Verbesserung von Saros in unzuverlässigen Netzwerken mit hoher Latenz

Bachelorarbeit

vorgelegt am
Institut für Informatik,
Arbeitsgruppe Software Engineerin
Prof. Dr. Lutz Prechelt
Freie Universität Berlin

am

23. November 2010

von
Sebastian Bauch

Betreuer
Dr. Karl Beecher

Zusammenfassung

Die vorliegende Bachelorarbeit hat zum Ziel die Überarbeitung der Netzwerksschicht der Eclipse-Erweiterung Saros und die Erweiterung der Sandor-Testumgebung. Die Basis für meine Arbeit waren die Diplomarbeiten von den Studenten Sandor Szücs und Henning Staib. Die Untersuchungen begannen mit den Tests an den Netzwerkprotokollen. Dafür wurden die Parameter Betriebssystem und Netzstruktur festgelegt. Von den vier von Saros verwendeten Protokollen erwiesen sich zwei als einsetzbar. Auf dieser Grundlage wurde die Netzwerkschicht überarbeitet. Im Ergebnis laufen die Netzwerkverbindungen stabil. Im weiteren Verlauf der Arbeit wurde die Testumgebung erweitert.

Inhaltsverzeichnis

1 Einleitung

Ein Forschungsgebiet der Arbeitsgruppe *Software Engineering* an der Freien Universität behandelt die unterschiedlichen Methoden der Softwareentwicklung. Als wesentliche Methoden sind hier die Paarprogrammierung (Pair-Programming) und die *Side-by-Side* Programmierung zu nennen. Diese Konzepte werden mit Hilfe des Forschungsprojektes Saros untersucht. Dieses Projekt entstand als Diplomarbeit des Studenten Riad Djemili[Dje06] und wurde in nachfolgenden Studien- und Abschlussarbeiten in der Arbeitsgruppe weiterentwickelt.

Saros ist eine *Open Source* Eclipse-Erweiterung[Ecl10] zur verteilten, gemeinsamen und gleichzeitigen Programmierung. Es unterstützt die Konzepte der verteilten Paarprogrammierung und *Side-by-Side* Programmierung. In verschiedenen Rollen können mehrere Entwickler gleichzeitig an einem gleichen Quellcode arbeiten.

Als Forschungsprojekt soll Saros helfen, die Methoden der verteilten Programmierung näher zu untersuchen. Dazu wird versucht, die Software in verschiedenen Firmen zu integrieren. Besonders interessant ist Saros für Firmen, die an unterschiedlichen Standorten vertreten sind, ihre Projekte trotzdem gemeinsam entwickeln wollen. Exemplarisch soll hier ein Projekt zwischen zwei Firmen in Österreich und Indien genannt werden. Hier entwickelten Mitarbeiter beider Firmen paarweise an einem Projekt. Der Datenverkehr wurde über das Internet vollzogen.

Die Übertragungszeit von Paketen in einem Netzwerk variiert in Abhängigkeit der Paketlaufstrecke. Bei hoher Auslastung des Netzwerkes kann es zu Verzögerungen oder im schlimmsten Fall zum Verlust von Paketen kommen. Des Weiteren kann es auch dazu kommen, dass Pakete nicht in der versendeten Reihenfolge ankommen.

Auf der Netzwerkebene muss Saros mit den oben genannten Problemen umgehen können. Es kommt hier vor allem auf die Reihenfolge der versendeten Pakete an. Zum einen muss gewährleistet werden, dass die getätigten Interaktionen der Teilnehmer in der richtigen Reihenfolge übertragen werden und zum anderen sollte bei der Initiierung einer Sitzung versucht werden, die schnellste Übertragungsverbindung zu bekommen.

Gerade im zweiten Punkt hatte Saros Probleme. Zu Beginn meiner Arbeit konnte im besten Fall eine TCP Verbindung mit Jingle[SA10] aufgebaut werden. Im schlimmsten Fall wurde immer eine langsame *In-Band Bytestream*[KSA10] Verbindung aufgebaut. Hierbei werden die Daten in XMPP-Nachrichten[Inet04a][Inet04b] zerlegt und versandt. Diese Art des Versendens machte einen Austausch von großen Datenmengen zu einem Geduldsspiel.

Ziel der Arbeit war es, Saros in diesem Punkt zu untersuchen und zu verbessern. Ausgangspunkt hierfür waren die Arbeiten der Diplomstudenten Sandor Szücs und Henning Staib. Der Student Staib untersuchte in seiner Arbeit die von Saros zu XMPP-Kommunikation verwendete Bibliothek Smack[Sma10]. Er führte Verbesserungen an den Verbindungsprotokollen *In-Band Bytestream* und *SOCKS5* durch. Mit diesen Verbesserungen war es nun möglich, das Protokoll SOCKS5 auch für Saros zu verwenden.

Der Student Szücs entwarf in seiner Arbeit eine Testumgebung, welche die Saros-Testprozedur[Sar10a] vereinfachen sollte. Die Testumgebung soll zum einen die Personenarbeitszeit verringern und zum anderen ein Testen mit vorgegebenen Parametern ermöglichen.

Auf der Grundlage der oben genannten Arbeiten ergaben sich folgende Aufgaben:

1. Überarbeitung des Netzwerkmoduls und Behebung von Defekten.

2. Erstellen von Testfällen und Mitarbeit an der Verbesserung der Testumgebung.

Diese beiden Punkte sollen zur Verbesserung von Saros auf der Netzwerkebene beitragen.

2 Theoretische Grundlagen

Dieses Kapitel soll einen Einblick in die theoretischen Grundlagen und die Entwicklungsabläufe von Saros geben. Des Weiteren werden die wichtigsten Funktionen von Saros vorgestellt.

2.1 Paarprogrammierung

Die Paarprogrammierung ist eine Arbeitsweise, die mit dem Softwareprozess *Extreme Programming*[Bec99] populär wurde. Hierbei arbeiten zwei Programmierer nebeneinander vor demselben Monitor an der gleichen Aufgabe. Während einer Sitzung nimmt ein Entwickler die Rolle des *Drivers* ein. Er bedient die Eingabegeräte und erläutert während der Arbeit seine Vorgehensweise. Der zweite Entwickler nimmt die Rolle des Beobachters, auch *Observer* genannt, ein. Der *Observer* beobachtet die Eingabe des Drivers und macht auf Fehler aufmerksam. Seine passive Sicht ermöglicht es ihm, den Überblick über die gesamte Aufgabe zu behalten und über Entwurfsentscheidungen und Problemlösungen nachzudenken.

In zahlreichen Experimenten und Befragungen wurden die Vor- und Nachteile der Paarprogrammierung gegenüber der Einzelprogrammierung untersucht. Sie zeigen, dass im Vergleich zur Einzelprogrammierung eine bessere Codequalität durch weniger Defekte und Codezeilen, ein besserer Wissenstransfer und eine höhere Zufriedenheit bei den Entwicklern erzielt werden kann [BN08] [NW01] [CW01] [Nos98] [Wil00]. Ein wesentlicher Nachteil liegt in der Bindung von zwei Entwicklern an einer Aufgabe. Während Nosek[Nos98] und Williams[Wil00] eine Reduzierung der Arbeitszeit um 40 Prozent messen, kommen Nawricki et. al.[NW01] in ihrer Studie auf eine Reduzierung von maximal 20 Prozent. Allerdings betrachtet die Studie von Nawricki et. al. nur die reine Arbeitszeit und lässt die Zeitersparnis durch weniger Defekte außen vor.

2.2 Side-by-Side-Programmierung

Eine abgeschwächte Form der Paarprogrammierung ist die *Side-by-Side* Programmierung. Bei dieser Arbeitsweise werden Merkmale der Einzel- und der Paarprogrammierung vereint. Jeder Entwickler arbeitet an seinem eigenen Rechner. Eine räumliche Nähe ermöglicht bei Bedarf Paarprogrammierung.

Eine Studie an der Universität von Poznan[NJOL05] hat gezeigt, dass diese Form der Paarprogrammierung die Entwicklungszeit auf 60 Prozent gegenüber der Einzelprogrammierung verringert. Allerdings sinkt hierbei das Wissen über den Code [NJOL05] und eine Reduzierung der Defekte durch eine laufende Durchsicht ist nicht mehr gegeben.

2.3 Verteilte Paarprogrammierung

Eine Variante der Paarprogrammierung ist die verteilte Paarprogrammierung. Wie bei der Paarprogrammierung arbeiten die Entwickler am gleichen Code, sitzen aber an ihren eigenen Rechnern. Die Rechner können sich im gleichen Raum oder auf verschiedenen Kontinenten befinden. Damit beide Entwickler die gleiche Arbeitsumgebung wie bei der Paarprogrammierung haben, wird spezielle Software benötigt. Hierfür stehen zwei verschiedene Ansätze zur Verfügung, das *Desktop Sharing* und die *Collaburative Awareness*[Dje06]. Bei dem *Desktop Sharing* wird der gesamte Bildschirminhalt vom Host an den Client übertragen. Die Arbeiten finden dann bei dem Host statt und Änderungen werden auch nur dort gespeichert.

Bei dem Ansatz der *Collaburative Awareness* ist der Mehrbenutzermodus in das Werkzeug integriert und die Änderungen an der Software werden auf den jeweiligen Rechnern gespeichert und bei Bedarf synchronisiert. Diese Art der Synchronisierung verringert, gegenüber dem *Desktop Sharing*, die Netzauslastung. Als wesentliche Vorteile zählt Djimili in seiner Arbeit den höheren Datenschutz, die hohe Parallelität und immer eine optimale Darstellung des Codes auf. Der hohe Datenschutz ist dadurch gegeben, dass der Entwicklungspartner nur Zugriff auf den Code, nicht aber auf den Rechner hat. Auch können hier beide Entwickler parallel arbeiten, da jeder von ihnen eine identische Kopie vom Code hat.

Baheti et. al.[BWGSS92] zeigen in ihrer Studie, dass mit geeigneten Werkzeugen die Vorteile der Paarprogrammierung erreicht werden.

2.4 Saros

Saros ist eine Erweiterung der Entwicklungsumgebung Eclipse[Ecl10] und entstand im Rahmen der Diplomarbeit von Riad Djimili [Dji06] an der Freien Universität Berlin. Im Zuge mehrerer Seminar- und Studienarbeiten wurde die Software weiterentwickelt.

Saros erweitert die Entwicklungsumgebung Eclipse um Funktionen zur Ausübung von Paarprogrammierung und *Side-by-Side* Programmierung. Es verwendet den Ansatz der *Collaburative Awareness*.

Das Starten einer Saros- Sitzung erfordert ein XMPP-Konto, über das die Teilnehmer im Netz erreichbar sind, und die gleiche Version von Saros. Die Forderung einer gleichen Version ist mit der stetigen Weiterentwicklung und damit eventuellen Inkompatibilität mit älteren Versionen begründet.

Zum Initialisieren einer Saros-Sitzung wählt ein Telnehmer in seiner Entwicklungsumgebung ein Projekt aus. Damit wird dieser zum Host der Sitzung. Im Zuge des Einladungsprozesses wählt der Host alle weiteren Teilnehmer der Sitzung aus. Nachdem diese die Einladung bestätigt haben, wird die Replikation des Projektes eingeleitet. Dazu überprüft der Host, welche Dateien der Client schon besitzt. Danach werden die fehlenden Dateien an ihn übertragen. Nach dem Einladungsprozess besitzen alle Teilnehmer eine identische Kopie des Projektes.

Eine Sitzung wird immer im Paarprogrammiermodus gestartet. Das heißt, der Host hat am Anfang der Sitzung die Rolle des *Drivers* und die Clients die Rolle der *Observer* inne. Er kann die Rollen in der Paarprogrammierung tauschen, indem er einem anderen Teilnehmer exklusive Schreibrechte erteilt und selbst die Rolle des Beobachters einnimmt. Oder er wechselt in den *Side-by-Side* Programmiermodus, indem er allen Sitzungsteilnehmern Schreibrechte erteilt. Die Beobachter können in einen *Follow Mode*[1] wechseln. Damit folgen sie automatisch allen Arbeitsschritten des *Drivers*.

Zur Wahrung der *Awareness* Informationen weist Saros jedem Teilnehmer eine Farbe zu. Diese Farben markieren den aktuellen Aufenthaltsort der Teilnehmer, dienen der Kennzeichnung von Codepassagen und machen die unabhängig erstellten Codepassagen sichtbar. Zusätzlich zu den Teilnehmerkennzeichnungen wird im *Package Explorer* farblich gekennzeichnet, welche Dateien gerade bearbeitet werden oder zum Lesen geöffnet wurden. Hiermit ist jederzeit ersichtlich, wo sich die einzelnen Sitzungsteilnehmer befinden.

[1] Im *Follow Mode* verfolgt der Beobachter alle Bewegungen des *Drivers*. Er befindet sich immer im gleichen Sichtfeld.

Wie am Anfang erwähnt, ist ein wesentlicher Faktor der Paarprogrammierung und *Side-by-Side* Programmierung die Kommunikation zwischen den Entwicklern. Saros stellt den Entwicklern einen Gruppenchat und eine *Voice over Ip*[2] Möglichkeit zur Verfügung. Eine weitere Möglichkeit der Kommunikation stellt das *Screen Sharing* dar. Diese soll der visuellen Darstellung zum Beispiel von Bildern dienen.

Im jetzigen Entwicklungsstadium können die VoIp und die *Screen Sharing* Funktionalität nur zwischen zwei Teilnehmern angewandt werden. Die folgende **Abbildung 1** zeigt die einzelnen Sicht- und Kommunikationsfenster sowie die farblichen Unterteilungen zur Darstellung der *Awareness* Informationen von Saros.

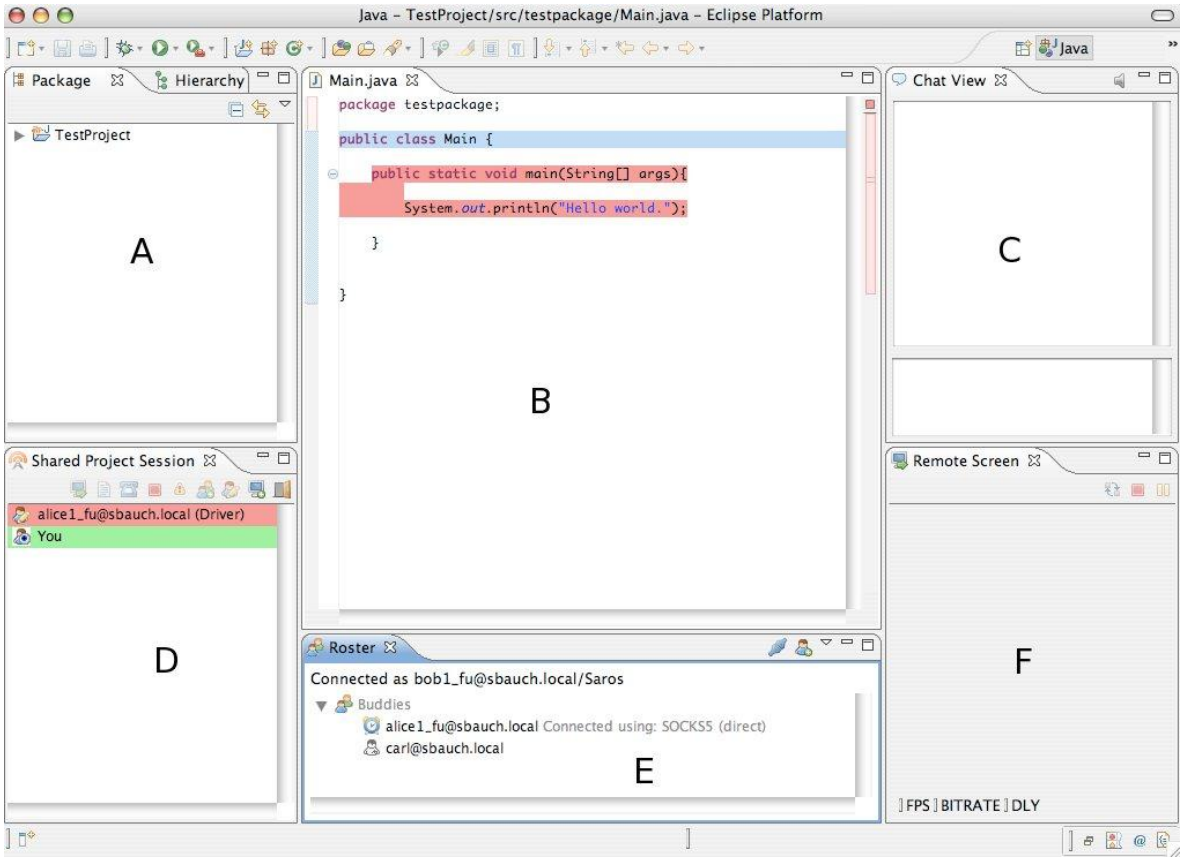

Abbildung 1 Saros Benutzersicht: A) Das geteilte Projekt wird mit einem „@" gekennzeichnet; B) Eclips-Editor mit farblicher Trennung der Teilnehmer C) Chatfenster; D) *Shared Project Session* Fenster zur Darstellung der Sitzungsteilnehmer; E) Das *Roster* Fenster zeigt die Jabber-Freundesliste an. F) Im *Remote Screen* Fenster

[2] *Voice over Ip* (VoIp) bezeichnet eine Technologie zur Sprachübertragung.

2.5 Softwareprozesse bei Saros

Die Entwicklung von Saros wird iterativ durchgeführt[Sar10a]. Die Veröffentlichung einer neuen Version erfolgt in der Regel alle vier Wochen. Hierbei unterteilt sich ein Entwicklungszyklus in zwei Phasen. In der ersten Phase, die drei Wochen andauert, arbeiten die einzelnen Teammitglieder an ihren Einzelprojekten. Jeder ist angehalten, Codedurchsichten von Patches zu machen, welche im *Reviewboard*[Rev10] aufgelistet sind. In der zweiten Phase, in der Regel die letzte Arbeitswoche eines Monats, findet die Veröffentlichung einer neuen Version statt. Hierbei werden vier Projektmitgliedern die Rollen des *Releasemanagers* (RM), des *Assistant Release Manager* (ARM), des Testmanagers (TM) und dessen Gehilfen (ATM) zugeteilt.

Der RM erstellt am Anfang der Woche eine Liste aller Neuheiten und Verbesserungen von Saros und macht sie den Projektmitgliedern zugänglich. Im Anschluss eröffnet er einen Veröffentlichungszweig im *Sourcevorge-SVN*[Sor10].

Daraufhin überlegen sich der TM und der ATM, in Absprache mit den Entwicklern, geeignete Testfälle für den Testtag. Der Mittwoch ist für die Durchführung der Testfälle vorgesehen. Der TM und der ATM koordinieren die Durchführung der einzelnen Testfälle und protokollieren die Ergebnisse. Nach Beendigung des letzten Testfalls erstellt der TM eine Liste alle aufgetreten Defekte, trägt diese in die Fehlerliste auf *Sourceforge* ein und weist sie den einzelnen Projektmitgliedern zur Behebung zu. Der Donnerstag und der Freitagvormittag dienen der Behebung der zugewiesenen Defekte.

Am Freitagnachmittag setzen sich der TM, ATM, RM und der ARM zusammen und führen letzte Tests an der Veröffentlichungsversion durch. Bei positivem Ausgang erstellt der RM ein neues *Release*, lädt alle Daten bei *Sourceforge*[Sor10] hoch und aktualisiert die Internetseiten[Sar10b][Sor10].

Mit einem Dank an alle Projektmitglieder wird die Woche beendet.

3 Erweiterung der Sandor - Testumgebung

Dieses Kapitel soll einen Einblick in das Sandor-Testumgebung geben und meine Arbeit bei der Erstellung von geeigneten Testfällen wiedergeben.

3.1 Motivation

Zu Beginn meiner Arbeit war die Testumgebung gerade fertig gestellt. Es konnten einige wenige Testfälle auf einem Arbeitsgruppenrechner ausgeführt werden. Die Testumgebung wurde vom Diplomstudenten Sandor Szücs entwickelt. Auch wenn die vorhandenen Beispieltests funktionierten, machten sich bei genauerem Hinsehen viele Schwächen der ausgewählten Bibliothek bemerkbar. Als Beispiel soll hier die Zeitintensivität und das nicht Unterstützen von Komponenten der SWT-Bibliothek[3] genannt sein. Die Diplomstudentin Lin Chen überarbeitete die Testumgebung und führte neue Funktionen ein. Hierdurch wurde es möglich, mit dem Schreiben von geeigneten Testfällen anzufangen. Zum heutigen Tag sind diese Tests nur lokal ausführbar, denn die Zeitintensivität und Probleme von Saros mit Ubuntu verhindern ein Ausführen der Tests in der Testumgebung.

Zu meinen Aufgaben gehörte das Schreiben von Testfällen für die Testumgebung. Diese sollen später helfen, die Veröffentlichungswoche für die Projektmitglieder durch Automatisierung der Testfälle zu vereinfachen. Dieses Kapitel soll weiteren Testentwicklern eine Hilfestellung geben, einen einfacheren Einstieg in das Schreiben von Testfällen zu finden und leichter mit bekannten Hindernissen fertig zu werden.

3.2 Architektur

Die Architekturbeschreibung ist in zwei Teile gegliedert. Der erste Teil behandelt den Aufbau der Testumgebung und deren Protokollebene. Der zweite beschreibt zentrale Objekte und die Architektur von Testfällen.

3.2.1 Aufbau der Testumgebung

Zum jetzigen Zeitpunkt besteht die Testumgebung aus drei virtuellen Maschinen. Eine virtuelle Maschine enthält die XMPP-Server und dient der Simulierung von verschiedenen

[3] Die SWT-Bibliothek ist eine Bibliothek für die Erstellung von grafischen Oberflächen in Java.

Netzwerkeigenschaften. Die anderen sind für die Musikanten vorgesehen. Die einzelnen Testfälle werden vom *Build-Server[BS10]* ausgeführt.

Zum Testen werden die Musikantenrechner neu gestartet, um eine saubere Umgebung zu haben. Im aktuellen Entwicklungsstand wird eine ausführbare Version von Saros erzeugt und auf die einzelnen Rechner verteilt und mit den zugewiesenen Ports gestartet. Dieser Verteilungsansatz hat den Nachteil, dass die einzelnen Versionen auf den Rechnern verbleiben und zurzeit von Hand ausgeführt werden müssen. Über ein Script werden dann die einzelnen Testfälle aus dem *Build-Server* heraus gestartet. Dieser protokolliert die aufgetretenen Fehler und gibt sie auf der Benutzerschnittstelle aus.

Bei entsprechendem Wunsch können auf den Musikanten von den einzelnen Schritten Aufnahmen gemacht werden. Diese sollen helfen, aufgetretene Fehler besser aufspüren zu können.

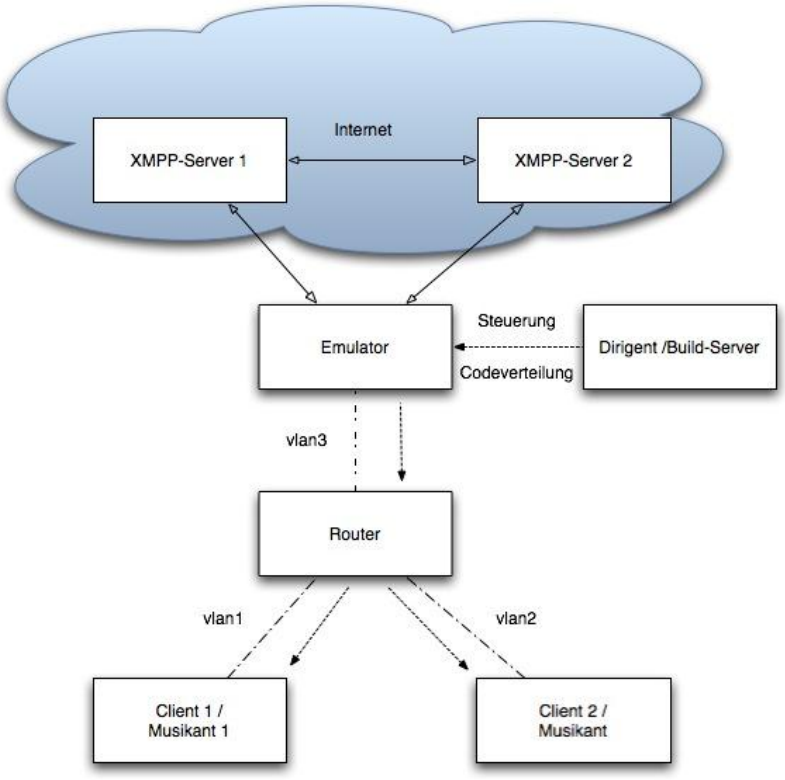

Abbildung 2 Saros-Testumgebung; Der *Build-Server* verteilt erstellt und verteilt die Saros-Version an die Clients. Des Weiteren steuert er die Clients/Musikanten. Die Clients befinden sich in eigenständigen virtuellen Netzwerken(vlan) und kommunizieren über den Router. Der Emulator emuliert die Netzwerkeigenschaften.

3.2.2 Erstellen von Testfällen

Die Testumgebung wurde so programmiert, dass die Saros Funktionen und die *SWTBot*[4] Funktionen in unterschiedlichen Paketen gekapselt sind und die Musikanten auf die Funktionen über Schnittstellen zugreifen.

Die wichtigsten Klassen zur Erstellung von Tests sind:

- BotConfiguration
- SarosState
- Musician
- SarosRmiWorkbenchbot
- WaitUntilObjekt

Die *BotConfiguration* Klasse dient zur Konfiguration der Musikanten. Hier sind alle für eine Saros-Verbindung wichtigen Parameter gespeichert. Dazu zählen die Jabber-ID[5], der Port und die Clientinternetadresse.

Das *SarosState* Objekt enthält Funktionen, die den internen Ablauf von Saros steuern, beziehungsweise den Status von Saros-Funktionen abfragen. Anwendung finden diese Funktionen, wenn es keine Möglichkeit gibt, eine äquivalente *SWTBot* Funktion zu schreiben oder visuelle Statusmeldungen mit dem internen Status von Saros auf Richtigkeit zu vergleichen.

Das für die *SWTBot* Funktionen interessante Objekt ist das *SarosRmiWorkbenchbot* Objekt. Hier werden alle auszuführenden Steuerungsfunktionen definiert. Zu diesem Zeitpunkt findet eine weitere Kapselung statt, welche die einzelnen Funktionen Eclipse Komponenten zuordnet. Diese soll die Zuordnung von Funktionen zu den Komponenten erleichtern, und das *SarosRmiWorkbenchbot* Objekt etwas lesbarer machen.

Die Interaktionen des *SWTBots* sind an unterschiedlichen Vorbedingungen gebunden. Als Beispiele soll hier das Warten auf die Aktivierung einen *Buttons* oder das Eintreffen einer XMPP-Nachricht genannt werden. Diese Wartebedingungen werden im *WaitUntilObject* gepappselt.

---

[4] Der SWTBot dient dem funktionalen Testen von Eclipse-Anwendungen.

Die Struktur der Testumgebung hat sich seit der Fertigstellung so stark geändert, dass sich auch die Vorgehensweise während der Testerstellung geändert hat. Auf die Erstellung eines JUnit4-Testfalls[6] möchte ich hier nicht weiter eingehen und verweise auf eine gute Beschreibung in der Diplomarbeit von Sandor Szücs.

Zum Verständnis der Arbeit möchte ich auf die Erstellung von relevanten Funktionen eingehen und dies an dem Test *TestChatFunktions* erläutern.

Der Test hat die Aufgabe, die Funktionsweise der Chatansicht zu überprüfen. Dazu wird geprüft, ob wir uns um Chatraum befinden und ob die von Alice an Bob gesandte Nachricht richtig versandt wird.

Um diese Funktionen überprüfen zu können, müssen folgende Vorbedingungen erfüllt werden:

- Alice und Bob sind mit einem XMPP-Server verbunden und haben sich gegenseitig in der Freundesliste.
- Alice und Bob befinden sich in einer Saros-Sitzung.
- Alice und Bob haben das Chatfenster geöffnet.

Aufbauend auf diesen Vorbedingungen können nun die *SWTBot* Funktionen definiert werden.

- Nachrichten aus dem Fenster lesen
- Nachrichten versenden

Die erste Aufgabe wird durch die Funktion *compareChatMessage* gelöst.

```
public boolean compareChatMessage(String jid, String string)
throws RemoteException;
```

Abbildung 3 Vergleichsfunktion

[5] Die Jabber-ID hat die Form alice@jabber.ccc.de und ermöglicht eine Adressierung im XMPP-Netzwerk
[6] JUnit4 ist eine Testumgebung zum Erstellen und Ausführen von automatisierten Testfällen. Die Testfälle werden in Java programmiert.[Jun10]

Diese Funktion bekommt eine als Parameter eine Jabber-ID und den zu vergleichenden Text übergeben und wirft, wenn der Musikant nicht mehr erreichbar ist, eine *RemoteException*.

```
public boolean compareChatMessage(String jid, String message)
        throws RemoteException {
        if (!isChatViewOpen())
            openChatView();
        activateChatView();
        SWTBotView view = delegate.activeView();
        SWTBot bot = view.bot();
        SWTBotStyledText text = bot.styledText();
        return Comperator.compareStrings("[" + jid, message,
text.getLines());
}
```

Abbildung 4 Funktionsdefinition

Eine Funktion muss am Anfang immer überprüfen, ob die entsprechende Ansicht geöffnet wurde und sie gegebenenfalls öffnen. Die *activateChatView* Funktion stellt den Fokus des *SWTBots* auf das Chatfenster. Im Normalfall besitzt der *SWTBot* immer den Fokus auf dem zuletzt geöffneten Fenster, doch durch aufspringende Saros-Meldungen kann er diesen verlieren. Dies führt dann zu einem Scheitern der Funktion und damit des Testfalls.

Mit dem Erlangen des Fokusses kann nun der *SWTBot* der SWT-Komponente geholt werden. Dieser ermöglicht es, auf die verschiedenen Felder der Komponente zuzugreifen. Mit der Funktion *styledText()* wird der *SWTBot* auf das richtige Feldelement delegiert.

Das *ComperatorObject* enthält die Vergleichsfunktion. Ein Defizit dieser Funktion ist, dass der komplette Fensterinhalt übergeben wird. Die Funktion überprüft die Liste auf eine bestimmte Wortfolge und liefert den Wert *True,* wenn diese enthalten ist. Wird der Test mehrmals ausgeführt und liefert der erste Test den Wert *True*, wird jeder folgende Test dies auch tun, da sich diese Wortfolge in der Liste befindet, da das Fenster nach einem Sitzungsende nicht gelöscht wird.

Da allerdings die Testumgebung nach jedem Test zurück auf den Urzustand gebracht wird, fällt dies nicht weiter ins Gewicht.

Nachdem die Textnachricht gelesen werden kann, wird nun noch eine Funktion zum Versenden benötigt. Diese bekommt als Parameter nur einen *String* übergeben. Da wir zurzeit nur einen allgemeinen Chatraum besitzen, wird bei dieser Funktion keine Jabber-ID benötigt.

```
public void sendChatMessage(String string) throws RemoteException;
```

Abbildung 5 Definition der sendChatMessage Funktion

```
public void sendChatMessage(String message) throws RemoteException {
        if (!isChatViewOpen())
            openChatView();
        activateChatView();
        SWTBotPreferences.KEYBOARD_LAYOUT = "EN_US";
        SWTBotView view = delegate.activeView();
        SWTBot bot = view.bot();
        SWTBotText text = bot.text();
        text.setText(message);
        text.pressShortcut(0, SWT.CR);
    }
```

Abbildung 6 sendChatMessage Funktion

Auch bei dieser Funktion muss am Anfang geprüft werden, ob das Chatfenster geöffnet ist und diese danach aktiviert werden. Die Anfangsschritte sind identisch der Vergleichsfunktion und bedürfen keiner weiteren Erläuterung. Zum Versenden einer Nachricht sind dann wieder einige Schritte zu beachten.

Ein Nachricht wird nach dem Betätigen der *Enter* Taste gesendet. Dafür muss mit der *pressShortCut()* Funktion ein Tastendruck simuliert werden. Dieses Signal ist von dem Tastaturlayout abhängig. Das Layout wird vom *SWTBot* am Anfang bestimmt. Zurzeit unterstützt dieser nur das Layout der *EN_US*. Sind andere gewünscht, muss ein Entsprechendes definiert werden oder die Variable *SWTBotPreferences.KeyboardLayout* manuell gesetzt werden.

Im Fall des *SWTBots* wurde die Variable manuell gesetzt, da nur eine Taste simuliert werden soll.

3.2.3 Bekannte Probleme des SWT-Bots

Die Verwendung des *SWBots* bringt drei wesentliche Probleme mit sich. Das erste Problem ist die fehlende Unterstützung von *Native Dialogs*[7]. Dazu zählt zum Beispiel der *FileChooser,* der bei der Auswahl von zu versendenden Dateien Verwendung findet, oder die *MessageBox,* die für die Anzeige von „Datei versenden" Anfragen zuständig ist.

In der *SWTBot* Gemeinde wurde ein *Workaround*[8] für einige dieser Dialogfenster entwickelt. Eine Verwendung dieser Bibliotheken halte ich allerdings für nicht nötig, da die wenigen NativeDialog Fenster, die in Saros Verwendung finden, das Einfügen dieser Bibliothek nicht rechtfertigen. Durch wenige Codezeilen kann das Problem einfach umgangen werden. Bei dem Entwurf der Tests ist die Entscheidung zu treffen, ob ein Überarbeitung der Komponente gemacht wird, oder für die Tests ein neues Dialogfenster implementiert werden muss. Im Fall der MessageBox kann das Problem durch eine Überarbeitung behoben werden. Hierbei wird diese durch ein *MessageDialog* ersetzt, welche vom *SWTBot* erkannt wird und so in den Tests ansprechbar ist. Für die Auswahlfenster existiert keine Möglichkeit einer Überarbeitung. Hier sind für die Testfälle Ergänzungen an Saros vorzunehmen. Für den Testmodus kann das Dialogauswahlfenster durch einen Texteingabedialog ersetzt werden. Dort kann zum Beispiel der absolute Dateipfad vom *SWTBot* eingefügt werden. Der Nachteil dieser Änderung besteht darin, dass nicht die eigentliche Dateiauswahl getestet wird und so deren Funktionsfähigkeit nicht sichergestellt werden kann.

Ein weiteres Problem stellt die Zeitabhängigkeit dar. Bei der Entwicklung von Testfällen ist aufgefallen, dass das Ausführen eines Tests auf verschiedenen Rechnern zu verschiedenen Ergebnissen führte. Im Wesentlichen hatte das zwei Ursachen. Entweder der Test wurde durch eine *TimeoutException* abgebrochen oder eine für den Ablauf wichtige Bedingung war nicht erfüllt.

Tests haben gezeigt, dass diese Fehlermeldung auftritt, wenn die zu verwendende Sichtfenster nicht geöffnet wurde oder nicht mehr fokussiert ist. Das letztere Problem tritt vor allem bei *Balloon Notifications*[9] auf.

---

[7] *http://wiki.eclipse.org/SWTBot/FAQ#How_do_I_use_SWTBot_to_test_native_dialogs_.28File_Dialogs.2C_Color_Dialogs.2C_etc.29.3F* , Stand November 2010

[8] Ein *Workaround* bezeichnet das Umgehen von bekannten Problemen.

[9] *Balloon Notifications* sind ereignisabhängig Nachrichten. Diese werden in einem eigenen Sichtfeld angezeigt.

Das Problem der nicht erfüllten Bedingungen wird durch ein zu schnelles Ausführen der Tests verursacht. Nachvollziehbar ist dies am *ChatMessageTest*. Hier muss der *SWTBot* auf eine Nachricht warten, bis er den Test ausführen kann. Diese Wartezeit ist abhängig von der Latenz der Netzwerkverbindung. Zum jetzigen Zeitpunkt wird gewartet, bis eine *TimeoutException*[10] ausgelöst wird. Bei weiteren Überarbeitungen an der Testumgebung ist hier eine Fehlermeldung zu finden, die dem Tester eine eindeutigere Fehlermeldung liefert.

3.3 Arbeiten an der Testumgebung

Während meiner Arbeit im Saros Team habe ich an der Testumgebung mitgearbeitet. Mit dem neu eingeführten *Buddy-System*[11] habe ich der Studentin Lin Chen Hilfestellungen bei der Überarbeitung der Testumgebung gegeben. Hierbei habe ich bei dem Erstellen von Testfällen und der Suche von Fehlern im Testsystem mitgewirkt.

Während der Mitarbeiter Florian Thiel die Umstellung der Testumgebung und der Saros-Server vornahm, arbeiteten die Studentin Lin Chen und ich an neuen Testfällen. Zu dieser Zeit wurden die Testfälle nur lokal getestet, was später zu Problemen führte.

Wir beide arbeiteten mit verschiedenen Betriebssystemen. Dadurch wurden Probleme des *SWTBots* mit unterschiedlichen Betriebssystemen aufgedeckt. Die Tests wurden nach der Fertigstellung auf einem Windows- und einem OSX System getestet. Damit konnten einige betriebssystembedingte Probleme aufgedeckt werden.

Ich habe an den folgenden Tests gearbeitet:

- Invitation
- Stresstest
- Screensharing
- TestChat

[10] Die *TimeOutException* wird ausgelöst, wenn eine SWT-Komponente nach einer bestimmten Zeit nicht ausgeführt werden konnte.
[11] Das *Buddy-System* teilt die Projektmitglieder in zweier Gruppen ein. Diese Gruppen unterstützen sich gegenseitig bei der Entwicklung. Das Gruppenmitglied ist erster Ansprechpartner bei Durchsichten von Patches.

Alle Tests liefen lokal fehlerfrei. Zu Problemen kam es erst bei dem Testen auf der neuen Serverumgebung. Hier wurden die Schwächen von Saros auf dem Betriebssystem Ubuntu ersichtlich.

Auf den Servern war eine Version von Ubuntu-Server 64-Bit installiert. Die ersten Testversuche fielen negativ aus und machten eine Untersuchung des Problems erforderlich.

Aus vorhergehenden Tests war bekannt, dass es Verzögerungen bei dem Herstellen einer Saros-Sitzung unter Ubuntu gibt. Die Umstellung auf *In-Band Bytestream* führte nicht zu einer Lösung des Problems. Bei der Herstellung einer Sitzung zwischen den beiden Ubuntu Systemen kam es unregelmäßig zu einem *Deadlock*[12].

Zur Lösung des Problems haben wir uns ein eigenes Testsystem auf einer *Virtuelbox*[13] installiert. Getestet wurde als Erstes, ob es sich um ein generelles Problem mit Linuxsystemen handelte, oder das Problem auf Ubuntu beschränkt ist. Dazu wurde auf einer virtuellen Maschine Debian-Server 64-Bit installiert und eingerichtet. Tests auf diesem System haben die üblichen Verzögerungen von Saros gezeigt, dennoch konnte bei mehreren Tests eine Verbindung hergestellt werden.

Als nächstes wurden Tests an Systemen mit Ubuntu 32-Bit Server und Clients, sowie an Ubuntu 64-Bit Server und Clients durchgeführt. Die 32-Bit Varianten zeigten die gleichen Eigenschaften wie bei den Tests mit Debian. Es konnte eine Verbindung hergestellt werden, mit einer Verzögerung von 1 bis 3 Minuten.

Etwas anders sah es bei den 64-Bit Varianten aus. Hier kam es unregelmäßig zu Problemen beim Einladungsprozess. Erste Vermutungen, das Problem könnte am installierten *openJDK*[14] liegen, wurden nicht bestätigt. Auch nach Installation Java-Version von Oracle trat das Problem auf. Bei weiteren Tests fiel auf, dass die Verzögerungen bei der Annahme einer Sitzung auftraten. Es wurde herausgefunden, dass das benötigte Bestätigungspaket vom Client an den Server nicht ankam. Dieses Problem wurde vom Diplomstudenten Michael Jurke behoben. Er änderte den Einladungsprozess derartig, dass ein Bestätigungspa-

[12] Ein *Deadlock* bezeichnet das Warten auf einen Bedingung die niemals eintritt.
[13] *Virtuelbox* ist eine Visualisierungssoftware für Linux, Windows und OSX. Online Referenz: http://www.virtualbox.org/, Stand 1.11.2010
[14] *openJDK* ist eine freie Implementierung von Java. Online Referenz: http://openjdk.java.net/, Stand 1.11.2010

ket vor der Annahme gesendet wird. Nach dieser Änderung konnten die Fehler nicht mehr festgestellt werden.

4 Arbeiten an der Netzwerkschicht

In diesem Kapitel erläutere ich meine Arbeit an der Netzwerkschicht. Zunächst möchte ich auf die Ausgangssituation und vorangegangene Arbeiten eingehen. Am Ende dieses Kapitels werde ich noch Anregungen für weitere Arbeiten an der Netzwerkschicht geben.

4.1 Motivation

Zu Beginn meiner Arbeit hatte Sandor Szücs seine Arbeit an der Netzwerkschicht abgeschlossen. Saros benutzt für die Kommunikation die *Open Source* Bibliothek *Smack*[Sma10]. Diese Bibliothek dient dem Erstellen von XMPP-Clients. Die darin enthaltenen Kommunikationsprotokolle SOCKS5[SMSAK10], Jingle[SA10a] und *In-Band Bytestream*[KSA10] werden zum Datenaustausch in Saros verwendet. Das Jingle Protokoll war auf zwei Arten implementiert. Es wurde periodisch wechselnd versucht eine TCP oder eine UDP Verbindung aufzubauen. Da die Jingle Spezifikation nur eine TCP Verbindung vorsieht wurde mit einer RUDP-Bibliothek[Inet99] eine solche simuliert.

Die RUDP Verbindung sollte zum Herstellen einer Direktverbindung zwischen zwei Rechnern hinter einem NAT verwendet werden.

Während meiner ersten *Release* Woche wurden verschiedene Verbindungstests durchgeführt. Die Tests sind mehrmalig ausgeführt worden und es wurden folgende Häufigkeiten bei den Verbindungsarten festgestellt:

Verbindungsart	Häufigkeit	NAT	Betriebssystem
Jingle-TCP	oft	nein	Windows
Jingle-UDP	selten	ja	Windows
SOCKS5	nie	ja	Windows
In-Band Bytestream	oft	ja	Windows

Tabelle 1 Verbindungsarten während einer Sitzung

Aus der Tabelle 1 wird deutlich, dass im Regelfall nur eine Jingle-TCP und eine *In-Band Bytestream* Verbindung aufgebaut wurde. Zu meiner Arbeit gehörten das Untersuchen dieses Problems, sowie eine mögliche Behebung.

4.2 Untersuchung der Verbindungsprotokolle

Der Aufbau der Netzwerkschicht wird in der Arbeit vom Studenten Szücs[Szu10] ausführlich erklärt. Die Arbeitsweise der einzelnen Protokolle wird vom Diplomstudenten Staib[Sta10] ausführlich dargestellt. Ich möchte hier auf die Fehlersuche bei den einzelnen Protokollen eingehen.

Für eine Eingrenzung von Fehlerquellen wurden die Netzwerkarten und die Betriebssysteme als feste Parameter gewählt. Diese Festlegung sollte helfen, die Fehlerquellen etwas einzuschränken. Die Tests sollten auch zur Ermittlung der Verbindungsaufbaudauer dienen. In der folgenden Tabelle 2 werden die Untersuchungsergebnisse dargestellt.

Verbindungsart	Betriebssysteme	Initiierungszeit	stabil	Nat	Häufigkeit
Jingle-TCP	WIN / WIN	ca. 3 min	Ja	Nein	Immer
Jingle -TCP	OSX/WIN	ca. 5 min	Ja	Nein	Überwiegend
Jingle -TCP	OSX/OSX	ca.4 min	Ja	Nein	Immer
SOCKS5	WIN/OSX/Linux	unendlich	Nein	Ja/Nein	Nie
Jinlge-UDP	WIN/WIN	ca. 4 min	Nein	Ja	Überwiegend
Jingle-UDP	OSX/OSX	ca. 6 min	Nein	Ja	Überwiegend
Jingle-UDP	OSX/WIN	ca. 4 min	Nein	Ja	Selten

Tabelle 2 Protokollarten und deren Häufigkeit beim Aufbau: Überwiegend entspricht etwa 70 Prozent der aufgebauten Verbindungen; Selten entspricht ca. 30 Prozent der aufgebauten Verbindungen

Bei der Betrachtung der Tabelle fällt das Fehlen des *In-Band Bytestream* Protokolls auf. Dieses Protokoll wird von Saros als letzte Möglichkeit verwendet. Das heißt, immer dann, wenn keine Verbindung mit einem anderen Protokoll aufgebaut werden kann. Des Weiteren fällt auf, dass nie eine SOCKS5 Verbindung aufgebaut werden konnte und eine Jingle-*UDP* Verbindung nicht stabil aufrechterhalten werden konnte.

Als Ersten komme ich zum Jingle-UDP Protokoll. Dieses Protokoll sollte helfen, mithilfe von UDP Packeten eine Verbindung zwischen Rechnern hinter einem NAT aufzubauen.

Bei der Implementierung dieses Verbindungsprotokolls wurde nicht auf die Spezifikation geachtet. Gründe dafür könnten die unzureichende Spezifikation und das Fehlen des Protokolls in der Bibliothek Smack sein. Saros versuchte, eine RUDP-Verbindung zwischen zwei Rechnern aufzubauen und darauf das Jingle-TCP anzuwenden. Die RUDP-Verbindung garantierte eine für Saros wichtige Bedingung – eine gesicherte Reihenfolge beim Versenden und Empfangen von Paketen. Zur Aufrechterhaltung der Paketreihenfolge wurde jedes Packet bestätigt. Fehlte diese Bestätigung oder kam ein Paket vor einem anderen an, wurde das Paket neu gesendet. Dies führte in vielen Fällen zu einem unkontrollierten Versenden von Paketen und hatte eine Reduzierung des Paketdurchsatzes zur Folge. Bei zu starkem Zirkulieren brach die Verbindung ab.

Ein weiterer Nachteil dieser Verbindung war die Ermittlung der benötigten IP-Adressen. Diese sollten mithilfe eines JSTUN-Servers[Kin05] ermittelt werden. Die Implementierung in der Bibliothek wies die Schwäche auf, dass bei mehreren Netzwerkkarten in den meisten Fällen die falsche Adresse ermittelt wurde, und damit keine Verbindung aufgebaut werden konnte.

Die Missachtung der Spezifikation und die Fehler in der verwendeten Bibliothek stellten die Verwendung dieses Verbindungstyps in Frage.

Als nächstes möchte ich zum SOCKS5 Protokoll kommen. Wie aus der Tabelle ersichtlich, konnte am Anfang nie eine solche Verbindung hergestellt werden. Hier hat man sich bei der Implementierung zu sehr auf die Bibliothek verlassen. Diese sollte selbständig das mögliche Protokoll auswählen. Ein falsch gesetztes *TimeOut*[15] und fehlerhafte Implementierung machte es jedoch unmöglich, damit eine Verbindung aufzubauen.

[15] *TimeOut* ist eine zeitabhängige Abbruchbedingung.

4.3 Änderungen an der Netzwerkschicht

Wie im vorhergehenden Kapitel erwähnt, gab es bei Saros einige Probleme bei dem Aufbau einer Verbindung. Damit Saros für Firmen interessant werden konnte, mussten diese Probleme behoben werden. Der Student Henning Staib untersuchte in seiner Arbeit die Probleme in der Bibliothek Smack und implementierte das Protokoll SOCKS5 und *In-Band Bytestream* neu. Damit war es nun möglich, auf die Protokolle einzeln zuzugreifen. Zur besseren Handhabung stellte er den Entwicklern ein Schnittstelle zur Verfügung.

Mit diesen Erneuerungen wurde eine Überarbeitung der Netzwerkschicht beschlossen. Während meiner Zeit bei Saros habe ich daran mitgewirkt. Ich möchte nun die Veränderungen darstellen.

Teil meiner Aufgabe war der Überarbeitung der Jingle Implementierung . Als erster Schritt wurde Jingle-UDP entfernt. Der Grund dafür lag vor allem in der Unzuverlässigkeit. Saros benötigt zum Versenden von großen Datenmengen eine stabile Socket-Verbindung. Diese sollte, wenn möglich, während der ganzen Sitzung aufrechterhalten werden. Das war die Grundlage für weitere Erneuerungen am Programm, wie z.B. das Versenden von Daten, VoIp oder *Video Sharing*. Bei Verlust einer Verbindung wurde immer wieder eine neue aufgebaut. Es konnte nicht sichergestellt werden, dass die alte Verbindungsart wieder verwendet wurde.

Als nächster Schritt wurden die vorhandenen Klassen vereinfacht. Hierdurch konnten unnötige Abhängigkeiten entfernt und damit der Code lesbarer gemacht werden.

Während meiner Arbeit an Jingle arbeitete der Student Jurke an einer richtigen Implementierung des SOCKS5 Protokolls. Mit seiner Hilfe wurden die Protokollklassen besser strukturiert, damit sie für andere Anwendungen benutzbarer sind.

Nach den Überarbeitungen konnten folgenden Änderungen erzielt werden:

- Saros kann eine Verbindung mit SOCKS5 direkt oder vermittelnd aufbauen und stabil aufrechterhalten.
- Die *In-Band Bytestream* Klassen wurden drastisch reduziert und damit handhabbarer und lesbarer gemacht.
- Saros verwendet bis auf weiteres nur noch SOCKS5 und *In-Band Bytestream* Verbindungen.

Aus dem letzten Punkt wird deutlich, dass im Moment das Jingle Protokoll nicht verwendet wird. Der Grund dafür liegt darin, dass die JSTUN Implementierung der Smack[Sma10] Bibliothek zurzeit die falschen IP-Adressen zurück liefert, was einen Verbindungsaufbau verhindert. Bis keine Änderungen an der JSTUN Implementierung in der Bibliothek vorgenommen wurden, wird bei Saros auf Jingle verzichtet.

Dennoch ermöglicht diese Überarbeitung bei zukünftigen Änderungen ein leichteres Verstehen des Protokolls.

Die **Abbildung 7** zeigt die aktuell verwendeten Übertragungsprotokolle und die unterschiedlichen Wege des Datenstroms.

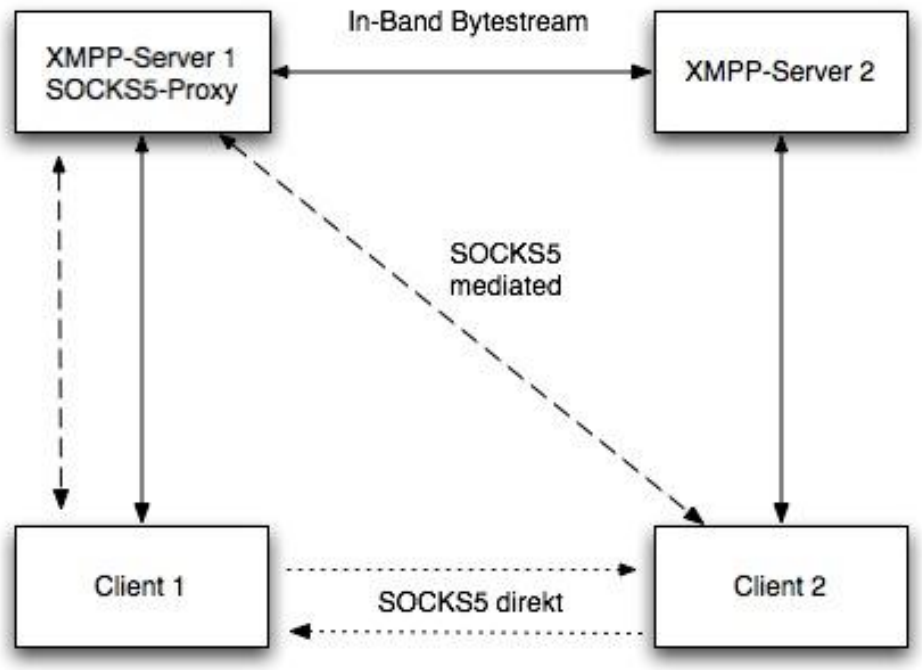

Abbildung 7 Datenübertragungswege von Saros.

4.4 Behebungen von Defekten bei Saros

Die Arbeit bei Saros beinhaltete auch die Untersuchung und Behebung von Defekten. In diesem Kapitel möchte ich auf einige eingehen.

4.4.1 Wiederverbindungsproblem (Reconnection Problem)

Aus verschiedenen Gründen konnte es passieren, dass die Verbindung zwischen dem Klienten und dem Host abbricht, aber sehr schnell wieder hergestellt werden konnte. Saros konnte diesen Verlust auf zwei Arten registrieren. Erstens zeigte ein Listender den Verlust der Direktverbindung und zweitens wurden die Pakete während der Abwesenheit für eine bestimmte Zeit an den XMPP-Server gesandt und dort zwischengelagert. Nach dem Ablauf dieser Zeit wurde der betreffenden Teilnehmer aus der Sitzung entfernt.

Hierzu muss darauf hingewiesen werden, dass die Aktivitätsnachrichten als XMPP-Nachricht versendet werden, und dabei nicht geprüft wird, ob sich noch ein Teilnehmer in der Sitzung befindet.

Die Ursache dieses Defektes lag in der langen Zeitspanne zwischen der Kenntnisnahme eines Teilnehmerverlustes und der Entfernung dieses Teilnehmers. Wie weiter oben beschrieben, sendet Saros die Aktivitätsnachrichten ohne weitere Prüfung der Zugehörigkeit zu einer Sitzung an jeden Teilnehmer. Ist dieser unerwartet nicht mehr erreichbar, werden die Nachrichten für einen bestimmten Zeitraum gepuffert. Nach Ablauf dieser Zeit wurde der Teilnehmer aus der Sitzung entfernt. Gelang es dem Teilnehmer schnell genug, wieder online zu sein, wurden die gepufferten und die neuen Nachrichten an diesen wieder übertragen. Saros erkannte also nicht, dass der Teilnehmer sich nicht mehr in der Sitzung befand und entfernte ihn aus diesem Grund auch nicht aus dieser. Damit konnte dieser auch nicht wieder eingeladen werden.

Wurde die Verbindung zum Host verloren, befanden sich die beiden Klienten noch in der Sitzung, aber durch das Fehlen von Rechten konnten diese auch keinen einladen.

Hierzu ist zu bemerken, dass alle logischen Angelegenheiten einer Sitzung bei dem Host stattfinden. Er allein hat das Recht, jemanden einzuladen. Alle betreffenden Änderungen werden über seinen Rechner gesteuert.

Nach Absprache mit dem Betreuer habe ich ein PingPong-System implementiert. Damit sollte die Erreichbarkeit der Sitzungsteilnehmer überprüft werden.

Als erstes musste die Entscheidung getroffen werden, ob die Überprüfung zwischen allen Teilnehmern stattfinden soll, oder ob es reicht, wenn nur der Host dies übernimmt. Ich habe mich für das Letztere aus folgenden Gründen entschieden. Die logische Steuerung einer Sitzung findet beim Host statt. Ist dieser nicht mehr erreichbar, kann die Sitzung beendet werden. Des Weiteren wird damit auch der Datenverkehr verringert, da nur einer Daten in periodischen Abständen sendet. Als Nächstes musste der Übertragungsweg festgelegt werden. Diese Entscheidung war leicht zu fällen, da ich eine Pufferung der Daten vermeiden wollte. Die Anfrage wird also als Nachrichtenpaket über die Direktverbindung gesendet. Dies geschieht alle 30 Sekunden und wird von den Klienten mit einer XMPP-Nachricht beantwortet. Meldet sich der Host nach 30 Sekunden nicht mehr, wird die Sitzung beendet. Die Clients haben 15 Sekunden Zeit, sich zu melden. Geschieht das nicht, werden sie aus der Sitzung entfernt.

In beiden Fällen ist es nun möglich, die Sitzung ordnungsgemäß zu beenden oder bei Verlust einen Sitzungsteilnehmer wieder einzuladen.

4.4.2 Überarbeitung von JUnit4 Testfällen

Während der Veröffentlichungswoche werden die vorhandenen JUnit4-Testfälle ausgeführt. Mit diesen Testfällen sollen ausgewählte Komponenten von Saros auf ihre Funktionstüchtigkeit getestet werden.

In Abhängigkeit der Leistungsfähigkeit einzelner Rechner werden diese Testfälle unterschiedlich schnell ausgeführt. Dieser Unterschied führt gerade bei den Testfällen für den Jupiter Algorithmus zu Problemen. Die Testfälle geben bei falscher Anwendung oder einer Verzögerung im Testverlauf ein falsches Ergebnis aus.

Während einer Veröffentlichungswoche sind wegen falscher Ausführung alle JUnit4-Testfälle fehlgeschlagen. Nach Rücksprachen konnte ich auf die Zeitabhängigkeit der Tests aufmerksam machen. Ein wiederholtes Ausführen brachte die richtigen Ergebnisse.

Im Zuge dieser Tests wurde beschlossen, die regelmäßig fehlschlagenden Testfälle zu überarbeiten. Gemeinsam mit dem Studenten Andreas Haferburg sind die fehlgeschlagenen Tests untersucht und durch wenige Änderungen korrigiert worden.

An der Zeitabhängigkeit einzelner Testfälle konnte nichts geändert werden. Diese geben nach korrekter Handhabung die richtigen Werte aus.

4.4.3 Beitrag zu Saros

Die Arbeit im Saros-Projekt besteht aus drei wesentlichen Säulen. Als erste Säule ist die eigene Arbeit im Projekt zu sehen, gefolgt von der Einnahme einer Managerrolle, der Untersuchung und Behebung von auftretenden Defekten, sowie die Durchsicht von bereitgestellten Patches.

Eine wesentliche Rolle im Projekt wird der Durchsicht von Patches zugemessen. Dieses liegt darin begründet, dass diese erst in das Projekt einfließen, wenn sie von anderen Projektmitgliedern geprüft wurden. Das verhindert zum einen das Auftreten neuer Defekte, die Einhaltung von Konventionen und zum anderen, ob sie das Problem auch lösen. Ein nicht unwesentlicher Punkt der Durchsicht ist das Kennenlernen des Saros-Codes. Durch Durchsichten lernt jedes Gruppenmitglied unterschiedliche Teile von Saros kennen.

Es ist vorgesehen, dass ein Patch von zwei Projektmitgliedern überprüft und von diesen mit einer „+1“ freigegeben wird. Bei negativer Beurteilung muss dieser noch einmal verbessert und für eine erneute Durchsicht eingereicht werden.

Die folgende Tabelle zeigt die von mir beurteilten Patches mit den dazugehörigen Beurteilungen und dem Themengebiet, die sie behandelt.

Patch	Autor	Bewertung	Anmerkungen
Generalise the roster menu invitation	Karl Beecher	-1, +1	Erlaubt die Einladung eines Benutzers zu einer neuen Sitzung.
User deselect when session view loses focus	Karl Beecher	+1	Dieser Patch hebt die Auswahl eines Benutzers auf, sobald der Fokus nicht mehr auf der *Session-View* ist.
Two Chat Fixes	Andreas Haferburg	+1	Startet den Chat im Hintergrund und versucht erst einem beizutreten, bevor ein neuer erstellt wird.
Exception in Saros#Stop	Andreas Haferburg	+1	
Inviting peer without saros does not cause an error	Michael Jurke	+1	

5 Zusammenfassung

Mit dieser Arbeit wurde ein Beitrag zur Verbesserung der Netzwerkschicht und zur Erweiterung der Sandor Testumgebung geleistet.

Es wurden Defekte der Netzwerkschicht untersucht und behoben. Die Schwierigkeiten der Untersuchungen lagen in der Reproduzierbarkeit der Defekte. Diese Defekte wurden in den meisten Fällen nur bei bestimmten Konstellationen Netzwerkeigenschaften und Betriebssystemen sichtbar.

Es war ersichtlich, dass Funktionen von Saros oft nur lokal oder unter einem Betriebssystem getestet wurden. Hierdurch konnten Netzwerk- oder Betriebssystem bedingte Defekte nicht frühzeitig erkannt werden. Das führte während der Arbeit zur andauernden Fehlersuche.

Für weitere Entwicklungen an Saros sollte zum Testen auf die Testumgebung zurückgegriffen werden. Hier können verschiedene Netzwerkeigenschaften simuliert und mit einem Linux Betriebssystem die Funktionen unter erschwerten Bedingungen getestet werden. Untersuchungen haben gezeigt, dass Funktionen, die unter Linux funktionieren, auch unter den gängigen Betriebsystemen Windows und OSX richtige Ergebnisse liefern. Der Weg, die Tests in regelmäßigen Abständen automatisch ablaufen zu lassen, sollte beibehalten werden. Hiermit kann schneller auf eventuelle Fehler reagiert werden, da diese nicht erst im schlimmsten Fall in der *Release*-Woche auftreten. Das würde auch die Arbeits- und Zeitbelastung der Entwickler in dieser Zeit enorm verringern.

Für eine Verbesserung der Netzwerkeigenschaften von Saros könnte eine weitere Untersuchung der Smack Bibliothek hilfreich sein. Die Untersuchung sollte sich vor allem mit dessen Kompatibilität mit verschiedenen Betriebssystemen befassen. In den Untersuchungen wurde ersichtlich, dass auftretende Netzwerkprobleme, wie zum Beispiel, unerklärliche Verzögerungen von bis zu vier Minuten, auf diese zurückzuführen sind.

6 Literatur

6.1 Zitierte Quellen

[Bec99] Kent Beck: *extreme Programming explained – Embrace Change.* In: Addison Wesley, September 1999

[BN08] Andrew Begel, Nachiappan Nagappan: *Pair Programming: What's in it for me?.* In: ESEM `08: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement: Seiten 120-128, Oktober 2008

[BWGSS02] Prashant Baheti, Lauri A. Williams, Edward Gehringer, David Stotts, Jason McC. Smitsch: *Distributed Pair Programming: Empirical Studies and Supporting Environments.* In: Technical Report TR02-010, Dep. of Computer Science, Univ. of North Carolina at Chapel Hill, 2002

[CW01] Alistair Cockburn, Lauri Williams: T*he Costs and Benefits of Pair Programming.* In: In eXtreme Programming and Flexible Processes in Software Engineering XP2000: Seiten 223-247, Addison-Wesley, 2000

[Dje06] Riad Djemili: *Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierung verteilter Paarprogrammierung.* In: Diplomarbeit, Freie Universität Berlin, Institut Informatik, August 2006

[Nos98] John T. Nosek: *The Case for Collaborative Programming.* In: Communications of the ACM, Vol. 41, No. 3, März 1998

[NW01] Jerzy Nawricki, Adam Wojciechowski: *Experimental Evaluation of Pair Programming.* In: Proceedings of the 12th European Software Control and Metrics Conference, Seiten 269-276, London, UK, 2001

[NJOL05] Jerzy Nawricki, Michal Jasninski, Lukasz Olek, Barbara Lange: *Pair Programming vs. Side-by-Side Programming.* In: Software Proecess Improvement, Vol. 3792 of Lecture Notes in Computer Science, Seiten 28-38, Springer-Berlin/Heidelberg, 2005

[Rie08] Oliver Rieger: *Weiterentwicklung einer Eclipse Erweiterung zur Realisierung und Protokollierung Verteilter Paarprogrammierung im Hinblick auf Kollaboration und Kommunikation.* In: Diplomarbeit, Freie Universität Berlin, 2008.

[Rin09] Marc Rinsch: *Agile Weiterentwicklung eines Woftware-Werkezeuges zu verteilten, kollaborativen Programmierung in Echtzeit.* In: Diplomarbeit, Freie Universität Berlin, 2009

[SOBS10] Stephan Salinger, Christopher Oezbek, Karl Beecher, Julia Schenk: *Saros: An Eclipse Plug-in für Distributed Party Programming.* In: CHASE 2010'10, Cape-Town, South Arfrica, , May 2-8 2010

[SST09] Peter Saint-Andre, Kevin Smith, Remko Troncon: *XMPP – The Definitve Guide Building Real Time Applications with Jabber Technologies.* In: O'Reilly Media, Inc., 2009

[Sta10] Henning Staib: *Verbesserung einer XMPP-Bibliothek für den Einsatz in verteilter Paarprogrammierung.* In: Diplomarbeit, Freie Universität Berlin, 2010

[Szü10] Sandor Szücs: *Behandlung von Netzwerk -und Sicherheitsaspekten in einem Werkzeug zur verteilten Paarprogrammierung.* In: Diplomarbeit, Freie Universität Berlin, Februar 2010

[Wil00] Laurie A. Williams: *The Collaborative Software Process.* In: Dissertation, The University of Utah, Department of Computer Science, August 2000

[WKWJ00] Laurie A. Williams, Robert R. Kessler, Ward Cunningham, Ron Jeffries: *Srenghthening the case for pair Programming.* In: IEEE Software, 17(4):Seiten 19-25, 2000

[Zil09] Sebastian Ziller: *Behandlung von Nebenläufigkeitsaspekten in einem Werkzeug zur Verteilten Paarprogrammierung.* In: Diplomarbeit, Freie Universität Berlin, 2009

6.2 Internet Quellen

[Ecl10] Eclipse: quelloffene und frei Entwicklungsumgebung. Online Referenz: www.eclipse.org, Stand 01.11.2010

[FSK05] Bryan Ford, Pyda Srisuresh, Dan Kegel: *Peer-to-Peer Communication Across Network Address Translation.* Online Referenz: http://bford.info/pub/net/p2pnat, 2005

[KSA10] Justin Karneges, Peter Saint-Andre: *XEP-0047: In-Band Bytestreams.* Online Referenz: http://xmpp.org/extensions/xep-0047.html, 2010

[Int04a] Internet Engineering Task Force; *Extensible Messaging and Presence Protocol (XMPP): Core.* Online Referenz: http://tools.ietf.org/html/rfc3920, 2004

[Int04b] Internet Engineering Task Force: *Extensible Messaging and Presence Protocol(XMPP): Instant Messaging and Presence.* Online Referenz: http://tools.ietf.org/html/rfc3921, 2004

[Int99] Internet Engineering Task Force: *Reliable UDP Protocol.* Online Referenz: http://tools.ietf.org/html/draft-ietf-sigtran-reliable-udp-00, 1999

[Jun10] JUnit4. *Resources for Test Driven Development.* Online Referenz: http://www.junit.org/, Stand 1.11.2010

[Kin05] Thomas King: *„JSTUN" – Java Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translation(NAT).* Online Referenz: http://jstun.javawi.de/, 2005

[SA10a] Peter Saint-Andre : *XEP-0234: Jingle File Transfer.* Online Referenz: http://xmpp.org./extensions/xep-0234.html, 2010

[Sar10a] Saros-Projekt: *Our Rlease Schedule.* Online Referenz: http://www.saros-project.org/HowToDevel#Our_Release_Schedule, Stand 22.11.2010

[Sar10b] Saros-Projekt Seite. Online Referenz: http://www.saros-project.org/, Stand 1.11.2010

[Sma10] Smack: Open-Source XMPP-Bibliothek. Online Referenz: http://www.igniterealtime.org/projects/smack/, Stand 01.11.2010

[SMSAK10] Dave Smith, Matthew Miller, Peter Saint-Andree, Justin Karneges: *XEP-0065: SOCKS5 Bytestreams.* Online Referenz: http://xmpp.org/extensions/xep-0065.html, 2010

[Sor10] Sourceforge: Portal zur Verwaltung des Saros-Projektes: http://sourceforge.net/projects/dpp/, Stand 01.11.2010

[Wik10] Wikipedia.org: *Reliable User Datagram Protocol.* Online Referenz: http://en.wikipedia.org/wiki/Reliable_User_Datagram_Protocol, Stand 01.11.2010

Abkürzungen

PP	Pair Programming
DPP	Distributed Pair Programming
SbS	Side-by-Side Programming
TM	Testmanager
ATM	Assistant Testmanager
RM	Realese Manager
ARM	Assistant Release Manager
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
RUDP	Reliable User Datagram Protocol
JID	Jabber-ID
SWT	Standart Widget Toolkit
NAT	Network Address Translation

Eidesstattliche Erklärung

Ich versichere, dass ich meine Bachelorarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Berlin, den 23.11.2010

Nachname: Bauch Vorname: Sebastian

Matrikelnummer: 3986983 Unterschrift: _____