Bachelor-Arbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Software Engineering

Design und Implementierung der neuen Saros Benutzeroberfläche

Patrick Bitterling
Matrikelnummer: 4214170
patbit@zedat.fu-berlin.de

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

Berlin, 02.02.2011

Zusammenfassung

Diese Arbeit befasst sich mit der Optimierung der Benutzeroberfläche von Saros. Saros ist ein Eclipse-Plugin, das es ermöglicht verteiltes Party Programmierung zu betreiben. Diese Bachelorarbeit beschreibt wie anhand von Design Prinzipien und Design Mustern, die Benutzeroberfläche verbessert wurde.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

02.02.2011

Patrick Bitterling

Inhaltsverzeichnis

1 Einleitung

1.1 Aufbau dieser Arbeit

Diese Abschlussarbeit ist in 8 Kapitel aufgeteilt. In Kapitel 2 und 3 werden die Grundlagen geklärt, die benötigt werden um zu verstehen womit diese Abschlussarbeit beschäftigt. Kapitel 4 gibt einen Überblick, über die ursprüngliche Saros Benutzeroberfläche. Die Kapitel 5 und 6 behandeln meine ersten Designs. Im 7. Kapitel wird Implementierung der neuen Saros-Benutzeroberfläche beschrieben. Im letzten Kapitel werden die Aufgaben eines Mitglieds der Arbeitsgruppe Software Engineering behandelt.

1.2 Notation

Alle Wörter die in *kusiv* geschrieben sind werden im Kapitel 2 Grundlagen erklärt.

Alle Wörter in `serifenloser, nicht proportionaler` Schrift sind Wörter aus dem Quellcode.

Andere relevante Fachwörter werden mit **dick gedruckter** Schrift kenntlich gemacht.

2 Grundlagen

2.1 Was ist Saros ?

Saros ist Eclipse-Plugin, das 2006 von Riad Djemili [Dje06] als Bestandteil seiner Diplomarbeit entwickelt wurde. Saros erweiterte Eclipse um die Fähigkeit der verteilten Paar-/Partyprogrammierung. Das PlugIn wurde im Laufe der Jahre durch weitere wissenschaftliche Arbeiten und den Mitglieder der AG Software Engineering.

2.2 Was ist Paarprogrammierung ?

Die Paarprogrammierung ist ein Vorgehen aus der agilen Softwareentwicklung. Bei der Paarprogrammierung sitzen zwei Softwareentwickler vor einem Computer. Dabei nehmen sie unterschiedliche Rollen an. Ein Benutzer übernimmt die Rolle des **Drivers**. Der Driver ist die Person, die die Tastatur benutzt und den Quellcode schreibt und den **Observer** darüber informiert was er grade tut. Die andere Person nimmt die Rolle des **Observers** bzw. des **Navigators** an. Der Observer ist ein Beobachter und weist auf Fehler hin die der Driver begeht. Wenn Probleme auftreten stimmen sich Driver und Observer über ein weiteres Vorgehen ab. Nach einem gewissen Zeitabstand tauschen Driver und Observer ihre Rollen.

2.3 Eigenschaften der Paarprogrammierung

Es gibt viele wissenschaftliche Arbeiten, zu den Thema Paarprogrammierung. Die wissenschaftliche Untersuchung [AC00] betrachtet dabei acht Punkte.

Ökonomie

> Die Programmierer brauchen 15% mehr Zeit, aber begehen dabei 50% weniger Fehler,wodurch die Kosten insgesamt eingespart werden.

Zufriedenheit

> Programmierer die noch nicht Paarprogrammierung gewöhnt sind brauchen erst eine Eingewöhnungszeit. Ein Großteil findet nach dieser Zeit das Zusammenarbeiten besser, als einzeln zu Programmieren.

Design Qualität

> Die zwei Programmierer können verschiedene Lösungen finden und die beste davon wählen. Hinzu kommt, dass die Anzahl der geschriebenen Quellcodezeilen abnimmt.

Kontinuierliche Codedurchsichten

Die meisten Programmierer machen Codedurchsichten nicht gerne und führen diese nur durch wenn sie angeordnet werden. Die Inspekteure kennen sich mit den Quellcode aber nicht aus. Deshalb soll die kontinuierliche Codedurchsicht effektiver sein. Es findet eine Zeitersparnis statt, weil Defekte gefunden werden bevor der Quellcode kompiliert wurde. Ein weiterer Vorteil ist, dass Programmiervorschriften stärker eingehalten werden.

Problembewältigung

Auftretende Probleme werden beim Paarprogrammieren schneller gelöst als von einem einzigen Programmierer.

Bildung

Durch das kontinuierliche Zusammenarbeiten wird ständig vorhandenes Wissen weitergegeben.

Teamentwicklung und Kommunikation

Durch die Paarprogrammierung wird die Zusammenarbeit des Teams gefördert. Die Kommunikation zwischen den Teammitgliedern fällt dadurch leichter.

Belegschaft und Projekt Management

Das Projektmanagement profitiert von der ständigen Wissensweitergabe und reduziert die Wahrscheinlichkeit Programmierer in Schlüsselpositionen zu verlieren, weil sich mehrere Programmierer mit den Quellcodeabschnitte des anderen auskennen.

In der Studie wird auch angemerkt, dass es einen kleinen aber beständigen Prozentsatz von Programmierern gibt, die keine Paarprogrammierung betreiben können.
Nach Angabe von [Wik11] die ihr Wissen aus [JEHS09] bezieht, kann Software die durch Paarprogrammierung geschrieben wurde nicht alle Aspekte von günstiger, besser und schneller erfüllen.

2.4 Was ist verteile Paarprogrammierung ?

Die verteilte Paarprogrammierung hat die selben Regeln wie die Paarprogrammierung. Die Programmierer sitzen jedoch nicht mehr vor dem selben Computer, sondern betrachten den Quellcode ortsunabhängig vor einem eigenen Computer.

2.5 Eigenschaften verteilter Paarprogrammierung

Nach [TS09] würde die Anwendung der agilen Softwareprozesse stark eingeschränkt werden, wenn alle Teammitglieder sich in örtlicher Nähe befinden müsste. Es wäre nicht möglich eine globale Softwareentwicklung zu betreiben, weil die Kosten und Zeitverlust durch Reisen ökonomisch nicht tragbar sind. Hinzu kommt die Möglichkeit, dass ein Observer die Möglichkeit hat noch eigene Aktivitäten durchzuführen wie z.B. Recherchen über APIs.

Durch das verteilte Arbeiten ist es erforderlich zusätzlichen technischen Aufwand zu leisten, damit sich Programmierer auf einander abstimmen können. Das betrifft nach [TS09] die Bereiche Koordination, Programmierung, Kommunikation, Lehre und Prüfung von Quellcode.

2.5.1 Koordination

Aktivität-Indikatoren

> Die Benutzeroberfläche muss den Benutzern ermöglichen zu sehen was der Andere tut.

Geteilte Dateiablage

> Die geteilte Dateiablage dient zu Austausch der aktuellen bearbeiteten Dateien.

2.5.2 Programmierung

Geteilter Dateieditor

> Es ist nötig einen Dateieditor zu haben, in dem beide Benutzer gleichzeitig Arbeiten können.

Kollaborative Sitzung

> Es muss eine Modellierung einer Session geben, die anzeigt wer in einer Session ist, beitritt, verlässt oder beendet.

Zugangsberechtigungen

> Einige gleichzeitig ausgeführte Interaktionen können zu Konflikten führen und die Benutzer irritieren. Deshalb ist es dann nötig Zugangsberechtigungen in diesem Bereich zu verteilen und eine Möglichkeit diese Rechte weiterzugeben.

2.5.3 Kommunikation

Eingebetteter Chat

> Der eingebettete Chat soll es ermöglichen schnell Informationen zwischen den Benutzern auszutauschen.

Fern-Selektieren

> Es muss den Benutzer in auf der anderen Seite ermöglicht werden Artefakte auszuwählen die der andere Benutzer vor seinem Bildschirm erkennen kann. Dadurch kann Interesse an speziellen Artefakten leicht vermittelt werden.

2.5.4 Lehre

Mentor

> Es sollte ermöglicht werden das Neuankömmlinge mit erfahrenden Benutzern zusammenarbeiten können.

Zuschauer

> Benutzer die noch wenig Erfahrung mit der Umgebung haben, sollten in einem Modus arbeiten in dem sie die nur Vorgänge in einer Sitzung beobachten können. Dadurch stören diese Benutzer nicht die Interaktionen der anderen Teammitglieder.

Wiedergabe

> Wenn ein Benutzer erst später in die Sitzung eintritt sollte es ihm ermöglicht werden, die bereits durchgeführten Änderungen sehen zu können.

2.5.5 Testen von Quellcode

Dezentrale Befehle

> Befehle die auf den lokalen Computer ausgeführt werden, sollen zum entfernten Computer gesendet und angewandt werden können. Dadurch wird es zum Beispiel ermöglicht ein Programm gemeinsam bis zum nächsten Stopppunkt laufen zu lassen.

2.6 Was ist Party Programming ?

Das Party Programming ist eine Erweiterung der Paarprogrammierung, die die Anzahl der Programmierer von zwei auf unbestimmte Anzahl erhöht. Jeder dieser Benutzer hat dann die Rolle eines Observers oder eines Drivers.

2.7 Wie unterstützt Saros DPP ?

Saros erweitert die Entwicklungsumgebung Eclipse an vielen Stellen um DPP zu ermöglichen. Saros unterstützt den Benutzer durch Annotationen und

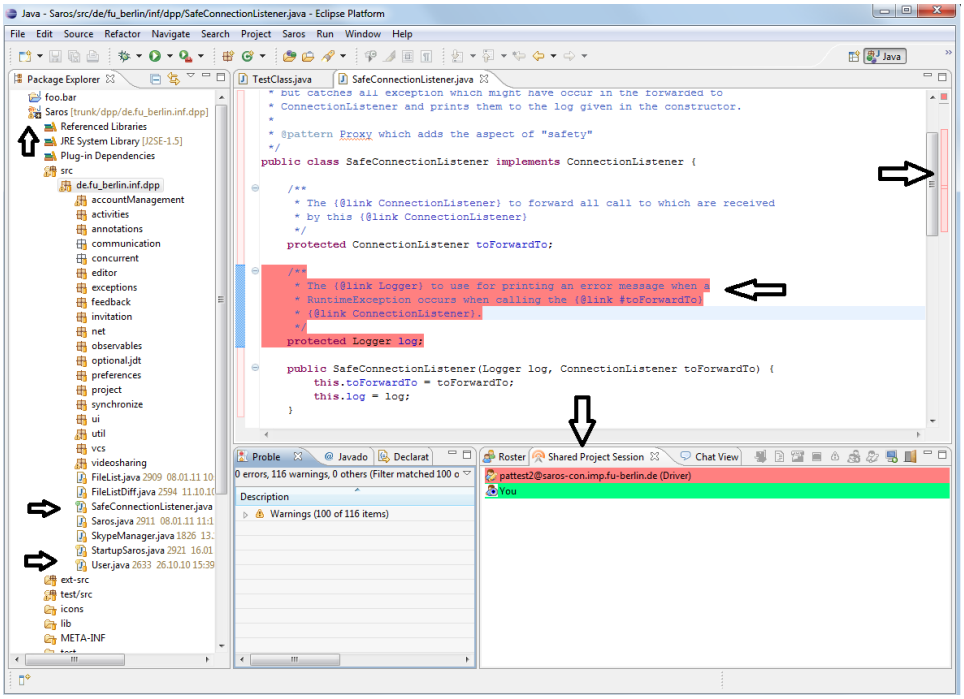

Abbildung 1: Eclipse mit Saros-PlugIn

Views die nützliche Informationen anzeigen.

Annotationen:

> Saros zeigt im Package Explorer an welche Projekte zu einer Saros Session gehören und welche Dateien geöffnet und beschrieben werden. Im Editor zeigt Saros an welche Passage von welchen Benutzer geschrieben wird und welchen Abschnitt sich ein Observer ansieht.

Views:

> Die Views geben Aufschluss darüber wer welche Kontakte online sind , wer sich in der Saros Session befindet und was im Chat geschrieben wird. Hinzu kommen noch weitere Views für Desktop Sharing und das Whiteboard.

2.8 SWT

SWT steht für Standard Widget Toolkit und ist eine Bibliothek zur Erstellung von Benutzeroberflächen in Java. SWT wurde im Jahre 2001 von IBM erstellt und wird derzeitig von der Eclipse Foundation weiterentwickelt. SWT wurde als eine Alternative zu AWT[1] entwickelt. SWT soll es ermöglichen soll nativ[2] wirkende Benutzeroberflächen für Applikationen zu erstellen und dabei eine möglichst hohe Leistung zu erzielen. SWT soll eine höhere Leistung als AWT leisten, weil SWT Ressourcen des Betriebssystems nutzt anstatt wie AWT eine Emulationsschicht zu benutzen. Die Benutzeroberfläche von Eclipse ist mit SWT geschrieben worden.

SWT beruht hauptsächlich auf fünf Arten von Komponenten, (Compsites, Controls, Layout und Events/Listeners) um Benutzeroberflächen zu gestalten.

Controls:

> Controls sind die kleinste Einheit in SWT. Zu den Controls gehören Buttons, Slider, Sash, ProgressBars, Scale, Spinner, Text (bzw. Label). Diese Controls werden als Inhalt in größeren Einheiten untergebracht den *Composites*.

Composites:

> Composites dienen unterschiedlichen Zwecken, zu einem können in ihnen Controls und andere Composites eingebettet werden. Andere Composites haben speziellere Aufgaben, wie z.B. die Canvas[3], die zum Zeichen dient, die Toolbar oder der *CTabFolders*. Hinzu kommen noch Tabellen, Bäume und Listen. Composites sind in einer Baumstruktur von einander abhängig. Composites können als Inhalt weitere Composites und/oder Controls haben.

Layouts:

> Jedes Composite besitzt ein Layout. Das Layout legt fest wie die Komponenten die zum Composite gehören angeordnet werden und wie viel Platz sie einnehmen.

Events/Listener:

[1] Abstract Window Toolkit eine Biliothek von Java zur Erstellung von Benutzeroberfläche

[2] als ob sie zum Betriebssystem gehören

[3] zu deutsch Leinwand

Für viele Interaktionen, die der Benutzer mit Benutzeroberfläche tätigt werden Events[4] ausgelöst, wie zum Beispiel Mausklicks oder die Selektion von Komponenten. Mit einem Listener[5] kann man auf diese Events horchen und reagieren und so zum Beispiel bei einem Rechtsklick ein Menü erscheinen lassen.

2.9 Benutzte SWT Komponenten

Die folgenden SWT Komponenten wurden während Erstellung der Designs verwendet.

2.9.1 Composites

CTabFolder:

Der CTabFolder ist in SWT eine Metapher für ein Notizbuch. In jeden Tab (Karteireiter) können eigene Composites untergebracht werden. Dadurch können auf dem selben physikalischen Raum unterschiedliche Inhalte anzeigen. Eines der bekanntesten Beispiele sind Internetbrowser, die in jedem Tab einen unterschiedliche Internetseite anzeigen kann.

ExpandBar:

Die ExpandBar kann ein Composite untergebracht werden. Der Inhalt des untergebrachten Composites kann per Mausklick versteckt oder wieder angezeigt werden.

Group:

Die Group gruppiert den Inhalt, indem es einen Rand um ihren Inhalt zieht und optional einen Titel anzeigen kann.

Sashform:

Eine Group gruppiert den Inhalt, indem es einen Rand um ihren Inhalt zieht und optional einen Titel anzeigen kann.

ScrolledComposite:

Das ScrolledComposite funktioniert ähnlich einem normalen Composite. Nimmt der Inhalt des Composites mehr Platz ein als zur Verfügung steht. Blendet dieses Composite eine Scrollbar (Rollbalken) ein, durch die der restliche Inhalt erreicht werden kann.

[4] zu deutsch Ereignisse
[5] zu deutsch Zuhörer

2.9.2 Layouts

FillLayouts:

> Das FillLayout ordnet den Inhalt nebeneinander oder untereinander an. Jedes Element hat dabei die gleiche Größe.

GridLayout:

> Das GridLayout ermöglicht es den Inhalt genau zu platzieren. Dazu wird zu jedem Element ein GridData zugeordnet. Das GridData bestimmt dann die Postion und Größe des Elements.

StackLayout:

> Dieses Layout ordnet den Inhalt schichtweise an. Alle Schichten nehmen gleich viel Platz ein und Größe die das Layout annimmt ist die der größten Schicht. Der Anwender kann nur die oberste Schicht des StackLayouts sehen.

TableColumnLayout:

> Das TableColumnLayout ermöglicht es die Größe jeder Spalte einer Tabelle selbst festzulegen.

eigene Layouts:

> Einige Composites wie z.B. die Sashform, Group oder CTabFolder besitzen ihr eigenes Layout, welches den Inhalt entsprechend dem Composite automatsch anordnet.

2.10 JFace

JFace erweitert SWT um "Model View Controller". JFace beinhaltet viele Helferklassen um Inhalte von Listen, Tabellen und Bäumen zu erstellen, sortieren und filtern. JFace definiert außerdem standardisierte Wizards und Dialoge.

3 Die Benutzbarkeit von Software

Die Konstruktion einer neuen Benutzeroberfläche ist es sinnvoll, sich zuerst mit Benutzbarkeit von Software zu beschäftigen um eine möglichst benutzerfreundliche Benutzeroberfläche zu gestalten. Viele Softwareingenieure haben sich bereits an dieses Thema gewagt und ein gewisse Anzahl von Design-Prinzipien aufgestellt, die es ermöglichen sollen gute Benutzeroberflächen zu entwickeln.

3.1 Die Kriterien für die Benutzbarkeit von Software

Das Buch [AD95] wählt einen Ansatz, durch den diese Prinzipien in eine der drei folgenden Kategorien eingeordnet werden:

Die *Erlernbarkeit* gibt an, wie schnell ein Anfänger sich beibringen kann, die Software effektive zu verwenden.

Die *Flexibilität* gibt an, auf wie viele Möglichkeiten ein Benutzer zurückgreifen kann, um mit der Software Informationen auszutauschen.

Die *Robustheit* gibt an, wie stark die Software den Benutzer dabei unterstützt, dass der Benutzer durch seine Interaktionen sein Ziel erreicht auch wenn er dabei Fehler begeht.

3.2 Prinzipien für die Erlernbarkeit von Software

Berechenbarkeit ermöglicht es, dass der Benutzer vorherzusagen kann welches Ergebnis zu erwarten ist, wenn er versucht mit der Software zu interagieren.

Synthetisierbarkeit ermöglicht es, dass der Benutzer den Zustand[6] der Software nach einer Aktion einschätzen kann.

Vertrautheit bewirkt, dass das Wissen was Benutzer bereits während der Benutzung mit Software erlangt hat, wiederverwenden kann um mit dieser Software zurecht zukommen.

Verallgemeinerung ermöglicht es dem Benutzer, sein Wissen aus einer Interaktion auf ähnliche Interaktionen wieder anwenden zu können.

Konsistenz in Software bedeutet, dass bei ähnlichen Aufgaben die Software ähnliche Eingaben verlangt bzw. eine ähnliche Ausgabe leistet.

3.3 Prinzipien für die Flexibilität von Software

Dialog-Initiative gibt an wer bei Interaktionen die Kontrolle hat. Es wird versucht das der Mensch möglichst viel Kontrolle hat und der Computer nur selten die Kontrolle übernehmen soll. Deshalb tauchen häufig Dialoge auf die den Benutzer fragen, ob die Software bestimmte Aufgaben ausführen darf.

[6] z.B. ob die Software noch arbeitet oder fertig ist und ob Fehler aufgetreten sind

Multi-Treading ermöglicht es mehrere Aufgaben nebenläufig auszuführen.

Task-Migration ist Möglichkeit eine Aufgabe des Benutzers an einem anderen Teil des Systems übertragen. (z.B die Rechtschreibüberprüfung in Dokumenten)
Substitution ermöglicht es äquivalente Werte gegeneinander auszutauschen. (Darunter fällt auch die Möglichkeit, unterschiedliche Ausgabearten bei der selben Eingabe zu erzeugen)
Anpassung ermöglicht es dem Benutzer, die Benutzeroberfläche umzugestalten und sich für Arbeitsabläufe eigene Makros zu erstellen.

3.4 Prinzipien für die Robustheit von Software

Beobachtung beschreibt den Vorgang, dass der Benutzer den internen Status der Software anhand wahrnehmbaren Darstellung zu erkennen. Unterstützt wird der Benutzer dabei durch z.B. Ausgaben auf Benutzeroberfläche oder Tonsignalen.

Wiederherstellbarkeit ermöglicht den Benutzer begangene Fehler wieder zu korrigieren. Ein Beispiel dafür sind Texteingaben, die meistens die Möglichkeit bieten getätigten eingaben wieder rückgängig zu machen. Software die Einstellungsoptionen haben, besitzen einen Button um alle Einstellungen auf ihre Standartwerte zurückzusetzen.

Antwortverhalten von Software unterteilt sich in 2 Aspekte:

1. die absolute Antwortzeit: Die Zeit die benötigt wird um einen Benutzer die Statusänderung anzuzeigen. Alle Aktionen deren Auswirkungen nicht sofort sichtbar sind, müssen den Benutzer signalisieren, dass sie grade aktiv am Arbeiten sind

2. stabile Antwortzeit: für ähnliche Aktionen müssen ähnliche Antwortzeiten auftreten (z.B müssen alle Pulldown-Menüs immer eine sofortige Antwortzeit liefern)

Task-Konformität sorgt dafür, dass der Benutzer alle Aufgaben die er durchführen will auch durchführen kann und zwar auf die Art und Weise wie der Benutzer es möchte.

4 Die ursprüngliche Saros Benutzeroberfläche

Als ich mit meiner Arbeit an der neuen Benutzeroberfläche begann gab es vier Views. Diese vier Views waren der Roster View, der Saros Session View, der Chat View und der Remote Screen View. Der Remote Screen View braucht je nach Benutzung sehr viel Platz und deshalb ist die Integrierbarkeit in einen anderen View stark eingeschränkt. Mein Idee war es daher die anderen drei Views in einem neuen zu integrieren.

4.1 Probleme der Benutzeroberfläche

Viele Interaktionen können über die vier bestehenden Views gesteuert werden. Diese Benutzeroberfläche ist sehr modular aufgebaut. Das bedeutet, dass für jeden Bereich in dem Informationen angezeigt werden sollen ein eigener View erstellt wurde. Der Vorteil ist, das die gewünschten Features vom Benutzer selbst angeordnet werden können bzw. komplett weggeschaltet werden können. Diese Modularität bringt aber auch Nachteile mit sich. Einige Features wie zum Beispiel der Roster und Saros Session View verbrauchen mehr Platz als nötig. Das liegt daran das alle Views von Saros nebeneinander in Karteireitern angeordnet sind und z.B. der Roster View viel weniger Platz in der Breite verbraucht als der Chat View. Außerdem fällt es dem Benutzer schwer mehrere Informationen auf einmal im Blick zu behalten, weil der Benutzer nur den Inhalt eines Karteireiters sehen kann. Saros besteht derzeit aus jeweils einem View für die Kontaktliste (auch Roster genannt), das Anzeigen der Mitglieder in einer Saros Session und einen für den Gruppenchat, der bei jeder Saros Session angelegt wird. Hinzu kommen noch der Desktop Sharing View und das Whiteboard. Während sich der Anwender in einer Saros Session befindet müsste der Anwender seine Arbeit unterbrechen und die Views wechseln wenn er eine dieser Fragen hat:

- Wer ist grade in meiner Buddy-List online?
- Wer ist in der Session?
- Was wird im Chat besprochen ?

Die derzeitige Benutzeroberfläche hat noch einige weitere Mängel. Einige der Icons beinhalten unnötige Symbole, die den Benutzer irritieren könnten oder es unkenntlich machen wofür dieses Icon dient. Hinzu kommt, dass Saros teilweise dem Benutzer zu wenig Feedback gibt. So fehlen zum Beispiel Töne, die der Benutzer aus Instant Messangern kennt (Kontakte kommen online oder gehen offline, der Benutzer bekommt oder sendet Nachrichten). Töne haben den Vorteil, dass sie schneller vom Benutzer verarbeitet werden können und weniger ablenken als visuelle Anzeigen [AD95].

Die alte Aufteilung ist auch nicht geeignet für einige in der Zukunft geplanten Features. Saros soll es ermöglichen, dass sich Anwender in mehreren Gruppenchat befinden können. Auch der direkte Chat mit anderen Benutzern soll eingebaut werden. Die ursprüngliche Oberfläche würde es aber keinem View gestatten, diese Funktionalität visuell dazustellen..

Im folgenden Abschnitt werden die Views, die ich umbauen werde näher erläutert.

4.2 Der Roster View

Saros benutzt XMPP (früher genannt Jabber) um Daten zwischen den Benutzern zu übertragen. XMPP steht für Extensible Messaging and Presence Protocol und ist ein auf XML[7] basierendes Protokoll, welches hauptsächlich für Instant Messanger benutzt wird. Dafür wird ein XMPP-Account benötigt. Gleichzeitig bietet der Account die Möglichkeit den Namen von Benutzern zu speichern und ihre Präsens anzugeben. Die abgespeicherten Benutzer können dann noch in Gruppen organisiert werden, wobei die Standard Gruppe die Bezeichnung Buddies hat. Der Roster besteht aus fünf

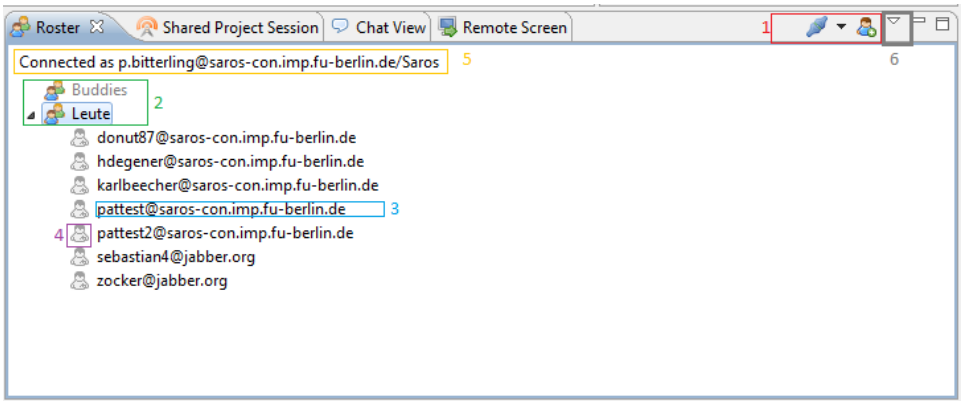

Abbildung 2: Der Roster View von Saros

Komponenten

1. Die Toolbar besteht aus 2 Buttons. Der linke Button dient zur Accountauswahl und zum verbinden bzw. trennen der XMPP-Verbindung. Der Rechte Button wird benutzt um weitere Kontakte hinzuzufügen.

2. Das sind die Gruppen in der Kontaktliste, sie können aufgeklappt werden um Kontakte anzuzeigen die zu dieser Gruppe gehören.

3. Das ist ein Kontakt (der im Bild eingerahmte gehört zur Gruppe Leute). Jeder Kontakt besitzt ebenfalls einen einen XMPP-Account. Das

[7] steht für extensible markup language und ist eine generischen Auszeichnungssprache

Icon nehmen den Namen stellt die Präsenz des Kontaktes dar (und ob der Kontakt Sarosunterstützung hat).

4. Das Bild nehmen den Namen stellt die Präsenz des Kontaktes dar (und ob der Kontakt Sarosunterstützung hat)
5. Der Text zeigt an, ob und welchen Account man verbunden ist
6. Das Menü des Roster Views besitzt nur, einen Eintrag um weitere Benutzer in eine Saros Session einladen zu können

Hinzu kommt ein Kontextmenü, das mit einem Rechtsklick auf dem Kontakt folgende Aktionen mit ihm ermöglicht: per Skype telefonieren, zur Session einladen, umbenennen, löschen, Verbindung testen.

4.3 Der Saros Session View

Der Saros Session View zeigt die Benutzer an, die sich in der aktuellen Session befinden mit der dazugehörigen Farbe. Die Farbe des Benutzers erscheint im Chat und im Editor, wo der Quellcode geschrieben wird. Der Saros Ses-

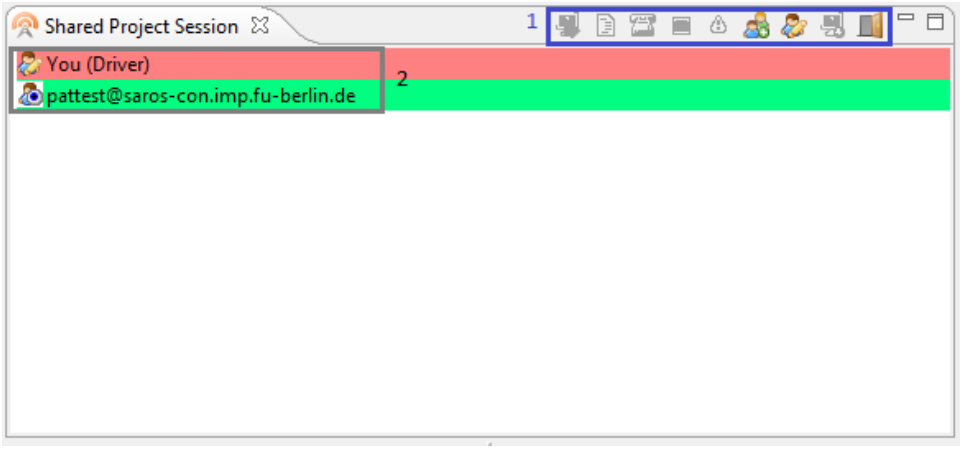

Abbildung 3: Der Saros Session View

sion View besteht aus zwei Hauptkomponenten:

1. Die Toolbar mit Aktionen zum Dateiübertragen, VoIP, Share Desktop, weitere Benutzer einladen, Rechteverteilung und zum Stoppen und Inkonsistenzen prüfen der Session.
2. Die Benutzer der Session sind in einer Tabelle angeordnet. Eine Reihe besteht aus einem Symbol, das anzeigt welche Rechte der Benutzer hat und seinen Namen. Ist der Benutzer ein Driver steht dies extra hinter dem Benutzernamen. Folgt man einem anderen Benutzer wird dieser fettgedruckt dargestellt.

Das Kontextmenü stellt auch Aktionen für die Rechteverwaltung bereit, sowie Möglichkeiten um zum Benutzer zu springen oder ihn zu verfolgen und um die Farbe des Benutzers einzustellen.

4.4 Der Chat View

Der Chat View dient zum Chatten mit Benutzern die in der Session sind. Der Chat View besteht aus drei Komponenten:

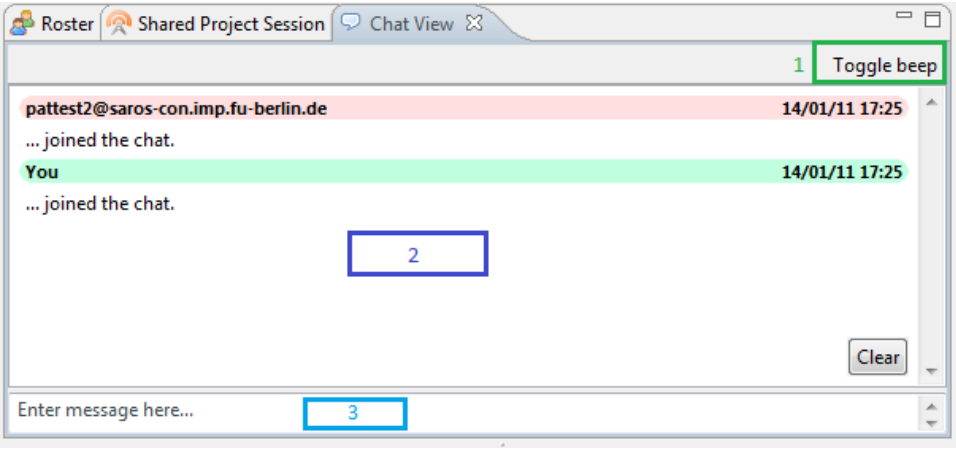

Abbildung 4: Der Chat View von Saros

1. Der Toolbar: Sie besteht aus einen Button, um den Piep-Ton an-/auszuschalten den man erhält, wenn man eine Nachricht erhält.
2. Chat Display: Hier werden Nachrichten angezeigt, die geschrieben wurden.
3. Chat Input: Das ist der Bereich, wo Nachrichten die abgeschickt werden sollen, eingegeben werden können.

5 Die ersten neuen Designentwürfe

Nachdem das letzte Kapitel die Probleme und Bestandteile der ursprünglichen Benutzeroberfläche behandelt hat, geht es in diesem Kapitel um die Erstellung der ersten Designentwürfe.

5.1 Das neue Grunddesign

Die ersten Überlegungen beim neuen Design betreffen das Grunddesign des neuen Views. Der Roster View und der Chat View zusammen ergeben zusammen die Komponenten eines Instant Messenger. Die meisten Instant

Messenger haben den Chat und Roster in zwei eigenen Fenstern. Eine Ausnahme stellt die Chat-Funktionalität der VoIP-Software Skype unter Windows dar..

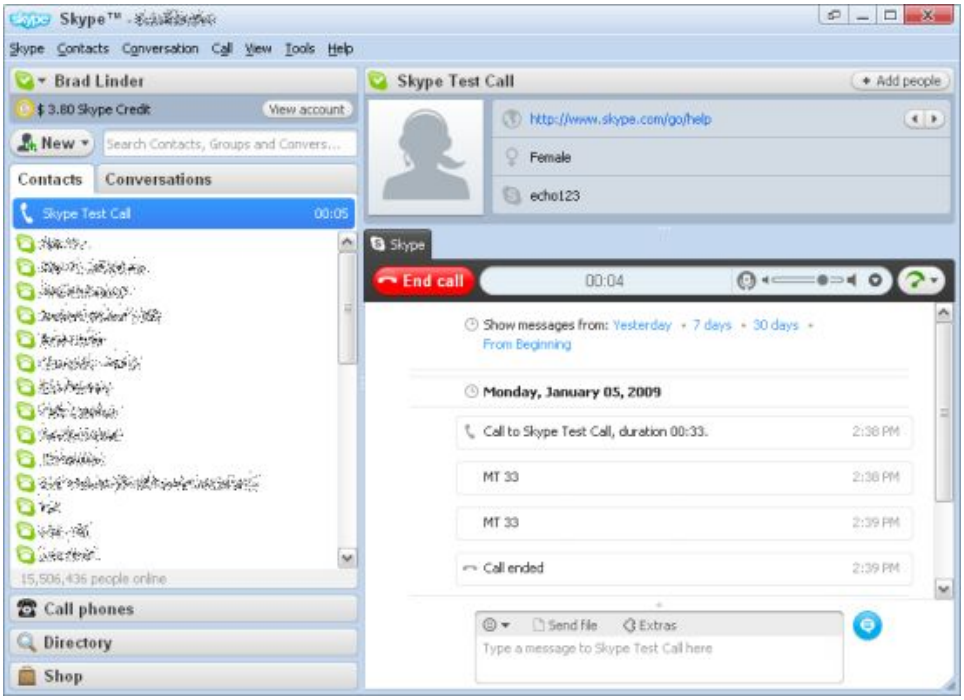

Abbildung 5: Die Oberfläche von Skype

Skype ordnet den Chat nach rechts und die Skype-Kontakte nach links. Da diese Software weit verbreitet ist, haben ich beschlossen das Design der Benutzeroberfläche ähnlich zu gestalten, weil viele Benutzer dann eine gewisse Vertrautheit empfinden werden. Der Saros Session View gehört mit zur linken Seite bei dieser Designüberlegung. Das liegt an der engen Verwandtschaft mit dem Roster und der Tatsache, dass derzeitig mit den Kontakten aus dem Session Session gechattet wird. Die Kontakte in der Session haben für den Benutzer daher auch eine höhere Relevanz als die aus dem Roster. Da alle Kontakte der Saros Session aus dem Roster kommen lohnt es sich, den Roster und Kontakte aus der Saros Session in einer Komponente zu integrieren. Das führt auch dazu, dass einige Kontakte nicht mehr unnötig doppelt (in der Kontaktliste und der Saros Session) angezeigt werden müssen. Die Zusammenlegung ist möglich, weil die Elemente aus Saros Session eine andere (nicht weiße) Hintergrundfarbe und andere Icons haben, die sie von Kontaktlisteneinträgen klar unterscheiden.

5.2 Die Erstellung der Designs mit WindowBuilder Pro

Um möglichst schnell neue Designs vorlegen zu können, habe ich nach einer Software gesucht mit der man schnell Benutzeroberflächen designen kann. Es hat sich Angeboten den **WindowBuilder Pro** von Google zu wählen. Mit diesen Eclipse-Plugin kann man per **Drag&Drop** SWT - Anwendungen bauen.

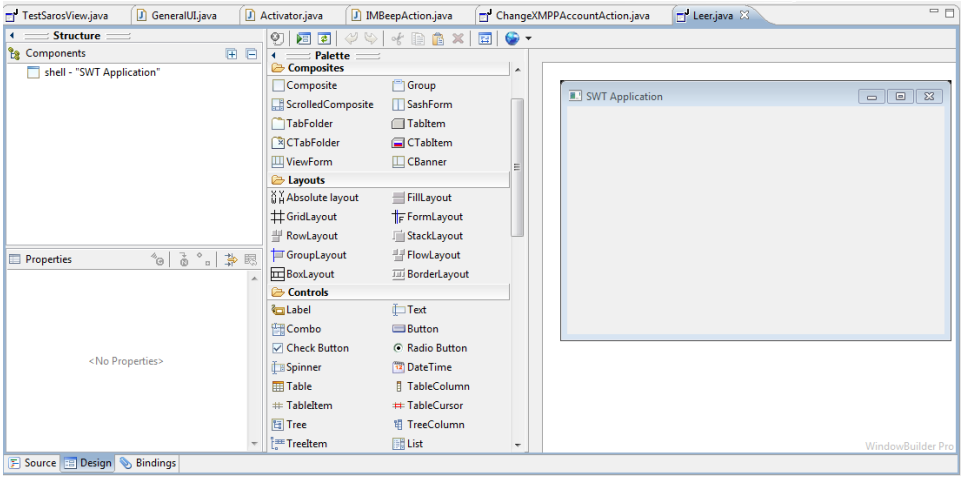

Abbildung 6: Die Oberfläche WindowBuilder Pro

Die Benutzung des WindowBuilders Pro ermöglicht es sich schneller in SWT einzuarbeiten. Der erste Vorteil ist, dass man alle Komponenten von SWT, die zur Verfügung stehen sieht. Außerdem kann man jederzeit die aktuelle Änderung in der GUI erkennen ohne nach jeder Änderung eine Java-Anwendung starten zu müssen. Der nächste Vorteil besteht in der Code-Generierung. Man kann dadurch schnell lernen wie man den Code zu schreiben hat, wenn man diese Design erreichen möchte. Durch die Gestaltung direkt in SWT ist zudem noch sichergestellt, dass alles technisch machbar ist was ich Designe und das Aussehen entspricht auch in Etwa der echten View später. Der generierte Quellcode kann zudem eventuell als einen Prototyp dienen.

5.3 Die angewandten Design Patterns

Entwurfsmuster (eng. Design Pattern) sind Lösungs-Schablonen für wiederkehrende Entwurfsprobleme [Wik10].

5.3.1 Die angewandten Design Patterns

Als physikalische Anordnung habe ich die gekachelte Ausschnitte [Tid05, Tiled Panes S.28] gewählt. Die Nachteile der mehren Fenster wurde bereits in Kapitel 4 ausführlich erörtert. Die linke Seite dient zur Informationsanzeige für die Kontakte und die rechte Seite dient für die Handhabung des Gruppenchats.

5.3.2 Gruppierung und Anordnung

Es gibt vier Aspekte [Tid05, Grouping and Alignment S.94] die dem Benutzer helfen herauszufinden welche Teile der Benutzeroberfläche zusammen gehören.

Nähe:

> Objekte in einer Benutzeroberfläche dicht bei einander sind, werden vom Betrachter als zusammengehörig assoziiert.

Ähnlichkeit:

> Objekte die gleiche Form, Größe, Farbe oder Ausrichtung haben werden vom dem Beobachter als zusammengehörig assoziiert.

Stetigkeit:

> Objekte die nebeneinander liegen und einen weichen Übergang haben werden vom Beobachter als zusammengehörig assoziiert.

Abschluss:

> Das Auge des Beobachters mag am liebsten einfache Formen wie Rechtecke oder Kreise. Deshalb werden Lücken vom Benutzer automatisch ausgefüllt.

5.4 Die ersten Designs

In meiner ersten Phase habe ich drei Entwürfe gemacht. Die Designs benutzen eine unterschiedliche Granularität bei der Verschmelzung des Saros Session Views und des Roster Views. Außerdem wurden unterschiedliche Darstellungsweisen für die Verschmelzung gewählt.

5.4.1 Design 1

Bei diesem Design werden der ehemalige Saros Session View und der Roster View in einer Baumstruktur integriert. Als Wurzelelemente dienen für die einzelnen Gruppen des Rosters. Hinzu kommen die Wurzelelemente für Benutzer die sich in der Saros Session befinden, als auch die Benutzer, die sich später nur im Chat befinden.

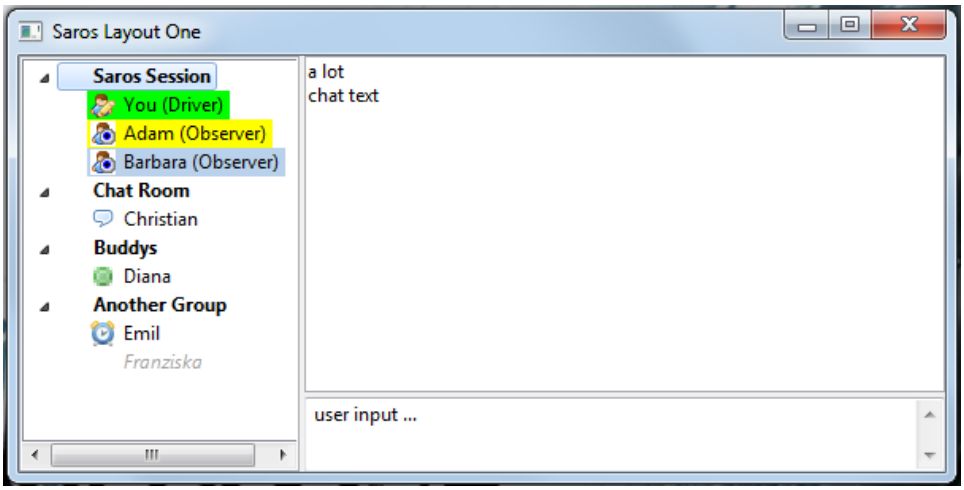

Abbildung 7: Das erste Design

5.4.2 Anwendung der Prinzipien

Nähe

Durch die großen Ränder kann man klar die Elemente unterscheiden.

Ähnlichkeit

Alle Kontakte sind in einem Baum angeordnet und Bestehen aus einem Symbol und einem Namen. Alle Kontakte die online sind haben die selbe Schriftfarbe.

Stetigkeit

In den Designs wird keine Berücksichtigung der Stetigkeit benötigt, weil es keine Elemente gibt, die nebeneinander liegen und zusammengehörig sind.

Abschluss

SWT bietet den Benutzer immer rechteckige Formen für seine Komponenten an. Bei der Gestaltung der Designs brauchte ich deshalb nicht auf die Eigenschaft des Abschlusses zu achten.

5.4.3 Design 2

Das 2. Design ist ein modifizierte Variante des ersten Designs. Dieses Design benutzt statt einer Baumstruktur expandierende Balken (*ExpandBar*). Die einzelnen Gruppen werden durch dieses Design stärker getrennt. Hinzu kommt, dass die ExpandBar mächtiger ist als ein Blatt in einer Baumstruktur. Blätter einer Baumstruktur können nur ein Icon und einen Text

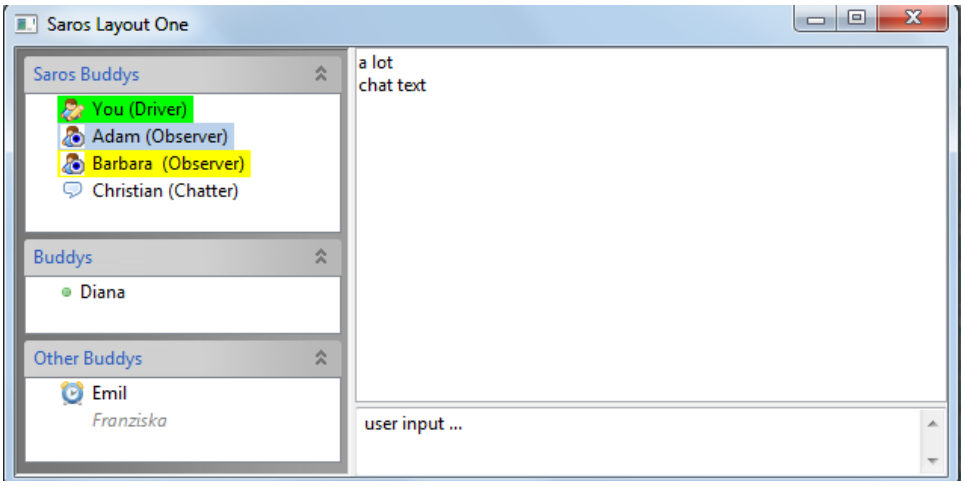

Abbildung 8: Das zweite Design

anzeigen. Die ExpandBar kann jedes beliebige Composite anzeigen bzw. verstecken.

Ausblendbare Elemente

> Ausblendbare Elemente [Tid05, closable panels S.111] werden benutzt, wenn es mehr anzuzeigen gibt als physikalisch Platz ist. Der Benutzer kann so entscheiden welche Informationen er sehen möchte und kann bei Bedarf mit nur einem Mausklick sich schnell zusätzliche Informationen anzeigen oder ausblenden (verstecken).

5.4.4 Design 3

Alternativer View:

> Es ist möglich, dass ein Design nicht alle Szenarien in der Benutzung abdecken kann. Es ist sinnvoll dem Benutzer dann eine Möglichkeit zu bieten, sich die Elemente anzeigen zu lassen wie es für diese Situation am besten ist. Dieses Design ist für Benutzer gedacht, die den View nicht an die untere rechte Ecke von Eclipse platzieren möchten. Dieses Design ist besser geeignet für Benutzer, die den View direkt an die Seite platzieren möchten. Dieser View ist wesentlich schmaler als bei dem anderen Designs. Außerdem lassen sich bei diesen Design bereits alle Komponenten minimieren, weil sich jetzt jede Komponente in der *ExpandBar* befindet.

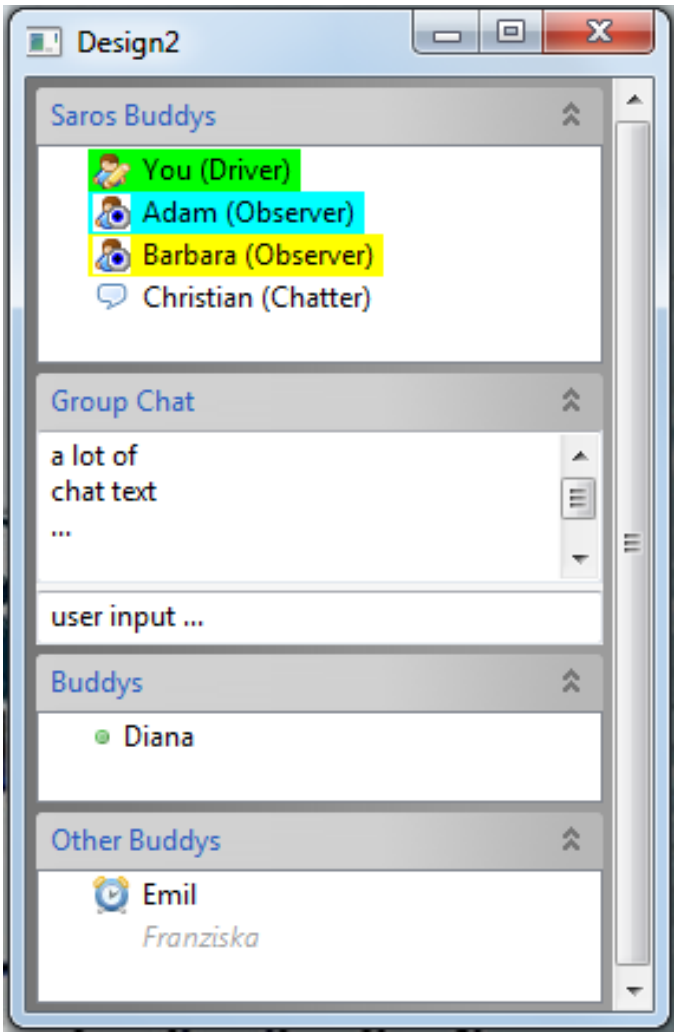

Abbildung 9: Das dritte Design

Ein Unterschied zwischen Design 1 und 2 bzw. 3 besteht darin, dass im ersten Design alle Benutzer, die im Chat aber nicht in der Saros Session sind, eine eigene Gruppe bekommen. Im 2. und 3. Design ist der Gruppenchat und die Saros Session zusammen gefasst. Da sowohl Chatter als auch Session-Benutzer in einen Gruppenchat sind müssen, diese nicht durch Gruppen getrennt sein. Um die Session-Benutzer leicht zu finden, werden diese nach oben sortiert und sind durch ihren farblichen Hintergrund und Symbol leicht von den Chattern zu unterscheiden. Das ersparen eines Eintrages in der *ExpandBar* reduziert die vertikale Raumeinnahme des Views.

Besonders wichtig bei der Gestaltung des Design war der Punkt, dass der Benutzer alle Informationen, die er nicht sehen möchte minimieren kann.

Diese Informationen lenken den Benutzer dann nicht mehr ab und der View verbraucht zudem weniger Platz.

5.5 Die Evaluierung der ersten Designs

Die Evaluation des ersten Design erfolgte im Rahmen eines Meeting mit weiteren Mitgliedern des Saros-Team, um möglichst viel Kritik an der Benutzeroberfläche zu äußern und mögliche alternative Gestaltungen zu finden. Am meisten wurde das 2. Design favorisiert, weil es Elemente klarer von einander trennt als im ersten Design. Hinzu kamen folgende Vorschläge zur Verbesserung des Designs:.

1. Der Bereich des Gruppenchats soll den Benutzer darin unterstützen sich in mehreren Gruppenchats befinden zu können. Als Designlösung dient ein *CTabFolder* in dem jeder Gruppen- oder Einzelchat später seinen eigenen Tab bekommen kann.

2. Ich hatte den Vorschlag gemacht, dass Benutzer die offline sind nicht im Roster anzeigen zu lassen, wie es bei Instant Messengern meist einstellbar ist. Das hätte den Vorteil, das der View weniger Platz einnehmen muss. Dieser Vorschlag erwies sich jedoch als ungeeignet, hinsichtlich der Tatsache, dass es den Benutzer irritieren kann, dass man hinzugefügte Kontakte nicht mehr sehen kann.

Die Toolbar hatte ich den Design noch nicht berücksichtigt, weil ich noch nicht wusste welche Aktionen ich daraus entfernen durfte. Dieses Vorgehen ist notwendig, weil die Toolbar sonst sehr viele Aktionen beinhaltet hätte und so die Effektivität wegen mangelnder Übersichtlichkeit gesengt wäre. Wenn der benötigte physikalische Platz der Toolbar länger wäre als die Breite des Views, dann würde sich die Toolbar über mehrere Zeilen erstrecken und es wäre weniger Platz für anderen grafischen Elemente. Es wurde beschlossen, dass die Aktionen: **Share Screen with selected user**, **Send a File to selected user** und **Start VoIP Session** in das Kontextmenü des Kontakts zu verschieben. Das hat den Vorteil, dass die Toolbar nicht zu überfüllt ist und unterstützt zudem Fitts' Gesetz [AD95, S.40] wodurch die Zeit um diese Aktionen auszuführen sinkt
Da auch der Chat nicht immer sichtbar sein muss, sollte es eine Funktionalität eingebaut werden, die es dem Benutzer gestattet auch diesen zu minimieren.

Außerdem gab es die Überlegung, dass es schön wäre wenn neben dem Benutzer ein Button mit einem Zahnrad-Symbol wäre. Dieser Button sollte wenn man mit der Maus auf dem Button stehen bleibt das Kontextmenü des Benutzer aufspringen lassen. Durch diesen Button könnten Mausklicks vermieden werden und der Anwender würde sehen, das mit den Benutzern

Aktionen möglich sind.

Hinzu kommt die Erkenntnis, das einige Icons nicht gut ausgearbeitet sind. Viele Icons haben als Hintergrundbild eine Person abgebildet. Dieses Symbol lenkt von dem 2. abgebildeten Symbol und macht es weniger kenntlich.

Die Möglichkeit zur Unterstützung von Gruppen im Roster ist den meisten Anwendern nicht bekannt, was zu einer Gruppe mit vielen Einträgen führt. Das neue Konzept sieht vor alle Kontakte aus der Session, Chat und dem Roster in eine Komponente zu vereinigen. Dabei werden die Kontakte nach Wichtigkeit sortiert und als zweites nach Alphabet.

6 Die erneute Gestaltung eines Designs

Dieses Kapitel beschreibt die erneute Gestaltung eines Designs, das der Evaluation gerecht wird. Dieses Design besitzt eine Toolbar, eine Möglichkeit den Chat zu minimieren und einen Button mit einem MouseOver[8]-Menü.

6.1 Die um zum neuen Design

Mit den neuen Anforderungen ist der WindowBuilder pro an seine Grenzen gelangt. Außerdem hatte ich noch keine Erfahrung, wie ich in Eclipse einen View einbauen kann. Eclipse benutzt aber nicht nur SWT sondern zusätzlich auch noch JFace. Durch JFace werden einige Gestaltungen anders gehandhabt als in SWT. Darunter fällt auch die Implementierung der Toolbar..

Nach einigen Anfangsschwierigkeiten mit der Erstellung eines Eclipse-Plugins und dem Benutzen von Bildern die Innerhalb eines Projektes liegen, gelang es mir die Funktionalität des alten Designs nachzubauen. Die meisten Anforderungen, die an das neue Design gestellt wurden waren relativ leicht umzusetzen.

Es gab nur zwei schwerere neue Funktionalitäten, zu einen die Minimierung des Chats und zum anderen die oben erwähnte Button der vor jeden Benutzer stehen soll.

Als Lösung für die Minimierung des Chats habe ich auf die linke Seite des Chats einen Button eingebaut, der beim anklicken den Chat minimiert.

Das zweite Problem konnte ich nach einiger Zeit zum einen großen Teil lösen. Um einen Button vor jedem Benutzer anzeigen zu lassen, habe ich ei-

[8] Mouseover: Zustand wenn die Maus für eine gewisse Zeit über einen Punkt hält

ne Tabelle erzeugt. Eine Tabelle fast eigentlich nur Texteinträge. Mit einiger Mühe kann man aber auch Buttons in eine Tabellenzelle einfügen. Also ist in der Spalte der Tabelle der Button gefolgt von einer Spalte um das Icon anzeigen zu lassen welches auch früher vor dem Benutzernamen steht und in der dritten Spalte steht dann der Name der Benutzername.

Dennoch haben sich bei dieser Gestaltung zwei Probleme ergeben:

1. Das erste Problem besteht in den Grafikfehlern die auftreten wenn die Größe des Views verändert wird und wieder lösen wenn die Blätterleiste (Scrollbar) benutzt wird.

2. Das 2. Problem ist der Aufruf des Kontextmenüs. Das Menü erscheinen zu lassen war kein Problem, aber es gibt keine Möglichkeit das Kontextmenü wieder zu schließen ohne es per Mausklick wieder zu schließen.

6.2 Die Präsentation des Designs im Seminar-BSE

Das Seminar-BSE (Beiträge zum Software Engineering) ist ein Arbeitsgruppenseminar, in dem laufend Vortrage zu Forschungsarbeiten stattfinden. Jedes Mitglied, das eine Arbeit im Bereich Software Engineering hält einen Vortrag (Konzeptvorstellung) nachdem er seinen Abschlussarbeit begonnen hat. Das Seminar bat mir die Gelegenheit die neue Benutzeroberfläche von einem breiten Publikum evaluieren zu lassen. Außerdem konnte ich dadurch auf Kritik zurückgreifen, von Anwendern die mit der Software nicht vertraut sind. Komponenten der neuen Benutzeroberfläche:

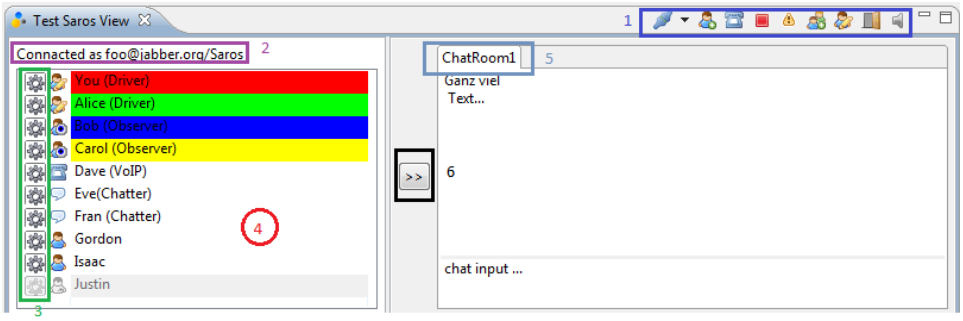

Abbildung 10: Das überarbeitete Design

1. Die Toolbar vereinigt die Toolbar aus dem Roster, der Saros Session und dem Chat.

2. Der Verbindungsstatus ist aus dem Roster übernommen und zeigt an, ob und welchem XMPP-Account der Benutzer verbunden ist.

3. Das ist der neue Button, der vor vor jedem Kontakt steht und beim MouseOver das Kontextmenü des Kontakts anzeigt.

4. Das sind die Kontakte aus der Saros Session, dem Roster und dem Chat/VoIP die nun in einer Komponente vereinigt sind.

5. Der rechte Teil der Benutzeroberfläche enthält den Chat. Jeder Chat resultiert in einem eigenen Tab.

6. Das ist der Button, der den Chat minimieren soll. Ein weiterer Druck auf den Button maximiert den Chat wieder.

Karteistapel:

> Der *CTabFolder* dient in der neu entworfenen Benutzeroberfläche als Karteistapel [Tid05, card stack S. 109]. Es ist nicht genug Platz verfügbar um später mehrere Chats nebeneinander zu platzieren. Der Karteistapel kann mehrere Chats im selben Bereich anzeigen, ohne dass der Benutzer die Übersicht bei den Chats verliert. Der Karteistapel hat den Vorteil, dass der Benutzer diese Anordnung bereits von andern Instant Messangern gewohnt ist.

Einige Anforderungen aus der letzten Evaluation wurden in dieser Fassung der Benutzeroberfläche noch nicht vollständig umgesetzt. Darunter gehört die Toolbar die noch nicht alle Elemente enthält und auch noch nicht in der richtigen Reihenfolge sind. Hinzu kommt, dass die Icons vor den Kontakten noch nicht überarbeitet wurden.

6.3 Die Evaluation des überarbeiteten Designs

Nach meinen Vortrag konnten die Zuhörer fragen zu dem Design stellen. Die gestellten Fragen brachten einige Kritikpunkte zum Vorschein:

1. Ein Einwand war, dass einige Benutzer lieber die Kontakte minimieren wollten. Das ist vor allem für die Benutzer interessant, die nur an Chat- bzw. VoIP-Konferenzen teilnehmen, aber sich nicht in einer Session befinden.

2. Der Button vor dem Benutzer brachte einige Konfusion hervor. Die Ursache liegt wohl an der Unbekanntheit dieses Features, welches aus einem Instant Messangers von Apple adaptiert wurde. Außerdem gibt es das technische Problem, das erscheinende Menüs wieder verschwinden zu lassen. Als Lösung wurde vorgeschlagen, dass nicht das Menü erscheint sonder ein Fenster welches so aussieht als wäre es das Menü.

Die Diskussion hat gezeigt, dass einige Elemente der Benutzeroberfläche überdacht werden sollten. Da ich in meiner Arbeit nicht genug Zeit habe um eine Umfrage unter Saros Benutzern durchzuführen, sollte es später in einer anderen Arbeit getan werden, um einen Konsens zu finden welche Features zu implementieren sind.

7 Vom Entwurf zur Implementierung

Nach dem Seminar-BSE war es an der Zeit sich mit dem Quellcode der derzeitigen Benutzeroberfläche zu beschäftigen und festzustellen wie sich mein Design am besten integrieren lässt.

7.1 Von der Theorie zur Praxis

Nach ausführlicher Auseinandersetzung mit dem für meine Arbeit relevanten Quellcode kam ich zu den Entschluss, dass es mir nicht möglich sein wird, mein vorgeschlagenes Design im Rahmen meiner Arbeit vollständig umzusetzen. Was in der Theorie einfach Aussah war in der Praxis viel komplizierter. Die Views benutzen ein große Anzahl von *Listener* des Funktionsweise ich erst vollständig verstehen musste. Hinzu kommt, die Reimplementierung der Baumstruktur im Roster. Eclipse benutzt für die Baum- und Tabellenstruktur nicht mehr direkt SWT sondern JFace. In JFace war ich zu diesem Zeitpunkt noch nicht eingearbeitet und den Einsatz des **Pico-Containers** für die Aktions aus dem Saros Session View war mir in diesem Moment auch noch nicht bekannt.

7.2 Planen der Implementierungsphase

Nachdem ich festgestellt hatte, dass ich im Rahmen meiner Arbeit nicht genug Zeit haben werde um mein Design vollständig zu realisieren, habe ich die zu realisierenden Features bzw. Anforderungen priorisiert. Die Priorisierung habe ich unter dem Aspekt vorgenommen, dass am Ende meiner Arbeit eine funktionierende SarosView zu haben, auch wenn diese nicht alle Anforderungen des neuen Designs vollständig enthält. Daher habe ich beschlossen erst mal alle Komponenten der alten Views zu verwenden. Danach sollten die drei Aktionen aus Toolbar in das Kontextmenü verschoben werden. Wenn noch Zeit übrig geblieben wäre hätte ich den Button vor den Kontakten hinzugefügt und danach an der Komponente gearbeitet um alle Kontakte in einer Komponente unterzubringen.

7.3 Saros neue Benutzeroberfläche

Die neue Benutzeroberfläche besitzt zwei Ansichten. Eine Ansicht wird benutzt, wenn sich der Benutzer nicht in einer Saros Session befindet, die andere während der Session. Die neue Benutzeroberfläche ist zweigeteilt. Auf der linken Seite werden dem Benutzer Informationen zu seinem Verbindungsstatus und seinen Kontakten angezeigt. Damit die Komponenten sich ähnlich sehen sind alle Komponenten mit einer Gruppierung umgeben. Die Gruppierung selber trennt die Komponenten leicht voneinander und durch den Titel jeder Gruppe weiß der Benutzer was für Informationen er wo zu erwarten hat.

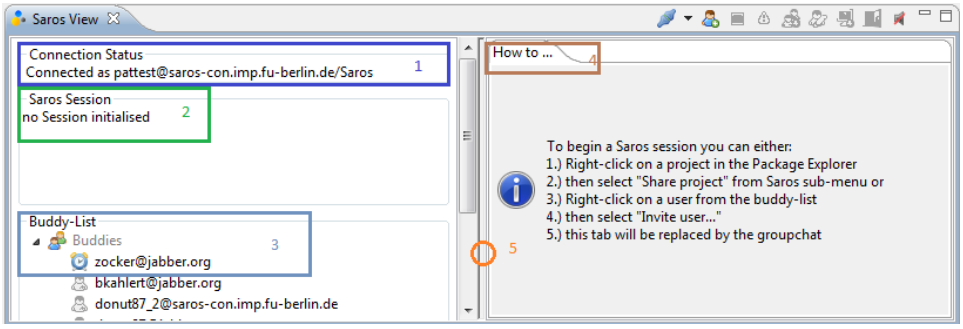

Abbildung 11: Neue Saros GUI mit Benutzer nicht in der Session

1. Hier wird der Verbindungsstatus des Benutzers angezeigt.
2. An dieser Stelle werden die Kontakte angezeigt, die sich in der Saros Session befinden. Die Darstellung ist die Wiederverwendung des früheren Saros Session Views.
3. Diese Gruppe dient zur Anzeige der Kontaktliste in der selben Baumstruktur des alten Roster Views.
4. Hier wird der Chat angezeigt bzw. eine Hilfe geboten, wie der Benutzer eine Saros Session starten kann.
5. An dieser Stelle ist eine Trennleiste, die der Benutzer verschieben kann um Platz zwischen der linken und rechten Seite zu gewichten wie er es möchte.

Das ist die Ansicht für den Benutzer, wenn keine Saros Session besteht. Da keine Saros Session besteht kann keine Tabelle bei 2) angezeigt werden. Dafür wird dem Benutzer ein Text angezeigt, der den Benutzer darüber informiert, dass er sich derzeitig nicht in einer Saros Session befindet.Derzeitig wird ein Gruppen-Chat erst betreten wenn eine Saros Session gestartet wurde. Es kann in der Zeit kein Chat in der Benutzeroberfläche dargestellt werden. Der Platz bei 4) wird stattdessen benutzt, um den Benutzer eine Hilfe zu geben wie er eine Saros Session starten kann. Die Abbildung ist die Ansicht, wenn die sich der Benutzer in einer Saros Session befindet. Die Stelle 2) wird jetzt durch die Tabelle ersetzt, die dem Benutzer anzeigt welche Kontakte sich in der Saros Session befinden. Da der Benutzer in einer Saros Session ist wird die Hilfe nicht benötigt und durch den Gruppenchat ersetzt. Das Icon in dem Karteireiter gibt an, ob ein Benutzer des Gruppenchats im Begriff ist grade etwas zu schreiben oder nicht. Wenn kein Benutzer des Gruppenchats schreibt wird eine Sprechblase angezeigt. Falls ein Benutzer etwas anfängt zu schreiben dann wird das Icon in einen Stift umgewandelt.

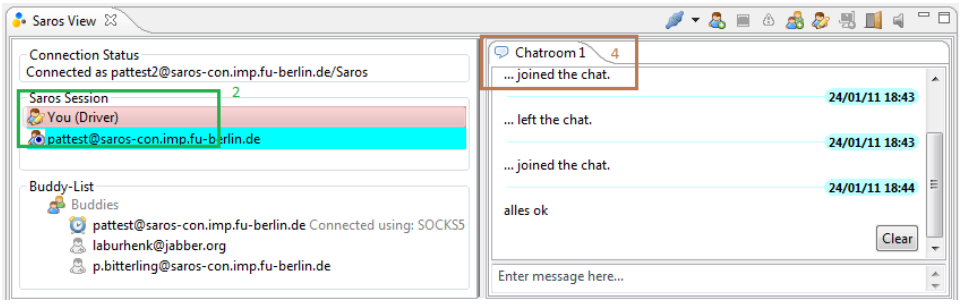

Abbildung 12: Neue Saros GUI mit Benutzer in der Session

7.4 Der Aufbau der neuen Benutzeroberfläche

Das folgende UML-Diagramm zeigt an wie die Elemente der Benutzeroberfläche miteinander verbunden sind. Hinzu kommen drei Markierungen bei denen noch Anmerkungen und Quellcodeabschnitte, die die Beziehung bzw. Funktionalität näher erläutern.

Die einzelnen Komponenten sind mit Rechtecken umgeben, die aus bis zu drei Zeilen bestehen. Die erste Zeile gibt den Typ der Komponente an, die zweite Zeile den Namen im Quellcode und die dritte Zeile gibt das Layout der Komponente an falls sie eins besitzt oder kein eigenes Verwendet.

Da die Komponenten miteinander in einer Baumstruktur Verknüpft sind, zeigen die Pfeile an welche Komponenten zusammen gehören.
Markierung 1:

> Das *StackLayout* sorgt für den Tausch zwischen dem dem *noSessionLabel* und dem *tableComp* welche die Tabelle zur Anzeige der Saros Sessoin Informationen enthält.
>
> Das Verhalten wird an drei Stellen beeinflusst (`createPartControl`, `sessionStarted`,`sessionEnded`).
>
> In `createPartControl` wird beim Start der Benutzeroberfläche zuerst das *noSessionLabel* angezeigt.

```
public void createPartControl(Composite
    parent) {
...
// Place for Saros Session
...
this.stackLayout = new StackLayout();
...
this.sessionGroup.setLayout(stackLayout);
```

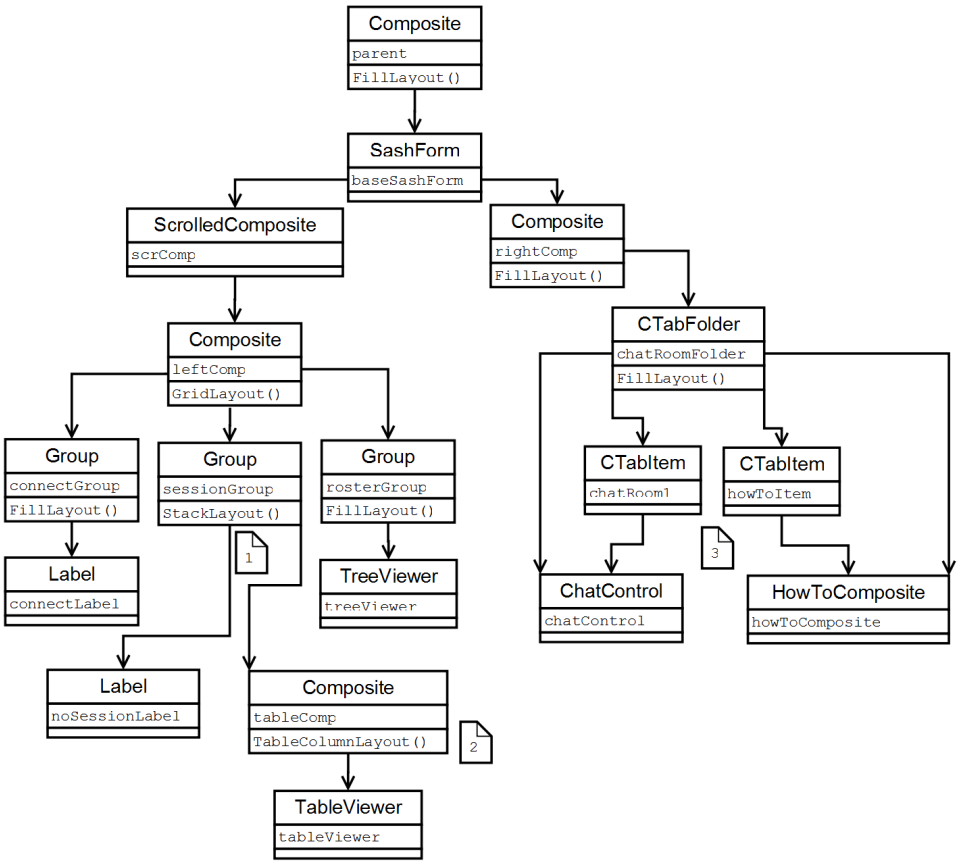

Abbildung 13: Strukturdiagramm der neuen Saros GUI

```
...
this.stackLayout.topControl = noSessionLabel;
this.sessionGroup.layout();
```

In `sessionStarted` wird der Anzeige zur Tabelle gewechselt. `SessionStarted` ist ein *Listener* der darauf horcht, dass eine Saros Session gestartet wurde.

```
public void sessionStarted(final
    ISarosSession newSarosSession) {
...
SarosView.this.stackLayout.topControl =
    SarosView.this.tableComp;
SarosView.this.sessionGroup.layout();
```

In `sessionEnded` wird der Anzeige wieder zurück zum `noSessionLabel` gewechselt. `SessionEnded` ist ein *Listener* der darauf

horcht, dass eine Saros Session beendet wurde.

```
public void sessionEnded(ISarosSession
    oldSarosSession) {
...
SarosView.this.stackLayout.topControl =
    SarosView.this.noSessionLabel;
SarosView.this.sessionGroup.layout();
```

Makierung 2:

Das `tableComp` Composite ist notwendig, weil das Group-Composite und das `TableColumnLayout` Grafikfehler bei der Umrandung verursacht hat.

Makierung 3:

Diese Stelle ist verantwortlich für die Anzeige der Karteireiter auf der rechten Seite der Benutzeroberfläche. Für das Anzeigen des Tabs ist das *CTabItem* verantwortlich. Mit der Funktion `setControl` wird festgelegt welchen Inhalt die Karteireiter anzeigen soll. Das Verhalten wird an drei Stellen definiert (`createPartControl, joined, left`).

In `createPartControl` wird beim Start der Benutzeroberfläche zuerst das `howToComposite` angezeigt.

```
public void createPartControl(Composite
    parent) {
...
howToItem = new CTabItem(chatRoomFolder, SWT.
    NONE);
howToComposite = new HowToComposite(
    chatRoomFolder, SWT.NONE);
...
this.chatControl = new ChatControl(
    chatRoomFolder, SWT.BORDER, white, white,
    2);
howToItem.setControl(howToComposite);
```

Die Funktion `joined` ist ein *Listener* der horcht ob der Benutzer einem Chat beitritt. Wenn der Benutzer einem Chat beigetreten ist dann befindet er sich derzeitig in einer Saros Session und das `howToItem` kann entfernt werden und dafür der `chatRoom1` als Tab angezeigt werden mit dem `chatControl` als Inhalt.

```
public void joined(final JID jid) {
...
SarosView.this.howToItem.dispose();
SarosView.this.chatRoom1 = new CTabItem(
   SarosView.this.chatRoomFolder, SWT.NONE);
SarosView.this.chatRoom1.setText("Chatroom 1"
   );
SarosView.this.chatRoom1.setImage(COMMENT);
SarosView.this.chatRoom1.setControl(
   chatControl);
```

Die Funktion `left` ist ein Listener der horcht ob der Benutzer einem Chat verlässt. Wenn der Benutzer einem Chat verlassen hat dann befindet er sich nicht mehr in einer Saros Session und der `chatRoom1` kann entfernt werden und dafür der `howToItem` als Tab angezeigt werden mit dem `howToComposite` als Inhalt.

```
public void left(final JID jid) {
...
SarosView.this.chatRoom1.dispose();
SarosView.this.howToItem = new CTabItem(
   SarosView.this.chatRoomFolder, SWT.NONE);
SarosView.this.howToItem.setControl(SarosView
   .this.howToComposite);
SarosView.this.howToItem.setText("How to ..."
   );
SarosView.this.chatRoomFolder.setSelection(0)
   ;
```

7.5 Toolbar und Kontextmenü

Bei der Gestaltung der Benutzeroberfläche wurden die drei Aktionen **Share Screen with selected user**, **Send a File to selected user** und **Start VoIP Session** in das Kontextmenü des Kontakts zu verschobe, um eine Überfüllung der Toolbar zu vermeiden.
Der Typ wurde intern von `Action` zu `SelectionProviderAction` geändert, damit sie wie anderen Aktionen im Kontextmenü angezeigt werden können.

7.6 Das HowToComposite

Die alte Benutzeroberfläche von Saros blendete in den Views Informationen ein, wenn der View noch nicht aktiv benutzt werden kann. Da in der neuen Benutzeroberfläche nur ein Teil des Views davon betroffen ist, habe ich eine eigenes *Composite* erstellt was diese Funktionalität nachahmen kann. Dieses *Composite* befindet sich im Package `de.fu_berlin.inf.dpp.ui.widgets`

und besitzt drei Funtionen (`setTitle, setImage, addStep`) und ein eigenes Layout.

setTitle:

> Diese Funktion setzt die erste Textzeile im Composite und dient zur Beschreibung was erreicht werden soll.

setImage:

> Diese Funktion entscheidet welches Bild neben dem Text angezeigt werden soll.

addStep:

> Diese Funktion fügt einen nummerierte Textzeile hinzu. Die Nummerierung erfolgt dabei automatisch. Jeder Aufruf von addStep soll einen Teilschritt zu Lösung des Problems liefern.

7.7 Sound als Feedback

Der Benutzer bekommt derzeitig nur einen Piep-Ton wenn er Chatnachricht erhält. Im Package`ui` habe ich eine neue Klasse SoundPlayer hinzugefügt. Diese Klasse ist basiert auf den Quellcode von http://www.anyexample.com/programming/java/java_play_wav_sound_file.xml . Unnötige Quellcodeabschnitte wurden entfernt, die Exceptions wurden um die Funktionalität des Protokollierens (Logging). In der Klasse `SarosUI` wurde eine Hilfsklasse angelegt um Dateien aus einem Eclipse-Projekt benutzen zu können. Der Soundplayer kann WAV-Dateien abspielen (keine MP3s).

Der Soundplayer soll es ermöglichen den Benutzer ein besseres Feedback zu geben. Die neue Tonausgabe soll dem Benutzer signalisieren, wenn Kontakte online oder offline gehen und beim Schreiben und Senden von Nachrichten. Die Töne sind dabei dem Instant Messanger **Pidgin** nachempfunden.

7.8 Kleine Probleme mit der neuen Benutzeroberfläche

TableViewer:

> Der TableViewer, der zur Anzeige der Tabelle dient, verbraucht am Anfang ungewöhnlich viel Platz. Da im *StackLayout* die Größe sich nach dem größten Element richtet, verbraucht das `noSessionLabel` viel mehr Platz als nötig.

SoundPlayer:

> Der Soundplayer spielt keine Töne auf dem Mac von Apple.

HowToComposite:

> Wenn die Breite des `HowToComposites` stark verkleinert wird, sieht der Text nicht mehr gut aus.

7.9 Was bleibt zu tun ?

Die neue Benutzeroberfläche entspricht noch nicht dem Design was ich beim Seminar-BSE vorgestellt habe. Um dieses Design zu erreichen sind noch einige Schritte zu tun.

1. Die Einträge aus dem Roster müssten in eine Tabelle gebracht werden.
2. Für die Kontakte aus dem Roster und der Saros Session muss ein gemeinsames Objekt geschaffen werden.
3. Die Einträge müssen nach Wichtigkeit sortiert werden.
4. Die Kontextmenülogik muss angepasst werden.

Wenn zusätzliche Features hinzukommen müssen weitere Anpassungen vorgenommen werden. Wenn es möglich ist einzeln zu chatten und/oder in einer Unterhaltung mit einem Kontakt zu sein, dann muss der Benutzer das im SarosView gut erkennen können. Dafür muss noch eine gute Darstellung gefunden werden, weil das Symbol vor dem Kontakt dafür nicht mehr ausreichen wird. Außerdem muss der Benutzer ermitteln können in welchen Gruppenchats sich der Kontakt befindet. Es sollte eine Anzeige geben wodurch der Benutzer ermitteln kann wer alles in einem Gruppenchat ist. Eine Teillösung könnten **DataTips** [Tid05, datatips S.176] sein. Datatips blenden zusätzliche Informationen ein wenn man über Punkt von Interesse geht. So könnte zum Beispiel dere Karteiteiter als ToolTip die Kontakte enthalten, die sich im Gruppenchat befinden.

Da die Testfunktionen in Saros derzeitig noch stark umgebaut werden, kann ich die nötigen Tests erst nach Abgabe dieser Arbeit schreiben.

8 Die Arbeitsgruppe Software Engineering

Während man seine Arbeit an der Fachgruppe Software Engineering schreibt ist man auch Mitglied der Arbeitsgruppen. Als Mitglied kommen neben der Arbeit noch zwei weitere Aufgaben hinzu. Eine Aufgabe ist das Beheben von Fehlern an der Software. Die zweite Aufgabe ist das Mithelfen bei einem Saros-Release.

8.1 Fehlerbehebung

Während meiner Zeit habe ich an zwei Stellen Fehler behoben. Der erste Bereich ist das AccountManagment und der zweite Bereich ist Benutzeroberfläche.

8.2 Hilfe beim Release

Bei den monatlichen Release von Saros werden vier Rollen verteilt: (Assisten-)Release-Manager und (Assisten-)Test-Manager. Ich habe währen meiner Zeit einmal als Release-Manager und einmal als Release-Assistent. Als Release-Manager hat man die Aufgabe einen Changelog zu erstellen. Das erstellen eines SVN-Branches und das spätere wieder zusammenführen des SVN-Branches. Des weiteren kommen die Erstellung des Dropins des neun Releases sowie die Anpassung der Update-Seite. Der Release-Assistent ist im Regelfall die Person, beim letzten Release der Release-Manger war und hilft dem Release-Manger falls Probleme auftreten sollten.

9 Anhang

Quellcode des SoundPlayers:

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

import java.io.File;
import java.io.IOException;

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.SourceDataLine;
import javax.sound.sampled.
   UnsupportedAudioFileException;

import org.apache.log4j.Logger;
import org.eclipse.ui.PlatformUI;

import de.fu_berlin.inf.dpp.preferences.
   PreferenceConstants;
import de.fu_berlin.inf.dpp.util.Util;

public class SoundPlayer {

    private static final Logger log = Logger.getLogger
       (SoundPlayer.class
        .getName());

    private final static int EXTERNAL_BUFFER_SIZE =
       524288; // 128Kb

    public static void playSound(final String fileName
       ) {

        if (!PlatformUI.getPreferenceStore().
           getBoolean(
            PreferenceConstants.BEEP_UPON_IM)) {
            return;
        } else {

            Util.runSafeAsync(log, new Runnable() {
                public void run() {

                    File soundFile = SarosUI.
                       getSoundFile(fileName);
                    if ((soundFile == null) || !
                       soundFile.exists()) {
```

```
            log.warn("Wave file not found
                at " + fileName);
            return;
        }

        AudioInputStream audioInputStream
            = null;
        try {
            audioInputStream = AudioSystem
                .getAudioInputStream(
                    soundFile);
        } catch (
            UnsupportedAudioFileException
            e1) {
            log.warn("Unsupported File",
                e1);
            return;
        } catch (IOException e1) {
            log.error("IO-Error while
                getting AudioInputStream",
                e1);
            return;
        }

        AudioFormat format =
            audioInputStream.getFormat();
        SourceDataLine auline = null;
        DataLine.Info info = new DataLine.
            Info(
            SourceDataLine.class, format);

        try {
            auline = (SourceDataLine)
                AudioSystem.getLine(info);
            auline.open(format);
        } catch (LineUnavailableException
            e) {
            log.error("No Audioline
                available", e);
            return;
        } catch (Exception e) {
            log.error("unknowen error
                while playing sound", e);
            return;
        }

        auline.start();
        int nBytesRead = 0;
        byte[] abData = new byte[
```

```
                        EXTERNAL_BUFFER_SIZE];

                    try {
                        while (nBytesRead != -1) {
                            nBytesRead =
                                audioInputStream.read(
                                abData, 0,
                                 abData.length);
                            if (nBytesRead >= 0)
                                auline.write(abData,
                                    0, nBytesRead);
                        }
                    } catch (IOException e) {
                        log.error("IO-Error while
                            write auline", e);
                        return;
                    } finally {
                        auline.drain();
                        auline.close();
                    }

                }
            });
        }
    }
}
```

Quellcode des HowToComposites:

```
package de.fu_berlin.inf.dpp.ui.widgets;

import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Label;

public class HowToComposite extends Composite {

    private Image showImage;
    private static final int numColumns = 2;
    private int steps = 0;

    private Label headline;
    private Composite imageSide;
    private Composite explanationSide;
    private Composite labelHolder;
```

```
    private Label iconLabel;

    public HowToComposite(Composite parent, int style)
        {
        super(parent, style);
        this.setLayout(new GridLayout(numColumns,
            false));
        this.showImage = Display.getDefault().
            getSystemImage(
             SWT.ICON_INFORMATION);

        this.imageSide = new Composite(this, SWT.NONE)
            ;
        this.imageSide.setLayoutData(new GridData(SWT.
            TRAIL, SWT.CENTER, false,
             true));
        this.imageSide.setLayout(new FillLayout());

        this.iconLabel = new Label(imageSide, SWT.NONE
            );
        this.iconLabel.setImage(showImage);

        this.explanationSide = new Composite(this, SWT
            .NONE);
        this.explanationSide.setLayoutData(new
            GridData(SWT.FILL, SWT.FILL,
             true, true));
        this.explanationSide.setLayout(new GridLayout
            (1, true));

        this.labelHolder = new Composite(
            explanationSide, SWT.NONE);
        this.labelHolder.setLayoutData(new GridData(
            SWT.LEAD, SWT.CENTER, true,
             true));
        this.labelHolder.setLayout(new FillLayout(SWT.
            VERTICAL));

        this.headline = new Label(labelHolder, SWT.
            WRAP);
        this.headline.setText(" ");

    }

    public void setImage(int systemImageNum) {
        this.showImage = Display.getDefault().
            getSystemImage(systemImageNum);
    }
```

```
    public void setTitle(String title) {
        this.headline.setText(title);
    }

    public void addStep(String nextStep) {
        this.steps++;
        Label step = new Label(this.labelHolder, SWT.
            WRAP);
        step.setText(this.steps + ".) " + nextStep);

    }

}
```

Literatur

[AC00] Laurie Williams Alistai Cockburn. The costs and benefits of pair programming. Technical report, Humans and Technology, University of Utah Computer Science, 2000.

[AD95] Gregory Abowd und Russell Beale Alan Dix, Janet Finlay. *Mensch Maschine Methodik*. Prentice Hall International, 1995.

[Dje06] R. Djemili. Entwicklung einer eclipse-erweiterung zur realisierung und protokollierungverteilter paarprogrammierung. Master's thesis, Freie Universität Berlin, Inst. für Informatik, 2006.

[JEHS09] Erik Arisholm Jo E. Hannay, Tore Dybå and Dag I.K. Sjøberg. The effectiveness of pair programming: A meta-analysis. Technical report, Simula Research Laboratory, Department of Software Engineering, University of Oslo, Department of Informatics, SINTEF Information and Communication Technology, 2009.

[Tid05] Jenifer Tidwell. *Designing Interfaces*. O'Reilly, 2005.

[TS09] Stephan Lukosch Till Schümmer. Understanding tools and practices for distributed pair programming. Technical report, FernUniversit¨ at in Hagen, Department for Mathematics and Computer Science, Cooperative Systems, Germany; Delft University of Technology, Faculty of Technology, Policy, and Management, Systems Engineering Section, The Netherlands, 2009.

[Wik10] Wikipedia. Entwurfsmuster. http://de.wikipedia.org/w/index.php?title=Entwurfsmuster&oldid=83818970, Januar 2010.

[Wik11] Wikipedia. Pair programming. http://en.wikipedia.org/w/index.php?title=Pair_programming&oldid=410351810, Januar 2011.