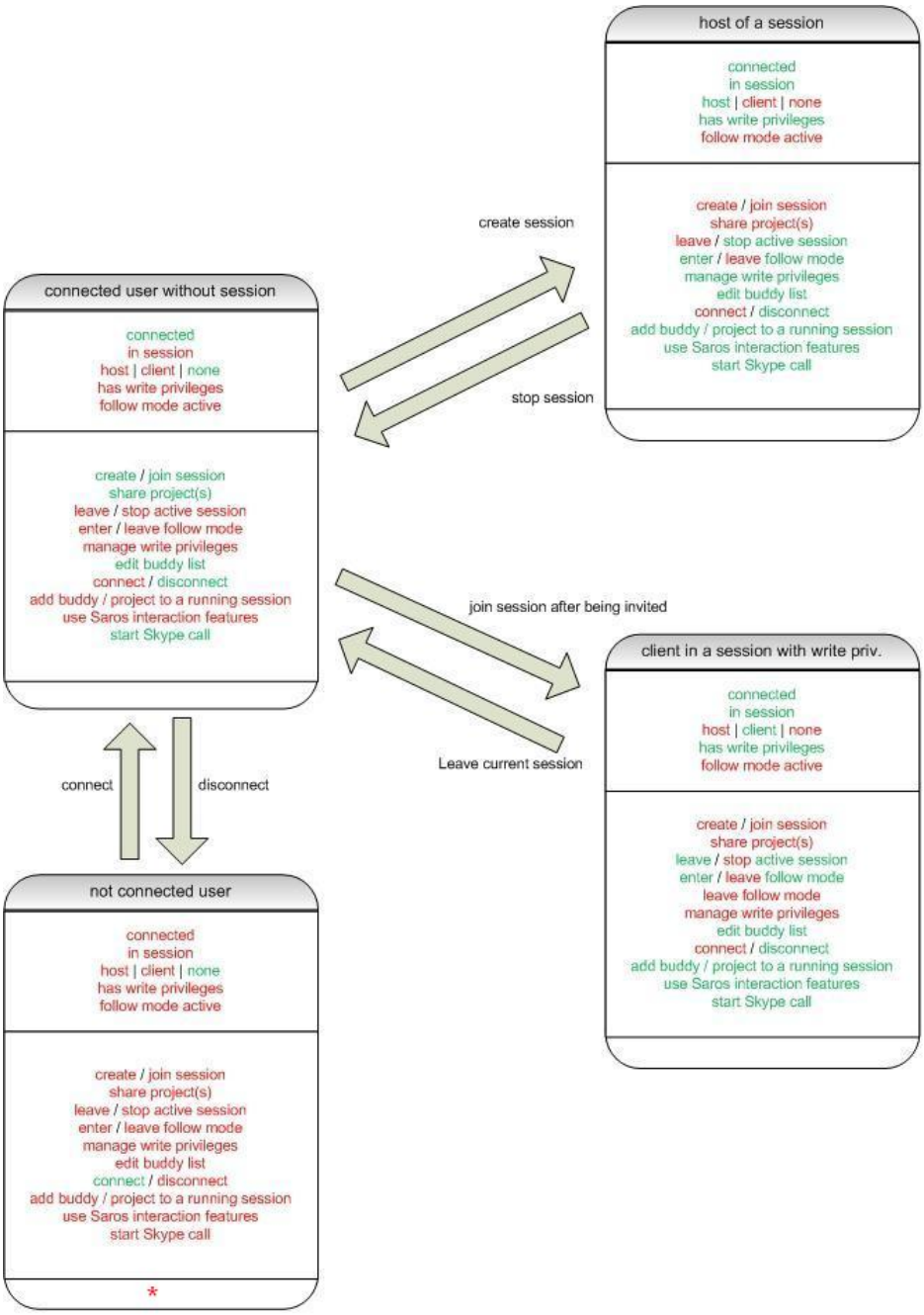

Abbildung 2: Zustandsübergänge, welche das Verbinden mit dem XMPP-Account und das Beitreten zu / das Erstellen einer Session darstellen

3.3 Überblick über Userzustände und die damit verbundenen Interaktionsmöglichkeiten

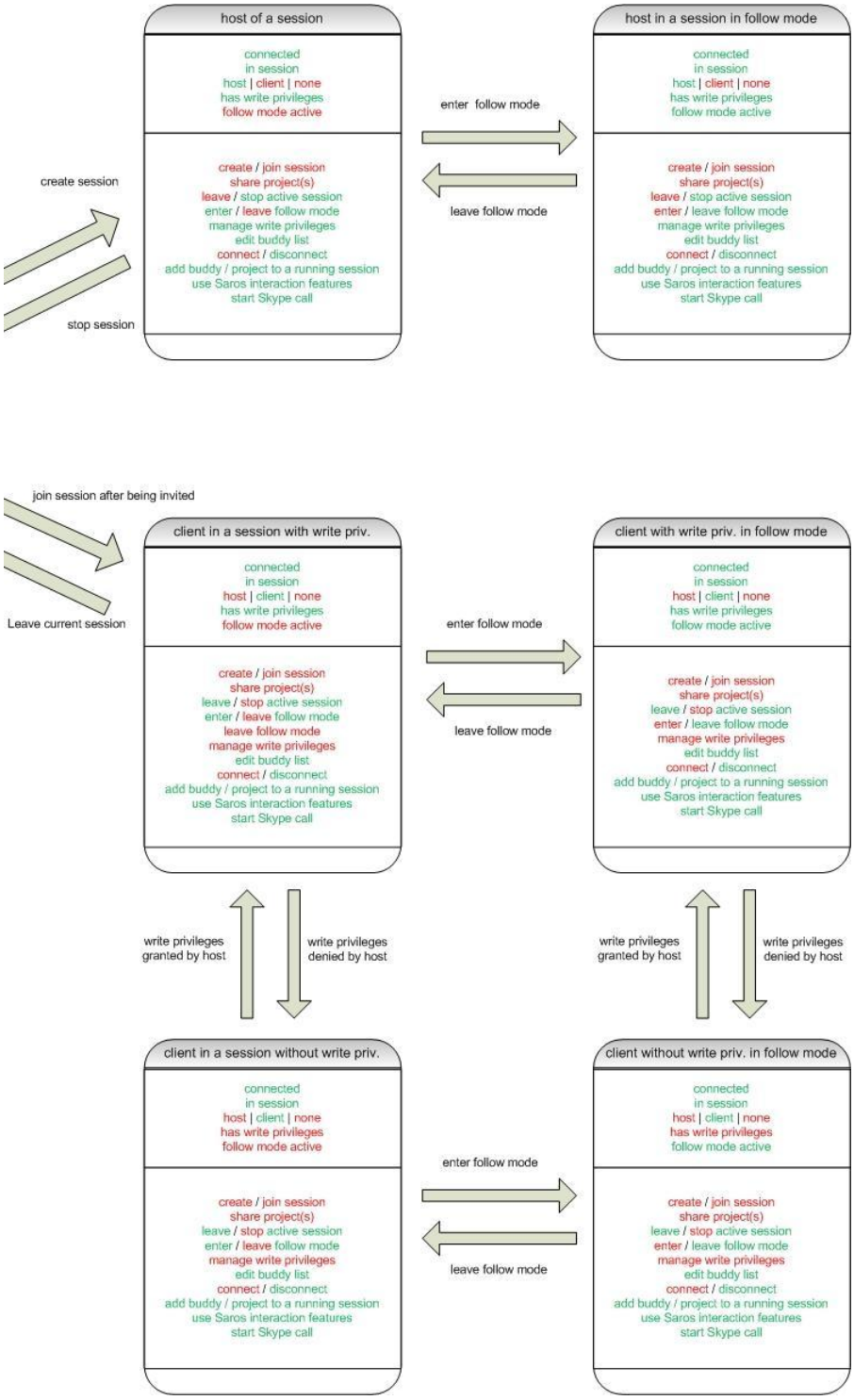

Abbildung 3: Fortsetzung von Abbildung 2: mögliche Zustandsübergänge für einen User innerhalb einer Session

4 Arbeitsansatz: Analyse von Saros und Vergleiche mit anderen Projekten

Wie im vorangegangenen Kapitel erwähnt, diente die Aufschlüsselung der verschiedenen Zustände, die ein User in Saros annehmen kann, der Überprüfung, ob alle Interaktionsmöglichkeiten dieser einzelnen Zustände den Erwartungen entsprechen. Nach der Untersuchung verschiedener Szenarien (beispielsweise Hinzufügen und Entfernen von Teilnehmern, mehrfaches Ändern der Schreibberechtigungen in Kombination mit dem Einsatz des Follow Modes etc.), welche jede der in Kapitel 3 dargestellten User-Aktionen und Zustände abdeckten, stellte sich heraus, dass es keinerlei Verstöße gab. Daher fasste ich den Entschluss, meine Analyse auf die Kontextmenüs im Allgemeinen - über einen Abgleich mit den verschiedenen Zuständen hinaus - zu erweitern. So war es mir möglich, im Feld der Interaktionsmöglichkeiten zu bleiben, welches stark mit der User Experience verknüpft ist. Überdies bot sich eine weitergehende Untersuchung der Kontextmenüs an, da diese sowohl vor als auch während einer Saros-Session zum Einsatz kommen und somit von hoher Wichtigkeit sind.

Der Fokus lag hierbei stets auf der in Kapitel 2 gegebenen Definition von User Experience - mein Ziel war, Einträge und dynamische Änderungen dieser innerhalb der Kontextmenüs daraufhin qualitativ zu untersuchen, ob sie Erwartungen enttäuschen, beim User Verwirrung hervorrufen oder auf andere Weise die Zufriedenheit des Nutzers gefährden. In Kapitel 5 werden gefundene Probleme dargestellt und diskutiert.

Das zweite analytische Verfahren, welches im Laufe der Recherche zu dieser Arbeit Anwendung fand, war eine Bestandsaufnahme anderer Projekte, welche ebenso wie Saros die verteilte virtuelle Zusammenarbeit ermöglichen. Infolgedessen konnte eine Defizitanalyse in Hinsicht auf Mechanismen, welche eine positive User Experience begünstigen, aufgestellt werden. Um ein möglichst breites Spektrum abzudecken, wurden nicht nur Projekte untersucht, welche sich auf die verteilte Paarprogrammierung spezialisieren, sondern auch solche, die einen gänzlich anderen Verwendungszweck haben. Die ausgewählten Projekte werden in Kapitel 6 vorgestellt. Entdeckte Features und Konzepte werden in den darauf folgenden Kapiteln evaluiert und eine mögliche Integration in Saros diskutiert.

5 Erkannte Probleme in den Kontextmenüs

Dieses Kapitel beschreibt die Probleme, welche sich bei einer Analyse der Kontextmenüs in Saros ergeben haben. Diese werden hier lediglich vorgestellt und diskutiert - Bildmaterial und Erläuterungen zu den vorgenommenen Änderungen finden sich in Kapitel 10, welches eine Darstellung der von mir vorgenommenen Codeanpassungen bietet. Dort werden ebenso codebedingte Zusammenhänge erklärt, welche bei der Implementierung beachtet werden mussten.

5.1 Buddies einladen, die offline sind

Hierbei handelt es sich um einen Aspekt, welcher aus User Experience-Sicht untragbar ist. Zum einen stiftet es enorme Verwirrung beim User, dass er in der Lage sein soll, jemanden zu einer Zusammenarbeit einzuladen, der gar nicht da ist (vgl. Abbildung 4). Zum anderen kommt hinzu, dass kurz nach dem Einladungsversuch eine Fehlermeldung angezeigt wird, welche den Nutzer darauf hinweist, dass er versucht hat, jemanden einzuladen, der zu diesem Zeitpunkt nicht mit seinem XMPP-Account eingeloggt war. Weiterhin kann man in Abbildung 5 sehen, dass bereits eine Session zur Kollaboration angelegt wurde - nach dem Verlassen des Fehlermeldungsdialogs wird diese allerdings sofort wieder geschlossen. Alles in allem sorgt dieses Verhalten für Verwirrung und eventuell sogar Frustration beim User, da eine ihm gebotene Interaktionsmöglichkeit den beschriebenen Verlauf nimmt und ihn an den Ausgangspunkt, den Einladungsprozess, zurückversetzt.

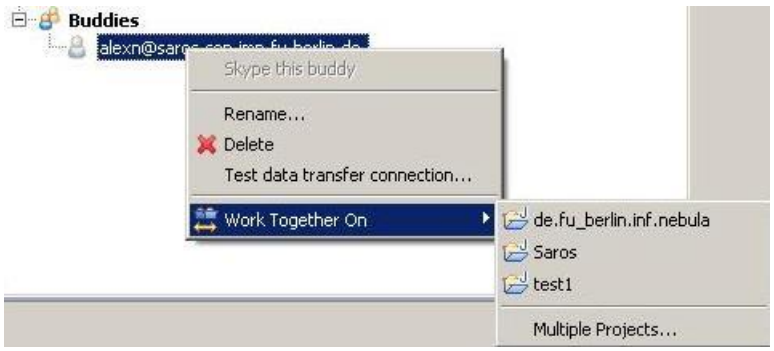

Abbildung 4: Ein Buddy, der nicht connected ist, kann in eine Session eingeladen werden

In einem Gespräch mit meiner Betreuerin, welches dieses Phänomen zum Inhalt hatte, stellte sich allerdings heraus, dass dieses Verhalten vorerst geduldet bleiben soll. Dies hängt damit zusammen, dass es gelegentlich noch

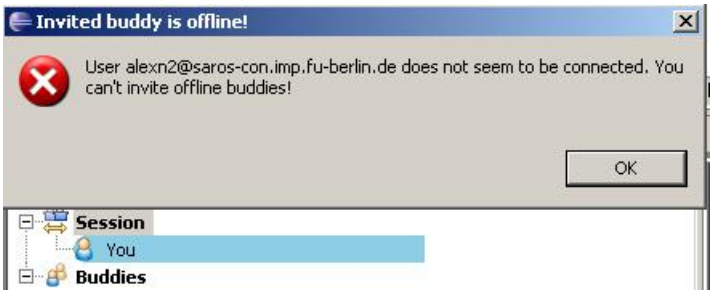

Abbildung 5: Feedback an den User, der versucht hat, einen nicht verbundenen Buddy einzuladen

Fälle gibt, in denen die Präsenzinformation über den Onlinestatus eines Buddies nicht richtig übertragen wird. Das heißt, es kommt vor, dass ein Eintrag in der Buddyliste als „not connected“ angezeigt wird, obwohl dies nicht dem wahren Zustand entspricht. Es ist in diesem Fall auch möglich, eine Session zu initiieren, da der potentielle Teilnehmer nur zum Schein nicht verfügbar ist. Es erfolgte eine Einigung zwischen meiner Betreuerin und mir, dass erst sichergestellt werden muss, dass die Präsenzinformationen zuverlässig und fehlerfrei übertragen werden, bevor dieses Problem unter dem Gesichtspunkt der User Experience angegangen werden kann.

5.2 Irreführende Nicht-Selektierbarkeit

Die Menüpunkte „Share with -> Multiple Buddies“ und „Work Together On -> Multiple Projects“, welche sich unter den Auflistungen aller verfügbaren Buddies bzw. Projekte befinden, werden grau hinterlegt und somit unselektierbar, wenn sich der Nutzer nicht im Status „connected“ befindet. Dies ist von der Ablauflogik her korrekt, jedoch ist irreführend, dass über den grau hinterlegten Flächen noch Feedback durch Einblenden von „not connected“ an der Stelle gegeben wird, welche im Normalfall durch die Einzelauflistungen eingenommen wird. Dies suggeriert, dass die Nicht-Selektierbarkeit des multiplen Sharings einen anderen Grund haben könnte, als die Tatsache, dass man nicht mit seinem XMPP-Account eingeloggt ist - dies jedoch ist falsch.

Da beide Menüpunkte aber, was ihre Aufrufbarkeit angeht, wie auch die Einzelauflistungen darüber, allein vom Status „connected“ abhängen, habe ich mich hier entschlossen, das Feedback für die beiden Punkte zu jeweils einem Punkt pro Menü zusammenzufassen.

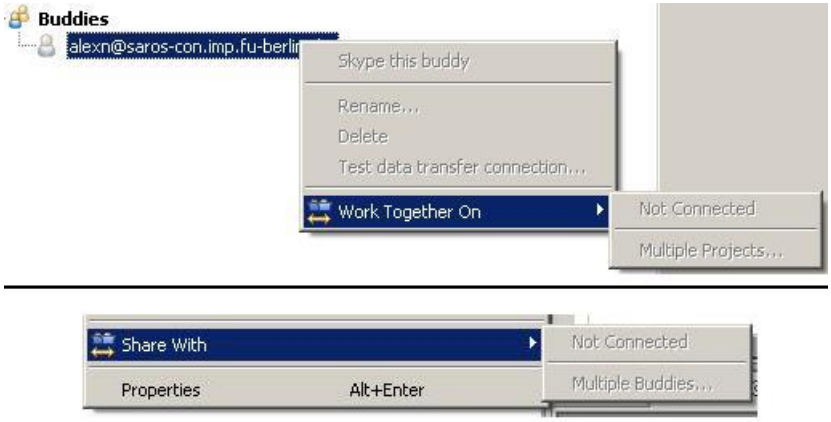

Abbildung 6: Irreführendes Feedback in den Kontextmenüs für multiples Sharing (oben) und multiple Buddy-Einladung (unten)

5.3 Multiples Sharing und Einladen trotz nicht vorhandener Elemente

Die im oberen Abschnitt bereits behandelten Menüs weisen noch eine weitere Schwachstelle auf: Hat der Nutzer einen leeren Workspace, so kann er dennoch die Option des multiplen Sharings auswählen (vgl. Abbildung 7). Selbiges gilt für eine leere Buddylist im Kontext einer Mehrfacheinladung zu einer Session. Ich habe mich dafür entschieden, eine Überprüfung einzubauen, welche die Optionen erst freigibt, wenn mindestens zwei Projekte bzw. Buddies vorhanden sind, da vom Sprachgebrauch her erst ab zwei Elementen von „multipel“ gesprochen werden kann.

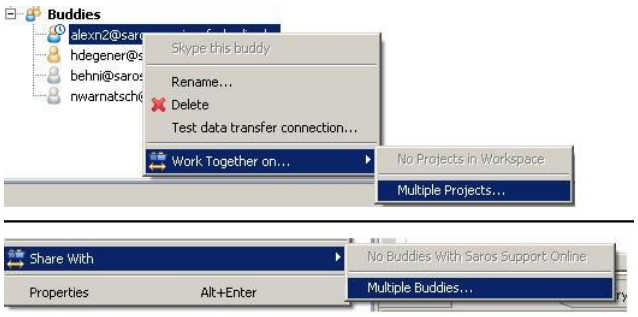

Abbildung 7: Obwohl kein Projekt im Workspace vorhanden ist, wird Multiples Sharing angeboten (oben). Analog dazu der Einladungsprozess (unten)

5.4 Keine Skype-Calls innerhalb einer Session

Saros bietet dem User die Möglichkeit, mittels Skype eine Sprachverbindung zu einem Buddy herzustellen. Diese Auswahl besteht allerdings nur für alle jene Kontakte, mit denen der Benutzer zu diesem Zeitpunkt nicht in einer Session zusammenarbeitet - dabei hat Riad Djemili in seiner Arbeit [Dje06] in einem Testlauf mit zwei Kommilitonen festgestellt, dass gerade die verbale Kommunikation in einer Kollaboration über Saros von großer Bedeutung ist. Daher muss auch innerhalb einer Session die Möglichkeit bestehen, diese Funktionalität zu nutzen.

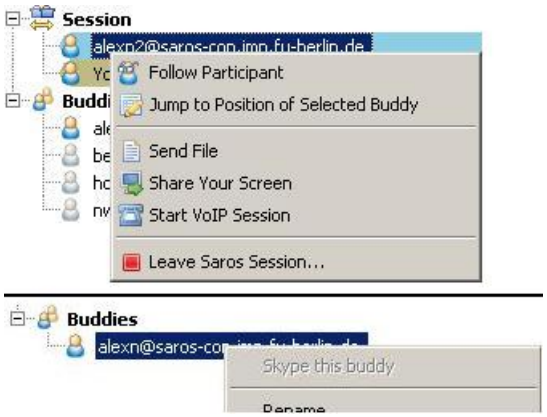

Abbildung 8: Das Herstellen einer Skype-Verbindung mit einem Session-Teilnehmer ist nicht möglich.

6 Vorstellung der Vergleichsprojekte

Dieses Kapitel vermittelt einen Eindruck der Projekte, die zu qualitativen Vergleichen mit Saros in Hinsicht auf potentielle Verbesserungsmöglichkeiten der User Experience herangezogen wurden. Eine Auflistung der konkreten Ideen, welche sich aus den Vergleichsprojekten ergeben haben, findet sich in Kapitel 8.
Dieser Abschnitt beinhaltet ausschließlich Projekte, die ich selbst ausprobieren und analysieren konnte, da so sichergestellt ist, dass ein tatsächlicher Eindruck über vorhandene Mechanismen der User Experience entstehen konnte.
Weiterhin gibt es Projekte, die ich mir zwar angesehen habe, die allerdings keinerlei Mehrwert für diese Arbeit bieten. Dies liegt einerseits daran, dass einige betrachtete Projekte anderen so ähnlich sind, dass sie keinen Beitrag zu der in Kapitel 8 aufgeführten Feature-und-Konzept-Liste leisten können - so ist beispielsweise der kollaborative Notizzettel „notepub“ (`http://notepub.com/`) zwar um einige Textbearbeitungsfunktionen reicher als das in Abschnitt 6.4 vorgestellte „MeetingWords“, hat allerdings aus User Experience-Sicht keinerlei Mechanismen, die nicht durch die Analyse letzteren Projektes bereits abgedeckt wären.
Andererseits kam es vor, dass einige Projekte auch nach viel Aufwand nicht lauffähig waren. In diese Gruppe fällt unter anderem „Gobby“ (`http://gobby.0x539.de/`), bei welchem es mir auch nach Installationsversuchen auf verschiedenen Betriebssystemen und der Verwendung eines Linux-Servers als Verbindungsendpunkt nicht gelang, ein kollaboratives Texteditieren nachzustellen.
Projekte, welche in eine von diesen beiden Kategorien fallen, sind nicht aufgeführt, da sie keinen Mehrwert liefern.

6.1 PeP - Pair Eclipse Programming

PeP ist ebenso wie Saros ein Plugin für Eclipse, dessen Weiterentwicklung im Jahre 2007 eingestellt wurde. In der Menge der untersuchten Projekte ist es das einzige, welches direkt auf die Progammierung zugeschnitten ist. Dies zeigt sich vor allem dadurch, dass nicht einzelne Dateien betrachtet und bearbeitet werden, sondern auf Wunsch auch ganze Packages oder Projekte. Im Gegensatz zu Saros kann es nicht per Eclipse direkt installiert werden, sondern ist ausschließlich per Dropin verfügbar, welches man auf der Webseite http://pep-pp.sourceforge.net/ findet.
Sobald dieses in den „dropins"-Ordner von Eclipse eingefügt wurde, wird die obere Eclipse-Menüleiste um einen Bereich für das Plugin ergänzt, welcher drei von fünf der Funktionen des Plugins enthält (vgl. Abbildung 9). PeP hält sich strikt an das Konzept der Paarprogrammierung, daher ist die Gesamtzahl der Mitwirkenden auf zwei begrenzt und es gibt zu jeder Zeit genau einen Driver und einen Observer. Der Observer hat dabei keine Schreibrechte im Quellcode, sondern verfolgt die Aktionen des Drivers. Dies ist dadurch möglich, dass jede Handlung des Drivers in der Eclipse-Instanz des Observers simuliert wird und der Fokus stets auf der aktuellen Position des Drivers liegt. Es handelt sich hierbei um das gleiche Verfahren, nach welchem der „Follow Mode" in Saros arbeitet, mit dem Unterschied, dass dieser in Saros manuell durch den User eingeschaltet werden kann, während sich der Observer in PeP permanent im Follow Mode befindet.

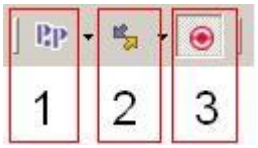

Abbildung 9: PeP-Menü in der Eclipse Menüleiste

- Mit „1" markiert ist hierbei der Button, mit welchem sich die User zum gemeinsamen Arbeiten zusammenfinden. Während Saros über das XMPP-Protokoll eine Kontaktliste mit allen potentiellen Mitwirkenden unterstützt, wird in PeP eine direkte Verbindung der Nutzer per IP-Protokoll verwendet. Man hat die Wahl, ob man sich als „Server" oder als „Client" verbinden will - diese Begriffe sind Synonyme zu Driver und Observer. Sobald beide User miteinander verbunden sind, kann ein Projekt zur gemeinsamen Bearbeitung per Eclipse-Menü vom Server an den Client übertragen werden, um die Zusammenarbeit zu starten (vgl. Abbildung 10). Dies ist die vierte Funktionalität, die PeP dem User bietet.
- Bevor die Driver-Observer-Verbindung aufgebaut ist, ist der mit „2"

markierte Button nicht selektierbar. Sobald die Zusammenarbeit begonnen hat, dient er zum Finden von Synchronitäts-Problemen. Neben der Möglichkeit, die lokalen Exemplare der jeweiligen Dateien auf Unterschiede zu prüfen, kann durch Auswahl der Option „Force Resynchronize“ eine Neuübertragung der Dateien initiiert werden. Dies ist die gleiche Funktion, welche das Package `concurrent.watchdog` in Saros hat, mit dem Unterschied, dass letzteres Inkonsistenzen regelmäßig überprüft und daher automatisch feststellt.

- Auch der durch „3“ markierte Button lässt sich erst benutzen, sobald eine Verbindung herstellt ist. Durch ihn können - wie in der klassischen Paarprogrammierung - die Rollen von Driver und Observer jederzeit getauscht werden.

Abbildung 10: Übertragungs-Funktion in PeP

Die letzte Funktionalität, die Pep mit sich bringt, ist ein Chat, welcher manuell als View in Eclipse aktiviert werden muss. Dieser Chat unterscheidet zwischen Nachrichten vom „local user“ und dem „remote user“ und dient gleichzeitig als Protokollierung darüber, wann die Rolle des Drivers getauscht wurde (vgl. Abbildung 11).

Abbildung 11: Screenshot des Chats in PeP

6.2 google docs

Bei google docs handelt es sich um eine Onlineversion bekannter Office-Services wie Textbearbeitung und Tabellenerstellung, nur dass es möglich ist, diese Dokumente gleichzeitig mit mehreren Usern zu bearbeiten. User können auf der Webseite http://docs.google.com/ Dokumente verschiedener Arten erstellen (vgl. Abbildung 12) und dann durch eine Einladung per Email-Adresse anderen Benutzern die Rechte zur Einsicht und Bearbeitung geben. Die Dokumente werden in der Cloud gespeichert, daher ist die einzige technische Voraussetzung für einen Zugriff eine bestehende Internetverbindung.

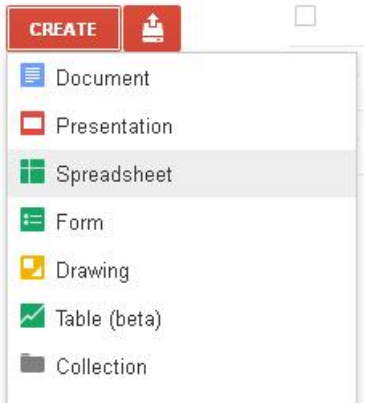

Abbildung 12: Erstellung eines Dokumentes in google docs

Als Kommunikationsweg besteht neben der Nutzung der Dokumente selbst die Möglichkeit, Dateien hochzuladen, welche für jeden Beteiligten erreichbar sind. Um über die letzten Veränderungen informiert zu werden, stellt google docs eine Übersicht zur Verfügung, welche den Urheber und das Datum dieser Veränderung darstellt. Aktuelle Bearbeitungen werden in den Dokumenten selbst durch Angabe des jeweiligen Users dargestellt und die davon betroffenen Bereiche sind für weitere Veränderungen solange gesperrt, bis die aktuelle Bearbeitung abgeschlossen ist (vgl. Abbildung 13).

Abbildung 13: Während ein Bereich bearbeitet wird, sperrt google docs diesen für alle anderen User

6.3 MoonEdit

Dieses von Tom Dobrowolski im Jahre 2005 entwickelte Programm unterscheidet sich von allen anderen in dieser Arbeit untersuchten Projekten darin, dass es sich hierbei nicht um ein Plugin oder einen Webservice handelt, welches Texteditierung mit mehreren Beteiligten ermöglicht, sondern um eine Software im klassischen Sinne. Diese ist plattformunabhängig und kann unter der Webseite http://moonedit.com/ heruntergeladen werden.
Inhaltlich umfasst das Programm gängige Funktionitäten einer Textbearbeitung, bietet zusätzlich allerdings auch die Möglichkeit, die Datei mit anderen Usern zusammen zu bearbeiten. Ähnlich wie bei PeP geschieht dies über eine direkte Verbindung zu einem User, der als Host bezeichnet wird. Dieser muss, um die Zusammenarbeit zu starten, seine lokale Datei über einen frei wählbaren Port zur Verfügung stellen. Teilnehmer können sich dann über diesen Port und die IP-Adresse des Hosts zu diesem verbinden. Gegebenenfalls wird auch noch die Eingabe eines Passwortes verlangt, welches vom Host festgelegt werden kann (vgl. Abbildung 14).

Abbildung 14: Dialog zur Teilnahme an einer gemeinsamen Bearbeitung einer Datei in MoonEdit

Auffällig ist bei diesem Projekt der hohe Grad an Individualisierungsmöglichkeiten: Neben der Wahl einer Farbe, welche zur Darstellung der jeweiligen geschriebenen Textpassagen verwendet wird, ist der User auch in der Lage, einen Namen anzugeben, unter welchem er dann für alle Teilnehmer sichtbar in einer Liste der momentan verbundenen User aufgeführt wird. Ebenso ist es möglich, die verwendeten Farben zur besseren Lesbarkeit zu invertieren, durch eine Audio-Benachrichtigung über die Schreibvorgänge von anderen Usern informiert zu werden oder die Länge des Wirkungsbereichs der <TAB>-Taste festzulegen (vgl. Abbildung 15). Hilfreich ist weiterhin die Anzeige der Maus-Cursor anderer Beteiligter durch einen vertikalen Strich an der jeweiligen Position, was eine Fokussierung auf den entsprechenden Bereich unterstützt (vgl. Abbildung 16).

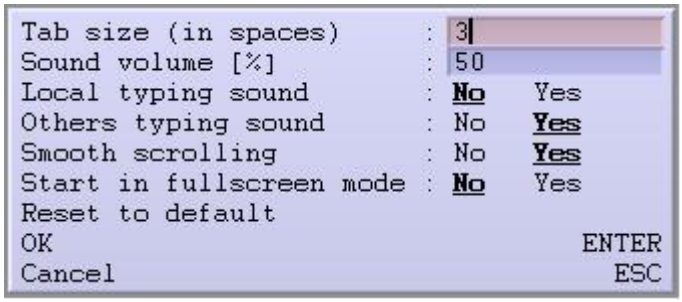

Abbildung 15: Ein Teil der Individualisierungs-Möglichkeiten in MoonEdit

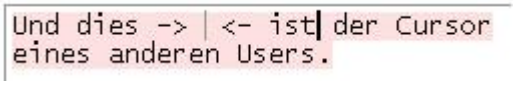

Abbildung 16: Maus-Cursor der anderen Teilnehmer werden in grau dargestellt

Weiterhin ist bemerkenswert, dass sich die Existenz der Datei auf die Benutzung von MoonEdit beschränkt. Will man sie über die gemeinsame Bearbeitung hinaus persistieren, so muss sie per Menü exportiert werden. Auch gibt es die Möglichkeit, die Sitzung aufzuzeichnen und die einzelnen Schritte dann in MoonEdit selbst wiederzugeben und sie somit zu protokollieren.

6.4 MeetingWords

MeetingWords, erreichbar unter http://meetingwords.com/, nennt sich ein Webservice zum gemeinsamen Editieren eines Textdokumentes. Dieser verfügt über grundlegende Gestaltungsfunktionen wie beispielsweise den Einsatz kursiver Schrift und bietet den Import einer lokalen Datei an. Auch ist hier die Möglichkeit gegeben, einen eigenen, für andere sichtbaren, Namen anzugeben und eine Farbe zu wählen.
Herausstellungsmerkmal dieses Projektes ist der Einladungsprozess: Jedes in MeetingWords erstellte Dokument erhält eine einmalige URL, welche dann an alle Teilnehmer verteilt werden kann. Nach Öffnen des Links gelangt man direkt in den Bearbeitungsbereich und kann zugleich mit der Zusammenarbeit beginnen (vgl. Abbildung 17).

Abbildung 17: Die Einladung in MeetingWords wird durch Verteilung eines einmaligen Links realisiert

Überdies ist in MeetingWords eine User-bezogene „undo“-Funktion integriert, welche Änderungen anderer Benutzer unangetastet lässt. Weiterhin ist es möglich, händisch Revisionen anzulegen, welche jederzeit aufgerufen werden können, um einen bestimmten Zustand wiederherzustellen. Auch ist ein sogenannter „Time Slider“ vorhanden, welcher ähnlich der Sitzungsaufzeichnung in MoonEdit einen Schnellablauf aller Änderungen am Dokument darstellt. Das Dokument selbst wird in der Cloud gespeichert und ist auf Wunsch in ein lokales Exemplar exportierbar.

Abbildung 18: Das Optionsmenü von MeetingWords - geöffnet ist die Revisionsübersicht

6.5 Adobe Connect

Adobe Connect (http://www.adobe.com/de/products/connect/) ist eine Software, deren Fokus nicht auf der gemeinsamen Texteditierung liegt. Vielmehr ist das Ziel, Präsentationen und Workshops über das Internet veranstalten zu können. Da Adobe Connect komplett auf Adobe Flash basiert, ist es ohne Hindernisse benutzbar und zugleich sehr performant. Es werden „Meeting Rooms“ angelegt, die jeweils über eine einmalige URL verfügen und den Rahmen für die Zusammenarbeit bilden. In einem Meeting Room werden alle nutzbaren Funktionalitäten durch sogenannte „Pods“ dargestellt - so gibt es beispielsweise je einen Pod für Audioübertragung, für eine Teilnehmerliste, für ein Whiteboard etc. .
Innerhalb einer Präsentation gibt es drei Rollen:

- Der **Host** kann den Meeting Room nach seinem Belieben gestalten und einrichten. Dazu zählt die Auswahl der zu verwendenden Pods genauso wie die Verteilung der Berechtigungen an die übrigen User. Er kann, wie auch die Präsentatoren, Dateien auf den zentralen Server von Adobe Connect hochladen und in dem „Share Pod“ (vgl. Abbildung 19) für alle Teilnehmer anzeigen lassen.

- **Präsentatoren** bestimmen, welche (zuvor hochgeladenen) Inhalte den Teilnehmern dargestellt werden. Ebenso können sie Interaktionen mit den übrigen Usern ausführen, indem Sie beispielsweise eine Umfrage starten.

- Der **Teilnehmer** ist die dritte Rolle, die ein User in Adobe Connect einnehmen kann. Teilnehmer sind die Zuschauer der Präsentation und haben daher lediglich die Möglichkeit, Notizen vorzunehmen und ihren Anzeigestatus zu ändern.

Dieser Status ist ein wichtiger Feedback-Mechanismus in Adobe Connect (vgl. Abbildung 20 [oP11]). Die Teilnehmer sind in der Lage, ihre Meinung über die Präsentationsweise zu äußern, indem sie Lautstärke und Geschwindigkeit des Präsentierenden bewerten, können durch das Setzen eines Status jedoch auch aktiv an dem Meeting mitwirken (beispielsweise durch Anlehnung einer These).

Abbildung 19: Der „share pod“ von Adobe Connect, über welchen die Präsentationen dargestellt werden

Abbildung 20: User-Statusänderungen geben den Präsentierenden Feedback

Auch hinter den „Breakout Rooms“ verbirgt sich ein interessantes Konzept. Auf Wunsch des Präsentators werden alle Teilnehmer gleichmäßig in verschiedene Räume aufgeteilt, mit dem Ziel, Gruppenarbeit zu leisten. Innerhalb dieser Breakout Rooms haben alle Beteiligten erweiterte Berechtigungen, sodass sie für den Zeitraum dieser Zusammenarbeit ebenfalls in der Lage sind, Dateien untereinander auszutauschen und zu präsentieren (vgl. Abbildung 21 [Bat11]).

Abbildung 21: „Breakout Rooms“ dienen der Arbeit in Gruppen

6.6 SubEthaEdit

SubEthaEdit ist ein kollaborativer Texteditor für Mac OS. Es wird das Bonjour-Konzept von Apple genutzt, welches automatisch Netzwerkdienste in Netzwerken erkennt. Somit ist es der Software möglich, eine Liste von potentiellen Teilnehmern zur Verfügung zu stellen, ohne dass eine Konfiguration oder ein Hinzufügen von Kontakten zu einer Kontaktliste nötig ist (vgl. Abbildung 22 [PW11]).

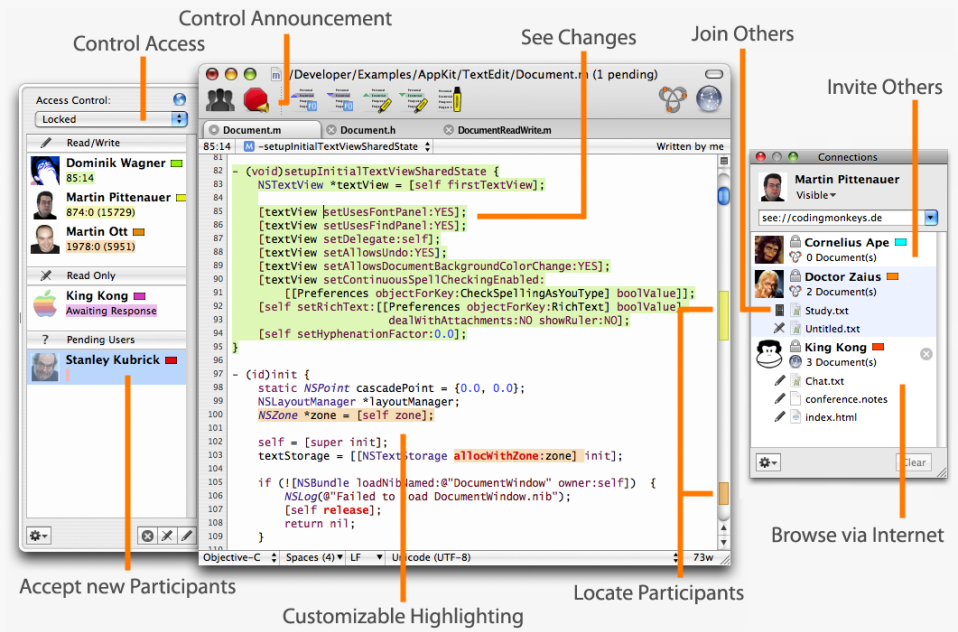

Abbildung 22: Die Nutzeroberfläche von SubEthaEdit

In SubEthaEdit gibt es zwei Möglichkeiten, die Zusammenarbeit an einem Dokument zu initiieren: Entweder lädt man durch das Bonjour-Feature gefundene Personen per drag-and-drop ein, oder man veröffentlich das Dokument, sodass User der Kollaboration auf Eigeninitiative beitreten können. In letztem Fall muss der Besitzer der Datei neuen potentiellen Teilnehmern erst gestatten, an dem Editieren mitwirken zu dürfen.

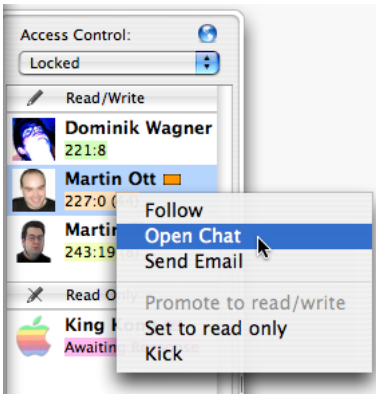

Abbildung 23: Aktionsmöglichkeiten des Hosts in Bezug auf einzelne Teilnehmer in SubEthaEdit

Ebenso können die Schreibberechtigungen für jeden User, wie in Saros, entfernt oder vergeben werden. Die Interaktionsmöglichkeiten in Bezug auf einzelne User gehen von demjenigen aus, der die Zusammenarbeit gestartet hat. Es gibt in diesem Kontext Parallelen zu Saros, SubEthaEdit bietet zudem die Möglichkeit, einen bestimmten Teilnehmer vom Weitereditieren auszuschließen (vgl. Abbildung 23 [PW11]).

7 Kriterien zur Evaluation

Infolge der Untersuchung der in Kapitel 6 vorgestellten Projekte entstand eine Liste von Features und Konzepten, deren Implementierung in Saros erwägenswert wäre. Aus diesem Grund werden diese im nächsten Kapitel genau dargestellt und nach gewissen Kriterien diskutiert und bewertet. Dies hat den Zweck, eine reduzierte Liste zu erhalten, deren Einträge letztendlich einen Mehrwert für Saros in Hinsicht auf eine Verbesserung der User Experience bieten, sofern sie implementiert werden - diese findet sich in Kapitel 9. Die Evaluation hat drei Kriterien zur Basis, welche im Folgenden erläutert werden. Ursprünglich war auch eine Kosten-Nutzen-Rechnung in dem Kriterienkatalog vorgesehen, ich entschied mich jedoch gegen diese, da die Kosten einer Umsetzung - also der tatsächliche Aufwand - nur von Personen zuverlässig abgeschätzt werden können, welche die Implementierung vornehmen.

7.1 Technische Umsetzbarkeit

Saros ist als Eclipse-Plugin sehr stark von den technischen Gegebenheiten von Eclipse selbst abhängig. So muss beispielsweise berücksichtigt werden, dass die grafischen Oberflächen in Eclipse auf SWT und JFace beruhen. Ebenso bietet Eclipse eine Architektur aus Views, Dialogen, Wizards etc. an, die zwar vielseitig einsetzbar ist, jedoch möglicherweise nicht die Umsetzung jeglicher Ansätze ermöglicht, welche in Kapitel 8 aufgeführt sind. Für einen Einblick in die Architektur von Eclipse verweise ich auf eine Ausarbeitung von Tilman Walther [Wal05].
Weiterhin ist zu beachten, dass der gesamte Datenverkehr in Saros auf dem XMPP-Protokoll basiert, welches auf das Versenden von Nachrichten und Präsenzinformationen ausgelegt ist. So ist zum Beispiel die Benutzerverwaltung in Saros durch den Einsatz sogenannter „Jabber Identifier“ realisiert, welche der Benutzer-Identifizierung innerhalb eines XMPP-Netzwerkes dienen. Auch der in Saros integrierte Multi-User-Chat beruht auf Funktionen des XMPP-Protokolls. [Nie06] Um dieses Kriterium zu erfüllen, muss eine Umsetzbarkeit unter Berücksichtung dieser Faktoren gegeben sein.

7.2 Mehrwert für die User Experience

Anhand der Definition von User Experience in Kapitel 2 dient dieses Kriterium einer Einschätzung, ob und wie das jeweilige Konzept oder Feature den Nutzern von Saros ein gesteigertes positives Benutzungserlebnis verschaffen kann (möglicherweise auch als Ersatz für bereits implementierte Mechanismen).

7.3 Passt das Konzept zu Saros?

Der letzte für mich relevante Punkt ist die Frage, ob das untersuchte Konzept oder Feature in den Kontext des Projektes Saros passt. Beispielsweise wäre ein erweitertes Benutzerprofil wie in gängigen sozialen Netzwerken sicherlich technisch realisierbar und hätte unter Umständen auch positive Auswirkungen auf die User Experience (bedingt durch das erweiterte Wissen über die Teilnehmer der Kollaboration), inwiefern dies allerdings mit dem programmier-orientierten Konzept von Saros vereinbar wäre, ist fraglich.

8 Features und Konzepte der anderen Projekte

Dieses Kapitel beinhaltet nun eine Auflistung der Features und Ansätze der zuvor vorgestellten Projekte, welche Auswirkungen auf die User Experience haben. Hierbei werden all diejenigen nicht dargestellt, welche entweder

- bereits in dieser oder ähnlicher Form in Saros vorhanden / integriert sind
- oder Duplikate von Funktionen eines anderen Vergleichsprojekt sind.

Die einzelnen Tabellen enthalten überdies jeweils eine kurze Diskussion und eine Überprüfung, ob die in Kapitel 7 genannten Kriterien erfüllt werden können. In der Kopfzeile ist stets eine Kurzbeschreibung samt der Angabe enthalten, aus welchem Projekt dieser Eintrag generiert wurde.

Nicht-Driver befindet sich permanent im Follow Mode		
gesehen in: PeP		
Die strikte Einhaltung der Rollenverteilung von Driver und Observer in PeP wird besonders durch die Tatsache, dass letzterer zu jedem Zeitpunkt die Aktionen des Drivers verfolgt, unterstützt. Dies orientiert sich sehr stark am Konzept der Paarprogrammierung selbst. Für Saros bietet sich eine Umsetzung eines dauerhaften Follow Modes, welcher auch zu keinem Zeitpunkt verlassen werden kann, jedoch nicht an - dies begründet sich damit, dass Saros darauf konzipiert ist, mehrere Driver und Observer zu haben.		
technisch umsetzbar: ✅	Mehrwert für UX: ❌	passt zu Saros: ❌

Force Resync Button		
gesehen in: PeP		
In PeP gibt es die Möglichkeit, die lokalen Code-Versionen der Teilnehmer auf Inkonsistenzen zu überprüfen - dies geschieht per Knopfdruck. Der Nachteil aus UX-Sicht ist hierbei jedoch, dass die User stets befürchten müssen, dass es zu Inkonsistenzen kommt, wenn sie diese Funktion nicht nutzen. Dies wirkt sich auch auf das Arbeitsverhalten aus, da man durch Benutzung des Buttons eventuell aus dem Kontext gerissen wird. Weiterhin stellt Saros bereits eine automatische Überprüfung durch das Package `concurrent.watchdog` zur Verfügung.		
technisch umsetzbar: ✅	Mehrwert für UX: ❌	passt zu Saros: ❌

Teilnehmer werden projektbezogen hinzugefügt		
gesehen in: google docs		
Das gemeinsame Bearbeiten von Dokumenten beruht in google docs auf dem einzelnen Hinzufügen aller gewünschten Teilnehmer zu dem jeweiligen Dokument durch den Besitzer. Diese können dann durch jeden dieser Teilnehmer jederzeit bearbeitet werden. Um dieses Verhalten zu übernehmen, müsste ein Server zur Verfügung stehen, um Projekte online zu speichern, womit der Gedanke der „Session“ verloren ginge. Somit würde die Zusammenarbeit in Saros nicht mehr das zeitgleiche Editieren zum Hauptaufgabenpunkt haben.		
technisch umsetzbar: ✅	Mehrwert für UX: ❌	passt zu Saros: ❌

User können individuelle Informationen übertragen		
gesehen in: Adobe Connect		
Die Präsentierenden erhalten durch Statusänderungen von Teilnehmern Feedback und können dieses zur laufenden Verbesserung ihrer Präsentation nutzen. In Saros wäre dies sehr nützlich, um weitere Awareness-Informationen zu übermitteln, wie beispielsweise eine kurzfristige Abwesenheit eines Session-Teilnehmers. Da das XMPP-Protokoll in der Lage ist, zusätzlich zu den Nachrichten an sich weitere Daten an die Pakete anzuhängig, wäre dies nicht nur wünschenswert, sondern auch technisch realisierbar.		
technisch umsetzbar: ✅	Mehrwert für UX: ✅	passt zu Saros: ✅

Webcam-Übertragung		
gesehen in: Adobe Connect		
In Adobe Connect können sich die Präsentierenden den Teilnehmern durch den Einsatz einer Webcam zeigen. Auch in der physischen Paarprogrammierung spielen visuelle Faktoren dahingehend eine Rolle, als dass sie Rückschlüsse auf den Zustand des Partners zulassen. Da sich die wesentlichen Aktionen in Saros jedoch im Editor View von Eclipse abspielen, wie z.B. Markierungen von Veränderungen im Code, ist durch eine zusätzliche, sich bewegende Darstellung ein negativer Einfluss auf die Produktivität zu erwarten.		
technisch umsetzbar: ✅	Mehrwert für UX: ✅	passt zu Saros: ❌

User connecten per IP, um zusammenzuarbeiten		
gesehen in: MoonEdit		
In MoonEdit können verschiedene User der Bearbeitung eines Dokuments beitreten, indem sie eine IP-Verbindung zu demjenigen herstellen, der die Rolle des Hosts einnimmt. Auf die Zusammenarbeit an sich hat dies im Vergleich zu der in Saros genutzten Buddyliste zwar keinen Einfluss, jedoch würde mit der Buddylist selbst ein nach [CM09] wichtiges Element der Echtzeit-Interaktion fehlen. Weiterhin müsste auch auf die mitgelieferten Präsenzinformationen verzichtet werden, was der Awareness in Saros abträglich wäre.		
technisch umsetzbar: ✅	Mehrwert für UX: ❌	passt zu Saros: ❌

Bearbeitete Textstellen werden uneditierbar gemacht		
gesehen in: google docs		
Wenn ein User in google docs beispielsweise eine Zelle einer Tabelle bearbeitet, so ist ausgeschlosen, dass ein anderer in dieser Veränderungen vornehmen kann. Um ein Durcheinanderschreiben in Saros zu vermeiden, scheint dies ein guter Ansatz zu sein. Ähnlich wie bei der Beschränkung auf Leserechte wäre es denkbar, eine gerade editierte Codezeile für andere unmodifizierbar zu machen. Ein angenehmer Nebeneffekt wäre hierbei, dass dies auch als Awareness-Information über die momentanen Arbeitsvorgänge dient.		
technisch umsetzbar: ✅	Mehrwert für UX: ✅	passt zu Saros: ✅

User können Medien-Dateien auf einen Server übertragen		
gesehen in: google docs		
Diese hochgeladenen Dateien können dann von allen an der Zusammenarbeit beteiligten Nutzern abgerufen werden. Betrachtet man diese Idee im Kontext von Saros, so fällt auf, dass man bereits per Direktverbindung Daten an einzelnen Buddy übertragen kann, zum anderen ist durch das Konzept der Session zu keinem Zeitpunkt eine derartige Verfügbarkeit von Mediendateien nötig, da Saros keine asynchrone Arbeit an einem gemeinsamen Dokumenten vorsieht.		
technisch umsetzbar: ✅	Mehrwert für UX: ❌	passt zu Saros: ❌

Gruppenarbeiten in separaten Räumen		
gesehen in: Adobe Connect		
Auf Wunsch des Hosts einer Präsentation können alle Teilnehmer in Gruppen aufgeteilt und in verschiedene Userräume versetzt werden, um dort in Teams zu arbeiten und später die Ergebnisse zusammenzuführen. Für Saros ist diese Idee aus zweierlei Gründen nicht praktikabel: Zum einen impliziert dies einen mehrfachen Ab- und wieder Aufbau der Session und zum anderen wird der Teil des Konzepts von Saros verletzt, welcher die zeitgleiche gemeinsame Editierung aller Teilnehmer vorsieht.		
technisch umsetzbar: ✗	Mehrwert für UX: ✗	passt zu Saros: ✗

benutzerbezogenes „undo“ und „redo“		
gesehen in: MeetingWords		
Während der Bearbeitung eines Dokuments werden laufend Revisionen angelegt, welche es ermöglichen, die letzten Aktionen eines Benutzers rückgängig zu machen, ohne dass dies Auswirkungen auf die Arbeit anderer Nutzer hat. Bei Saros verhält es sich aufgrund des Synchronitätsverfahrens so, dass auch die Änderungen anderer bei einem „undo“ betroffen sind. Durch den Einsatz von benutzerbezogenen Revisionen sollte dies auch im Rahmen von Eclipse umsetzbar sein.		
technisch umsetzbar: ✓	Mehrwert für UX: ✓	passt zu Saros: ✓

Anmerkung: Es gibt in Saros bereits das entsprechende Package `concurrent.undo`, einer Aussage von Karl Beecher im Feature-Request-Bereich von Saros (http://sourceforge.net/tracker/?func=detail&aid=3414948&group_id=167540&atid=843362) zufolge war dieses jedoch bisher nicht funktionstüchtig.

User können ihren Nickname selbst bestimmen		
gesehen in: MoonEdit		
Bevor die Zusammenarbeit in MoonEdit möglich ist, wird der Nutzer aufgefordert, einen Namen zu bestimmen, unter welchem er den anderen in der Liste der Teilnehmer angezeigt wird. Eine Nickname-Vergabe ist in Saros bisher nur für die Buddies in der eigenen Buddylist möglich. Da es vorkommen kann, dass jemand an der Kollaboration teilnehmen möchte, der einem selbst noch nicht bekannt ist (z.B. bei der Zusammenarbeit verschiedener Unternehmen), so ist eine individuelle Nickname-Vergabe sinnvoll.		
technisch umsetzbar: ✓	Mehrwert für UX: ✓	passt zu Saros: ✓

Passwortgesicherte Zusammenarbeit		
gesehen in: MoonEdit		
Um unerwünschte IP-Verbindungen auf die Sitzung zur Dokumentenbearbeitung zu vermeiden, bietet MoonEdit die Möglichkeit, eine Passwortabfrage einzubinden. Da Saros-Sessions jedoch nur auf Einladung und nicht durch Eigeninitiative betreten werden können, ist eine solcher Schutzmechanismus nicht nötig und wirkt höchstens abschreckend denjenigen gegenüber, die an einer Kollaboration teilnehmen möchten.		
technisch umsetzbar: ✅	Mehrwert für UX: ❌	passt zu Saros: ❌

Aufzeichnung aller Änderungen		
gesehen in: MeetingWords		
Mit dem „TimeSlider“ ist es in MeetingWords möglich, in einem Schnelldurchlauf alle Veränderungen des bearbeiteten Dokuments zu betrachten. Dies ist ein hervorragendes Mittel zur Protokollierung der Zusammenarbeit und von daher ist eine Implementierung in Saros erwägenswert. Durch das Anlegen von Revisionen nach jeder Änderung ist eine solche Wiedergabe auch in Eclipse realisierbar.		
technisch umsetzbar: ✅	Mehrwert für UX: ✅	passt zu Saros: ✅

User aus der Session entfernen		
gesehen in: SubEthaEdit		
Klickt der Besitzer des Dokuments im Kontextmenü eines anderen Users, der momentan an der Bearbeitung beteiligt ist, auf „kick“, so wird dieser aus der Kollaboration entfernt. Eine solche Funktionalität ist in Saros wünschenswert, um der Rolle des Hosts, der die komplette Kontrolle über die Session haben soll, gerecht zu werden. Nach heutigem Stand müsste eine Session geschlossen und wieder neu eröffnet werden (inkl. Sharen der Dateien), was nicht als gute Lösung angesehen werden kann.		
technisch umsetzbar: ✅	Mehrwert für UX: ✅	passt zu Saros: ✅

Cursor-Positionen anderer anzeigen		
gesehen in: MoonEdit		
In MoonEdit werden die Mauscursor aller anderen Teilnehmer dem User in einem Grauton angezeigt. Somit weiß er jederzeit auch ohne eine Textmarkierung, welche Abschnitte momentan im Fokus der anderen Nutzer liegen. Dies ist ein großartiger Mechanismus, um Awarenessinformationen zu vermitteln, jedoch hat ein Saros-User bereits die Möglichkeit, durch einen Doppelklick auf einen Sessionteilnehmer zu dessen aktueller Position zu springen (auch dateiübergreifend).		
technisch umsetzbar: ✅	Mehrwert für UX: ✅	passt zu Saros: ❌

Interaktionen durch drag&drop		
gesehen in: SubEthaEdit		
Die Menüführung in SubEthaEdit unterstützt den Einsatz von drag&drop. So kann ein Eintrag der Kontaktliste zum Beispiel in das Bearbeitungsfenster gezogen werden, um diese Person zu der Liste der Mitwirkenden hinzuzufügen. Eine Recherche ergab, dass drag&drop in Eclipse zwar nicht trivial, aber durchaus umsetzbar ist. [Art03] Insgesamt ist zu erwarten, dass drag&drop positive Auswirkungen auf die intuitive Benutzung von Saros hat.		
technisch umsetzbar: ✅	Mehrwert für UX: ✅	passt zu Saros: ✅

9 User Stories zu ausgewählten Features

Das Saros-Team der Arbeitsgruppe Software Engineering (AG SE) orientiert sich in einigen Aspekten der Entwicklung an der agilen Entwicklungsmethode Scrum - so gibt es beispielsweise regelmäßige StandUps der momentanen Mitglieder, es existiert ein Continuous Integration-System und auch ein Releaseplan wurde festgelegt. Um diesen Prozess in der AG SE zu unterstützen, formuliert dieses Kapitel nun User Stories, also Anforderungen aus Nutzersicht, zu den Features und Konzepten der Vergleichs-Projekte, welche alle der in Kapitel 7 genannten Kriterien erfüllen. User Stories sind ein elementares Artefakt im Scrum-Prozess und bestehen jeweils aus einem Titel und einer Beschreibung. Somit soll ein solider Grundbaustein für eine eventuelle Umsetzung gelegt werden, da diese im Rahmen dieser Arbeit aufgrund des nötigen Konzeptionsaufwandes nicht möglich war. Hierbei wird ein Augenmerk darauf gelegt, dass jeweils immer nur das „was“ beschrieben wird - das „wie“ entscheidet der Programmierer durch seine Implementierung.

Übertragen von persönlichen Status-Informationen
Als Saros-Nutzer möchte ich die Möglichkeit haben, selbstbestimmte Status-Informationen anzugeben. Diese sollen sich bei weiteren Änderungen aktualisieren und allen anderen Teilnehmern an der Session an geeigneter Stelle (z.B. in der Buddylist) angezeigt werden.

Uneditierbarkeit von derzeit bearbeiteten Stellen
Als Saros-Nutzer möchte ich, dass andere Session-Teilnehmer die Stelle, die ich gerade bearbeite, solange nicht editieren können, bis ich mit meiner aktuellen Bearbeitung fertig bin.

benutzerbezogenes „undo“ und „redo“
Als Saros-Nutzer möchte ich in der Lage sein, meine letzten Änderungen unabhängig von den Änderungen anderer User rückgängig zu machen.

Eigenen Nickname angeben
Als Saros-Nutzer möchte ich einen eigenen Nicknamen für meinen XMPP-Account angeben können, welcher meinen Buddies in deren Buddylist angezeigt wird.

Aufzeichnung der Session
Als Saros-Nutzer möchte ich eine Aufzeichnung der Session, in der ich mich gerade befinde, anfertigen und ansehen können.

User aus laufender Session entfernen
Als Host einer Saros-Session möchte ich in der Lage sein, einzelne Teilnehmer aus der laufenden Session zu entfernen, ohne dass die Session an sich oder andere Teilnehmer davon betroffen sind.

Drag&drop-Hinzufügen von Buddies/Projekten
Als Saros-Nutzer möchte ich Projekte und Buddies durch drag&drop zu einer laufenden Session hinzufügen können.

10 Im Rahmen dieser Arbeit umgesetzte Änderungen

Nachdem im letzten Kapitel eine Arbeitsgrundlage für die Umsetzung der genannten Features, welche aus den erwähnten Gründen nicht realisiert werden konnten, geschaffen wurde, widmet sich dieses nun den von mir vorgenommenen Änderungen im Code von Saros.
Zu diesem Zweck wird zunächst eine Einführung in die Funktion der Datei `plugin.xml` gegeben, da diese Hauptbestandteil der in Abschnitt 10.4 vorgestellten Implementierung ist. Die darauf folgenden Abschnitte nehmen Bezug auf die in Kapitel 5 herausgestellten Probleme im Rahmen der Kontextmenüs und zeigen auf, wie diese bearbeitet wurden.
Aus einer Recherche [CM09] ergab sich weiterhin, dass die Umformulierung einiger Textpassagen große Wirkung auf die User Experience haben kann - auf diesen Aspekt wird in Abschnitt 10.5 eingegangen.

10.1 Grundlage: Benutzung der plugin.xml

Dieses Kapitel enthält Änderungen, welche in der Datei `plugin.xml` vorgenommen wurden - dies betrifft die Bearbeitung der Kontextmenüs. Jeden Menüpunkt bezeichnet man als `command` - ein `command` kann als die Beschreibung des Menüeintrags angesehen werden. Jeder `command` ist mit einem `handler` verknüpft, welcher dann das Verhalten des `commands` bei dessen Aufruf bestimmt. Diese `handler` befinden sich in Saros im Package `ui.commandHandlers` (z.B. gibt es einen `addBuddyHandler`). In der Klasse `ui.views.SarosView.java` kann zwar eine grundsätzliche Zuweisung von Menüpunkten zu einem Menü erfolgen (für Genaueres siehe Abschnitt 10.2), wenn jedoch die Anzeige oder Verfügbarkeit eines Eintrages mit zusätzlichen Bedingungen verknüpft sein soll, so wird der Einsatz von sogenannten `MenuContributions` nötig. Diese erweitern bereits vorhandene Menüs (wie z.B. das normale Eclipse-Menü im oberen Bereich der Eclipse-Umgebung oder auch das Menü von Saros, welches Interaktionsmöglichkeiten für Einträge im Roster darstellt) um einzelne Menüpunkte.

`MenuContributions` enthalten auch eine Liste von `commands`, welche allerdings mit den gewünschten Bedingungen logisch verknüpft werden können. So ermöglicht beispielsweise das Keyword `visibleWhen` ein Nicht-Anzeigen eines Menüpunktes, wenn bestimmte Bedingungen erfüllt werden. Diese nennen sich `definitions` und liefern Werte, welche zur Evaluation durch logische Verknüpfungen genutzt werden. Es gibt eine Reihe an `definitions`, welche Eclipse von sich aus anbietet (beispielsweise die Überprüfung, ob ein Workspace vorhanden ist). Möchte man jedoch Überprüfungen ausführen, welche im Zusammenhang mit dem jeweiligen Projekt und nicht Eclipse im

Allgemeinen stehen, so werden sogenannte `propertyTester` benötigt. Diese sind in der Lage, Methoden innerhalb einer Klasse aufzurufen und die zurückgelieferten Ergebnisse weiter zu verarbeiten. Die in Saros verwendeten `propertyTester` finden sich im Package `propertyTesters`.

Da allerdings eine ständige Neuauswertung dieser Bedingungen starke Auswirkungen auf die Performance von Eclipse selbst hätte, wird eine solche Re-Evaluation nur vorgenommen, wenn sich in dem von einem `propertyTester` abgedeckten Umfeld etwas geändert hat - dies wird über `services` realisiert (zu finden im Package `serviceProviders`). Diese verfügen über Listener, welche auf Änderungen reagieren, indem sie mit der Methode `fireSourceChanged()` den jeweiligen `propertyTestern` diese Information weitergeben und die Neuauswertung aller selbst erstellten `definitions` einleiten. Dies ist für Abschnitt 10.4 relevant.

10.2 Skype-Button innerhalb der Session

Die grundlegende Struktur der Saros-Kontextmenüs findet sich in der Klasse `ui.views.SarosView.java`. Alle Interaktionsmöglichkeiten, welche im Package `ui.views.actions` implementiert sind, können hier den Menüs - Toolbar, Rostermenü oder Sessionmenü - zugewiesen werden (vgl. Abbildung 24). Dies geschieht über einen `MenuManager`, welcher von Eclipse bereitgestellt wird und der Verwaltung von Menüs dient. Um eine Fallunterscheidung vornehmen zu können, ob das Menü für die Session oder das Roster angezeigt werden soll, besteht die Möglichkeit, vor dem Hinzufügen der einzelnen Menüpunkte Bedingungen anzugeben:

```
/*
 * Do not display the following actions if
    participants are
 * selected.
 */
List<User> participants = SelectionRetrieverFactory
    .getSelectionRetriever(User.class).getSelection();
if (participants.size() > 0)
    return;
/*
 * Do not display the following actions if no buddies
    are
 * selected.
 */
List<JID> buddies = SelectionRetrieverFactory
    .getSelectionRetriever(JID.class).getSelection();
if (buddies.size() == 0)
    return;
```

Abbildung 24: Überblick über die Saros-eigenen Menüs

Bei diesem Codeausschnitt wird beispielsweise sichergestellt, dass ein Buddy ausgewählt wurde, nicht jedoch ein Eintrag im Session-Bereich. Daraufhin wird eine `action` dem Menü durch `manager.add(action);` hinzugefügt, wobei `manager` eine Ausprägung des oben genannten `MenuManagers` ist. Um nun den Skype-Button im Session-Menü einzubauen, muss ein entsprechender Eintrag in der Methode `addSessionMenuItems(MenuManager menuManager)` vorgenommen werden, da diese für genau dieses Menü zuständig ist. Hierzu wird zuerst ein Objekt der sogenannten `SkypeAction` instanziiert und daraufhin dem `MenuManager` hinzugefügt:

```
final SkypeAction skypeAction = new SkypeAction();}
...}
manager.add(skypeAction);}
manager.add(new Separator());}
```

Bei dem `Separator` handelt es sich hierbei um eine Trennlinie, welche zwischen zwei Einträgen eines Menüs eingefügt werden kann. Dies mache ich an dieser Stelle, da sich der Skype-Menüpunkt an der gleichen Position befindet, welche er auch bei einem Menüaufruf im Roster innehat - und auch dort wird ein `Separator` verwendet. So soll der Erwartungshaltung entsprochen werden, dass gleiche Funktionen an gleicher Stelle vorgefunden werden können.

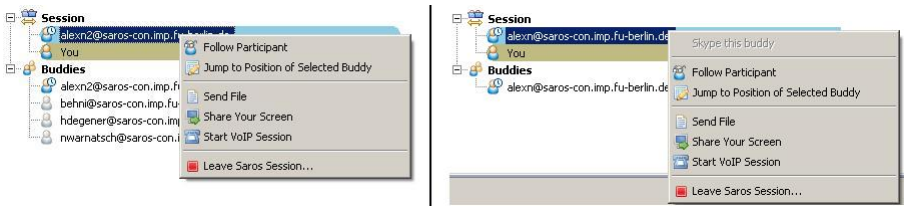

Abbildung 25: Vergleich vorher(links)-nachher(rechts): Der Skype-Menüpunkt ist nun auch in einer Session verfügbar

10.3 Irreführende Nichtselektierbarkeit

In Kapitel 5 wurde dargelegt, dass die Menüpunkte „Share with Multiple Buddies“ und „Share Multiple Projects“ zwar nicht ausführbar sind, wenn der User nicht im Status „connected“ ist, jedoch ist das gegebene Feedback irreführend. Während die Einzelauflistung der Projekte/Buddies durch den Eintrag „not connected“ ersetzt wird, ist dies bei den benannten Menüpunkten nicht der Fall - diese werden lediglich grau hinterlegt (vgl. Abbildung 26). Ein Nutzer könnte daraus schlussfolgern, dass die Nichtverfügbarkeit der „Multiple“-Funktionen mit einem anderen Grund zusammenhängt als dem Nichtverbundensein, dies ist aber falsch. Um hier einer Irreführung vorzubeugen, werden die beiden betroffenen Menüpunkte nun nur noch unter der Bedingung in dem Menü angezeigt, dass der User mit seinem XMPP-Account angemeldet ist. Somit gilt das Feedback von „not connected“ sowohl für die Einzelauflistungen als auch die „Multiple“-Funktionen.

Abbildung 26: Menüeinträge des Projekt-Sharens in den Zuständen „connected“ (links) und „not connected“ (rechts)

Zu diesem Zweck wurden die beiden `MenuContributions` „Share with“ und „Work Together On...“ bearbeitet. Die `commands` „Multiple Buddies...“ und „Multiple Projects“ besitzen nun jeweils eine Überprüfung auf den Status des Users. Sollte dieser nicht mit seinem Account angemeldet sein, so werden die Menüs durch die Verwendung von `visibleWhen` ausgeblendet (vgl. Abbildungen 27 und 28). „isConnected“ ist hierbei eine `definition`, welcher den `propertyTester propertyTesters.SarosPropertyTester.java` benutzt, um zu erfahren, ob der Saros-User mit seinem XMPP-Account angemeldet ist oder nicht.

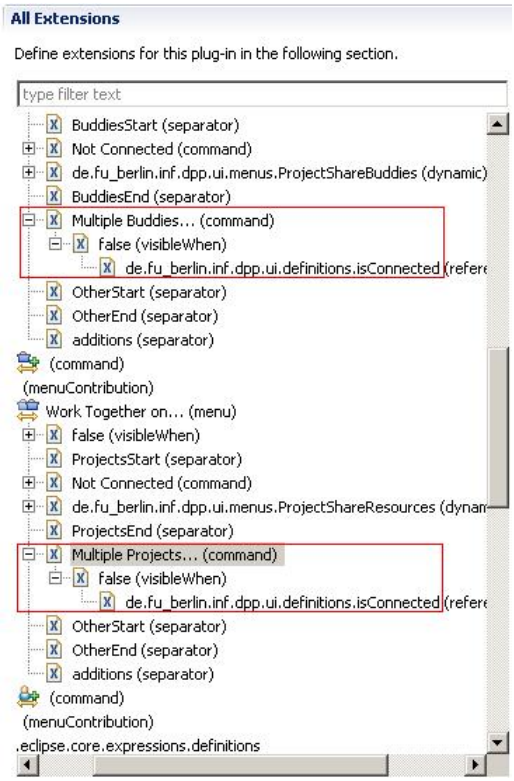

Abbildung 27: Grafische Darstellung der plugin.xml, markiert sind die neu hinzugefügten Überprüfungen

Abbildung 28: Das Feedback „not connected“ gilt nun auch für die „Multiple“-Funktionen (Beispiel multiple Projektauswahl)

10.4 „Multiple“-Funktionen erst ab zwei Einträgen verfügbar machen

Ein weiterer in Kapitel 5 herausgestellter Makel ist die Tatsache, dass der User die Optionen „Share Multiple Projects“ und „Share With Multiple Buddies“ auch dann hatte, wenn ein oder sogar kein Element dieser Typen vorhanden war. Das multiple Sharing von Projekten wurde nun so angepasst, dass mindestens zwei Projekte im Workspace vorhanden sein müssen, damit dieser Menüeintrag anklickbar ist. Warum eine ähnliche Einschränkung für das mehrfache Einladen von Buddies im Rahmen dieser Arbeit nicht möglich war, wird nach einer Erklärung der getätigten Änderungen für erstgenannten Menüpunkt erläutert.

Eingangs wurde erwähnt, dass projektbezogene Überprüfungen über `propertyTester` und `services` realisiert werden müssen - genau das war hier der Fall, da es eine Aussage über die Anzahl der Projekte im Workspace seitens Eclipse nicht gibt. Der erste Schritt war also, einen neuen `service` zu implementieren. Analog zu dem bereits vorhandenen `SarosSourceProvider` wurde dieser `WorkspaceSourceProvider` benannt. Da das Interface „AbstractSourceProvider“ erweitert wird, müssen hierbei

- der Konstruktor `public WorkspaceSourceProvider()` erstellt
- und die drei Methoden `public void dispose()`, `public Map getCurrentState()` und `public String[] getProvidedSourceNames()` implementiert werden.

Weiterhin wurde ein `IResourceChangeListener` hinzugefügt, welcher reagiert, sobald die Ressourcen - also Projekte - im Workspace eine Veränderung erfahren. Dieser ruft in diesem Fall die Methode `workspaceChanged();` auf, welche wiederum die Ausführung von `fireSourceChanged(ISources.WORKBENCH, WORKSPACE, workspace);` dem weiter unten aufgeführten `propertyTester` mitteilt, dass sich eben diese Änderung ereignet hat. Der Quellcode der Klasse `WorkspaceSourceProvider.java` ist im Anhang einsehbar.

Danach wurde der `propertyTester WorkspacePropertyTester` angelegt. Dieser dient nun der Überprüfung der gewünschten Bedingung an sich - in diesem Fall also, ob sich mehr als ein Projekt im Workspace befindet. Dies geschieht durch:

```
if ("hasMultipleProjects".equals(property))} {
  return workspace.getRoot().getProjects().length >=
     2;}
}
```

„hasMultipleProjects“ ist hierbei die Bezeichnung, auf welche der `propertyTester` reagiert. Daher ist ersichtlich, dass, so es denn gewollt ist, mehrere Methoden in einem `propertyTester` mit verschiedenen Bezeichnungen integriert werden können. Der vollständige Quellcode ist auch im Anhang dieser Arbeit dokumentiert.

Nachdem dies getan war, konnte nun die `definition` angelegt werden. Der Namenskonvention der bereits vorhandenen Definitionen folgend, erhielt sie den Namen „workspaceHasMultipleProjects“. Die Verknüpfung mit dem `WorkspacePropertyTester` in der plugin.xml ist in Abbildung 29 dargestellt.

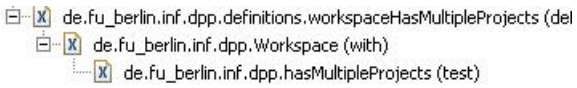

Abbildung 29: Darstellung der Definition „workspaceHasMultipleProjects“ in der plugin.xml

Der letzte Schritt war nun, die entsprechenden Änderungen im Menü selbst vorzunehmen. Da im letzten Abschnitt das von Saros gegebene Feedback kritisiert wurde, war es an dieser Stelle auch wichtig, dem User eindeutig aufzuzeigen, weswegen die Option in bestimmten Fällen nicht benutzbar ist. Ich entschloss mich dazu, einen neuen Menüpunkt anzulegen, welcher die Bezeichnung „Less Than 2 Projects in Workspace“ erhielt und nur angezeigt werden soll, wenn das Multiple Sharing aufgrund der zu geringen Anzahl vorhandener Projekte nicht verfügbar ist. Die neue `definition` wurde durch ein „and“ logisch mit der Änderung im letzten Abschnitt verknüpft. Für den neuen Menüpunkt wurde das gleiche getan, jedoch mit dem Unterschied, dass die neue `definition` logisch negiert wurde und der zugehörige `handler` mit dem `command` „Empty“ verknüpft ist, was dafür sorgt, dass hinter dem Menüpunkt keinerlei Funktionalität steht und er lediglich grau hinterlegt dargestellt wird.

Eine Darstellung des neuen Zustandes der betroffenen `MenuContribution` in der `plugin.xml` ist in Abbildung 30 gegeben, eine Visualisierung der neuen Anzeige in Saros selbst findet sich in Abbildung 31.

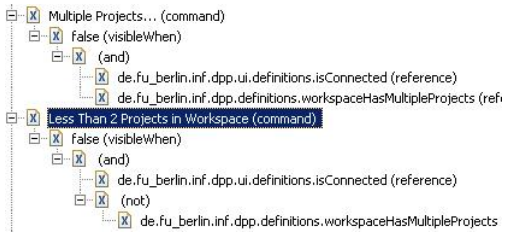

Abbildung 30: Der betroffenen Abschnitt der plugin.xml nach den vorgenommenen Änderungen

Abbildung 31: Multiples Sharing ist nun nur noch mit mehr als zwei vorhandenen Projekten möglich

Die Umsetzung einer ähnlichen Anzeige im Menü für das multiple Einladen von Buddies beruht auf den gleichen Ansätzen, wird allerdings dadurch erschwert, dass die Buddylist aus mehreren Komponenten aufgebaut ist: Dem `Roster` und den sogenannten `RosterEntries` (welche die jeweiligen Buddies sind). Dadurch zieht sich eine Überprüfung auf Änderungen durch mehrere Ebenen, denn eine Änderung eines `RosterEntries` ist nicht gleichbedeutend mit der des `Rosters` an sich. Mit dem oben beschriebenen Vorgehen könnte man einzelne Einträge im `Roster` überprüfen, was allerdings nicht zielführend wäre, weil jeder `RosterEntry` einzeln betrachtet werden müsste - denn beispielsweise wirkt sich ein neu hinzugefügter Buddy nicht auf das `Roster` an sich aus. Analog zu einer Überprüfung auf Änderungen im Workspace, wie sie von mir vorgenommen wurde, erscheint mir eine Überprüfung des `Rosters` selbst die einzige sinnvolle Handhabung zu sein. Da dies jedoch aus genanntem Grund ein Konzept erfordert, welches beschreibt, wie `RosterEntry`-Änderungen zu einer `Roster`-Änderung führen, ist dies wesentlich komplexer als die von mir getätigte Umsetzung. Daher war eine derartige Implementierung für das multiple Einladen von Buddies im Rahmen dieser Arbeit leider nicht möglich.

10.5 Einladung zu Interaktionen

In dem Buch „Designing Social Interfaces“ [CM09] wird auf den Mehrwert hingewiesen, welcher entsteht, wenn man die Aufforderungen zur Interaktion als Fragen und Einladungen formuliert. Dieser Effekt begründe sich darin, dass beim User das Gefühl entstehe, die Kontrolle über das Geschehen zu haben, was bei einer Aufforderung nicht gegeben sei. Infolge dessen werde auch die Freude im Umgang mit dem Produkt gesteigert.

In Saros gibt es sogenannte Wizards, welche Oberflächen zur Benutzerinteraktion darstellen. In Abbildung 32 ist als Beispiel der Wizard zum Hinzufügen eines neuen Buddies gezeigt. Bei einer Untersuchung aller Saros-Wizards fiel mir auf, dass an vielen Stellen eine Umformulierung in eine Frage einen angenehmeren Eindruck entstehen lässt. Der Inhalt aller dem Nutzer gezeigten Wizards wird in Klassen des Packages `ui.wizards.pages` festgelegt, welche das JFace-Interface „WizardPage“ erweitern. In diesen können unter anderem der Titel und die Beschreibung des jeweiligen Wizards angepasst werden. Die zweite Quelle für die dargestellten Texte in Wizards ist die Datei `ui.messages.properties`, in welcher sich alle übrigen Texte befinden und von dort per Aufruf durch die jeweiligen `WizardPages` eingebunden werden können.

Abbildung 32: Beispiel des Einsatzes eines Wizards, um einen Buddy hinzuzufügen

Im Folgenden werden die Umformulierungen durch Präsentation von vorher-nachher-Vergleichen dargestellt. Ebenso wird der Kontext genannt, in welchem die jeweilige Textpassage zum Einsatz kommt.

Es sei erwähnt, dass diese Liste nicht alle in Saros vorhandenen Wizards umfasst. Dies liegt daran, dass sich nicht bei allen eine Umformulierung zu einer fragenden Aufforderung angeboten hat - beispielsweise bei der Erstellung eines neuen Accounts müsste das gesamte Layout überdacht werden, um solche Fragen sinnvoll einzubauen, da logistisch kein Platz für einladende Formulierungen gegeben ist (vgl. Abbildung 33).

Abbildung 33: Der Wizard zur Erstellung eines neuen XMPP-Accounts

Kontext: Hinzufügen eines neuen Kontaktes
ehemals:
Enter the XMPP/Jabber ID of the buddy you want to add.
überarbeitet:
What is the XMPP/Jabber ID of the buddy you want to add?

Kontext: Anlegen eines neuen Accounts
ehemals:
New Account
überarbeitet:
Want to create a new account?

Kontext: Benutzen eines bereits vorhandenen Accounts
ehemals:
Existing Account
überarbeitet:
Already got an account? Feel free to use it!

Kontext: Auswahl der Projekte zum Sharen
ehemals:
Select the file(s) to work on.
überarbeitet:
Which file(s) would you like to work on?

Kontext: Auswahl der Buddies, welche eingeladen werden sollen
ehemals:
Select the buddy(s) to work with.
überarbeitet:
Which buddies would you like to work with?

Kontext: Eingehende Übertragung eines geteilten Projektes
ehemals:
Select local project.
überarbeitet:
Use an existing project or create a new one?

Kontext: Umbenennung eines Buddies
ehemals:
Enter the new nickname of this buddy ’0’
überarbeitet:
Which nickname do you want to use for this buddy?

11 Resümee und Ausblick

Das Ziel dieser Arbeit war, eine Grundlage für die Verbesserung der User Experience in Saros zu schaffen. Zu diesem Zweck wurden zum einen andere Projekte, welche die verteilte Zusammenarbeit ermöglichen, untersucht und mit Saros verglichen. Es ergab sich eine Liste von Features und Konzepten, deren Implementierung im Rahmen dieser Arbeit leider nicht möglich war. Dennoch liefert sie Ansätze, wie die Erfahrungen, die ein Nutzer im Umgang mit dem Projekt macht, positiv beeinflusst werden können. Hierzu wurden auch formale Anforderungen in Form von User Stories verfasst.

Zum anderen wurden die Interaktionsmöglichkeiten von Nutzern innerhalb von Saros auf Sinnhaftigkeit und Konsistenz überprüft. Eine Aufschlüsselung aller Rollen und Zustände, die ein User innehaben kann, und der darauf folgende Abgleich der tatsächlich jeweils verfügbaren Interaktionen mit den vom Saros-Konzept vorgesehenen ergab keinerlei Verstöße oder Ungereimtheiten. Infolge dessen widmete ich mich den Kontextmenüs, welche eine zentrale Rolle in der Benutzung von Saros darstellen. Hier konnten einige Probleme gefunden und behoben werden. Dies sollte dazu beitragen, das Ausmaß von Irreführung, Verwirrung und Frustration, was gegebenenfalls bei der Benutzung entstehen könnte, zu reduzieren.

Weiterhin wurden die grafischen Interaktionsoberflächen, genannt Wizards, daraufhin untersucht, inwieweit sie zur Aktion einladen. Nach Literaturrecherche wurden Eingabeaufforderungen - sofern dies zielführend war - als Fragen formuliert, um so beim Nutzer das Gefühl zu stärken, dass er tatsächlich Herr der Dinge ist, was ein wesentlicher Aspekt der User Experience ist.

Insgesamt bleibt festzuhalten, dass die Ergebnisse dieser Arbeit ein erster Schritt in Richtung einer Verbesserung der Nutzerfreundlichkeit und somit der UX sind, jedoch bei weitem nicht ausreichen, um eine gänzlich zufriedenstellende User Experience in Saros zu garantieren. Dazu ist dieser Themenkomplex zu umfangreich und bedarf weiterer Untersuchung. Ein Aspekt, der im Rahmen dieser Arbeit nicht behandelt werden konnte, ist das Einholen von Feedback von Usern - die Validierung beruht auf einschlägiger Literatur und Rücksprache mit meiner Betreuerin. Gerade User-Feedback erscheint mir für die Zukunft jedoch wichtig, wenn es um die Optimierung der in Saros verwendeten Prozesse geht, welche für eine ganzheitliche Verbesserung der User Experience unumgänglich ist.

Bei der Analyse der Kontextmenüs fiel mir überdies auf, dass die wichtige Funktion des Projekt-Teilens bei Auswahl eines Buddies die unterste Position des Menüs belegt. Es wäre eine enorme Verbesserung, wenn die gesammelten Daten über das Saros-Nutzerverhalten in Hinsicht auf die Häufigkeit der Verwendung bestimmter Funktionen ausgewertet würden, um infolgedessen die Menüpunkte nach Priorität und Relevanz anzuordnen. Dies würde die intuitive Benutzung von Saros unterstützen und somit auch positive Auswirkungen auf die User Experience haben.

12 Zusatz: Konkurrenzfähigkeit von Saros

Dieses Kapitel beschäftigt sich mit der Frage, inwiefern Saros gegenüber anderen Projekten, welche sich auf die verteilte Programmierung spezialisiert haben, konkurrenzfähig ist. Darum werden, um einen direkten Vergleich zu ermöglichen, in erster Linie Plugins für Eclipse betrachtet. Nach einer Internetrecherche hat sich ergeben, dass es neben Saros vier bekannte Plugins gibt:

- Pair Eclipse Programming (http://pep-pp.sourceforge.net/)
- XPairtise (http://xpairtise.sourceforge.net/)
- Sangam (http://sangam.sourceforge.net/)
- DocShare (http://wiki.eclipse.org/DocShare_Plugin)

Das Projekt Pair Eclipse Programming wurde bereits in Kapitel 6 vorgestellt. Es ist funktionstüchtig, beschränkt sich jedoch auf eine geringe Anzahl von Funktionen, welche die Paarprogrammierung unterstützen sollen. Es besteht eine direkte Netzwerk-Endverbindung zwischen den beiden Teilnehmern, von welchen einer der Driver und der andere der Observer ist.

XPairtise hält sich wie PeP strikt an das Konzept der Paarprogrammierung, weswegen es auch hier genau einen Driver und einen Observer (genannt „Navigator“) gibt. Leider ist es mir nicht gelungen, dieses Projekt selbst auszuprobieren, da die Verbindung zu einem nach Anleitung selbst eingerichteten Server nicht funktionierte. Aus diesem Grund beruhen alle hier wiedergegebenen Informationen auf Bild- und Textmaterial von der XPairtise-Internetseite. Das Grundkonzept beruht auch auf dem Einsatz einer Session, jedoch kommt diese anders zustande: Während in Saros Buddies direkt eingeladen werden, gibt es bei XPairtise eine „Session Gallery“, in welcher alle eingerichteten Kollaborationen aufgelistet sind. Der User kann wählen, welcher Session er beitritt und gibt dabei an, welche Rolle er gerne innehaben würde (vgl. Abbildung 34 [XP008]).

Wie in der Abbildung ersichtlich, existiert in XPairtise eine dritte Rolle: Spectator. User in dieser Rolle befinden sich automatisch in einem Modus, welcher mit dem Follow Mode in Saros übereinstimmt. Sollten der Driver und der Navigator einer Session bereits feststehen, so betritt man sie automatisch in dieser Beobachterfunktion. Andere User können, sobald man sich selbst innerhalb der Session befindet, eingeladen werden. Der Navigator kann einen Rollenwechsel mit dem Driver beantragen, welchem letzterer vor dem Tausch zustimmmen muss.

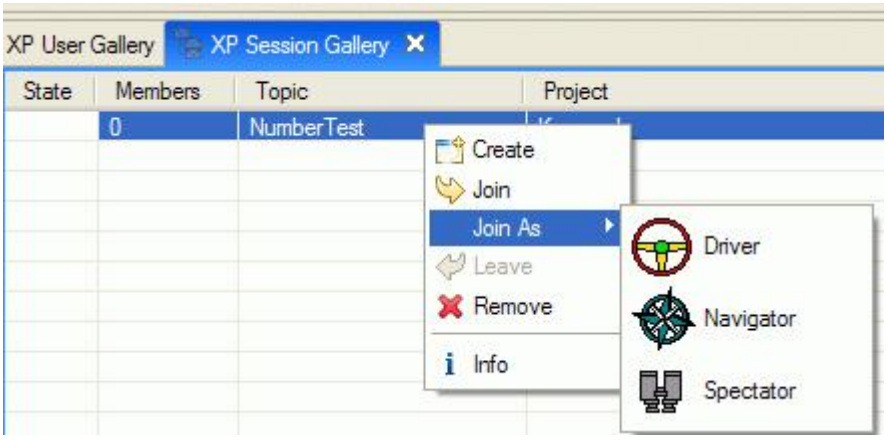

Abbildung 34: Beitreten einer Session in XPairtise

Die Arbeiten an XPairtise wurden im Jahre 2008 eingestellt. Möglicherweise ist dies auch der Grund, warum das Projekt in der aktuellen Eclipse-Version 3.7 nicht lauffähig war. Das Konzept der Session Gallery ist interessant, da es eine Zusammenarbeit vom Projekt selbst und nicht von der Anwesenheit der beteiligten Personen abhängig macht.

Sangam benutzt auch einen selbst konfigurierten Server, um die verteilte Paarprogrammierung zu realisieren. Es wurde seit 2004 nicht mehr weiterentwickelt und war leider auch nicht lauffähig, weswegen die zugehörige Webseite auch hier als Informationsquelle dient. Der Dokumentation zufolge nutzte es das Kizna Syncshare Messaging Platform, jedoch ist die Internetseite dieses Projektes nicht mehr erreichbar. Es gibt wieder die feste Aufteilung in einen Driver und mehrere Observer, wobei letztere per Knopfdruck einen Rollenwechsel initiieren können. Die Kommunikation unter den Teilnehmern erfolgt ebenso wie in PeP und XPairtise über einen integrierten Chat.

Den Screenshots zufolge geschieht die Installation per Dropin. Danach ist eine Konfiguration in Eclipse nötig, welche Java-Dateien mit dem sogenannten „Sangam Java Editor“ verknüpft. Das Anlegen der Accounts wurde durch Registrierung auf der Webseite durchgeführt. Dieses Projekt kam laut der Startseite nie über den Status einer Alphaversion hinaus, weswegen auch der dargestellte Funktionsumfang gering ist.

DocShare ist ein ein Plugin von den Eclipse-Entwicklern selbst. Der Dokumentation zufolge wurde es bis in Version 3.5 von Eclipse unterstützt, mit dem heutigen Stand ist die Anleitungsseite jedoch nicht mehr verfügbar, obwohl diese von der oben genannten URL verlinkt ist. Es war Teil des „Eclipse

Communication Framework Project“, kurz ECF (Webseite: http://wiki.eclipse.org/Eclipse_Communication_Framework_Project), allerdings lässt die aktuelle Featureliste darauf schließen, dass DocShare nicht mehr in dem ECF integriert ist.
Einer Aussage auf der Saros-Webseite zufolge war es mit DocShare nicht möglich, ganze Projektstrukturen in eine Zusammenarbeit einzubringen, sondern lediglich einzelne Dateien gemeinsam zu bearbeiten. Eine Internetrecherche bezüglich eines aktuellen Standes in Hinsicht auf Eclipse 3.7 war ergebnislos.

Im Rahmen der Eclipse-Plugins lässt sich festhalten, dass generell nur eine sehr begrenzte Auswahl vorhanden ist. Unter den gefundenen Projekten ist Saros das einzige, welches noch aktiv entwickelt wird. Ein Großteil der gefundenen Plugins war auch nach genauem Befolgen der jeweiligen Dokumentation bzw. Anleitung nicht lauffähig, hierbei bildeten PeP und Saros die Ausnahme. Mit der „Session Gallery“ liefert XPairtise jedoch einen interessanten Ansatz für eine erweiterte Verwaltung von Sessions.

Um die Beurteilung über die Konkurrenzfähigkeit auszuweiten, ist es sinnvoll, nicht nur Eclipse zu betrachten, sondern auch andere Entwicklungsumgebungen mit einzubeziehen. Einer Umfrage von CertPal zufolge sind „NetBeans“ und „IntelliJ IDEA“, welche von Sun bzw. JetBrains entwickelt werden, neben Eclipse die beliebtesten Programmierumgebungen (vgl. Abbildung 35 [CP009]).

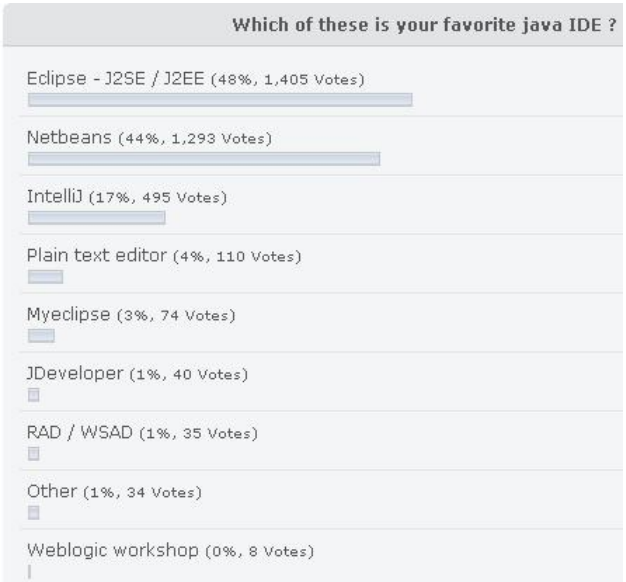

Abbildung 35: Umfrage zur Verwendung von Entwicklungsumgebungen aus dem Jahre 2009

Für NetBeans wurde das sogenannte „Developer Collaboration plugin“ (http://netbeans.org/kb/articles/quickstart-collaboration.html) entwickelt, welches wie Saros auf XMPP basierte und daher auch die Möglichkeit bot, Präsenzinformationen zu übertragen. User fanden in einer „Conversation“ zusammen, dem Äquivalent zu der Session in Saros. Jeder Teilnehmer war befugt, Dateien und Projekte in dieser Conversation einzubringen und sie somit allgemein editierbar zu machen.

Ab Version 6.7 wurde das Developer Collaboration plugin durch Suns Entwicklungsplattform Kenai ersetzt, welche dem etablierten Dienst SourceForge Konkurrenz machen sollte. Die verwendete Technologie wurde von XMPP in eine Cloud-Lösung geändert und um den Einsatz von „Maven“ erweitert, welches eine Verwaltung von Software-Builds ermöglicht. 'Kenai [enthält] neben den Versionskontrollsystemen Subversion und Mercurial gut integrierte Issue-Tracking-Software wie Bugzilla, JIRA, Continuous-Integration-Unterstützung mit Hudson, Foren, Wikis und einen in NetBeans integrierten Instant-Messaging-Client.' [Bie09] Diese Tools bieten Entwicklern bei ihren Umsetzungen viel Komfort und eine umfangreiche Palette an Unterstützungen. Durch die Speicherung im Internet sind die Projekte, welche zur gemeinsamen Bearbeitung freigegeben sind, jederzeit erreichbar - neue Teilnehmer werden per Einladung hinzugefügt.

IntelliJ IDEA (http://www.jetbrains.com/idea/) erfreut sich aufgrund zahlreicher integrierter Tools großer Beliebtheit. Dazu zählen die umfangreiche Syntax-Unterstützung für viele Programmiersprachen und die leicht bedienbaren Menüs zum Erstellen von grafischen Oberflächen genauso wie die sogenannten „Code Quality Features“, welche unter anderem die Quelltexte auf Code-Duplikate untersuchen und Abhängigkeiten zwischen Klassen darstellen können.

Im Rahmen dieser Entwicklungsumgebung gibt es das Feature „IDE Talk“, welches einen Chat zur Verfügung stellt. Darüber hinaus kann man sich Differenzen zwischen Projekten, die von mehreren Nutzern geöffnet sind, anzeigen lassen. Ein zeitgleiches Editieren ist hierbei jedoch nicht möglich, auch nach intensiver Internetrecherche wurde kein Weg gefunden, verteilte Paarprogrammierung direkt in IntelliJ zu betreiben. Einigen Useraussagen zufolge wird deswegen oft der Umweg über ein Screensharing-Tool in Kauf genommen.

Zusammenfassend lässt sich sagen, dass Saros den großen Vorteil genießt, sich aktuell in der Entwicklung zu befinden und einfach installierbar zu sein. In Bezug auf andere Eclipse-Plugins stellt Saros sehr viele Funktionalitäten zur Verfügung, welche andere Projekte nicht bieten können. XPairtise benutzt das Konzept einer „Session Gallery“, um die Zusammenarbeit zu

initiieren und hebt sich somit auch von anderen Plugins ab. Eine präzise Aussage kann hier aufgrund der Tatsache, dass ich XPairtise nicht selbst testen konnte, nicht gegeben werden, jedoch erweckt die Selbstpräsentation auf der Internetseite den Eindruck, dass dieses Projekt innovative Ideen umsetzt und somit Saros vom Funktionsumfang her in nichts nachsteht.
Der Internetrecherche zufolge ist eine verteilte Paarprogrammierung in der Entwicklungsumgebung IntelliJ IDEA derzeit nicht möglich. NetBeans hingegen überzeugt durch eine Vielzahl an Features und Services, welche den Fortschritt in einem gemeinsam bearbeiteten Projekt zusätzlich unterstützen.

13 Anhang

13.1 Quellcode

WorkspaceSourceProvider.java:

```
package de.fu_berlin.inf.dpp.serviceProviders;

import java.util.HashMap;
import java.util.Map;

import org.eclipse.core.expressions.Expression;
import org.eclipse.core.resources.IResourceChangeEvent
    ;
import org.eclipse.core.resources.
    IResourceChangeListener;
import org.eclipse.core.resources.IWorkspace;
import org.eclipse.core.resources.ResourcesPlugin;
import org.eclipse.ui.AbstractSourceProvider;
import org.eclipse.ui.ISources;

import de.fu_berlin.inf.dpp.SarosPluginContext;
import de.fu_berlin.inf.dpp.util.Utils;

/**
 * Adds variables to Eclipse's Core
   {@link Expression}s in order to be used in
 * plugin.xml.
 */
public class WorkspaceSourceProvider extends
    AbstractSourceProvider {

    /**
     * Corresponds to a serviceProvider variable as
         defined in Extension
     * org.eclipse.ui.services.
     */
    public static final String WORKSPACE = "de.
        fu_berlin.inf.dpp.Workspace";

    private IResourceChangeListener
        resourceChangeListener = new
        IResourceChangeListener() {
        @Override
        public void resourceChanged(
            IResourceChangeEvent event) {
            workspaceChanged();
        }
    };
```

```
    private IWorkspace workspace;

    public WorkspaceSourceProvider() {
        SarosPluginContext.initComponent(this);
        this.workspace = ResourcesPlugin.getWorkspace
            ();

        this.fireSourceChanged(ISources.WORKBENCH,
            WORKSPACE, this.workspace);
        this.workspace.addResourceChangeListener(
            resourceChangeListener);
    }

    public void dispose() {
        this.workspace.removeResourceChangeListener(
            resourceChangeListener);
    }

    public String[] getProvidedSourceNames() {
        return new String[] { WORKSPACE };
    }

    public Map<Object, Object> getCurrentState() {
        Map<Object, Object> map = new HashMap<Object,
            Object>(2);
        map.put(WORKSPACE, this.workspace);
        return map;
    }

    private final void workspaceChanged() {
        Utils.runSafeSWTAsync(null, new Runnable() {
            public void run() {
                fireSourceChanged(ISources.WORKBENCH,
                    WORKSPACE, workspace);
            }
        });
    }

}
```

WorkspacePropertyTester.java:

```
public class WorkspacePropertyTester extends
    PropertyTester {

    public boolean test(Object receiver, String
        property, Object[] args,
        Object expectedValue) {
        if (receiver instanceof IWorkspace) {
            IWorkspace workspace = (IWorkspace)
                receiver;
            if ("hasMultipleProjects".equals(property)
                ) {
                return workspace.getRoot().getProjects
                    ().length >= 2;
            }
        }
        return false;
    }

}
```

Abbildungsverzeichnis

1 Schema zur Darstellung des User-Zustandes und der daraus resultierenden Interaktionsmöglichkeiten 7
2 Zustandsübergänge, welche das Verbinden mit dem XMPP-Account und das Beitreten zu / das Erstellen einer Session darstellen 8
3 Fortsetzung von Abbildung 2: mögliche Zustandsübergänge für einen User innerhalb einer Session 9
4 Ein Buddy, der nicht connected ist, kann in eine Session eingeladen werden 11
5 Feedback an den User, der versucht hat, einen nicht verbundenen Buddy einzuladen 12
6 Irreführendes Feedback in den Kontextmenüs für multiples Sharing (oben) und multiple Buddy-Einladung (unten) 13
7 Obwohl kein Projekt im Workspace vorhanden ist, wird Multiples Sharing angeboten (oben). Analog dazu der Einladungsprozess (unten) 13
8 Das Herstellen einer Skype-Verbindung mit einem Session-Teilnehmer ist nicht möglich. 14
9 PeP-Menü in der Eclipse Menüleiste 16
10 Übertragungs-Funktion in PeP 17
11 Screenshot des Chats in PeP 17
12 Erstellung eines Dokumentes in google docs 18
13 Während ein Bereich bearbeitet wird, sperrt google docs diesen für alle anderen User 18
14 Dialog zur Teilnahme an einer gemeinsamen Bearbeitung einer Datei in MoonEdit 19
15 Ein Teil der Individualisierungs-Möglichkeiten in MoonEdit . 20
16 Maus-Cursor der anderen Teilnehmer werden in grau dargestellt 20
17 Die Einladung in MeetingWords wird durch Verteilung eines einmaligen Links realisiert 21
18 Das Optionsmenü von MeetingWords - geöffnet ist die Revisionsübersicht 21
19 Der „share pod“ von Adobe Connect, über welchen die Präsentationen dargestellt werden 23
20 User-Statusänderungen geben den Präsentierenden Feedback 23
21 „Breakout Rooms“ dienen der Arbeit in Gruppen 23
22 Die Nutzeroberfläche von SubEthaEdit 24
23 Aktionsmöglichkeiten des Hosts in Bezug auf einzelne Teilnehmer in SubEthaEdit 25
24 Überblick über die Saros-eigenen Menüs 38
25 Vergleich vorher(links)-nachher(rechts): Der Skype-Menüpunkt ist nun auch in einer Session verfügbar 38

26 Menüeinträge des Projekt-Sharens in den Zuständen „connected“ (links) und „not connected“ (rechts) 39
27 Grafische Darstellung der plugin.xml, markiert sind die neu hinzugefügten Überprüfungen 40
28 Das Feedback „not connected“ gilt nun auch für die „Multiple“-Funktionen (Beispiel multiple Projektauswahl) 40
29 Darstellung der Definition „workspaceHasMultipleProjects“ in der plugin.xml 42
30 Der betroffenen Abschnitt der plugin.xml nach den vorgenommenen Änderungen 43
31 Multiples Sharing ist nun nur noch mit mehr als zwei vorhandenen Projekten möglich 43
32 Beispiel des Einsatzes eines Wizards, um einen Buddy hinzuzufügen 44
33 Der Wizard zur Erstellung eines neuen XMPP-Accounts . . . 45
34 Beitreten einer Session in XPairtise 50
35 Umfrage zur Verwendung von Entwicklungsumgebungen aus dem Jahre 2009 51

Literatur

[Art03] John Arthorne. Drag and Drop in the Eclipse UI, 2003.

[Bat11] Batipi. http://www.batipi.com/adobe-connect-pro/, 2011.

[Bie09] Adam Bien. NetBeans 6.7 setzt Fokus auf Maven- und Kenai-Integration. http://www.heise.de/developer/meldung/NetBeans-6-7-setzt-Fokus-auf-Maven-und-Kenai-Integration-187367.html, 2009.

[CM09] Christian Crumlish and Erin Malone. Designing Social Interfaces, 2009.

[CP009] http://www.certpal.com/blogs/2009/08/your-favorite-java-ide/, 2009.

[CW00] Alistair Cockburn and Laurie Williams. The Costs and Benefits of Pair Programming, 2000.

[Dje06] R. Djemili. Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierung verteilter Paarprogrammierung. Master's thesis, Freie Universität Berlin, Inst. für Informatik, 2006.

[Ger11] Dr. Martin Gersch. Ökonomische Besonderheiten des E-Business. https://lms.fu-berlin.de/@@/A5E1E30EBE6B3196BEB27A3C71B11F02/courses/1/WIWISS_Pjs_109020_11S/content/_1337768_1/Plenum_ABV_E-Business_SS11-3.pdf, 2011.

[GG04] Carl Gutwin and Saul Greenberg. The Importance of Awareness for Team Cognition in Distributed Collaboration, 2004.

[Jet06] Hans-Christian Jetter. Die MCI im Wandel: User Experience als die zentrale Herausforderung?, 2006.

[Joh11] Meike Johannsen. Verbesserung und Pflege der Dokumentation der DPP-Software Saros. Master's thesis, Freie Universität Berlin, Inst. für Informatik, 2011.

[Jor08] Tobias Jordans. Donald Norman definiert User Experience, 2008.

[Kah11] Björn Kahlert. Verbesserung der Out-Of-Box-Experience in Saros mittels Heuristischer Evaluation und Usability-Tests. Master's thesis, Freie Universität Berlin, Inst. für Informatik, 2011.

[Nie06] Pin Nie. An open standard for instant messaging: eXtensible Messaging and Presence Protocol (XMPP), 2006.

[oP11] University of Pennsylvania. http://www.sas.upenn.edu/computing/image/view/1465/_original, Stand 2011.

[PW11] Martin Pittenauer and Dominik Wagner. http://www.codingmonkeys.de/subethaedit/, Stand 2011.

[use11] userfriend. http://www.userfriend.de/usability/was-ist-usability, 2011.

[Wal05] Tilman Walther. Architektur und Konzepte von Eclipse 3. http://www.tilman.de/uni/Eclipse3.pdf, 2004/05.

[XP008] http://xpairtise.sourceforge.net/gettingstarted/user_join.html, Stand 2008.