Freie Universität Berlin

Diplomarbeit zur Erlangung des akademischen Grades
Diplom-Informatiker

Behandlung von Netzwerk- und Sicherheitsaspekten in einem Werkzeug zur verteilten Paarprogrammierung

Sandor Szücs

Matrikelnummer: 3979980

sandor.szuecs@fu-berlin.de

Berlin, 28. Februar 2010

Department of Mathematics and Computer Science
Institute for Computer Science
Software Engineering Working Group

Responsible University Professor: Prof. Dr. Lutz Prechelt
Supervisor: Christopher Özbek and Stephan Salinger

Inhaltsverzeichnis

1 Einleitung **8**
1.1 Typografische Konvention 9

2 Kontext der Arbeit: Verteilte kollaborative Softwareentwicklung mit Saros **10**
2.1 Theoretische Grundlage 10
2.2 Softwareprozess des Saros-Teams 13
2.3 Saros 14

3 Arbeit Teil 1: Verbesserung von Netzwerkschicht und Einladungsprozess **17**
3.1 Motivation 17
3.2 Netzarchitektur 18
3.3 Arbeiten 23
3.4 Ausblick 33

4 Arbeit Teil 2: Netzwerk Testumgebung **35**
4.1 Motivation 35
4.2 Anforderungen 36
4.3 Auswahl eines Testframeworks 37
4.4 Umsetzung 41
4.5 Anwendung 51
4.6 Fazit 53

5 Ausblick **55**

6 Zusammenfassung **56**

Literaturverzeichnis **61**

A Messungen **62**
A.1 Durchsatz Messung der Synchronisierung von Saros 62
A.2 Test verschiedener Jabber Server 63

B Konfigurationen der Rechner **64**
B.1 FreeBSD 6.4 64
B.2 Debian Lenny 68
B.3 Kubuntu 69

C Testframework **73**
C.1 Verwendung des Testframeworks 73
C.2 E-Mail Konversation 80

Tabellenverzeichnis

4.1 grob klassifizierte Netzwerkszenarien zweier Teilnehmer einer Saros-Sitzung 37
4.2 Testframeworks 38
4.3 Annotationen und Ausführungszeiten in JUnit4 49

A.1 Durchsatzmessung bei der Synchronisierung. Suessapfel befand sich im WLAN Eduroam und brazzaville im Informatiknetz. 62

B.1 Die angegebenen Ports entsprechen den Portweiterleitungen für den Zugriff aus dem Informatiknetz. 64

Abbildungsverzeichnis

2.1 Zu sehen ist eine Saros-Sitzung mit 3 Benutzern (6). (5) zeigt eine Knopfleiste, in der der *Observermodus* aktiviert ist. Der aktuell verfolgte Benutzer ist in Fettbuchstaben geschrieben (6). Andere Teilnehmer sind im offenen Editorfenster mit unterschielichen Farben (1) und (2) dargestellt. Der Sichtausschnitt im Editor der anderen Teilnehmer kann im Balken rechts von (3) gesehen werden. (4) weist auf die Markierung einer von anderen aktuell geöffnete Datei im *Package Explorer* hin. 15

3.1 Zu sehen ist die Arbeitsweise von *DNS SRV*. Der *XMPP* Dienstanbieter `xmpp-server.l.google.com` wird durch eine *DNS SRV* Abfrage gefunden. 19
3.2 In Saros verwendete Dateitransfermöglichkeiten. (1) ist bevorzugt und baut eine *Peer-to-Peer* Verbindung auf. (2) hat nach Messungen ca. 80kB/s Durchsatz. Die Verfügabrkeit ist abhängig von der Serverkonfiguration. (3) hat nach Messungen nur 4kB/s Datendurchsatz. 20
3.3 Netzwerk Architektur beim Datenversand 21
3.4 Netzwerk Architektur beim Empfang von Daten 22
3.5 Schnittstelle zum Versand von Daten über den `DataTransferManager` 31
3.6 Schnittstelle zum Empfang von Daten über den DataTransferManager . 31

4.1 Komfortables konfigurieren von Netzwerkeigenschaften mit der Kombination *ipfw* und *dummynet*. 40
4.2 Orchester bestehend aus zwei Musikanten und einem Dirigent. Der Dirigent verwendet die von Musikanten angebotene Schnittstelle zur Steuerung. Ein Musikant ist ein Teil von Saros, falls Saros im Testmodus gestartet wird. 42
4.3 Das Bild zeigt den logischen Aufbau des virtuellen Testnetzwerks. Netzwerkverbindungen zwischen unterschiedlichen Netzen gehen über den Emulator. Der Emulator ist in der Lage Netzwerkeigenschaften zu bestimmen. 43
4.4 Emulator erzeugt eine *Pipe* mit den Eigenschaften Verzögerung, Bandbreiteneinschränkung und Paketverlust. Die Regel wird auf Netzwerkpakete in beide Richtungen von *$DELAY_NET* angewendet. Zwischen *Pipe* und Regelanwendung steht der Emulator Bypass, um in der Anwendungskette der Firewall vor der *Pipe* gefunden zu werden. 43
4.5 Die Änderung von einfachen Methoden am Beispiel *OK Button* drücken 45
4.6 Beispiel Signatur einer komplexen Methode 45
4.7 Erster Schritt der Annahme einer Einladung. Der *Button* mit der Aufschrift *Finish* ist nicht aktiviert. 45

4.8 Zweiter Schritt der Annahme einer Einladung. Der Button mit der Aufschrift *Finish* ist aktiviert. 45
4.9 Erster Schritt der Annahme einer Einladung. Der Bot wartet zur Synchronisierung auf einen, erst im nächsten Fenster aktivierten, Button. . . 46
4.10 Zweiter Schritt der Annahme einer Einladung. Der Bot wartet auf die Schließung des Fensters, was zu einer WidgetException beim Aufruf von `delegate.shell()` führen wird. 46
4.11 Auszug aus einer Testmethode. Die letzten Zeilen verifizieren aus `invitee` Perspektive, ob der `inviter` am Projekt teilnimmt und die Driverrolle hat. 47
4.12 Das Klassendiagramm zeigt die Klassen des Testframeworks und ihren Zusammenhang mit zwei verschiedenen Tests. 47
4.13 Konfiguration eines Musikanten für den Lauf im Testemulator innerhalb eines JUnit Tests. 49
4.14 Auszug einer Sitzung in `irc://freenode.net/#eclipse` 51

B.1 Netzwerkkonfigration von `xmpp1.internet.test` 69
B.2 DJBDNS (aka tinydns) Konfigurationsdatei 70
B.3 Erfolgreiche Tests der SRV Einträge für XMPP 70
B.4 Netzwerkkonfigration von `xmpp2.internet.test` 71
B.5 `start-rmiswtbot-project.sh`: Skript zum Starten von Saros im Testmodus 72
B.6 Veränderter Start der X-Sitzung für für Musikanten 72

C.1 Automatisch gesteuerte Situation einer Einladung im Testemulator . . . 74

Auflistungen

src/dnssrv_snippet 19
src/dtm_send.java 31
src/dtm_recv.java 31
src/delay_snippet 43
src/lowlevel.java 45
src/highlevel.java 45
src/bot_session_ack1.java 46
src/bot_session_ack2.java 46
src/musician_usage.java 47
src/configure_musician.java 49
mail_irc_log/eclipse_startup_cmd_eclipse_at_freenode 51
config/bsd/rc.conf 64
config/bsd/ipfw.sh 66
config/debian/xmpp1_interfaces 69
config/debian/tinydns_data 70
config/debian/test_DNS_SRV 70
config/debian/xmpp2_interfaces 71
src/vm_simulator/saros/start–rmiswtbot–project.sh 72
config/ubuntu/xsession_musikanten 72
src/vm_simulator/fap_deploy 75
mail_irc_log/sendmail_swtbot_ketan1.txt 80
mail_irc_log/recvmail_swtbot_ketan1.txt 81
mail_irc_log/sendmail_swtbot_ketan2.txt 81
mail_irc_log/recvmail_swtbot_ketan2.txt 82
mail_irc_log/sendmail_swtbot_ketan3.txt 83
mail_irc_log/recvmail_swtbot_ketan3.txt 83
mail_irc_log/sendmail_swtbot_ketan5.txt 84
mail_irc_log/recvmail_swtbot_ketan5.txt 84
mail_irc_log/sendmail_swtbot_ketan6.txt 84
mail_irc_log/recvmail_swtbot_ketan6.txt 85

Selbstständigkeitserklärung

Ich versichere, 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.

Berlin, den 28. Febraur 2010
Sandor Szücs

Danksagung

Ich danke meiner Freundin für die seelische Unterstützung und meinem Freund Florian Thiel für viele hilfreiche Diskussionen in der Zeit meiner Diplomarbeit.
Ein Besonderer Dank gilt meinen Betreuern Christopher Özbek und Stephan Salinger für ihre Hilfestellungen in unterschiedlichen Bereichen meiner Arbeit.
Weiterhin möchte ich *paulweb515* aus `irc://irc.freenode.net/#eclipse` für den Hinweis auf die Prozesseigenschaften in Eclipse danken.
Als Letztes möchte ich noch Ketan Padegaonkar für sein Interesse an Teilen meiner Arbeit und die freundliche E-Mail Korrespondenz danken.

Kapitel 1

Einleitung

Diese Arbeit behandelt das Netzwerkmodul und ein Testframework für Saros, einem Werkzeug zur verteilten Paarprogrammierung. In Kapitel [2] der Diplomarbeit wird die theoretische Einbettung der Arbeit und das Werkzeug Saros erklärt. Kapitel [3] beginnt mit einer praktischen Einleitung in verwendete Netzwerkprotokolle, gibt einen Überblick über Netzwerkmodule innerhalb von Saros und zeigt die Bearbeitung meiner dort angesiedelten Aufgaben. In Kapitel [4] beschreibe ich meine Motivation und Vorgehensweise in der Entwicklung eines Testframeworks für Saros. Es enthält getroffene Entscheidungen, handelt vom Bau, beleuchtet die Anwendung und schließt mit zukünftigen Einsatzszenarien. Der Schluss [5] der Arbeit versucht Hinweise auf weitere Arbeiten in den behandelten Bereichen zu geben und endet mit einer Zusammenfassung [6] aus meinem persönlichen Blickwinkel.

Die Notwendigkeit dieser Arbeit liegt insbesondere in der schweren Testbarkeit, der nicht optimalen Arbeitsweise und der Unzuverlässigkeit des Netzwerkmoduls von Saros. Software sollte in seinem Kern nicht regelmäßig Versagen, ansonsten ist sie für neue Anwender uninteressant. Saros hat eine wachsende aber noch keine große Anwendergruppe, daher ist die Lösung der Probleme relevant. Für die Arbeitsgruppe Software Engineering der Freien Universität Berlin bedeutet Saros eine Möglichkeit zur Datenerhebung in verschiedenen Arten der Paarprogrammierung zu erhalten. Eine große Benutzeranzahl ist für die Arbeitsgruppe daher von Interesse.

Vor dieser Arbeit gab es eine Reihe an Vorgängern, die Saros stetig erweiterten. Saros ist ursprünglich eine Entwicklung von Riad Djemili der eine lauffähige, noch nicht ausgereifte Software nach seiner Diplomarbeit [Dje06] hinterließ. Es folgte Björn Gustavs, der in einer Studienarbeit [Gus07] unter anderem die Zuverlässigkeit im Netzwerk bearbeitet hat. Oliver Rieger [Rie08] war der Nachfolger von ihm und fügte unter anderem die Möglichkeit der Peer-to-Peer Verbindung und NAT Traversierung hinzu. Christoph Jacob ist sein direkter Nachfolger und mein Vorgänger in der Bearbeitung des Netzwerkmoduls. Laut Aussagen [Jac09] in seiner Arbeit funktionierte Saros nur noch sehr unzuverlässig zu Beginn seiner Bearbeitungszeit. An eine Benutzung war nicht mehr zu denken. Gröbste Fehler hat er dokumentiert und in vielen Teilen der Software behoben. Sein Aufgabengebiet ließ es nicht zu sich eingehend mit allen Defekten in der Netzwerkschicht zu befassen.

Ziele der Diplomarbeit sind optimieren der Dateiübertragung, erhöhen der Robustheit im Netz, ermöglichen des Abbruchs und die Anzeige von Fortschritt in Netzwerk-

verbindungen. Außerdem soll eine Testsuite für die Netzkomponente aufgesetzt werden. Die Komponente soll auf Belastbarkeit und Zuverlässigkeit getestet werden können.

1.1 Typografische Konvention

Ich verwende folgende Konventionen zur besseren Lesbarkeit des Texts.

`Maschinenschrift` wird zur Markierung von Code, Objekten und Methoden innerhalb von Fileßtext verwendet.

- Klassenmethode `Class::method()`
- Instanzmethode `Class#method()`

„Doppelte Anführungszeichen“ werden für direkte Zitate verwendet.

[1.1] Geklammerte Ziffern mit Punkten sind Verweise auf Abschnitte innerhalb der Arbeit. Verweise auf den Anhang haben anfangs einen großen Buchstaben.

[Bec99] benutze ich für Zitate. Die Konvention sind Buchstaben gefolgt von der Jareszahl, optional durch ein Pluszeichen getrennt.

Kursivschrift wird für Abkürzungen, Fachwörter, Beispiele, nicht sinnvoll übersetzbare Wörter verwendet.

Kapitel 2

Kontext der Arbeit: Verteilte kollaborative Softwareentwicklung mit Saros

In diesem Kapitel stelle ich den Kontext der Arbeit vor. Der erste Teil [2.1] geht näher auf die theoretische Grundlage aus dem Bereich der Softwaretechnik ein. Es enthält Referenzen zu Literatur, in der der Leser mehr über die vorgestellten Softwareprozesse erfahren kann. Die Arbeitsmethodik des Saros-Teams wird im zweiten Abschnitt [2.2.2] näher beleuchtet. Zu guter Letzt [2.3] wird die Software Saros und ihre Möglichkeiten vorgestellt.

2.1 Theoretische Grundlage

Die theoretische Grundlage von Saros bilden unterschiedliche Arten der gemeinsamen Programmierung. Jede dieser Varianten wird in einem eigenen Abschnitt vorgestellt.

2.1.1 Paarprogrammierung

Paarprogrammierung ist eine agile Methode, die insbesondere in *eXtreme Programming*, auch bekannt unter der Abkürzung *XP*, Verwendung findet [Bec99]. Paarprogrammierung wird häufig auch durch *PP* abgekürzt. Während einer Sitzung arbeiten zwei Programmierer an einem Computer und bedienen abwechselnd die Eingabegeräte. Der Entwickler, der die Eingabegeräte führt wird im folgenden *Driver* genannt, der andere wird häufig als *Observer* bezeichnet. Der *Driver* programmiert, während der *Observer* die Übersicht behält und über alternative Lösungen nachdenkt. Das Paar tauscht regelmäßig die Rollen, um eine andere Sicht auf die erzeugten Artefakte zu bekommen [WKCJ00]. Es wird eine kontinuierliche Durchsicht praktiziert, die für eine höhere Codequalität und die Vermeidung von Fehlern ausschlaggebend ist [AGDIS07, CCG+07, HDAS09, Nos98, WKCJ00]. Aus Fehlern entstehen Versagen, deren Behebung zu einem späteren Zeitpunkt teurer als die sofortige ist [Hum97]. Mit dem Einsatz dieser Praxis kann eine Verbesserung der Wirtschaftlichkeit erreicht werden.
Laut diversen kontrollierten Experimenten mit Solo- und Paarprogramierern ist die reine Produktivität im Paar größer [HMDK04, LC03, Mad05, MWBF02, MWBF03, NWF+03, NWW+03, Nos98, WKCJ00, WU01]. Durch die gegenseitige Kontrolle bleibt der Fo-

kus stärker auf dem aktuellen Ziel. Bei *PP* werden verschiedene Abstraktionsebenen betrachtet, normalerweise arbeitet der *Driver* auf der mittleren und der *Observer* auf der unteren und oberen Schicht [BRdB08, CH07, FRdB07]. Daraus folgt ein besseres gesamtheitliches Verständnis des Bearbeiteten. Dies sollte eine Grundlage für höhere Qualität sein. Weitere Vorteile sind geteiltes Code Eigentum [Bec99], erhöhter Wissenstransfer und verbesserte Kommunikation im Team [CW01].

In der Tat läßt sich nicht leugnen das zwei Arbeitnehmer eingesetzt werden, um ein Problem zu lösen. Verschiedene Experimente kommen, für den vorhandenen Overhead, zu unterschiedlichen Zahlen. Nosek hat in seinem Experiment 30% weniger Zeit zu 40% mehr Aufwand [Nos98], Williams hat 40% weniger Zeit zu 20% mehr Aufwand [Wil, WKCJ00] und 20% weniger Zeit zu 60% mehr Aufwand hat Nawrocki [NW01] gemessen.
Maßgeblich für die genannten Verhältnisse können, laut den genannten Studien, die Einarbeitungszeit eines Teams, die Persönlichkeit der Teilnehmer und die Wissens- bzw. Erfahrungs-Unterschiede sein.

2.1.2 Verteilte Paarprogrammierung

Bei verteilter Paarprogrammierung, häufig *DPP* abgekürzt, sind die Partner räumlich getrennt. Es entstehen dadurch offensichtliche Unterschiede: Beide Teilnehmer brauchen je einen Computer und eine Lösung, um die gleiche Sicht auf das aktuell bearbeitete Artefakt zu erhalten. Häufig handelt es sich bei der Lösung, um Software, die eine synchronisierte Sicht zur Verfügung stellt. Generell gibt es zwei Ansätze zur Softwareunterstützung für die verteilte Paarprogrammierung, *Desktop Sharing* und *Collaboration Awareness* [Dje06]. Die naive Lösung einer synchronisierten Sicht ist das *Desktop Sharing*. Hierbei wird die Bildschirm- oder Programmsicht geteilt, wobei die zu bearbeitenden Artefakte an einem festen Ort bleiben. Lösungen, die eine Collaboration Awareness implementieren, synchronisieren Artefakte zwischen beiden Teilnehmern. Eine Ausführung des Resultats kann dabei auf beiden Rechnern geschehen.

Es gibt eine Reihe von Unterschieden zwischen klassischer und verteilter Paarprogrammierung.

Nachteile verteilter Paarprogrammierung: Das Netzwerk zwischen den Partnern **und** beide Rechner müssen funktionieren, ansonsten ist DPP nicht möglich. Häufig werden die fehlende Körpersprache, die erhöhte Lernkurve und eine fehlende Möglichkeit der grafischen Kommunikation bemängelt [SWN+03]. Stotts meint auch, dass das direkte Zeigen auf eine genaue Stelle nicht möglich oder nur schwer möglich sei.

Vorteile verteilter Paarprogrammierung: Verteilte Arbeitsplätze im Kontext der Paarprogrammierung sind unter anderem, die von Beck geforderten, bequemeren Arbeitsplätze [Bec99]. Sie können beispielsweise personalisiert eingerichtet werden. Ein höherer Grad an Parallelität ist durch die Nutzung zweier Computer gegeben [SWN+03]. Der *Observer* kann in der Dokumentation die Beschreibung verwendeter Methoden nachlesen und deren korrekte Benutzung prüfen, während der *Driver* ein Codefragment zu Ende schreibt.
Weiterhin sei kein Platzwechsel notwendig und es würde weniger *off topic* Diskussionen stattfinden [SWN+03]. Ein oft genannter Vorteil ist der Zwang Notizen

in elektronischer Form zu verfassen, die sich später zu Dokumentationszwecken verwenden lassen.
DPP ermöglicht den Einsatz von Paarprogrammierung in Situationen wenn sich potentielle Partner nicht an einem Ort befinden.

2.1.3 Side-by-Side-Programmierung

Side-by-Side-Programmierung, oft *SbS* abgekürzt, wurde als erstes von Alistair Cockburn erwähnt [Coc04]. Nawrocki et al beschreiben in ihrem Papier [NJuOL05] *SbS* folgendermaßen:

> „In SbS a task is assigned to a pair of programmers, and each programmer has his own computer. That allows them to split the task into subtasks and work on each subtask individually (similarly to the classical approach). The main difference between SbS and the classical approach is physical proximity of the pair members (which enhances communication) and unity of the goal (SbS programmers are working on the same task and they both are responsible for it).“

Es handelt sich bei *SbS*, um eine bewußte Nähe zweier Programmierer, die eine Aufgabe gemeinsam erledigen sollen. Dazu teilen sie die Aufgabe in Unteraufgaben ein, die sie einzeln lösen. Ein kontrolliertes Experiment von Solo- zu Paar- zu Side-by-Side-Prorgramierern hat gezeigt, dass der Overhead von 50% bei *PP* zu 20% bei *SbS* im Vergleich stark abnimmt. Es scheint also die Effizienz zu steigern und dabei vergleichbare Effekte, wie bei der *PP* zu haben.[NJuOL05]
Eine Untersuchung mit dem Ziel *SbS* genauer zu verstehen hat sieben Episoden ausfindig gemacht, die beschreiben welche Anwendungsfälle es bei Zusammentreffen während einer *SbS* Sitzung gibt [PSS08, PSS09].

2.1.4 Verteilte Side-By-Side-Programmierung

Verteilte Side-by-Side Programmierung ist eine interessante Alternative zu *DPP*, weil die in der vorigen Sektion [2.1.3] genannten Vorteile im Gegensatz zu *(D)PP* bestehen bleiben. Die Untersuchung der Kollaborationsepisoden in *SbS*, die in [PSS08] beschrieben sind, zeigen auf, dass meistens nur kurze Zusammenarbeit betrieben wird. Daher wird eine Möglichkeit benötigt, die erlaubt schnell die Sicht des Partners zu erhalten und danach wieder zurück zur Eigenen zu gelangen.
Eine von Dewan vorgestellte Lösung ist es, jeden Teilnehmer mit zwei Computern auszustatten. Auf einem *PC* arbeitet man wie gewöhnlich, während auf dem anderen die Sicht des Partners abgebildet ist. Diese Variante wurde in einer explorativen Studie verwendet, um die Arten von Zusammenarbeit in *SbS* und *DSbS* besser zu verstehen [DASH09].
Die naheliegende Schwachstelle der genannten Lösung ist der erhebliche Mehraufwand für eingesetzte Betriebsmittel. Weitere Nachteile sind erhöhter Platz- und Energiebedarf. Ein Vorteil ist es, keine Sichtwechsel in Software zu benötigen.
Ein anderer Ansatz wird von kollaborativen Editoren wie Gobby[1], Saros[2] oder SubEthaEdit[3] verfolgt. Teilnehmer können gleichzeitig in einem Dokument schreiben, so

[1]Gobby: `http://gobby.0x539.de/` (Online abgerufen am 23. Februar 2010)
[2]Saros: `http://dpp.sourceforge.net/` (Online abgerufen am 23. Februar 2010)
[3]SubEthaEdit: `http://www.subethaedit.net/` (Online abgerufen am 23. Februar 2010)

dass unabhängiges Arbeiten und die gleichzeitige Sicht auf ein aktuelles Artefakt möglich ist. Der Cursor und die Markierungen der anderen Teilnehmer sind für alle sichtbar. Im Gegensatz zum Desktop Sharing [SWN+03], ist daher genaues zeigen auch vom *Observer* zum *Driver* möglich. Darüber hinaus bietet Saros die Möglichkeit schnell zur Sicht eines anderen Programmierers zu gelangen. Die Rückkehr zur eigenen Sicht kann üder die umgebende *IDE*, Eclipse, geschehen.

2.2 Softwareprozess des Saros-Teams

In der Zeit, die ich an Saros arbeitete, wuchs das Entwicklerteam von zwei auf zeitweise bis zu sechs Studenten an. Entwickler hatten dabei relativ klar abgegrenzte Zuständigkeiten, was weniger gute Einflüsse auf den *Codeownership* hat. Bei Defekten rund um `ConcurrentDocumentManager` hatte ich mich z.B. nicht zuständig gefühlt, die vorhandenen Mängel zu beseitigen. Weil wir zeitweise keine zuständige Person für diese Klasse hatten blieben die Defekte lange erhalten. Meiner Meinung nach wäre daher ein mehr agiler Prozess für Saros förderlicher, weil dadurch die aktuell wichtigen Defekte im Fokus liegen. Die Kehrseite ist, dass die Benotung der Einzelleistung schwieriger ist, da Erfolge nicht mehr einer bestimmten Person zugeordnet werden können.

2.2.1 Beschreibung

Saros wird in einem iterative Prozess entwickelt. Ein Zyklus wird dabei in zwei Phasen, namentlich Implementierung und Test, unterteilt. Ein Zyklus bestand anfangs aus zwei Wochen. Nachdem wir Saros in die Betaphase einstuften, verlängerten wir einvernehmlich den Zyklus auf drei Wochen[4]. Mir kam die Verlängerung sehr gelegen, weil ich vorher nie den Eindruck hatte etwas für ein Release geschafft zu haben.
Während der Implementierungsphase sollen definierte Ziele bearbeitet, gelöst, einer Durchsicht unterzogen und integriert werden. Ein Patch muss vor Einbringung in die Hauptlinie von zwei weiteren Entwicklern akzeptiert werden, dabei hat insbesondere unser damaliger Technischerleiter mit Argusaugen jedes Detail bemerkt. Mich persönlich hat dies in meinem technischen Horizont weitergebracht, allerdings hat es auch meinen Fortschritt, zum Wohle von Saros, gebremst.
Generell gibt es im Saros-Team die Rollen *Testmanager*, *Releasemanager* und jeweils deren Assistenten. Die genannten Rollen werden mit *TM* und *RM*, bzw. *ATM* und *ARM* für die Assistenten, abgekürzt.
Die Test- und Releasephase dauert insgesamt eine Woche. Während der Testphase muss der *TM* vorbereitend passende Tests zur geschehenen Entwicklung aussuchen. Der *Releasemanager* bereitet am Dienstag einen separaten Entwicklungszweig für Test und Release vor. Das gesamte Team trifft sich darauf am Mittwoch und arbeitet die ausgewählten Tests an Hand einer Vorlage gemeinsam durch. Am Ende des Tages schickt der *TM* eine Liste, vor dem Release zu behebender Defekte, an alle Teilnehmer. Donnerstag und Freitag werden gefundene Defekte behoben. Am Freitagnachmittag wird vom *RM* das Release vorgenommen und alle ihre zuständigen Webeiten aktualisieren. Als letzte Tat am Freitag werden die Ziele für die nächste Veröffentlichung geplant, die zeitweise auf der Releaseseite[5] hinterlegt wurden.
Eine neue Version von Saros zu veröffentlichen bindet meistens zwei Personen den

[4] Aktuell ist der Zyklus auf vier Wochen verlängert worden.

[5] Releaseseite: `https://www.inf.fu-berlin.de/w/SE/DPPReleasePlan` (Online abgerufen am 23. Februar 2010)

ganzen Freitag Nachmittag[6]. Das liegt zum einen an den vielen Schritten und zum anderen an der langsamen Reaktionszeit der geteilten Ablage auf *Sourceforge*[7].

2.2.2 Zusammenarbeit

Teile meiner Arbeit überschnitten sich mit der Bachelorarbeit von Tas Sóti. Mit ihm habe ich die Schnittstelle zwischen Netzwerkschicht und Einladungsprozess [Só09] entwickelt. Wir haben eine zeitlang Paarprogrammierung erfolgreich praktiziert. Er konnte klären, wann, welche `Exception` geworfen werden soll, damit die Ausnahmebehandlung in der grafischen Oberfläche korrekt eintreffen. Ich konnte Hinweise für bessere Lesbarkeit des Codes im Einladungsprozess geben. Ich implementierte zeitnah die Abbrechbarkeit und Fortschrittsaktualisierungen an Stellen, die u.a. während der Einladung benutzt werden. Während einer Paarprogrammiersitzung zur Verwendung der Schnittstelle zwischen Geschäftslogik und Netzwerk, dem `XMPPTransmitter`, fielen mir Duplikate beim Senden über die `XmppConnection` auf. Duplikate sind als *Code Smell* bekannt und wurden deshalb gleich beseitigt [Fow99].
Die Arbeiten am Einladungsprozess wurden in einem zur Hauptlinie separaten Bereich entwickelt. Zum Schluss mußte dieser wieder zurück in die Hauptlinie integriert werden, was in einer Tripple-Paarprogrammiersitzung mit Christopher Özbek, Tas Sóti und mir innerhalb eines Tages geschah. Es war wichtig die Erfahrung des technischen Leiters zur Integration zu haben, weil wir zu zweit sicherlich nicht die komplette Übersicht behalten hätten.

Eine andere Sitzung hatte ich mit Henning Staib[8]. Gemeinsam evaluierten wir erfolgreich innerhalb eines Tages die Implementierung einer Datenstrom basierten Schnittstelle auf dem `DataTransferManager` [3.3.5] über Javas `PipedInputStream` und `PipedOutputStream`. Datenströme werden als zusätzliche Fähigkeit für die Audio- und Videokommunikation benötigt. Ein weiterer Einsatzzweck ist die initiale Versendung des Projektarchivs, das bisher vor der Versendung komplett in den Speicher geladen wird.

2.3 Saros

Saros ist ein Werkzeug für verteilte Paarprogrammierung. Software dieser Art kann in zwei grundsätzlich verschiedenen Ansätzen implementiert werden. Einerseits kann die Sicht und andererseits die angezeigte Ressource synchronisiert werden. Arten von *Desktop Sharing* synchronisieren die Sicht, während *Colaboration Awareness* versuchen, die im Programm gezeigten, Ressourcen abzugleichen.
Die Wahl hat maßgeblichen Einfluss auf die Parallelität, die in vorherigen Arbeiten auf Programm-, Sicht- und Schreibebene unterschieden wurden. Beim *Desktop Sharing* ist die Parallelität nur auf Programm- und Sichtebene gegeben. Im Gegensatz dazu ist Saros, eine Art der *Colaboration Awareness* und bietet die Möglichkeit der parallelen Arbeit auf Programm-, Sicht- und Schreibebene zur Verfügung.

[6] Die notwendige Schritte für die Erstellung eines Release ist auf der Webseite mit *Revision* 70 zu finden: `https://www.inf.fu-berlin.de/w/SE/DPPHowToDevel#How_to_create_a_new_release`
[7] Sourceforge ist der Name eines Webportal zur Verwaltung einer Vielzahl an Softwareprojekten
[8] Ein Student der z.Z. meines Schreibens seine Diplomarbeit im Saros-Projekt schreibt.

2.3.1 Beschreibung von Saros

Die nachfolgende Beschreibung von Saros bezieht sich auf den Einsatz mit dem Java Editor mit der Abkürzung *JDT*[9]. Zur Zeit des Schreibens sind in anderen Editoren Defekte bekannt, auf die ich nicht näher eingehen werde.

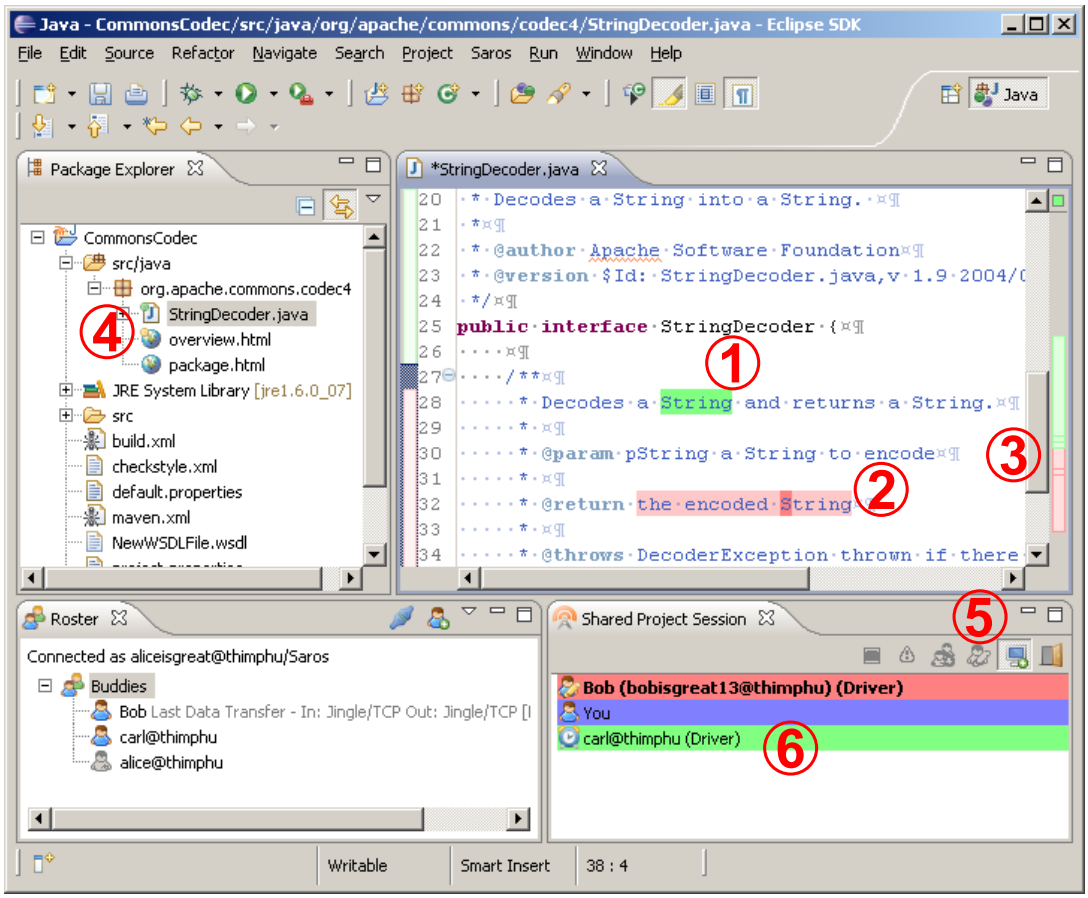

Abbildung 2.1: Zu sehen ist eine Saros-Sitzung mit 3 Benutzern (6). (5) zeigt eine Knopfleiste, in der der *Observermodus* aktiviert ist. Der aktuell verfolgte Benutzer ist in Fettbuchstaben geschrieben (6). Andere Teilnehmer sind im offenen Editorfenster mit unterschielichen Farben (1) und (2) dargestellt. Der Sichtausschnitt im Editor der anderen Teilnehmer kann im Balken rechts von (3) gesehen werden. (4) weist auf die Markierung einer von anderen aktuell geöffnete Datei im *Package Explorer* hin.

Anders als die meisten Kollaborativen-Echtzeit-Editoren, versucht Saros konsequent die in Abschnitt [2.1] beschriebenen Softwareprozesse zu unterstützen.
Paarprogrammierung, bzw. DPP, wird mit Hilfe eines *Observermodus* unterstützt. Dieser Modus ist in Abbildung 2.1 dargestellt. Es wird die Sicht des Betrachters, auf die, des aktuell observierten *Drivers* aktualisiert. Die gleichzeitige Sicht auf ein Artefakt ist dadurch leicht möglich. Direktes zeigen auf eine Stelle kann durch markieren von Textpassagen in beide Richtungen geschehen. Natürlich kann auch der *Observer* selbstständig den Code durchsehen.
Die Funktionalität des gleichzeitigen Schreibens ermöglicht Verfahren wie *SbS* oder

[9] *Java Development Tools* `http://www.eclipse.org/jdt/` (Online abgerufen am 23. Februar 2010)

DSbS anzuwenden. Durch ein Doppelclick auf den Partner im *Shared Project View* springt man zur aktuellen Sicht des anderen, zurück kommt man mit Eclipsebordmitteln[10]. Es kann schnell der Kontext des Partners erfasst, eine kurze gemeinsame Leistung vollbracht und danach wieder zurück zu seiner ursprünglichen Arbeit gewechselt werden. Der Einsatz von Saros kann daher einen Beitrag zur effizienten Umsetzung von *SbS* und *DSbS* leisten.

2.3.2 Möglichkeiten von Saros

Grob betrachtet kann Saros in allen, am Kapitelanfang [2.1], erwähnten Entwicklungsprozessen genutzt werden. Durch eine Diskussion mit Redhuan D. Oon, einem Entwickler, des *ADempiere*[11] Projekts, haben sich für mich noch weitere, feinere Einsatzszenarien ergeben:

Die Einführung neuer Entwickler in ein Projekt und

Erklärung des Entwurfs an Hand einer Führung durch den Quellcode.

Der Einsatz von Saros scheint mir auch für nicht agile, virtuelle Teams in den zwei genannten Fällen sinnvoll. Insbesondere Open Source Projekte könnten neue Entwickler schneller mit der Codebasis vertraut machen und an das Projekt binden. Häufig gibt es in Projekten sogenannte *Bug Day* zur Beseitigung von bekannten Defekten, warum nicht auch eine Einführung in die Codebasis?

[10] Zur alten Sicht zurück kommt man mit Hilfe des nach links gerichteten Pfeils in der Werkzeugleiste aus Abbildung 2.1

[11] ADempiere `http://www.adempiere.com/index.php/ADempiere` (Online abgerufen am 23. Februar 2010)

Kapitel 3

Arbeit Teil 1: Verbesserung von Netzwerkschicht und Einladungsprozess

In diesem Kapitel ist meine Arbeit an Arbeitspaketen und Netzwerkklassen beschrieben. Zunächst möchte ich auf die Gründe eingehen, warum eine bereits bestehende Komponente so intensiv überarbeitet wurde. In Abschnitt [3.2] zeige ich die Netzwerkarchitektur verwendeter Protokolle und beschreibe den in Saros eingesetzten Netzwerkstapel. In der darauffolgenden Sektion [3.3] erkläre ich meine Umsetzungen und Lösungsansätze mir aufgetragener Arbeitspakete. Zum Schluss [3.4] gebe ich Hinweise auf notwendige Arbeiten und erläutere Ideen zur Weiterentwicklung der Netzwerkschicht von Saros.

3.1 Motivation

Zu Beginn meiner Diplomarbeit befand sich Saros in der Alphaphase und hatte an vielen Stellen Defekte, die zu regelmäßigen Versagen führten. Auch das Netzwerkmodul enthielt viele Fehler. Zudem gab es schwer verständliche Bestandteile in der internen Arbeitsweise und nicht optimale Implementierungen. Ein konkretes Versagen existiert auch heute[1] noch zu Beginn einer Saros-Sitzung: Der Aufbau einer Peer-to-Peer Verbindung mit möglicher NAT Traversierung versagt in seltenen Fällen schon bei Teilnehmern, die eine direkte Internetverbindung ohne NAT besitzen.

Saros wurde von Riad Djemili im Jahr 2006 innerhalb seiner Diplomarbeit entwickelt [Dje06]. Zu dieser Zeit lief die Kommunikation zwischen Saros-Instanzen ausschließlich mit Hilfe von XMPP Nachrichten über XMPP Server. Oliver Rieger führte 2008 in seiner Diplomarbeit *Peer-to-Peer* Verbindungen zur Optimierung von Saros ein [Rie08]. Im Jahr 2009 entdeckte Christoph Jacob während seiner Diplomarbeit auftretende Probleme beim Aufbau von *Peer-to-Peer* Verbindungen in Saros [Jac09]. Er dokumentierte sie in Abschnitt 8.3 seiner schriftlichen Ausarbeitung.

Zu meinen Aufgaben im Saros-Projekt zählten Verbesserungen in der Arbeitsweise, die Beseitigung bestehender Defekte und dem Hinzufügen ausgewählter Funktionalitäten. Die Verbesserungen beinhalten das Optimieren bestehender Komponenten und das Refaktoring zentraler Klassen. Defekte waren an vielen Stellen, wie z.B. im Verbin-

[1] Heute ist der 12.1.2010.

dungsaufbau zu beseitigen. Eine neue Funktionalität ist z.B. die Fortschrittsanzeige in allen langlaufenden Prozessen des Netzwerkmoduls.

Außerdem möchte ich mit diesem Kapitel dem Leser mein Wissen über verwendete Protokolle und Saros' interne Arbeitsweise zu vermitteln. Meine Nachfolger im Saros-Projekt sollen in der Lage sein, das Verhältnis der zentralen Objekte im Netzwerkmodul und insbesondere die verschiedenen Datentransferkanäle über Javas `java.net.Socket` Klasse schneller zu verstehen.

3.2 Netzarchitektur

Die Architekturbeschreibung ist in zwei Teile gegliedert. Der erste Teil behandelt die Protokollebene. Der Zweite gibt Aufschluss über die Implementierung in Saros und der verwendeten *Smack* Bibliothek. Zentrale Objekte und Architektur der Netzschicht werden beschrieben.

3.2.1 Protokollschicht

XMPP [SAa], *Extensible Messaging and Presence Protocol*, ist ein von der *IETF* [IET] standardisiertes Protokoll und ist durch *XEPs* [SAb], *XMPP Extension Protocols*, offiziell erweiterbar. Erweiterungen in Form von *XEP* werden von der *XMPP Standards Foundation*, abgekürzt *XSF*, vorgenommen.

Bei *XMPP* handelt es sich um ein asynchrones, nachrichtenbasiertes Protokoll ähnlich dem wesentlich älteren *SMTP* [2]. *XMPP* ist aus dem ehemaligen *Jabber* Protokoll hervorgegangen und basiert auf *XML*. Die Hauptanwendung für dieses Protokoll ist der *Instant-Messaging*-Dienst, häufig deshalb auch noch *Jabber* genannt. Saros und andere kollaborative Editoren verwenden *XMPP* als Kommunikationskanal. Saros verwendet es insbesondere für den Austausch von Benutzeraktivitäten zur Anzeige zahlreicher *Awarenessinformationen*, z.B. die aktuelle Textselektion eines Entwicklers.

Benutzer von XMPP Diensten sind über ihre *JID*, *JabberID*, eindeutig identifiziert. Die Definition der *JID* ist im Kern von *XMPP* [SAa] enthalten. `Alice@jabber.org` ist ein Beispiel für eine *JID*.

Die Ressourcenangabe einer *JID* gibt das adressierte Softwarepaket an, z.B. `alice@jabber.org/Saros`. Die Angabe einer Ressource bestimmt die empfangende *XMPP* Klientensoftware. Ohne Ressourcenangabe ist es nicht sichergestellt, dass bei gleichzeitiger Verwendung unterschiedlicher *XMPP* Software, der richtige Klient die ankommende Nachricht erhält.

Dienstanbieter sind durch die Domänenangabe einer *JID* identifiziert. In dem oben genannten Beispiel wäre es `jabber.org`. Für die Hochverfügbarkeit von *XMPP* Servern verwenden größere Firmen wie *Google DNS SRV*[3]. *DNS SRV* lokalisiert einen Dienst dynamisch an Hand der Domäne. Abbildung 3.1 verdeutlicht die Funktionsweise der dynamischen Dienstauflösung.

Die Kommunikation zwischen `alice@jabber.org` und `bob@jabber.ccc.de` ist möglich, falls sich beide Dienstanbieter in der *XMPP* Föderation befinden, d.h. Anfragen von anderen *XMPP* Servern bearbeiten.

Auf Protokollebene definiert *XMPP* grundsätzlich zwei verschiedene Paketarten:

[2] SMTP - `http://www.ietf.org/rfc/rfc821.txt` (Online abgerufen am 23. Februar 2010)

[3] DNS SRV ist ein Standard der IETF nach RFC 2782 `http://tools.ietf.org/rfc/rfc2782.txt` (Online abgerufen am 24. Februar 2010)

```
$ host gmail.com
gmail.com has address 209.85.129.19
gmail.com has address 209.85.129.83
gmail.com has address 209.85.129.17
gmail.com has address 209.85.129.18
gmail.com mail is handled by 20 alt2.gmail-smtp-in.l.google.com.
gmail.com mail is handled by 30 alt3.gmail-smtp-in.l.google.com.
gmail.com mail is handled by 40 alt4.gmail-smtp-in.l.google.com.
gmail.com mail is handled by 5 gmail-smtp-in.l.google.com.
gmail.com mail is handled by 10 alt1.gmail-smtp-in.l.google.com.

$ host -t SRV _xmpp-server._tcp.gmail.com
_xmpp-server._tcp.gmail.com has SRV record 5 0 5269 xmpp-server.l.google.com.
_xmpp-server._tcp.gmail.com has SRV record 20 0 5269 xmpp-server1.l.google.
    com.
_xmpp-server._tcp.gmail.com has SRV record 20 0 5269 xmpp-server2.l.google.
    com.
_xmpp-server._tcp.gmail.com has SRV record 20 0 5269 xmpp-server3.l.google.
    com.
_xmpp-server._tcp.gmail.com has SRV record 20 0 5269 xmpp-server4.l.google.
    com.

$ host xmpp-server.l.google.com
xmpp-server.l.google.com has address 209.85.163.125
```

Abbildung 3.1: Zu sehen ist die Arbeitsweise von *DNS SRV*. Der *XMPP* Dienstanbieter `xmpp-server.l.google.com` wird durch eine *DNS SRV* Abfrage gefunden.

- **Message** ist eine unidirektionale Nachricht an einen bestimmten Teilnehmer.
- **IQ** ist eine Frage-Antwort Kommunikation, indem ein Anfragepaket versendet und eine Antwort mit gleicher *XMPP-Paket-ID* erwartet wird.

Saros verwendet `Message` u.a. zur Übertragung von Benutzeraktivität und `IQ` zur Versionsabfrage vor dem Beginn einer Sitzung.

Zur Identifizierung einer bestimmten Funktionalität kann grundsätzlich jede Zeichenkette dienen. Saros verwendet aktuell die Zeichenkette `de.fu_berlin.inf.dpp`, während von der *XSF* definierte *XEPs* zwei verschiedene Notationen verwenden:

- **URL** `http://jabber.org/protocol/disco#info`, verwendet *XML-Schema* Namensräume und wird von Erweiterungen genutzt.
- Notation mit **Doppelpunkt** als Trennzeichen `jabber:client`, dient hauptsächlich[4] zur Kennzeichnung von *XMPP RFCs*[5], die auch von der *IETF* standardisiert sind.

XMPP Klienten können die Unterstützung diverser Fähigkeiten bestätigen, indem sie eine *Service Discovery* nach *XEP-0030* [HMESA08] mit einer Liste von Namensräumen beantworten. Der anfragende Klient muss dann selbstständig in der Liste prüfen, ob

[4]Es scheint mir so zu sein, vgl. die *Revision History* unter `http://xmpp.org/registrar/namespaces.html` (Online abgerufen am 23. Februar 2010)

[5]Requests for Comments wird für Protokollveröffentlichungen genutzt und von der *IETF* [IET] betrieben `http://www.ietf.org/rfc.html` (Online abgerufen am 23. Februar 2010)

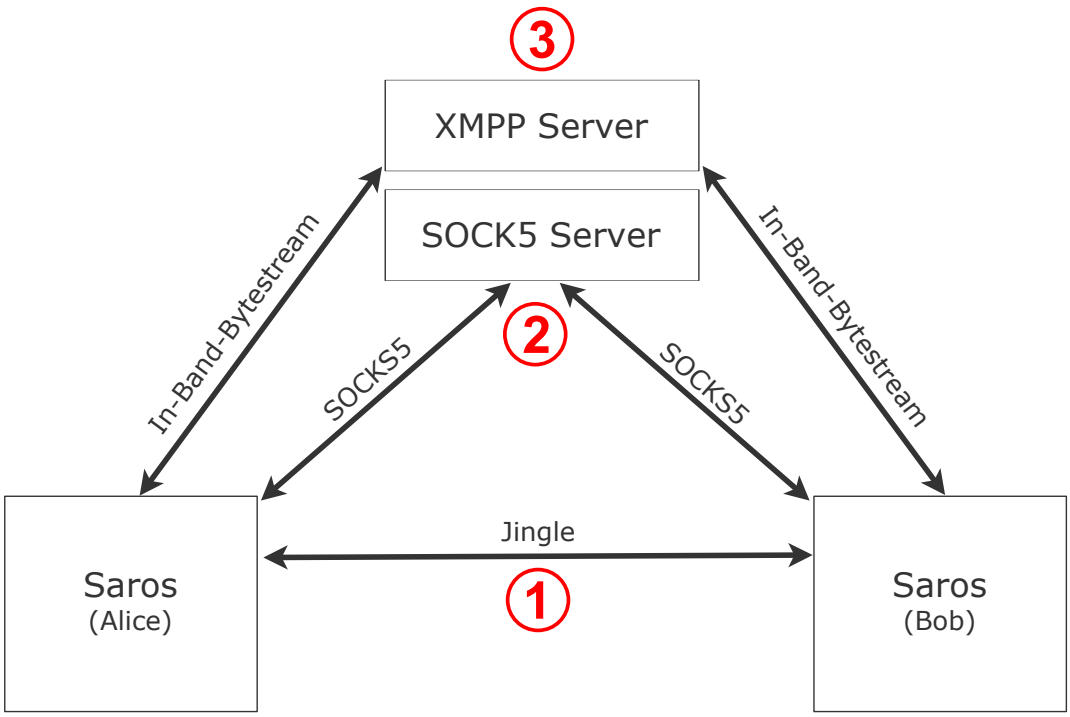

Abbildung 3.2: In Saros verwendete Dateitransfermöglichkeiten. (1) ist bevorzugt und baut eine *Peer-to-Peer* Verbindung auf. (2) hat nach Messungen ca. 80kB/s Durchsatz. Die Verfügabrkeit ist abhängig von der Serverkonfiguration. (3) hat nach Messungen nur 4kB/s Datendurchsatz.

das gewünschte Feature von der Gegenstelle implementiert ist.

Zu Beginn einer Saros-Sitzung wird das gemeinsame Projekt der Teilnehmer synchronisiert. Für den hierfür notwendigen Dateitransfer sind bereits einige *XEPs* definiert. Saros verwendet zur Dateiübertragung, je nach Verfügbarkeit, die *XEPs In-Band Bytestreams*, *Jingle* und *SOCKS5 Bytestreams*. Die Wege der genannten Protokolle können in Abbildung 3.2 nachvollzogen werden. *Jingle* wird bevorzugt, da es den Aufbau einer *Peer-to-Peer* Verbindung ermöglicht und daher den höchsten Durchsatz erzielt. Der Einsatz von *SOCKS5 Bytestreams* ist abhängig von der Serverkonfiguration. Als Rückfallverbindung wird *In-Band Bytestreams* für den Transfer größerer Datenmengen verwendet. *In-Band Bytestreams* kodieren Daten in *Base64* und versenden diese als *XMPP Message*. Server dürfen laut Standard den *In-Band* Datenverkehr drosseln, sodass nur wenige *kB/s* Daten transferiert werden. Im Gegensatz hierzu gehören *SOCKS5 Bytestreams* zu den *Out-Band* Protokoll. Die Durchsatzmessungen sind im Anhang [A.1] dokumentiert.

Es ist zu beachten, dass *SOCKS5 Bytestreams*[6] **nicht** *RFC1928*[7] konform ist, sondern eine eigene Spezifikation nach *XEP-0065* [SMSA07] besitzt.

Der Verbindungsaufbau über *Jingle* ist laut Spezifikation in der Lage verschiedene Methoden zur Überbrückung von *NAT* zu verwenden. Das *Jingle* Protokoll ist pluginbasiert und daher erweiterbar. Für den Verbindungsaufbau verwendet Saros derzeit

[6]XEP SOCKS5 Bytestreams: `http://xmpp.org/extensions/xep-0065.html` (Online abgerufen am 23. Februar 2010)

[7]RFC1928: `http://www.ietf.org/rfc/rfc1928.txt` (Online abgerufen am 23. Februar 2010)

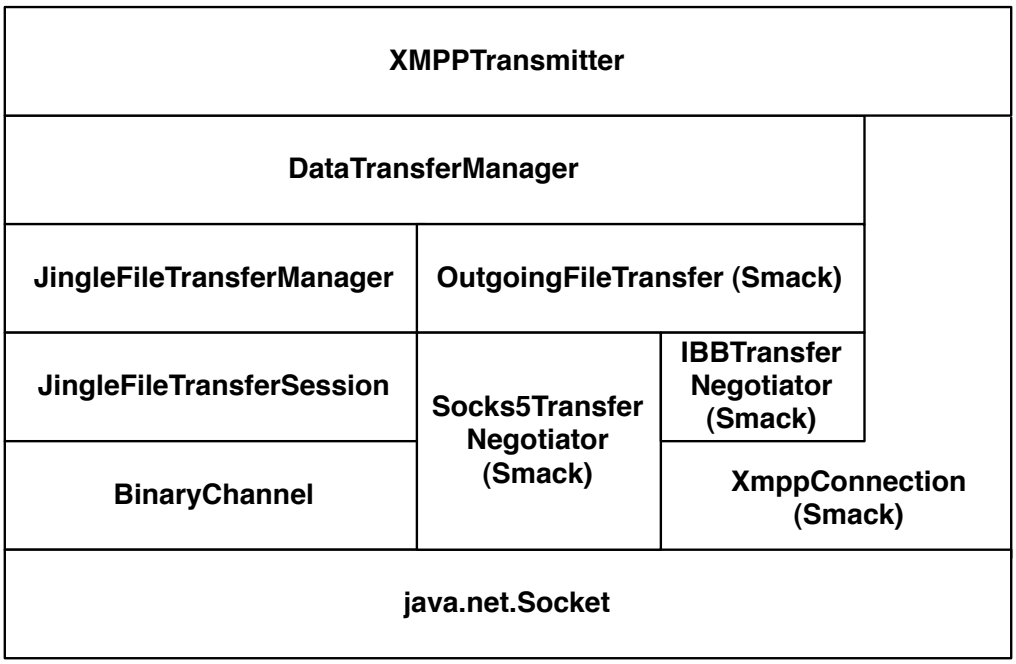

Abbildung 3.3: Netzwerk Architektur beim Datenversand

ICE [Ros07], das intern sowohl *STUN* [RWHM03] als auch *TURN* [RMM09] zur Traversierung von *NAT* verwenden kann. Die von Saros genutzte XMPP Bibliothek *Smack* unterstützt bisher allerdings ausschließlich *STUN*.

Unterschiedliche *NAT* Varianten und das sogenannte *Hole Punching* wurden von Oliver Rieger [Rie08] und Christoph Jacob [Jac09] bereits ausführlich in ihren Diplomarbeiten beschrieben, deshalb gehe ich an dieser Stelle nicht näher darauf ein.

3.2.2 Saros - Netzschicht

Das Netzmodul von Saros hat eine Schichtenarchitektur. Dies ist eine häufige Grundarchitektur in Netzwerkmodulen. Das Diagramm 3.3 zeigt den Netzwerkstapel im Saros *Plugin* beim Datenversand. Zur Ausfallsicherheit werden verschiedene Netzwerkprotokolle für die Übermittlung von Daten verwendet. Bei Ausfall einer Verbindung wird für den Benutzer transparent eine andere benutzt, um den Versand von Daten zu gewährleisten.

Sarosentwickler versenden Daten über den `XMPPTransmitter` und erhalten Daten vom `XMPPReceiver`. Wie die Architekturdiagramme 3.3 und 3.4 zeigen, kapselt der `DataTransferManager` die dateitransferartige Kommunikation. Unterhalb dessen wird dann auf die in Sektion [3.2.1] vorgestellten Protokollimplementierungen verwiesen.

Mehrere kleine Unterschiede bestehen zwischen Empfang und Versand: Einerseits werden Daten, solange keine Sitzung existiert, direkt über die `XMPPConnection` des `XMPPTransmitters` empfangen. Andererseits werden Daten, während einer laufenden Sitzung, über den `XMPPReceiver` empfangen. Statt direkt auf dem `XMPPTrans-`

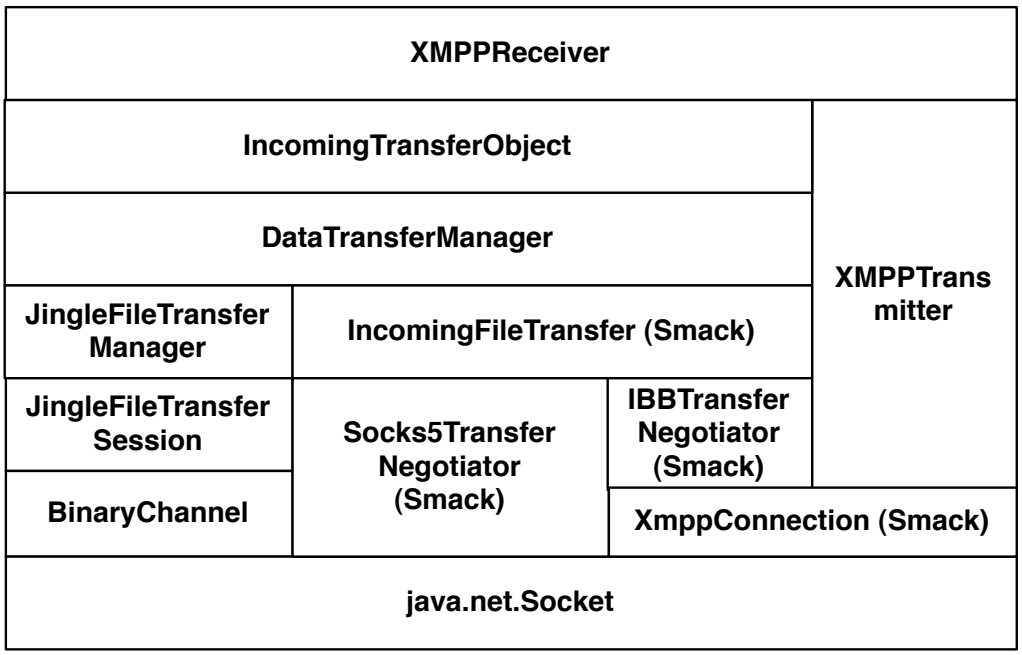

Abbildung 3.4: Netzwerk Architektur beim Empfang von Daten

`mitter` operieren zu müssen, bietet der `XMPPReceiver` die Möglichkeit Filter zu registrieren. Registrierte Objekte werden daraufhin nur bei eintreffen durch den Filter bestimmter Pakete aufgerufen. Der Empfang ist auf diese Art und Weise über ein *Observer Muster* [GHJV95] gelöst. Für eingehende Peer-to-Peer Verbindungen gibt es ein `IncomingTransferObject`, das ankommende Transferanfragen kapselt. Dies ermöglicht der Geschäftslogik, oberhalb des `XMPPReceiver`, einen Transfer anzunehmen oder abzulehnen.

Der Aufbau einer Saros-Sitzung geschieht mit Hilfe einer Einladung, die bereits von Tas Sóti genau beschrieben wurde [Só09]. Eingangs findet eine Synchronisation der Projektdaten statt. Der Versand und Empfang wird dabei über die oben erwähnten Dateitransfermöglichkeiten abgehandelt.

Aktivitäten versenden

Benutzeraktivität ist ein wichtiger Bestandteil von Saros. Aktivitäten werden auf Klassen abgebildet, die von `AbstractActivity` erben. Vor dem Versand werden sie in einer Liste gesammelt. Sie ist in der Klasse `TimedActivities` gekapselt und serialisiert Aktivitäten mit Hilfe des `ActivitiesExtensionProvider` nach *XML*. Regelmäßig werden sie vom `ActivitySequencer` über den `XMPPTransmitter` versendet.

Der `XMPPTransmitter` entscheidet je nach Größe des Datenvolumens, ob für den bevorstehenden Transfer eine *Peer-to-Peer* Verbindung aufgebaut werden soll. Normalerweise werden Daten über *XMPP* in einer `Message` versendet. Hier ist eine Schwäche in der Netzwerklogik zu erkennen. Leider kann man aufgrund der Protokollarchitektur nicht davon ausgehen, dass eine *Peer-to-Peer* Verbindung wiederverwenden zu können. Ein Problem besteht jedoch darin, dass die in Abschnitt [3.2.1] vorgestell-

ten Protokolle auf einem Dateitransfer basieren. Eine direkte Folge des Dateitransfers ist, dass die Verbindung nach dem Transfer wieder geschlossen wird. Hinzu kommt, dass Implementierungen aus der verwendeten *XMPP* Bibliothek *Smack* [Sma], die Angabe einer begrenzten Dateigröße für den Verbindungsaufbau benötigen. Aus diesen Gründen werden zur Laufzeit einer Sitzung häufig Daten über *XMPP* Server versendet werden.

3.3 Arbeiten

Im folgenden schildere ich meine Leistungen als Mitglied des Saros-Projekts. In den nächsten zwei Abschnitten beschreibe ich die Umsetzung meiner Arbeitspakete. Ein Arbeitspaket ist eine Aufgabe innerhalb des Saros-Projekts, die einer Person zugeteilt ist. Später gehe ich näher auf meine Implementierungen in zentralen Objekten aus dem Netzwerkmodul ein.

3.3.1 bearbeitete Arbeitspakete

AP2 Geschwindigkeit der Dateisynchronisation optimieren

Um gemeinsam über Saros an einem Projekt arbeiten zu können, muss jeder Teilnehmer dieselbe Version des Projektes zur Verfügung haben. Dafür wird zu Beginn einer Sitzung das gesamte, zu teilende Projekt synchronisiert. Ein Kunde der AG Software Engineering setzte Saros zur Entwicklung zwischen Wien und Indien ein. Das geteilte Projekt hatte eine Größe von insgesamt 80MB. Die Dateisynchronisation dauerte ungefähr 15 Minuten. Um ihnen einen schnelleren Einstieg in die Arbeit zu gewährleisten, sollte der Transfer optimiert werden.

Die Dateisynchronisation wurde vor meiner Tätigkeit am Saros-Projekt dateiorientiert vollzogen. Dazu berechnete das System zu versendende Dateien oder Dateiänderungen und übertrug sie dann einzeln.

Die von mir vorgenommenen Optimierungen betreffen den Versand als Archiv und die Kompression noch nicht komprimierter Metadaten wie Dateilisten und Transferbeschreibung. Vor dem Transfer des Projekts wird ein temporäres Projektarchiv unter der Verwendung der bestehenden Methode `FileZipper::createProjectZipArchive()` gebaut. Metadaten wie Transferbeschreibung und Dateilisten werden zusätzlich für den Sender transparent komprimiert. Als Kompressionsalgorithmus für die Metadaten wurde von mir `DEFLATED` gewählt, da es das einzige Verfahren aus der *Java* Standardbibliothek ist, das auf Byte Feldern arbeitet. Die Implementierung ist in den statischen Methoden `Util::deflate` und `Util::inflate` zu finden. Sie enthalten Fortschrittsanzeige und Abbruchsmöglichkeit. Das Verfahren kann leicht ausgetauscht werden, falls ein besserer Algorithmus aus einer externen Bibliothek benutzt werden soll.

Während der Bearbeitung nahm ich auch die Kompression für die dateiorientierte Übertragung vor. Sie verwendet die gleichen Methoden wie zur Kompression von Metadaten benutzt wird. Ausgenommen von der Kompression innerhalb der dateiorientierten Übertragung sind schon komprimierte Datenformate, wie das Bildformat *JPEG*. Für die Erkennung von Ausnahmen gibt es eine statische *Dictionary* Datenstruktur in der Klasse `TransferDescription`. Mit ihrer Hilfe werden Dateien vor der Kompression an Hand der Dateiendung auf schon bestehende Kompression geprüft.

Der archivorientierte Versand ist bei langsamen Verbindungen schneller als der dateiorientierte. Die Messungen zu dieser Optimierung sind im Anhang [A.1] dokumentiert. Die Tabelle A.1 zeigt auch einen Nachteil der Lösung. Die Kombination schneller Netzwerke mit relativ kleinen Projekten wird durch den Bau eines Archivs verlangsamt. Die Optimierung wurde beibehalten, da der Vorteil für langsame Verbindungen viel größer ist, als der kleine Nachteil für die Schnellen.

AP4 Übertragung von Nachrichten via Peer-to-Peer

Aktivitäten werden in Saros zur Übermittlung von Benutzeraktivitäten inklusive Erstellung neuer Dateien in einer Sitzung verwendet. Sie werden mit Hilfe des `XMPP-Transmitter` übertragen. Zu Beginn meiner Arbeit wurden Aktivitäten ausschließlich in Form von `Chat Message` über XMPP Server versendet. Die Übertragung hat häufig eine für den Benutzer merkbare Latenz. Eine Optimierung wäre die generelle Nutzung einer *Peer-to-Peer* Verbindung zur Übertragung, da sich der Umweg über den Server vermeiden lassen würde. Ein weiteres Problem ist das Senden großer Datenmengen, z.B. bedingt durch einfügen einer großen Textpassage über die Zwischenablage. Saros wäre durch die Verwendung einer *Peer-to-Peer* Verbindung in der Lage den Durchsatz zu optimieren.

Der `XMPPTransmitter` trifft nach meiner Implementierung die Entscheidung, ob es sich lohnt eine *Peer-to-Peer* Verbindung zu nutzen. Normalerweise werden, daher Aktivitäten als `Message` über die XMPP Serrver versendet. Eine *Peer-to-Peer* Verbindung wird benutzt, wenn die Datenpakete größer als *16kB* sind, da der Verbindungsaufbau einen nicht zu vernachlässigbaren Overhead enthält. Es wurde auf den Wechsel auf die *Peer-to-Peer* Verbindung verzichtet, da die Latenz in der Übertragung für das Benutzerempfinden wichtiger ist als der reine Durchsatz. Der Aufwand des Verbindungsaufbaus liegt in der Architektur des zu Grunde liegenden Dateitransfers. Ein Dateitransfer wird für die Übertragung von einer Datei aufgebaut und bleibt nur bis zum Abschluss dessen bestehen. Das Verhalten ist in den verwendeten Protokollen beschrieben und von uns nicht änderbar.

Für die Umstellung waren vorher einige Refaktorings in der Klasse `XMPPTransmitter` notwendig, weil die interne Logik zur Versendung unnötig kompliziert war. Es gab z.B. eine Methode, die eine Liste von Nachrichten zu mehreren Benutzern versendet. Das komplizierte war, dass einige verschachtelte Vorbedingungen geprüft wurden, die lesbarer innerhalb der im Iterator aufgerufenen Methode sind. Die vorherige Implementierung war daher nicht immer verständlich.

AP5 Cancelation und Progress-Support von Dateiübertragungen

Bei langlaufenden Prozessen ist dem Benutzer häufig nicht ersichtlich, ob die Aktivität Fortschritte macht oder das Programm stehen geblieben ist. Manchmal möchte ein Benutzer einen begonnen Prozess abbrechen, anstatt ihn beenden zu lassen. Ein Beispiel eines langlaufenden Prozesses in Saros ist der Sitzungsaufbau. Einen großen Teil der Zeit verbringt Saros dort mit der Dateisynchronisation. Für die Teilnehmer war während des Prozesses keine Fortschrittsanzeige vorhanden und daher kein Ende ersichtlich.

Das Arbeitspaket Abbruch und Fortschritt hatte einen verhältnismäßig großen Umfang. Zur Umsetzung war es notwendig an sämtlichen Stellen der Netzwerkkomponente die Klasse `SubMonitor` zu verwenden. Viele Änderungen betrafen Methodensignaturen in Schnittstellen und erzeugten daher Folgearbeiten. Der Fortschritt kann nur

korrekt angeben werden, wenn der `SubMonitor` innerhalb einer Schleife schrittweise erhöht wird. Prozesse, die früher eine Einzeloperation auf einem großen Datenstrom ausführten, mußten daher in die Verwendung von Schleifen umgeschrieben werden. Die Verwendung von `SubMonitor` ermöglicht auch die Erkennung eines Abbruchs vom Benutzer. Es konnte daher überwiegend gleichzeitig Fortschritt und Abbrechbarkeit implementiert werden. Die Umsetzung betraf den kompletten Netzwerkstapel aus den Abbildungen 3.3 und 3.4.

Neu eingeführt wurde unter anderem die Klasse `BinaryChannel`, dessen Beschreibung sich in Abschnitt [3.3.5] befindet. Es war vorher nicht möglich über *Jingle* hergestellte Verbindungen sauber zu beenden, oder eine realistische Fortschrittsanzeige einzubauen. Das ehemalige Protokoll verwendete einen `ObjectStream` aus der Standardbibliothek von *Java* zur Übertragung des kompletten Projektarchivs in einem Schreib- und Leseaufruf. Die Konsequenz war, dass ein Abbruch nur grafisch vollzogen, der Datenstrom jedoch noch fertiggestellt wurde.

Um die Anzeige regelmäßig zu aktualisieren und einen gewollten Abbruch von einem fehlerhaften Zusammenbruch unterscheiden zu können, wurde von mir ein einfaches Protokoll entwickelt. Das Protokoll ist bidirektional verwendbar und kann während einer längeren Übertragung abgebrochen werden, ohne den *Socket* schließen zu müssen. Erst habe ich ihn als Prototyp losgelöst von Saros entwickelt und später integriert, weil es eine größere Änderung war, die ich losgelöst testen wollte. In meiner ersten Idee habe ich den `BinaryChannel` mit einer asynchrone Schnittstelle versehen. Öffentlich zugängliche Methoden füllten nur eine Warteschlange. Interne Methoden verarbeiteten die Schlange und schrieben die Daten in den `Socket`. Der erste Gedanke war eine höhere Effizienz zu erzielen und gleichzeitig leicht eine Priorisierung über die Wartschlange vornehmen zu können. Die später integrierte Variante erhielt, eine blockierende Schnittstelle. Die asynchrone Schnittstelle der Erstimplementierung hätte eine komplexe Handhabung von `SubMonitor` Objekten notwendig gemacht. Zusätzlich kann der `BinaryChannel` von außen über *Threads* betrieben werden, damit keine Blockade des Gesamtsystems entsteht.

Die Verwendung des von Smack zur Verfügung gestellten Datenstroms der `FileTransfer` Schnittstelle mußte auch entsprechend geändert werden. Die Verwendung von `SubMonitor` war notwendig, um Fortschritt anzeigen zu können und eine Abbruchsmöglichkeit zur Verfügung zu stellen. Das Verhalten sollte von außen wie im Fall der *Peer-to-Peer* Verbindung über den `BinaryChannel` sein.

Zusätzlich zum Abbruch einer Verbindung wurde von mir akzeptieren und ablehnen einer Verbindung ermöglicht. Ein Abbruch bezieht sich auf einen bestehenden Transfer, während die Ablehnung keinen Transfer zu stande kommen läßt. Die Dateiübertragung über die `FileTransfer` Schnittstelle aus der *Smack* Bibliothek unterstützt bereits das Akzeptieren und Ablehnen einer Verbindung. Die Funktionalität mußte intern nur in den `BinaryChannel` implementiert werden. Zusätzlich mußte die Wahlmöglichkeit der Benutzerschnittstelle zur Verfügung gestellt werden.

Peer-to-Peer Verbindungen werden vom `DataTransferManager` gekapselt. Er hat in allen Varianten der Dateiübertragung die Gelegenheit den eintreffenden Transfer abzulehnen. Das neu implementierte `IncomingTransferObject` ist dafür zuständig und bietet eine einheitliche Schnittstelle für `FileTransfer` und *Peer-to-Peer* Verbindung an. Benutzer des `DataTransferManager`s empfangen Daten, falls sie dem zugehörigen Transfer über das nach oben gereichte `IncomingTransferObject` zustimmen.

Außerdem ist eine Fortschrittsanzeige auf Empfängerseite nur durch das `IncomingTransferObject` möglich, da das Netzwerkmodul einen `SubMonitor` von der grafischen Oberfläche von Eclipse erhalten muss. Die Methode `IncomingTransferObject#accept()` bekommt daher einen `SubMonitor` als Parameter übergeben.

AP7 Smack Community

Häufig ist es unverständlich, ob ein Defekt innerhalb von Saros oder in der Bibliothek liegt. Die verwendete XMPP Bibliothek *Smack* ist an vielen Stellen fehlerhaft. Ein Beispiel dafür ist die Authentifizierungsmethode *SASL Digest MD5*[8] während des XMPP-Logins.

Viel Unterstützung aus der *Smack Community* war nicht zu erwarten, da viele aktuelle Foreneinträge unbeantwortet blieben. Das letzte *Smack* Release ist mittlerweile[9] 15 Monate her. Ich hatte nicht den Eindruck, dass noch während meiner Diplomarbeitszeit innerhalb der Smack Entwicklung aktiv etwas passiert. Mehrere Verbesserungsvorschläge samt *Patch* im *Bugtracker* von Smack waren seit einiger Zeit vorhanden. Durchsichten von Entwicklern zur Aufnahme eingereichter Defektbehebungen hatten wenig bis keinen Fortschritt[10].

Als ich im wöchentlich stattfindenden *Community Chat* offen über einen *Fork* von *Smack* gesprochen habe, gab es keine Kritik. Es gab sogar zurückhaltenden Zuspruch für eine Spaltung des Projekts. Mein Einsatz in der *Smack Community* bezog sich daher auf für uns wichtige, vorhandene, aber noch nicht eingemeindete *Patches* für Smack. Eine vorzeigbare Verbesserung ist die Verfügbarkeit von *SASL Digest MD5* für die Authentifizierung beim *XMPP-Login*. Der damalige technische Leiter, Christopher Özbek, hat daraufhin den *Fork*[11] von Smack für Saros und die *Smack Community* vollzogen.

AP9 Testsuite für die Netzkomponente

Dieses Arbeitspaket konnte nicht, wie von der Projektleitung erwartet, mit Hilfe einer Bibliothek, die eine Netzwerkschnittstelle simuliert, umgesetzt werden. Es war ein erheblicher Aufwand für die Erstellung einer Testsuite notwendig. Die weitere Beschreibung ist daher in einem separaten Kapitel [4] zu finden.

3.3.2 Abgebrochene oder nichtbearbeitete Arbeitspakete

Die Flexibilität agiler Arbeitsweise ist nicht unbedingt förderlich zur Vollendung vorgenommener Arbeit. Häufig ist es wichtiger sich auf Kunden einzustellen und deren Wünsche zu erfüllen, als strikt seiner eigenen Planung zu folgen. In meiner Projektdurchführung führte dies zu einem Abbruch eines angefangenen Arbeitspakets. Wegen des großen Aufwands für die geforderte Testsuite habe ich drei Arbeitspakete nur sehr oberflächlich bearbeitet.

[8] SASL Digest MD5: `http://tools.ietf.org/rfc/rfc2831.txt` (Online abgerufen am 23. Februar 2010)

[9] Heute ist der 18.2.2010

[10] Sieben Monate bis ein einfacher Patch in den Trunk aufgenommen wurde: `http://www.igniterealtime.org/issues/browse/SMACK-264` (Online abgerufen am 23. Februar 2010)

[11] geforktes Smack Repository `http://repo.or.cz/w/Smack.git` (Online abgerufen am 23. Februar 2010)

AP1 Netzwerk Monitor

Die internen Zustände sind für Entwickler nicht ersichtlich. Ein Beispiel dafür ist der Füllstand von Warteschlangen. Ein sogenannter *Netzwerk-Monitor* soll interne Zustände in einem `View` anzeigen. Entwickler können mit einem solchen Werkzeug interne Zustände beobachten und eventuell Schwächen entdecken.

In meiner Planung hatte ich vor dieses Arbeitspaket als Erstes zu bearbeiten. Unsere Kunden, die Teles AG, hatten zu dieser Zeit ein akutes Problem. Der Sitzungsaufbau produzierte zu lange Wartezeiten und mußte deshalb optimiert werden. Dies führte zum Entschluss dem zweiten Arbeitspaket den Vorzug zu geben und die weitere Bearbeitung des *Netzwerk-Monitors* zu verschieben. Später stellte ich die Bearbeitung weiter nach hinten, weil ich andere Arbeiten für wichtiger hielt. Die Planungsänderung passierte in einvernehmen mit der Projektleitung. Ein Beispiel eines wichtigeren Arbeitspakets ist *Progress-* und *Cancelation-Support*, ohne dass kein erneuter Verbindungsversuch bei der Projektsynchronisierung, ohne Eclipse zu beenden, möglich ist. Falls der Verbindungstyp nicht *Jingle/TCP* ist, hat es erhebliche Geschwindigkeitseinbußen zur Folge, wie sich aus Tabelle A.1 im Anhang schließen läßt.

AP3 Find matching project

Ein Eingeladener kann während des Einladungsprozesses wählen, ob ein neues Projekt angelegt oder ein lokal vorhandenes zur Synchronisation angeglichen werden soll. Falls der Eingeladene sich zur Wiederverwendung eines vorhandenen Projekts entscheidet, hat er zwei Varianten zur Wahl. Er kann ein Projekt selbst aussuchen oder den vorhandenen *Workspace* automatisch durchsehen lassen.

Ein Problem erscheint wenn der *Workspace* zu groß ist und dadurch der Vorgang lange dauert. Zur Lösung erscheint mir eine geeignete Heuristik zum Abbruch als sinnvoll. Wenn ein gefundenes Projekt sehr ähnlich ist kann die Suche beendet werden. Es vermeidet die Untersuchung weiterer Projekte aus dem *Workspace*.

In meiner Planung hatte ich dieses Arbeitspaket spät angesetzt. Ich halte es als ein schönes aber nicht unbedingt notwendiges Feature von Saros. Während einer Fehlerbeseitigung habe ich die Abbruchsmöglichkeit über einen `SubMonitor` bereitgestellt. Falls ein Benutzer jetzt den Vorgang abbricht, wird das aktuell bestpassendste Projekt vorgeschlagen, statt alle gesammelten Daten zu verwerfen. Der Benutzer hat die Wahl den Vergleich abzubrechen und den bereitgestellten Vorschlag zu akzeptieren oder sich anders zu entscheiden. Eine Heuristik habe ich aus Zeitmangel nicht erarbeitet.

AP6 Verbindungsaufbau über NAT

Dieses Arbeitspaket halte ich für ein sehr wichtiges, da sich viele potentielle Kunden hinter einem *NAT* befinden. Man denke z.B. an die Zusammenarbeit von Informatikstudenten, die jeweils von zu Hause aus gemeinsam ihre Hausaufgaben bearbeiten wollen.

Der Verbindungsaufbau über *NAT* sollte generell durch die Verwendung des *ICE* [Ros07] Protokolls gelöst sein. Leider funktioniert dies nicht immer. Sogar bei Teilnehmern aus Netzen, die sich direkt im Internet befinden und keine *NAT* Traversierung benötigen, versagt der Verbindungsaufbau in unregelmäßigen Abständen. Ein Beispiel eines von mir bekannten Versagens erfolgt, wenn mehrere Netzwerkschnittstellen auf einem Computer existieren. Dabei fällt die Wahl gelegentlich auf ein Falsches. Eine simulierte Netzwerkkarte einer virtuellen Maschine sollte für ein Verbindungsaufbau

verwendet werden. Die Fehlersuche ist schwierig, weil sich das Versagen nicht zuverlässig wiederholen läßt.

Für die Untersuchung Probleme dieser Art ist ein Testframework notwendig, das einen beherrschbaren Aufbau hat und dessen Netzwerkparameter einstellbar sind. Ein Großteil meiner Arbeitszeit verbrachte ich daher mit dem Aufbau eines solchen Frameworks, das in Kapitel [4] beschrieben ist. Ich bin der Meinung das zukünftig der Verbindunsgaufbau untersucht und die auftretenden Probleme beseitigt werden können.

AP8 Latenz-Optimierung

Gemeinsames Arbeiten innerhalb eines Dokuments ist bei einer hohen Latenz unangenehm für Saros-Anwender. Insbesondere wenn sich die Teilnehmer weit entfernt und in Netzwerken mit hoher Latenz befinden ist der Einsatz von Saros nur schwer möglich.

Die Optimierung in Netzwerken mit hoher Latenz plante ich nach dem Bau der Testsuite aus Abschnitt [3.3.1] vorzunehmen. Eine Testsuite mit einer einstellbaren Latenz ist zur Optimierung von großem Wert.

Trotzdem habe ich eine nachvollziehbare Verbesserung, während der Dateisynchronisierung [3.3.1] vorzuweisen. Die Versendung eines Archivs statt einzelner Dateien hat den Vorteil eine *TCP* Verbindung zu nutzen, die nicht immer wieder aufgebaut werden muss. Ein häufiger Wiederaufbau ist teuer, wenn ein Netzwerk eine hohe Latenz besitzt. Leider kam ich aus Zeitmangel nicht mehr zu weiteren Optimierungen.

3.3.3 Fehlerbeseitigung

Die Behebung von Defekten ist ein Teil der Arbeit an Saros. Anfangs mußte ich eine Inkompatibilität der verwendeten Bibliothek des *limewire* Projekts zur damals von *Mac OS X* mitgelieferten *Java* Version beseitigen. Später bremste ich Initiativen, die die abwärts Kompatibilität zu *Java5* gebrochen hätten. Ein Beispiel war die zeitweise Verwendung des Konstruktors `PipedInputStream(int pipeSize)` aus der *Java* Standardbibliothek. Der Konstruktor ist in *Java6* vorhanden, nicht aber in *Java5*.

Zu den wichtigsten Fehlerbeseitigungen von mir zählen der *XMPP*-Login über Proxyserver und die Verwendungsmöglichkeit von *SASL Digest MD5*. Außerdem ist die korrekte Dienstermittlung [3.3.5] bei Verwendung mehrerer *XMPP*-Klienten zu nennen. Anfangs wurde nicht darauf geachtet während der *service discovery* den richtigen *XMPP*-Klienten anzusprechen. Eine Folge daraus war häufiges Versagen, wenn eine XMPP fähige *Instant-Messaging* Software zur gleichen Zeit verwendet wurde.

Ein anderer kurioser Defekt zeigte sich beim *Doppelclick* auf den *XMPP-Connect-Button*. Es gab dort eine *Race Condition*, die ich beseitigte.

3.3.4 Kundenbetreuung

Im Saros-Team hat jedes Mitglied die Aufgabe das Saros-Projekt bekannter zu machen. In meinen Zuständigkeitsbereich fiel ein Internet Marktplatz für Eclipse-Plugins namens *eclipsepluginscentral*. Später wurde es in *marketplace*[12] umbenannt. Der Marktplatz wird jetzt offiziell von `Eclipse.org` bereitgestellt. Neue Ausgaben von Saros kündigte ich zeitnah auf der Seite an.

[12] `http://marketplace.eclipse.org/` (Online abgerufen am 23. Februar 2010)

In den gleichen Bereich fällt für mich die Beantwortung von E-Mail. Anfragen der Kategorie Netzwerk, übernahm ich selbstverständlich. Meiner Meinung nach ist die zeitnahe Reaktion wichtig für die Kundenbindung.

Ein Sonderfall war die Betreuung des *ADempiere*[13] Projekts in `irc://irc.freenode.net/#adempiere`, die ich auf Hinweis von Eike Starkmann[14] übernahm. Redhuan D. Oon konnte weder eine Sitzung beitreten noch andere erfolgreich einladen. Leider ließ sich das Problem nicht beheben.

3.3.5 Zentrale Objekte der Netzkomponente

In den folgenden Abschnitten beschreibe ich die Arbeitsweise zentraler Objekte aus dem Netzwerkmodul von Saros.

DiscoveryManager

Der `DiscoveryManager` verwendet zur Dienstermittlung nach *XEP-0030* [HMESA08] die Smack Bibliothek. Wie in Abschnitt [3.2.1] erwähnt, identifizieren Ressourcen *XMPP*-Klienten, z.B. *Instant-Messaging*-Software oder auch Saros. *XMPP*-Klienten beantworten eine Anfrage auf Dienste mit einer Liste unterstützter Feature. Die Kommunikation zwischen den beiden Klienten geschieht mit Hilfe von *IQ* Paketen.

Anfangs war die Handhabung der Dienstermittlung falsch, weil nicht auf die Angabe der Ressource einer *JID* geachtet wurde. Zwingend notwendig ist es die Ressource anzugeben, da sonst womöglich die falsche Anwendung nach unterstützten Features befragt wird. Jetzt wird bei der Feststellung unterstützter Dienste, erst die Ressourcenangabe für Saros einer *JID* ermittelt, und danach die Ermittlung des Saros Feature eingeleitet. Für den Anwender bedeutet dies, dass neben Saros auch andere *XMPP*-Klienten gestartet werden können, ohne Fehler in der Anwendung von Saros zu provozieren.

Der Prozess des Auffindens ist langlaufend und aufwendig, darum werden einmal ermittelte Daten zwischengespeichert. Falls eine relevante Änderung, wie Ab- oder Anmeldung, eintrifft werden die Daten aus dem Zwischenspeicher gelöscht. Relevant sind nicht die Präsenznachrichten *away* und *available*, weil sie keine Ab- oder Anmeldung von Saros bedeuten. Im Fall von Saros ist der Fokus auf Eclipse gleichbedeutend mit der Präsenz *available*.

XMPP Login

Der *XMPP-Login* ist ein zentraler Vorgang und bisher noch kein eigenständiges Objekt. Ein Login wird durch die Methode `Saros#connect()` vollzogen.

Ich stellte auf Benutzeranfrage die alternative Konfiguration des verwendeten XMPP-Ports und die Unterstützung von `XMPPConnection` über Proxyverbindungen zur Verfügung. Socks5 Proxyverbindung über einen *SSH* Tunnel sind von mir getestet worden und funktionieren. Ein anderer Test mit einem separat aufgesetzten *Dante*[15] Proxyserver mit Authentifzierung schlug fehl. Meines Wissens unterstützt *Smack* keine Authentifzierung an Proxyservern. Die Unterscheidung verschiedener Proxyarten des Eclipse Frameworks und Smack stimmen nicht überein. Smack unterscheidet zwischen *Socks4*,

[13] ADempiere Projektseite: `http://www.adempiere.de/` (Online abgerufen am 23. Februar 2010)
[14] Ein Student im Saros-Team
[15] Dante `http://www.inet.no/dante/` (Online abgerufen am 23. Februar 2010)

Socks5 und *HTTP*, während Eclipse die Unterscheidung *Socks*, *HTTP* und *HTTPS* kennt. Vielleicht ist dies eine Ursache des Fehlverhaltens der Bibliothek.

Die Authentifizierung via *SASL Digest MD5*, konnte durch einen von mir getesteten Patch aus *SMACK-264* unterstützt werden. Bisher existiert eine Lösung nur im *Trunk* von Smack und in der von Saros genutzten inoffiziellen Smack *Community* Version[16].

XMPPTransmitter und XMPPReceiver

Anfangs war der `XMPPTransmitter` die zentrale Versender- und Empfängerklasse. Eine Folge daraus war, dass es eine große Anzahl verschiedener *Listener* Objekte besaß. *Listener* sind auch unter dem Namen *Observer* [GHJV95] Muster bekannt.

Die aktuelle Version leitet eintreffende Pakete an den `XMPPReceiver` weiter, an dem sich Interessenten registrieren können. Der `XMPPReceiver` bietet die Möglichkeit über Smacks `PacketFilter` nur bestimmte Pakete zu empfangen. Dies kann nicht blockierend als Ereignis über den *Listenermechanismus* oder mit Hilfe von `SarosPacketCollector#nextResult()` blockierend geschehen.

Der `XMPPTransmitter` versendet Daten transparent über zwei verschiedene Kanäle. Die erste Variante verwendet Smacks `XMPPConnection` Klasse, um ein `Packet` an den Empfänger zu leiten. Aktivitäten innerhalb einer Saros-Sitzung werden an die anderen Teilnehmer über diesen Weg versendet, es sei denn die zu verschickenden Aktivitäten sind zu groß. Ab Größen von *16kB* wird die zweite Variante, der Aufbau einer *Peer-to-Peer* Verbindung über den `DataTransferManager`, vorgezogen. Dadurch werden große Einfügeoperationen anwendbar. Andere Benutzeraktivitäten sollten diese Größenordnung nicht errreichen, darum ist die Wahlgröße legitim.

Saros ist eine Sitzungsorientierte Software und hat zur Identifizierung eine Sizungsidentifkationsnummer im `SessionManager`. Die Schnittstellenmethode `sendMessageToProjectUser(JID,PacketExtension)` beachtet vor dem Abschicken einer Nachricht, ob sich der angegebene Benutzer im gemeinsamen Projekt befindet, die andere, `sendMessageToUser()`, achtet nicht darauf. Auf der anderen Seite bietet der `XMPPReceiver` die Gelegenheit einen Sitzungsfilter zu verwenden, um ausschließlich Nachrichten der eigenen Sitzung zu erhalten. Ignorieren der Sitzungsnummer und der Sitzungsteilnahme ist notwendig, falls man sich im Sitzungsaufbau befindet.

DataTransferManager

Der `DataTransferManager` ist zuständig für ausfallsichere, benutzertransparente `Socket` Verbindungen. Bevorzugt ist der Aufbau eines `JingleFileTransfer`, zur Ausfallsicherung kommt *Smacks* `FileTransfer` zum Einsatz. `FileTransfer` teilt sich intern in `Socks5Transfer` und `IBBTransfer` auf. Wie sich aus den Name bereits schließen läßt werden die Protokolle *Jingle* [LBSA+09], *Socks5 Bytestreams* [SMSA07] und *In-Band Bytestreams* [KSA09] vom `DataTransferManager` gekapselt.

In Abbildung 3.5 ist die Schnittstelle für den Datenversand zu sehen. Sie blockiert beim Aufruf bis die übergebenen Daten in den `Socket` geschrieben worden sind. Mit Hilfe des übergegebenen `SubMonitor`s ist ein Abbruch des Datentransfers zu jeder Zeit möglich. Dazu muss der Monitor mit Hilfe von `SubMonitor#cancel()` abgebrochen werden.

[16] inoffizielles Smack Community Repository: `http://repo.or.cz/w/Smack.git` (Online abgerufen am 23. Februar 2010)

```
public void sendData(TransferDescription description, byte[]
    data, SubMonitor progress) throws IOException,
    SarosCancellationException
```

Abbildung 3.5: Schnittstelle zum Versand von Daten über den `DataTransferManager`

```
public interface IncomingTransferObject {

  /**
  * Accepts a transfer and returns the incoming data.
  */
  public byte[] accept(final SubMonitor progress) throws
      SarosCancellationException, IOException;

  /**
  * Rejects the incoming transfer data.
  */
  public void reject() throws IOException;

  public NetTransferMode getTransferMode();
  public TransferDescription getTransferDescription();
};
```

Abbildung 3.6: Schnittstelle zum Empfang von Daten über den DataTransferManager

Der Empfang ist nur wenig schwieriger zu verstehen als der Versand und ist in Abbildung 3.6 abgebildet. Beim Empfang wird den höheren Schichten ein `IncomingTransferObject` vorgelegt. Der Empfänger kann den Transfer mit `reject()` ablehnen oder mit `accept(SubMonitor)` annehmen. Nach der Annahme kann im weiteren Verlauf des potentiell langwierigen Prozesses wie oben beschrieben der Empfang abgebrochen werden. Der Abbruch ist daher beidseitig möglich. Der Empfänger wird asynchron benachrichtigt, damit ein weiterer Empfang nicht blockiert.

In Zukunft können im `DataTransferManager`, weitere Protokolle gekapselt werden. Meiner Meinung nach ist die Idee dieses *Singleton* [GHJV95] Objekts ausgereift und damit stabil für die Zukunft.

Jingle Klassen

`JingleFileTransferManager` verwaltet `JingleFileTransferSession`s und versucht über Smacks `ICETransportManager` die richtige Netzwerkschnittstelle zur Übertragung ausfindig zu machen. Der Verbindungsaufbau wird von der `JingleFileTransferSession` erstmal per *TCP* und 7,5 Sekunden verzögert per *UDP* versucht. Wenn 30 Sekunden abgelaufen sind und keine Verbindung aufgebaut werden konnte bricht er ab. Bei Erfolg wird ein `BinaryChannel` erstellt und für den Betrieb ein *Empfängerthread* gestartet. Ankommende Daten aus dem `DataTransferManager` werden an den `BinaryChannel` weitergereicht und von dort aus versendet.

Im Fall von *UDP* wird eine *RUDP*[17] Implementierung des limewire Projekts verwendet. Das Eintreffen von Daten in Reihe ist daher auch im Fall von *UDP* zuverlässig.

BinaryChannel

Der `BinaryChannel` ist eine Neuentwicklung zur Unterstützung von Abbrechbarkeit und Forstschrittsanzeige, beschrieben in Abschnitt [3.3.1].

Das ursprünglich verwendete Protokoll nutzte *Javas* `ObjectOutputStream` und `ObjectInputStream` zum Transfer von Daten. Die Probleme, die sich dabei ergeben betreffen insbesondere den Abbruch und die Fortschrittsanzeige. Zum einen war der extern anzuzeigende Fortschritt nicht messbar, zum anderen konnte der Transfer nicht abgebrochen werden ohne den darunterliegenden `java.net.Socket` zu schließen. Die Auswirkung für den Benutzer war, dass ein Abbruch einer laufenden Dateiübertragung, z.B. ein Projektarchiv bei der Einladung, nur in der grafischen Oberfläche stattfand. Die Daten wurden komplett übertragen und die einzige Abhilfe war das schließen von *Eclipse*!

Die Entwicklung eines neuen, einfachen, bidirektionalen Protokolls war notwendig, um während einer laufenden Verbindung einen Kommunikationskanal zum abbrechen zu haben. Meine Implementierung des `BinaryChannel` spaltet serialisierte Objekte in 64k große Pakete auf und versendet sie als `BinaryPacket`. Auf Empfängerseite werden ankommende Pakete in die zugehörige Liste einer `HashMap` gesammelt. Zusammengehörige Objektteile werden durch neu eingeführte Identifizierer namens `objectid` erkannt. Es können daher Teile verschiedener Objekte gleichzeitig eintreffen. Falls das letzte `BinaryPacket` eines Objekts eintrifft, wird die Hauptschleife verlassen und zur `JingleFileTransferSession` zurückgekehrt. Diese benachrichtigt aus der Empfangsschleife heraus über `IJingleTransferListener` den `DataTransferManager`, der das `IncomingTransferObject` erhält und an die Geschäftslogik weiterreicht. Die Empfangsschleife kehrt gleichzeitig zum Betrieb des `BinaryChannel` zurück, um weiter eintreffende Daten aus dem `Socket` lesen zu können. Für diesen Mechanismus findet ein Threadkontextwechsel im `DataTransferManager` statt, der den Empfang von der Annahme entkoppelt.

Die Annahme eines Transfers kann akzeptiert oder abgelehnt werden. Die vor dem Datentransfer versendete `TransferDescription` ist ein eigenständiges Objekt und wird dem Benutzer als Anfrage vorgelegt. Eine `TransferDescription` beschreibt ankommende Daten und wird immer vor den zugehörigen Daten versendet. Dank der zugrundeliegenden Transportschicht der Protokolle *TCP* und *RUDP* trifft die `TransferDescription` zuverlässig vor den Daten ein.

Ein Abbruch eines laufenden Transfers geschieht durch versenden eines `REJECT` Paket. Auf Empfängerseite löscht ein Abbruch die Liste der bereits eingetroffenen Objekte mit der angegebenen `objectid` aus dem `REJECT` Paket. Auf Senderseite beendet der `BinaryChannel` den Versand des angegebenen Objekts und signalisiert den Abbruch seinem Aufrufer mit einer `SarosCancellationException`. Wie der richtige *Threadkontext* eingehalten wird, um dem Aufrufer die Ausnahme zu überbringen ist in der Bachelorarbeit von Tas Sóti erläutert [Só09].

Das Protokoll kennt folgende Daten- und Kontrollpakete.

- Datenpakete

[17] RUDP garantiert eintreffen von Datagrammen in Reihe `http://en.wikipedia.org/wiki/Reliable_User_Datagram_Protocol` (Online abgerufen am 23. Februar 2010)

- TRANSFERDESCRIPTION: Das Paket enthält ein Teil einer `TransferDescription`.
- DATA: Das Paket enthält ein Teil eines serialisierten Objekts

- Kontrollpakete
  - CANCEL: Sender bittet Empfänger um Abbruch
  - REJECT: Empfänger lehnt Daten ab
  - FINISHED: bestätigt Empfang aller Daten des Transfers
  - SHUTDOWN: `BinaryChannel` soll beendet werden

3.3.6 Fazit

Die Netzwerkkomponente hat Verbesserungen in ihrer funktionsweise erhalten. Für Benutzer transparente Anwendung von Kompression, der Einsatz von *Peer-to-Peer* Kommunikation bei Versand großer Nachrichten und die Abbrechbarkeit und Fortschrittsanzeige in allen Teilen der Software sind die großen neuen Feature. Unterschiedliche Defekte konnten behoben werden, die Wichtigsten waren die Unterstützung von *Mac OS X* und die Handhabung verschiedener *XMPP*-Klienten. Ich denke der Netzwerkstapel ist gut gekapselt und insgesamt verständlicher und wartbarer geworden.

Was habe ich gelernt?

Ich habe gelernt, dass die Entwicklung von Verbesserungen in Netzwerksoftware nicht unerheblichen Mehraufwand enthält. Insbesondere Messungen sind erforderlich und müssen extra angefertigt werden, falls relevante Daten nicht automatisiert erhoben werden.

Nach dem Lesen des *Refactoring* Buchs [Fow99] von Martin Fowler habe ich Refaktoring als Werkzeug verstanden. Verstehen durch Refaktoring ist ein Ansatz den ich leider erst zum Schluss meiner Diplomarbeit gelernt habe.

3.4 Ausblick

Abschließend beschreibe ich bestehende Defekte und präsentiere Ideen die meiner Meinung nach Verbesserungen im Netzwerkmodul darstellen.

3.4.1 Defekte

Die folgenden Abschnitte enthalten kurze Beschreibungen Netzwerkmodul bezogener Defekte, die meiner Meinung nach in Zukunft beseitigt werden müssen. Ich hoffe, dass die Beschreibung anderen Entwicklern hilft die Problematik zu verstehen.

Verbindungsaufbau ist richtungsorientiert Teilnehmer A sendet Teilnehmer B etwas über eine *Peer-to-Peer* Verbindung und B auch zu A, dann werden in beide Richtungen *Peer-to-Peer* Verbindungen aufgebaut. Der Overhead ist nicht notwendig.

Persistenz der Peer-to-Peer Verbindung Nach Datenübertragung über eine *Peer-to-Peer* Verbindung wird diese wieder geschlossen. Es hört sich einfach an eine Verbindung persistent zu halten. Leider ist die Abhängigkeit `FileTransfer` Schuld.

Eine Behebung ist wahrscheinlich nur durch Änderungen in *Smack* möglich. Eine Änderung dieser Art scheint mir nicht für alle verwendeten Protokolle realistisch zu sein. Eine Teiländerung sollte meiner Meinung nach im Fall einer Jingleverbindung durchgeführt werden, da man dort innerhalb des Sarosquellcodes bleibt. Eine Verbesserung von Durchsatz und Latenz sollte an dieser Stelle leichter durchgeführt werden können.

Aufbau der Peer-to-Peer Verbindung funktioniert manchmal nicht An Testtagen gibt es immer wieder Probleme beim Aufbau der *Peer-to-Peer* Verbindung. Es gab sogar Fehlschläge, obwohl sich alle Teilnehmer im gleichen Netz befanden! Meines Wissens kann der Defekt nur auftreten, wenn mehrere Netzwerkschnittstellen auf einem Computer vorhanden sind. Der Mechanismus für *NAT* Traversierung wählt u.U. eine virtuelle Netzwerkkarte aus und führt zum beschriebenen Fehlverhalten.

Service Discovery Die Dienstermittlung ist in seltenen Fällen fehlerhaft. Bei der Einladung sollen eigentlich nur Teilnehmer aufgelistet werden, die auch Saros unterstützen. In der Voreinstellung ist der Filter deaktiviert, da in unregelmäßigen Abständen die herausgefundene Sarosunterstützung falsch ist.

3.4.2 Ideen

Meiner Meinung nach sollte die Komplexität im `JingleFileTransferManager` und im Verbindungsaufbau der Klasse `JingleFileTransferSession` reduziert werden. Dort befindet sich nach meiner Arbeit die höchste Komplexität. Die meisten wichtigen Schwachstellen sind dazu in den genannten Teilen enthalten. Beheben von Defekten ist durch eine gezielte Reduktion komplexer Bestandteile wahrscheinlicher [Fow99].

Eine umfangreiche Idee für die Netzkomponente wäre die Umstellung auf *ECF*[18]. *ECF* hat mehr Abstraktion in den Bereichen Dienstermittlung und Kommunikationskanal, als unsere bisherige Lösung. *ECF* unterstützt verwendet unter anderem *Smack* als Implementierung für *XMPP*, eine Migration sollte daher transparent für den Benutzer möglich sein. Der Einsatz von spontanen *Ad-Hoc* Sitzungen wäre über das implementierte Protokoll *zeroconf*[19] möglich. Ich bin der Meinung, dass *Ad-Hoc* Sitzungen einen Mehrwert darstellen.

Zu Beginn einer Sitzung findet durch den Replikationsansatz immer eine langlaufende Projektsynchronisation statt. Ein Verbesserung wäre ein Ansatz den ich mit *Lazy Replikation* bezeichne. Alle Dateien werden im Hintergrund übertragen und Eingeladene können sofort am Projekt teilnehmen. Dateien, die in das Sichtfeld des Benutzers gelangen, sollten eine Priorisierung in der Übertragung erhalten.

[18] ECF - Eclipse Communication Framework `http://wiki.eclipse.org/Eclipse_Communication_Framework_Project` (Online abgerufen am 23. Februar 2010)

[19] zeroconf `http://www.zeroconf.org/`, Protokoll `http://tools.ietf.org/rfc/rfc3927.txt` (Online abgerufen am 23. Februar 2010)

Kapitel 4

Arbeit Teil 2: Netzwerk Testumgebung

Im folgenden Kapitel beschreibe ich den zweiten Teil meiner Arbeit, die Entwicklung einer Netzwerk Testumgebung. Wie in Arbeitspaket 9 [3.3.1] kurz erwähnt war die Umsetzung einer Testsuite nicht in der dafür vorgesehenen Zeit möglich. Durch die Auseinandersetzung mit dem vorliegenden Problem wurde der Umfang dieser Arbeit erst deutlich. Es entwickelte sich zum Kernbestandteil meiner Diplomarbeit.

Ich beginne mit meiner Motivation und beschreibe im Anschluss die Anforderungen an eine Testsuite. Auf den Entscheidungsprozess wird in Abschnitt [4.3] eingegangen gefolgt von der Umsetzung [4.4]. Darauf folgend erkläre ich die Anwendung [4.5] des gebauten Frameworks. Abschließend ziehe ich mein Fazit [4.6] zum Testframework und gehe auf zukünftige Verwendungsmöglichkeiten ein.

4.1 Motivation

Innerhalb der Bearbeitung verschiedener Arbeitspakete traten immer wieder unregelmäßige Fehler in der Netzschicht auf [3.4.1]. Ein Beispiel für einen solchen Fehler ist die Dienstermittlung [1] nach *XEP-0030* [HMESA08] im `DiscoveryManager`.

Manchmal kommt es bei der Abfrage nach der Sarosunterstützung zu fehlerhaften Antworten. Das Problem ist zum einen die Wiederholbarkeit und zum anderen die Schwierigkeit der Identifizierung des Defekts. Es kann verschiedene Gründe haben warum ein Defekt solcher Art auftritt. Es könnten logische oder zeitliche Fehler bei *Caching*, in der Verwendung der *Smack-API*, Fehler innerhalb der *Smack-API* oder Fehler in der Beantwortung der Anfragen innerhalb verschiedener Serversoftware sein. Schlimmstenfalls ist es eine Kombination aus den oben genannten Fehlerquellen.

Die Entwicklung eines Testframeworks war notwendig, da es zur Fehleranalyse und dadurch zur Fehlerbehebung beiträgt. Durch parametrisierbare Tests lassen sich Bedingungen einstellen, die Versagen der Software reproduzieren und nachvollziehbar machen. Die Automatisierung ausgeführter Testschritte und deren Validierung bietet zusätzlich Wiederholbarkeit.

[1] *service discovery* nach *XEP-0030* `http://xmpp.org/extensions/xep-0030.html` (Online abgerufen am 23. Februar 2010)

4.2 Anforderungen

Das Testobjekt, Saros, ist ein Eclipse Plugin zur verteilten Paarprogrammierung. Das Minimalszenario zum Testen beinhaltet daher neben einer Teststeuerung mindestens zwei Sarosinstanzen. Zur Einfachheit unterscheide ich die zwei Instanzen mit den Namen *Quelle* eines Ereignisses und *Ziel*, von dem das Ergebnis zur Verifikation eingesammelt werden muss.
Die größten Herausforderungen eines automatischen Tests von Saros sind:

- *Quelle* und *Ziel* befinden sich nicht in der gleichen Applikation und vielleicht noch nicht einmal auf dem gleichen Rechner.
- Eine zu testende Aktion wird auf der *Quelle* durchgeführt. Das Ergebnis muss auf dem *Ziel* verifiziert werden.
  - Zeitablauf (*Timeout*)
  - Das Resultat muss vor dem nächsten Test überprüft werden.
- Verifikation der Ergebnisse in einer Umgebung mit mehr als zwei unterschiedlichen Instanzen.
- Testen der Mehrschreiberfunktionalität ist komplex

Die Kernanforderungen an ein Testframework zur Bewältigung der genannten Herausforderungen sind daher:

- Ablauf eines Tests muss synchronisiert und verteilt sein
- Ergebnisse müssen von einem *Nicht-Quellrechner* erhalten werden

4.2.1 funktionale Anforderungen

Saros hat unterschiedliche Benutzergruppen, manche befinden sich netztechnisch direkt im Internet, manche hinter einem *NAT* oder einem *Portfilter*. Schlechte Leitungsqualität, kabellose Netze oder große Entfernungen können zu hohen Latenzen der Verbindung zwischen Anwendern führen. Saros soll natürlich auch in solchen Situationen funktionieren. Die Testumgebung sollte daher eine gute Abdeckung von Netzwerkeigenschaften abbilden können. In Tabelle 4.1 sind Testszenarien abgebildet, die insbesondere für die Validierung des Verbindungsaufbaus wichtig sind. Eine große Latenz oder geringe Bandbreite darf nicht die Ursache für ein komplettes Versagen von Software sein. Daraus folgt die Anforderung Netzwerkeigenschaften einstellbar zu machen.

4.2.2 nichtfunktionale Anforderungen

Benutzbarkeit: Die Testsuite **soll** möglichst automatisch nach einem Startbefehl ablaufen.

Erlernbarkeit: Alle Entwickler des Saros-Projekts **müssen** mit vertretbarem Aufwand in der Lage sein das Testframework benutzen zu können.

Java: Tests **müssen** in der Programmiersprache Java geschrieben werden können.

Tabelle 4.1
grob klassifizierte Netzwerkszenarien zweier Teilnehmer einer Saros-Sitzung

Internet	Internet
Internet	NAT
Internet	Jingle blockiert
Internet	Latenz
NAT	NAT
NAT	Jingle blockiert
NAT	Latenz
Jingle blockiert	Jingle blockiert
Jingle blockiert	Latenz
Latenz	Latenz

Portierbarkeit: Die Testsuite **muss** leicht auf einen anderen Rechner portiert werden können.

Wartbarkeit: Es **soll** möglichst wenig Arbeitszeit der Systempflege gewidmet werden müssen.

4.3 Auswahl eines Testframeworks

Wiederverwendung ist eines der Kernprinzipien des Software Engineering. Durch eine geeignete Frameworkwahl kann wertvolle Zeit eingespart und die Testmaschine ein Erfolg werden. In diesem Abschnitt gehe ich auf den Entscheidungsprozess verwendeter Softwarepakete und Frameworks ein.

4.3.1 Alternativen

Java Testframeworks

Java Testframeworks, die in verteilten Umgebungen laufen, gibt es viele. Die meisten von ihnen haben das Ziel Unit Tests schneller ablaufen zu lassen. Einige von diesen Frameworks schaffen es im Fehlerfall den *Stacktrace* zum zugehörigen Test auszugeben. In diese Kategorie fielen *jtrunner*[2], Grinder[3] und GridGain[4]. Die Dokumenation von *jtrunner* habe ich komplett lesen müssen, um festzustellen, dass mein erdachter Einsatzzweck vom Framework nicht unterstützt wird. Bei *Grinder* und *GridGain* war es leicht ersichtlich, dass sie mir nicht weiterhelfen. Es war bei denen klar, dass sie nur Tests verteilt ablaufen lassen und die Ergebnisse aufbereitet zusammenstellen und daher keine Synchronisierung von Tests ermöglichen.

Wie sich herausstellte gibt es leider nur ein Framework, *STAF*[5], das in der Lage ist, außer des verteilten Ablaufs, eine Synchronisation zu ermöglichen. *STAF* ist ein verteiltes, *Cross-platform* Testframework entwickelt von *IBM*.

[2] `http://jtrunner.sourceforge.net` (Online abgerufen am 23. Februar 2010)
[3] `http://grinder.sourceforge.net/` (Online abgerufen am 23. Februar 2010)
[4] `http://www.gridgain.com/` (Online abgerufen am 23. Februar 2010)
[5] `http://staf.sourceforge.net/` (Online abgerufen am 23. Februar 2010)

Eclipsefernsteuerung ist eine andere Variante, die aus Abschnitt [4.2] genannten Anforderungen zu erfüllen. Mit Hilfe einer Fernsteuerung ist ein Test in der Lage mehrere Eclipse-Instanzen gleichzeitig bedienen zu können. Es ist daher ein geeigneter Ansatz.. *SWTBot*[6] ermöglicht die Bedienung von Eclipse. Benötigte Anforderungen wie verteilter Ablauf und Synchronisation müssen selbst implementiert werden.
Zur Übersichtlichkeit sind die Ergebnisse in Tabelle 4.2 übersichtlich dargestellt. Es folgt eine Erklärung zu den Einträgen in der Tabelle 4.2.

verteilt: Starten von Tests auf unterschiedlichen Computern ist möglich.

synchronisiert: Der Testablauf ist synchronisiert steuerbar. Eine Aktion A wird auf einem Computer C ausgeführt und danach die Aktion Aáuf einem Computer C, A,A,C und Cńüssen dabei wählbar für den Test sein.

JUnit: JUnit-Test oder eine andere Art Integration ist vorhanden. Die Einarbeitungszeit soll gering sein und es soll leicht ersichtlich sein, ob ein Testlauf erfolgreich war oder nicht.

Eclipse-Plugin: Eclipse-Plugins sind ohne großen Aufwand steuerbar.

Tabelle 4.2
Testframeworks

Framework	verteilt	synchronisiert	JUnit	Eclipse-Plugin
STAF	ja	möglich	keine Angabe	nein
jtrunner	ja	nein	ja	nein
Grinder	ja	nein	ja	nein
GridGain	ja	nein	ja	nein
SWTBot	nein	nein	ja	ja

Netzwerkframework

Bisher bin ich nur auf den Testablauf eingegangen. Verschwiegen wurde von mir die Umgebung, in der Tests eingebettet und die Netzwerkbedingungen aus Tabelle 4.1 prüfbar werden.
In der Erforschung von Netzwerken werden zwei verschiedene Ansätze verfolgt, Simulationen[7] und Emulationen[8]. Sie werden verwendet, um Tests in einer kontrollierten Umgebung durchführen zu können. Die Umgebung soll sich möglichst wie die echte Welt verhalten, dabei aber parametriesierbar sein, d.h. Variablen sollen beeinflußt werden können. In unserem Fall wollen wir Variablen wie direkte Verbindung, Netzwerkadressumsetzung, Verzögerung und Paketfilter einstellen. Andere Beispiele für spezialisierte Umgebungen zur Erforschung von Netzwerken sind *ORBIT* [ORB] für *WLAN*, *VINI* [VIN] für *Routing*, *Roofnet* [BABM05] für *MESH* und *ModelNet* [VYW+02] für Sensorennetzwerke.
Eine Netzwerkemulation erlaubt es Endsysteme in einem Netz zu haben, dessen Eigenschaften durch den Emulator erzeugt werden. Im Gegensatz zur Emulation kann

[6] `http://eclipse.org/swtbot/` (Online abgerufen am 23. Februar 2010)
[7] `http://en.wikipedia.org/wiki/Network_simulation` (Online abgerufen am 23. Februar 2010)
[8] `http://en.wikipedia.org/wiki/Network_emulation` (Online abgerufen am 23. Februar 2010)

in einer Netzwerksimulation kein realer Computer dem Netz hinzugefügt werden. Ein Vorteil der Simulation ist die einfache Benutzbarkeit, da sie auf einem Rechner läuft. Leider wurde keine geeignete Simulation gefunden, die alle Bedingungen aus Tabelle 4.1 simulieren kann. Die Implementierung einer Simulation kam nicht in Frage, weil man *DNS* mit *SRV*, *NAT*, *STUN*, *XMPP* sowie alle zu testenden *XEP* hätte simulieren müssen. Es gab daher keine Wahl zwischen Simulation und Emulation.
Drei verschiedene Emulatoren haben meine Aufmerksamkeit bekommen, weil sie frei verfügbar und scheinbar stabil waren:

Dummynet [Riz] ist Teil von FreeBSD und ist benutzbar über das Frontend *ipfw*

Netem [NET] ist Teil des 2.6er Linux Kernel und ist benutzbar über das Frontend *tc*

NistNET [Nis] ist unter Linux installierbar und wurde von der *NIST*[9] entwickelt.

Sowohl *Dummynet*, als auch *Netem* beherschen die Emulation realer Bedingungen wie Paketverlust, Verzögerung und Bandbreiteneinschränkung. Ipfw ist eine Firewallsoftware unter FreeBSD, die daher auch Portfilterung ermöglicht.
NistNET wurde nicht weiter in Betracht gezogen, weil sich mit *Netem* ein Kandidat schon im Linux Kernel etabliert hat.

4.3.2 Entscheidung

Entscheidung für ein Java Testframework

Es gab zwei wählbare Alternativen *STAF* und *SWTBot*. Ich habe mich für *SWTBot* entschieden und erläutere meine Entscheidung.
STAF ist ein großes sehr flexibles Framework zum verteilten Testen. Es unterstützt verschiedene Sprachen, darunter Java. Es existiert eine große Sammlung an Dokumentation[10]. Nach lesen der *Getting Started* Anleitung war mir die Verwendung in unserem Fall unklar. Mir scheint *STAF* für das Projekt nicht angebracht, weil Saros durch *STAF* nicht steuerbar wird. Weiterhin sieht mir das Projekt einfach zu komplex für den Anwendungsfall für das Testen von Saros aus.
Unter anderem aus Mangel an Zeit zur Evaluation von *STAF* und der bestehnden Schwierigkeit Eclipse-Plugins zu testen habe ich mich für *SWTBot* entschieden.
SWTBot ist ein Eclipse-Projekt und bietet die Möglichkeit *SWT* Komponenten und Eclipse zu testen. Es ist das einzig aktuelle von mir gefundene Projekt mit diesem Ziel. Es gibt in der Kategorie *SWT* noch das Projekt *Remote SWT*[11]. *Remote SWT* hat seit dem Jahr 2003 keine Änderungen mehr erhalten und die Projektseite auf die verwiesen wird ist nicht mehr erreichbar. Mein Schluss daraus ist, dass *SWTBot* das relevante Projekt ist.
SWTBot läuft normalerweise als spezieller Unittest mit einem eigenen *TestRunner*. Die Teststartklasse heißt `SWTBotJunit4ClassRunner` und ist für *JUnit4*[12] ähnliche Tests gedacht. Das Framework ist als *Fluent Interface* [Fow05] implementiert, um eine einfache Handhabung zu gewährleisten. Das Design Muster *Fluent Interface* ist auch unter den Namen *Cascade*[Cro08] oder *ExpressionBuilder*[13] bekannt.

[9] NIST: National Institute of Standards and Technology
[10] `http://staf.sourceforge.net/docs.php` (Online abgerufen am 23. Februar 2010)
[11] `http://sourceforge.net/projects/rswt/` (Online abgerufen am 23. Februar 2010)
[12] JUnit: `http://www.junit.org/` (Online abgerufen am 23. Februar 2010)
[13] Martin Fowler `http://martinfowler.com/dslwip/ExpressionBuilder.html` (Online abgerufen am 23. Februar 2010)

```
# define bandwidth and delay of the emulated link
ipfw pipe 1 config bw 3Mbit/s delay 32ms
# pass all traffic through the emulator
ipfw add pipe 1 ip from any to any
```

Abbildung 4.1: Komfortables konfigurieren von Netzwerkeigenschaften mit der Kombination *ipfw* und *dummynet*.

Um verteiltes Testen zu ermöglichen, ist eine Bedienung mehrerer Instanzen benötigt. Die Implementierung einer *Java RMI* Schnittstelle ist meine Lösung zur Fernsteuerung von mehreren Sarosinstanzen. Die Lösung der Synchronisierung ist die komplette Teststeuerung in einem separaten Prozess laufen zu lassen. Tests werden nicht mit dem `SWTBotJunit4ClassRunner` gestartet, da sie einen unabhängigen Ablauf zur Konsequenz hätte, die nachträglich synchronisiert werden müßte.

Entscheidung für ein virtuelles Netzwerk

Eine standardisierte Umgebung auf einem realen Computer zu bauen führt zu der Nutzung von virtuellen Maschinen. Der vorhandene Computer mit dem NAmen `ibadan` läuft unter Microsoft Windows 2003 Server. Auf ihm habe ich *VMware Server*[14] installiert. VMware ist der größte Hersteller von *VM* Produkten und bietet virtuelle Maschinen Kontainer zum freien Download über einen Marktplatz an. Darüber hinaus kann das gewählte Produkt auch komplexere Netze auf einem Host-Computer abbilden.

Die Netzwerkemulationen *Netem* und *Dummynet* haben eine sehr ähnliche Ausstattung. Im Gegensatz zu Netem hatte ich das Frontend *ipfw* vorher bereits verwendet. Um Risiken zu vermeiden, wurde daher Dummynet als Emulator gewählt. Ein weiterer Vorteil von *Dummynet* ist die Nutzung von *Portfilter* und *Netzemulator* aus dem gleichen Frontend.
Dummynet bietet eine komfortabele Schnittstelle in Form einer *Pipe*, die konfigurierbare Eigenschaften besitzt. Im Beispiel 4.1 wird die *Pipe Nummer 1* mit einer Bandbreitenbeschränkung von *3 MBit/s* und einer Verzögerung von *32ms* eingestellt. Die *Pipe* wird auf alle *IP* basierten Verbindungen angewendet [MC09]. Nach der Entscheidung für *Dummynet* war noch die Entscheidung über das Emulator Betriebssystem zu treffen. Eine natürliche Entscheidung war FreeBSD zu nutzen, weil Dummynet teil dieses UNIX Derivates ist. Ein weiterer Vorteil von FreeBSD ist die sehr übersichtliche Konfiguration von Netzwerkschnittstellen in der zentralen Konfiguration `/etc/rc.conf`. Die komplette Konfigurtion ist in Anhang [B.1] abgebildet.
Als XMPP Serversoftware kommt *Openfire*[15] zum Einsatz, weil es ein von uns empfohlenes Softwarepaket ist. *Openfire* verwendet die XMPP Implementierung Smack vom gleichen Herrsteller. Als XMPP Hostsystem kommt das aktuelle Debian stable zum Einsatz, weil es ein stabiles und leicht administrierbares System ist. Alle notwendigen Dienste wie *DNS* und *STUN* stehen als Pakete der Distribution zur Verfügung, was die Installation und Wartung vereinfacht.
Die Entscheidung des Betriebsystems der Sarosklienten fiel auf Ubuntu, weil es das von Debian entwickelte, einfach zu bedienende Paketsystem *dpkg* besitzt. Ubuntu spart

[14] `http://www.vmware.com/products/server/` (Online abgerufen am 23. Februar 2010)
[15] `http://www.igniterealtime.org/projects/openfire` (Online abgerufen am 23. Februar 2010)

Einarbeitungszeit, da ich Debian bereits gut genug verstehe. Der Vorteil von Ubuntu im Gegensatz zu Debian, als Sarosklienten-System ist die Aktualität von Softwarepaketen und der vermehrte Einsatz auf Benutzerrechnern. Der Vorteil von Ubuntu gegenüber Windows ist, dass alle Rechner gleich administriert und gesteuert werden können. Durch die relativ homogene Wahl der Betriebsysteme ist die vorhandene Komplexität reduziert.

4.4 Umsetzung

Zur Umsetzung wird eine Testumgebung auf `ibadan.imp.fu-berlin.de` mit VMware Server aufgebaut und eine Teststeuerung implementiert. Als erstes erläutere ich die Idee des selbstimplementierten Java Frameworks [4.4.1], danach beschreibe ich die Besonderheiten der Umgebung [4.4.2], den Bau der Einzelkomponenten [4.4.3] und zu guter letzt die Verteilung [4.4.4] auf die Testrechner.
Die Verwendung des Frameworks mit Hilfe von Unit Tests wird in einem späteren Abschnitt [4.5] beschrieben.

4.4.1 Idee des Java Testframeworks

Für das Framework müssen Sarosinstanzen steuerbar sein. Sarosinstanzen laufen im Testnetz und in der Realität auf verschiedenen Rechnern, d.h. sie müssen fernsteuerbar sein. In Abbildung 4.2 ist ein UML Komponenten Diagramm einer Testszene zu sehen, die über *RMI* fernsteuerbar ist. Eine Testszene entspricht einem **Orchester**. Es besteht aus zu steuernden Saros-Instanzen im folgenden auch **Musikanten** genannt, und der Teststeuerung, dem **Dirigent**.
Saros wird automatisch Musikant, wenn die Laufzeitumgebung, *JVM*, mit dem Parameter `-Dde.fu_berlin.inf.dpp.testmode` gestartet wird. Dazu habe ich eine Auswertung des genannten optionalen Parameters beim Start von Saros vorgenommen. Parameter können in der *Eclipse Runconfiguration* eingestellt werden. Diese Konfiguration enthält alle Parameter und Eigenschaften für den Start eines Plugins, eines Java Programms oder eines *JUnit* Tests aus Eclipse heraus.

Der Dirigent ist ein einfacher *JUnit4* Test. Er steuert die Musikanten über die von mir entwickelte *RMI* Schnittstelle, wie in Abbildung 4.2 zu sehen ist. Bisher von mir gebaute Testklassen beinhalten bisher immer nur eine Testmethode, weil viele Szenarien, z.B. eine Einladung, die Einhaltung eines bestimmten Ablaufs vorschreibt. Der strikte Ablauf eines Szenarios ist durch einen Test pro Datei trivial gegeben. Unit Tests dürfen auch innerhalb einer Datei per Definition keine Abhängigkeiten zueinander haben. Die Reihenfolge der Testausführung ist häufig undefiniert und könnte höchstens von einem eigenen `Runner`[16] sichergestellt werden.

4.4.2 Testumgebung

Der Aufbau der Testumgebung beinhaltete die Besorgung verschiedener Betriebsysteme. FreeBSD 6.4 und Ubuntu 9.0.4 wurden als *VM* aus dem Internet und Debian als Netzinstallation von `ftp.fu-berlin.de` geladen.
Die komplette Konfiguration aller Rechner kann im Anhang [B] nachgeschlagen werden.

[16] Runner ist die Oberklasse aller JUnit Test startenden Klassen

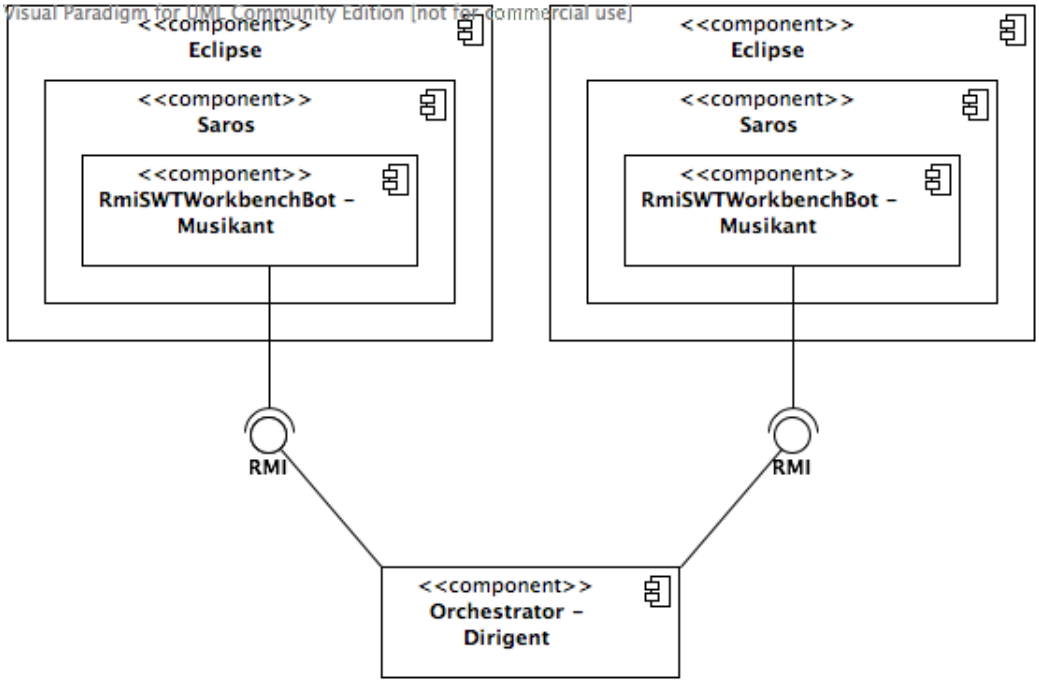

Abbildung 4.2: Orchester bestehend aus zwei Musikanten und einem Dirigent. Der Dirigent verwendet die von Musikanten angebotene Schnittstelle zur Steuerung. Ein Musikant ist ein Teil von Saros, falls Saros im Testmodus gestartet wird.

Alle virtuellen Maschinen kommunizieren nur innerhalb ihres Netzes direkt, mit anderen Netzen wird die Kommunikation über den Emulator erzwungen. Durchgesetzt wird der Datenfluss im Netzwerk über *Gateway* Einträge der jeweiligen Netzwerkkonfiguration.
Die Besonderheiten des Netzwerkes lassen sich an Hand des Schaubild 4.3 einfacher erläutern. Die *Emulator Bypass* betitelte Verbindung umgeht jegliche Einstellungen des Emulators in der Kommunikation von und zum Dirigenten. Dies ist zur Teststeuerung notwendig, weil *RMI* Verbindungen im *delay* Netz nicht verlangsamt oder im *kein P2P zu Musikanten* Netz nicht unterbrochen werden sollen. Dies kann nicht auf Portebene passieren, weil RMI Objekte dynamisch vergebene Ports erhalten. Das *NAT* Netz kann mittels *ipfw* und *natd* gebaut werden. Es ist bisher nicht implementiert, sollte sich jedoch relativ einfach bauen lassen. Es wurde vom Bau abgesehen, weil zum einen Fehler schon bei direkter Verbindungsmöglichkeit auftreten und zum anderen die Zeit für das Framework schon knapp bemessen war. Die Teststeuerung war zu der Zeit noch fertig nicht gebaut.
Latenz erzeugen und der Emulator Bypass verdeutlicht das Firewallskript in Abbildung 4.4. Mit Hilfe des Emulator Bypass wird die reibungslose Verbindung zwischen der Teststeuerung und den zu steuernden Eclipse-Instanzen auf Netzwerkebene gesichert.

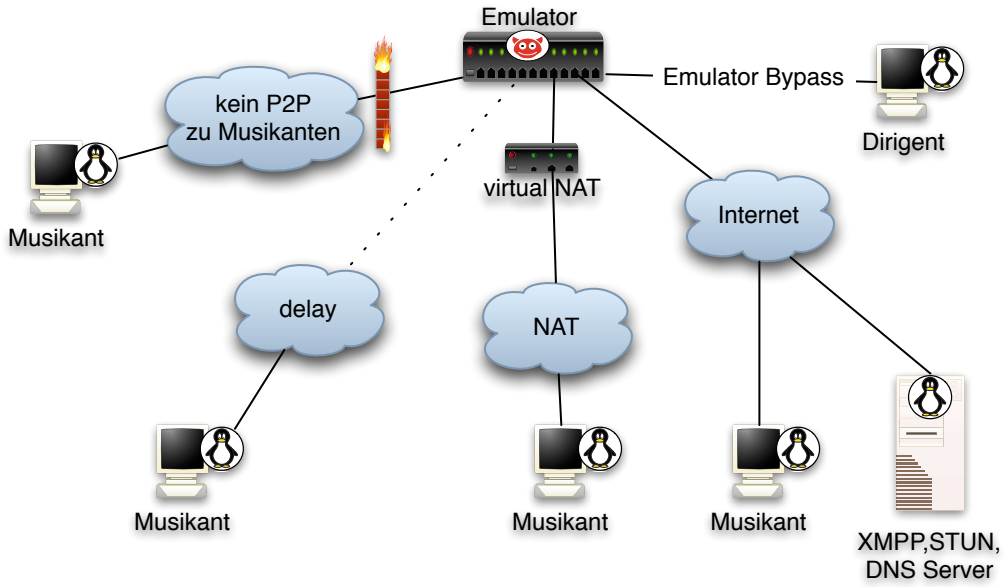

Abbildung 4.3: Das Bild zeigt den logischen Aufbau des virtuellen Testnetzwerks. Netzwerkverbindungen zwischen unterschiedlichen Netzen gehen über den Emulator. Der Emulator ist in der Lage Netzwerkeigenschaften zu bestimmen.

```
## add pipe with delay, set bandwidth and packet loss
## rate (5%) for inbound and outbound network
$PF_CMD pipe 1 config bw 1Mbit/s delay 500ms mask src-ip
    $DELAY_NET dst-ip $DELAY_NET plr 0.05

## conductor dummynet emulator bypass
$PF_CMD add 1000 allow ip from any to $CONDUCTOR
$PF_CMD add 1000 allow ip from $CONDUCTOR to any

## use the pipe 1 as rule nummber 2000 for $DELAY_NET incoming
## and outgoing for all besides ssh,vnc,rmi
$PF_CMD add 2000 pipe 1 in dst-ip $DELAY_NET not dst-port
    22,5901,1099
$PF_CMD add 2000 pipe 1 out src-ip $DELAY_NET not src-port
    22,5901,1099
```

Abbildung 4.4: Emulator erzeugt eine *Pipe* mit den Eigenschaften Verzögerung, Bandbreiteneinschränkung und Paketverlust. Die Regel wird auf Netzwerkpakete in beide Richtungen von *$DELAY_NET* angewendet. Zwischen *Pipe* und Regelanwendung steht der Emulator Bypass, um in der Anwendungskette der Firewall vor der *Pipe* gefunden zu werden.

4.4.3 Bau des Testframeworks

Bau der Musikanten

Der Bau der Musikanten ist zweiteilig. Die Klasse `RmiSWTWorkbenchBot` ist für die direkte Verwendung und Export des *SWTBots* zuständig. Die andere Klasse `Musician` ist für die einfache, objektorientierte und robuste Verwendung in Unit Tests gedacht.

RmiSWTWorkbenchBot: Die zentrale Komponente des *SWTBots* ist die Klasse `SWTWorkbenchBot`. Sie wurde von mir dahingehend erweitert eine *RMI* Schnittstelle anzubieten. `SWTWorkbenchBot` hat ursprünglich ein *Fluent Interface* [Fow05] und liefert deshalb bei der Benutzung meistens SWT-Objekte zurück. *SWT-Objekte* können nicht per *RMI* exportiert werden, weil sie plattformabhängig sind und daher nicht serialisiert werden können. Nur lokal verwendete Attribute wurden mit dem Schlüsselwort `transient` versehen. `Transient` verhindert die *Serialisierung* der angegebenen Eigenschaft.
`RmiSWTWorkbenchBot` hat ein Attribut `SWTWorkbenchBot` und ist per *Delegation* [GHJV95] implementiert, um die Methodensignaturen für den Export zu ändern. Einfache Methoden wurden umgeschrieben, wie die Verwendung in Abbildung 4.5 zeigt. Komplexere Aktionen wurden ähnlich zur Schnittstellenbeschreibung in Listing 4.6 implementiert.
Die Handhabung des *SWTBots* ist nicht immer einfach, weil der `SWTWorkbenchBot`, im Gegensatz zum Menschen, Aktionen ausgeführt und zur nächsten Aktion übergeht ohne auf ein Fenster zu warten. Ein klassisches Beispiel ist ein Fortschrittsbalken, der einem Benutzer anzeigt das es noch eine Weile dauert. Im Fall von Saros ist die Annahme einer Einladung mehrteilig und langlaufend. Nach der Einladung des Einladers erscheint, auf der Seite der Eingeladenen ein Fenster 4.7 zur Sitzungsannahme. Die Eingeladene nimmt die Sitzung an mit der Bestätigung auf den Knopf mit der Bezeichnung *Next* >. Dateilisten werden ausgetauscht und verglichen, bis das nächste Fenster erscheint und der Knopf mit der Aufschrift *Finish* aktiv ist. In diesem Fall wartet der Musikant solange bis im Fenster 4.8 die genannte Schaltfläche aktiviert ist. Der Codeausschnitt 4.9 verdeutlicht die Handhabung aus der Perspektive eines Programierers.
Der zweite Schritt zur Sitzungsannahme ist die Projektsynchronisierung und dauert häufig sehr viel länger als der Erste. Die Eingeladene tritt der Sitzung bei und möchte ein neues Projekt anfangen. Das Fenster mit dem Titel *Session Invitation* bleibt auf der Seite der Eingeladenen solange bestehen bis der Beitritt komplett abgeschlossen ist. Abbildung 4.10 zeigt die entsprechende Realisierung aus Sicht des Programmierers.

Musician: Ein `Musician` ist für die einfache Verwendung und Konfiguration der Klasse `RmiSWTWorkbenchBot` da. Es kapselt die notwendigen *RMI* und *XMPP* Daten mit der Ausführung von Methoden auf dem Bot. Die Konfiguration wird über den Konstruktor vorgenommen. Komplexe Aktionen werden wie in Abbildung 4.11 verwendet. Ich denke der Testcode ist dadurch eingehend verständlich und unkompliziert verwendbar.

Die Klassen der entfernten Workbench sind in verschiedene Teile aufgespalten, um den sarosspezifischen, aus dem allgemeinen Teil zu trennen. Ziel ist es den allgemeinen Teil in das *SWTBot* Projekt zu integrieren. Dies kann in Zukunft die Pflege der Komponente erleichtern. Saros könnte auch von einer breiteren Benutzermasse in Form von Stabilität und Vollständigkeit dieses Bausteins profitieren. Das Klassendiagramm in Abbil-

```
// Beispiel
// Verwendung von SWTWorkbenchBot
bot.button("OK").click();
// Verwendung von RMISWTWorkbenchBot
rmibot.clickButton("OK");
```

Abbildung 4.5: Die Änderung von einfachen Methoden am Beispiel *OK Button* drücken

```
public void addViewByName(String category, String nodeName)
    throws RemoteException;
```

Abbildung 4.6: Beispiel Signatur einer komplexen Methode

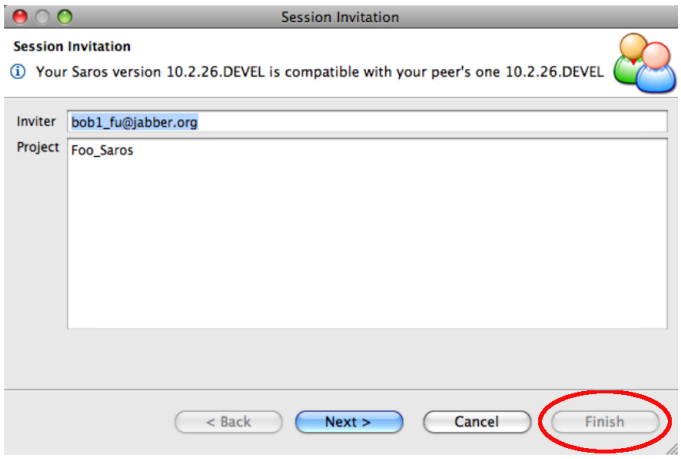

Abbildung 4.7: Erster Schritt der Annahme einer Einladung. Der *Button* mit der Aufschrift *Finish* ist nicht aktiviert.

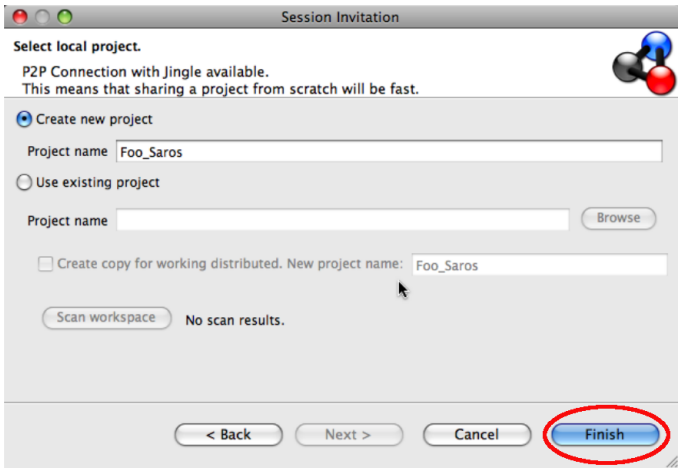

Abbildung 4.8: Zweiter Schritt der Annahme einer Einladung. Der Button mit der Aufschrift *Finish* ist aktiviert.

```
/**
 * First step: invitee acknowledge session to given inviter
 */
public void ackProject1(String inviter) throws RemoteException
    {
     activateShellByName("Session␣Invitation");

     if (!delegate.textWithLabel("Inviter").getText().equals(
        inviter))
         log.warn("inviter␣does␣not␣match:␣" + inviter);

     captureScreenshot(TEMPDIR + "/acknowledge_project1.png");
     delegate.button("Next␣>").click();
     captureScreenshot(TEMPDIR + "/acknowledge_project2.png");
     try {
         while (!delegate.button("Finish").isEnabled()) {
             delegate.sleep(100);
         }
     } catch (Exception e) {
         // next window opened
     }
}
```

Abbildung 4.9: Erster Schritt der Annahme einer Einladung. Der Bot wartet zur Synchronisierung auf einen, erst im nächsten Fenster aktivierten, Button.

```
/**
  * Second step: invitee acknowledge a new project
  */
public void ackNewProject2(String projectName) throws
    RemoteException {
     delegate.radio("Create␣new␣project").click();
     captureScreenshot(TEMPDIR + "/acknowledge_project3.png");
     delegate.button("Finish").click();
     captureScreenshot(TEMPDIR + "/acknowledge_project4.png");
     try {
         while (true) {
             delegate.shell("Session␣Invitation");
             delegate.sleep(100);
         }
     } catch (Exception e) {
         // window closed
     }
}
```

Abbildung 4.10: Zweiter Schritt der Annahme einer Einladung. Der Bot wartet auf die Schließung des Fensters, was zu einer WidgetException beim Aufruf von `delegate.shell()` führen wird.

```
/*Auszug aus einer Testmethode
  inviter und invitee sind jeweils vom Typ Musician*/
@Test
public void testShareProject() throws RemoteException {
    inviter.shareProject(invitee, PROJECTNAME);
    invitee.ackProject(inviter, PROJECTNAME);
    try {
        assertTrue(invitee.isParticipant());
        assertTrue(invitee.isObserver());
        assertTrue(invitee.isParticipant(inviter));
        assertTrue(invitee.isDriver(inviter));
...
}
```

Abbildung 4.11: Auszug aus einer Testmethode. Die letzten Zeilen verifizieren aus `invitee` Perspektive, ob der `inviter` am Projekt teilnimmt und die Driverrolle hat.

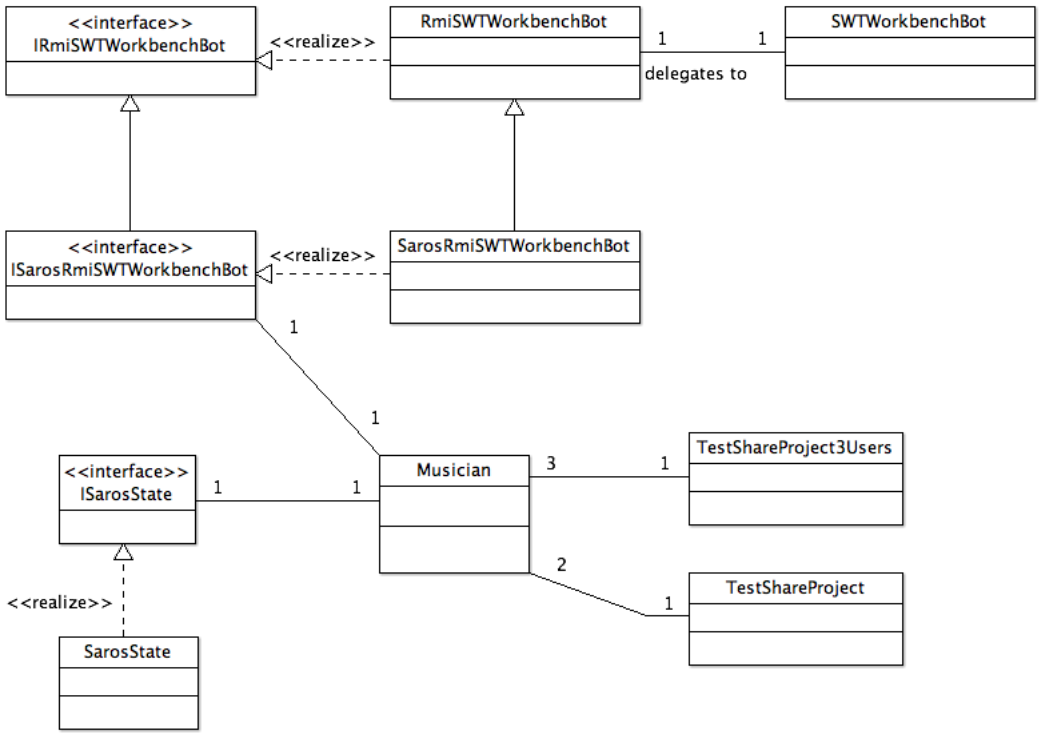

Abbildung 4.12: Das Klassendiagramm zeigt die Klassen des Testframeworks und ihren Zusammenhang mit zwei verschiedenen Tests.

dung 4.12 verdeutlicht das Verhältnis der Klassen innerhalb des Testframeworks. Der sarosspezifische Teil trägt im Namen das Präfix *Saros* und behandelt alle Aktionen, die für das spezielle Testen von Saros notwendig sind. Ein Beispiel dafür ist die Steuerung des Einladungsprozesses. Die zentrale Klasse für Testschreiber heißt `Musician`, während die, für das Framework zentrale, Bot Implementierung `SarosRmiSWTWorkbenchBot` heißt. `Musician` ermöglicht den Benutzern des Frameworks Aktionen des Bots auszuführen und den Zustand von Saros zu erfassen. Dies geschieht über das zweite *RMI Objekt*, `SarosState`. Es eröffnet dem Tester die Möglichkeit den aktuellen Zustand eines Musikanten zu verifizieren wie in Abbildung 4.11 gezeigt ist. `SarosState` selbst braucht verschiedene Objekte, um den internen Zustand zu berichten, und ist deshalb dem `PicoContainer` von Saros bekannt. `PicoContainer` [PIC06] löst mittels *Dependency Injection* [Fow04] Abhängigkeiten auf und wird für die Verwaltung von *Singletons* [GHJV95] eingesetzt. Die *Java Interfaces* sind im übrigen für die *RMI Stummel* notwendig.

Der Start eines Musikanten wird über die Klasse `StartupSaros` ermöglicht. Sie ist eine Implementierung von Eclipse `IStartup`, die nach Initialisierung der Workbench nach dem Laden eines Plugins automatisch die Methode `earlyStartup()` aufruft. In `StartupSaros#earlyStartup()` wird auf den entscheidenen Startparameter[17] geprüft, um Saros normal oder als Musikant zu starten. Der Start als Musikant umfasst den Start einer *RMI-Registry* und den Export von zwei verschiedenen *RMI-Objekten*[18]. Zum einen wird das Steuerobjekt `RmiSWTWorkbenchBot` exportiert und zum anderen das Zustandsobjekt `SarosState` angeboten.

Bau des Dirigenten

Der Dirigent ist puristisch als *JUnit4* Test implementiert. Er erzeugt sich als erstes, die für das implementierte Testszenario benötigte Anzahl an `Musician`. Die Steuerung geschieht im Folgenden über die in `Musician` gekapselten *RMI* Aufrufe. Implementierte Testszenarien sind derzeit:

- Kontakt hinzufügen mit Bestätigung,
- Verbindungsaufbau mit 2 Teilnehmern und
- Verbindungsaufbau mit 3 Teilnehmern.

JUnit4 ermöglicht die Trennung von Initialisierung, Test und Finalisierung mit Hilfe von Annotationen. Die Ausführungszeiten von annotierten Methoden kann in Tabelle 4.3 nachvollzogen werden. Weil Testsznarien normalerweise einen geradlinigen Anlauf haben und daher nur einen Test enthalten habe ich nur `@Before` und `@After` Annotationen verwendet. Sie setzen Vor- und Nachbedingungen für einen Testlauf durch. Für den Lauf einer gesamten `TestSuite`, die alle aufgeführten JUnit Tests ausführt müssen Nach- und Vorbedingungen konsistent gehalten werden. Ansonsten verfehlt ein Test womöglich sein Ziel, ohne einen Fehler in Saros aufzudecken.
Am Anfang eines Tests müssen die Musikanten konfiguriert werden, wie der Codeauszug 4.13 zeigt. Andere Ausführungen sind abhängig vom Testszenario. Es kann vor Testausführung sinnvoll sein sich per *XMPP* zu verbinden oder eine Saros-Sitzung

[17] Startparameter mit optionalem Argument `de.fu_berlin.inf.dpp.testmode=<port>`

[18] Für den Export von RMI-Objkten ist eine Policy Datei im `bin` Verzeichnis des Saros-Projekts notwendig

Tabelle 4.3
Annotationen und Ausführungszeiten in JUnit4

Annotation	Ausführungszeit
@BeforeClass	einmalig nach dem Start
@Before	vor jedem Test
@Test	Test
@After	nach jedem Test
@AfterClass	einmalig vor dem Ende

```
@Before
public void configureInvitee() throws RemoteException,
    NotBoundException {
  invitee = new Musician(
    new JID("alice@xmpp1.internet.test" + Saros.RESOURCE),
    "password", "rmiswtbot1.internet.test",
    1099
  );
  invitee.initRmi();
  invitee.openSarosViews();
  invitee.xmppConnect();
}
```

Abbildung 4.13: Konfiguration eines Musikanten für den Lauf im Testemulator innerhalb eines JUnit Tests.

aufzubauen. Tests, die den Verbindungsaufbau prüfen, verbinden sich beispielsweise vor dem Test per *XMPP*. Falls verschiedene Arten der Einladung mit gleichbleibender Anzahl an Teilnehmern getestet werden sollen macht es Sinn die `@BeforeClass` Annotation für die Einwahl und für die Auswahl `@AfterClass` zu verwenden.

4.4.4 Deployment

Nachdem der Emulator, die Fern- und die Teststeuerung gebaut sind ist der größte Teil des Frameworks fertig. Das Ausrollen der zu testenden Objekte soll so automatisch wie möglich geschehen, damit die Hemmschwelle für Sarosentwickler, das Framework zu benutzen, möglichst gering ist.
Ein Problem ist die aktuelle Version zu verteilen. Eclipse bietet von hause aus verschiedene Varianten zur Installation von Plugins. Die bekanntesten sind die Installation über eine *Updatesite* oder als *Dropin*. Eine *Updatesite* stellt Versionsinformationen bereit und eine URL[19], unter welcher die Software in der spezifizierten Version zu finden ist. *Dropins* werden zur Installation einfach in das Verzeichnis *dropins/* der Eclipseinstallation kopiert. Beim Start von Eclipse wird das neue Plugin automatisch mitgestartet.
Andere Ansätze sind die Verteilung von Quell- oder Objectcode. Quellcode hat den Vorteil das man als Tester bestimmte Patchfiles testen kann und nicht darauf angewiesen ist den gesamten Quellcodebaum mit Bibliotheken auf die Testsysteme zu verteilen. Der Nachteil ist das Quellcode vor dem Start noch übersetzt werden muss. Nach dem

[19] URL nach RFC 1738 `http://www.ietf.org/rfc/rfc1738.txt` (Online abgerufen am 23. Februar 2010)

Bauvorgang ist der Start dieser Arten gleich.
Von der Installation über eine *Updatesite* wurde von mir Abstand genommen, weil es für den Tester aufwendiger ist, als die Erstellung eines *Dropins*. Zusätzlich müßte eine lokale Webseite eingerichtet und zur Verteilung die *site.xml* geändert werden, um dem *Updatemanager* von Eclipse zur Installation einer neuen Version zu überreden.
Es wurden drei Varianten getestet, wobei die ersten zwei verlockend waren, aber nicht den gewünschten Erfolg brachten.

Ansatz Quellcode verteilen: Über das zentrale *Subversion Repository* von Saros ist es möglich Quellcode auf allen Testrechnern einfach konsistent zu halten. Falls der Code auf einem System noch nicht vorhanden ist, wird er aus dem *Repository* geladen. Einmal vorhanden ist es nur noch notwendig den Quellbaum in einen konsistenten Zustand zu bringen. Dies bewerkstelligen der Subversion-Befehl *revert* und ein nachfolgendes Löschen aller Dateien, die sich nicht im *Repository* befinden.
Nachfolgend besteht die Möglichkeit einen bestimmten *Branch* oder den aktuellen *Trunk* zu wählen, und auf die konsistente Basis einen *Patch* anzuwenden. Danach muss das Projekt übersetzt werden, um von der *JVM* lesbar zu sein.
Das Problem bei diesem Ansatz ist der letzte Schritt. Der Bau von Saros benötigt Teile von Eclipse, deren Abhängigkeitsauflösung nicht trivial ist. Es gibt leider kein *Maven*[20] *Repository*, das zu einer automatischen Abhängigkeitsauflösung für den Bau verwendet werden könnte. Ein naiver Bauvorgang mittels Java Compiler war nicht möglich. Eine Nichtautomatische Auflösung von Abhängigkeiten hätte große Nachteile in der Wartbarkeit und wurde nicht von mir versucht.

Ansatz Dropin verteilen: Naheliegend ist der lokale Bau eines *Dropins*, dass per Automatismus auf die Testrechner verteilt wird. Unschön ist der vorhandene Overhead für den Anwender, der zum Zweck eines Tests ein *Dropin* bauen muss. Auf der anderen Seite sollte man meinen, dass dieser Ansatz trivial zum Erfolg führt: *Dropin* verteilen, in den Ordner *dropins/* der Eclipseinstallation kopieren und Eclipse starten. Wie sich nach aufwendigen Tests herausstellte, führt der Start leider zu unvorhergesehenen Schwierigkeiten.

1. Das erste Problem war die Parameterübergabe. Sie ist unterschiedlich zum Start einer Software aus Eclipse heraus. Eine Internetrecherche brachte keinen Erfolg, weil zu viele Menschen das Problem haben, Parameter einer Applikation zu übergeben, die innerhalb Eclipse gestartet wird. Nach einigen Versuchen und zusätzlichem Raten gelang es mir den Start über die Konfigurationsdatei `eclipse.ini` zu beeinflußen und das Problem der Parameterübergabe zu lösen.
2. Das Zweite war die notwendige *Policydatei* für *RMI*, die nicht automatisch in die *Jar* verpackt wurde. Das ist leicht mit Hilfe von *zip* lösbar und kann daher leicht automatisiert werden.
3. Zu guter letzt kam beim Start von Saros im Testmodus eine `ClassNotFoundException`, die auf nicht auffindbare Dateien des *SWTBots* hinwies. Ein Export der notwendigen *SWTBot* Abhängigkeiten beim Bau des *Dropins* mit nachvollziehbaren Pfaden in der *Jar* brachte auch keinen Erfolg. Der Fehler verwies auf eine Klasse, die sich in dem Pfad der *Plugin-Jar* befand. Ich

[20] Maven ist ein häufig verwendetes Buildwerkzeug für Java Software.

```
14:42 <spawnyd> hi! Any idea to generate/export/.. an eclipse
   run configuration to an Ant/shell/.. script?

14:46 <paulweb515> spawnyd: there isn't, AFAIK. You can go to
   the debug view and pick your process properties. That will
   give you the full command line of the last launch

14:47 <spawnyd> paulweb515: ty that's enough for my use case.
```

Abbildung 4.14: Auszug einer Sitzung in `irc://freenode.net/#eclipse`

konnte auch keinen Fehler im *Classpath* feststellen, so dass ich diesen Versuch ebenfalls abbrechen mußte.

Ansatz Objectcode verteilen: Beim dritten Versuch wird ein gesamtes Projekt per *rsync*[21] auf die Testrechner verteilt. Der Entwickler muss dabei nur ein Skript[22] starten mit Pfadangabe zum lokal verwendeten Projekt. Es ist daher wenig Aufwand für den Benutzer sein aktuelles Projekt auf alle Rechner zu verteilen.
Ein Problem tritt erst auf, wenn plattformabhängige Komponenten getestet werden müssen. Der Start des verteilten Objektcodes ist schnell anpassbar, falls man die Möglichkeit des Eclipsedebuggers kennt, Prozesseigenschaften anzuzeigen. Der Eclipsedebugger hält unter den Eigenschaften eines Prozesses auch den kompletten Startbefehl [B.5], wie ich schließlich in einer *IRC*[23] Sitzung erfuhr.

4.5 Anwendung

Die Anwendung teilt sich auf in die Vorstellung des Testablaufs, und das Aufzeigen weiterer Ideen zum Einsatz des Frameworks. Eine detailierte Aufzeichnung ist in Anhang [C.1.1] zu finden.

4.5.1 Testablauf

Ein Test besteht aus dem Ablauf, der Konfiguration von Testrechnern und Emulator, sowie zu guter Letzt dem geschriebenen Unittest.

Ablauf: Ein Ablauf besteht der Reihenfolge nach aus der Verteilung von Dateien, dem Start der *Musikanten*, optional dem Start von Analysewerkzeugen, dem Start des auszuführenden Tests und deren Verifikation.
Die Verteilung der Dateien ist automatisiert über eine Skriptsammlung möglich. Das Frontend dafür ist in Anhang [C.1.2] näher beschrieben. Der *Musikantenstart* ist über einen Automatismus des *Xservers*[24] gelöst, der das vorher verteilte Startskript für Eclipse bei Initialisierung der *X* Sitzung startet. Die *X* Sitzung wird über den *VNC*[25] Dienst aus der Skriptsammlung indirekt gestartet. *VNC* ermöglicht

[21] rsync: `http://samba.anu.edu.au/rsync/` (Online abgerufen am 23. Februar 2010)
[22] Skripte zum verteilen sind in Abschnitt 4.5 erläutert.
[23] Internet Relay Chat ist ein Protokoll nach RFC 1459: `http://tools.ietf.org/html/rfc1459` (Online abgerufen am 23. Februar 2010)
[24] Xserver ist ein UNIX Dienst zur Anzeige einer grafischen Oberfläche.
[25] VNC ermöglicht teilen des Bildschirms auf Basis des Remote Framebuffer Protokolls.

dem Betrachter den verteilten Test von seinem Rechner aus, auf den entfernten Testsystemen, zu verfolgen.

Testkonfiguration: Die Testkonfiguration, wie die `JID` des Einladers, muss auf allen *Musikanten* und dem *Dirigenten* gleich sein, sonst ist die Ausführung fehlerhaft. Diese Konfiguration ist Teil eines Unittests und wurde in Abschnitt [4.4.3] ausführlich erklärt und an Hand des Beispiels 4.13 verdeutlicht.

Emulator: Das Emulationsnetz muss dem Test entsprechend eingerichtet werden. Das Verhalten von Saros bei größeren Verzögerungen zu messen bedarf einer eigenen Vorbereitung. Die Konfiguration ist optional über die zentrale Skriptsteuerung möglich. Ein vorbereitetes Firewallskript ermöglicht den Test unterschiedlicher Netzeigenschaften im Emulator. Das Emulatorskript ist in Anhang [B.1] abgedruckt. Die zentrale Skriptsteuerung verteilt im ersten Schritt alle Skripte auf Testrechner, Teststeuerung und den Router. Der zweite Schritt richtet die Testrechner ein und im dritten Schritt wird die Emulation auf dem Router konfiguriert. Es können vorgefertigte Emulationen verwendet werden, um Saros in verschiedenen Netzbedingungen oder Extremsituationen zu testen. Extreme Situationen sind bezogen auf Bandbreite, Durchsatz und Paketverlust. Netze sind von mir in verschiedene *WLAN* und *DSL* Standards unterteilt. Nach kleineren Messungen mittels *ping* wurden die Netzwerkeigenschaften im Emulatorskript vorbereitet.

Unittest: Auszuführende Unittests müssen auf dem *Dirigenten* hinterlegt sein. Nachdem alle Vorbereitungen getroffen worden, kann ein Testlauf gestartet werden.

Die Ablaufautomatisierung garantiert die Konsistenz aller Sarosinstanzen auf den Rechnern und die strikte Einhaltung der Schrittfolgen. Für die einfache Verwendbarkeit wurde ein einzelnes Frontend zur Verteilung und Konfiguration von Emulation und Rechnern implementiert.

4.5.2 Tests schreiben

Tests werden in Form von *JUnit4* Tests geschrieben. Empfehlen kann ich die Verwendung einer Datei pro Testkontext. Ein Testkontext besteht hier aus der Konfiguration der Teilnehmerzahl und dem Zustand der Sarosinstanzen. Vor Ausführung der Testmethode muss Saros in den entsprechenden Zustand gebracht und nachher wieder in den Testkontext zurückgestellt werden. Eine Vorbedingung kann eine aufgebaute *XMPP*-Verbindung oder auch eine vorhandene Saros-Sitzung sein. Nach dem Gesamtdurchlauf einer Testdatei muss Saros für die nächste in einen konsistenten Zustand gebracht werden. Nach dem Lauf einer Testszene sollten daher alle Sarosinstanzen wieder in den Grundzustand gebracht werden, d.h. nicht mehr mit *XMPP* verbunden sein.
Die Klasse `Musician` sollte als Frontend für den Bot benutzt und gegebenenfalls erweitert werden. Es ermöglicht dem Entwickler leicht verständliche Tests zu schreiben, wie in Abbildung 4.11 gezeigt. Ergebnisse werden, wie in JUnit Tests üblich, mit Methoden wie `assertEquals(expected,actual)` geprüft. Der interne Zustand von Saros kann auch über das `Musician` Objekt abgefragt werden. Falls die angebotene Schnittstelle nicht ausreicht kann der *RMI Export* `ISarosState` erweitert und eine passende *Delegation* [GHJV95] für die Erweiterung in `Musician` verwendet werden. Der Vorteil dieser Abstraktion ist eine einheitliche *Fassade* [GHJV95] für alle Testschreiber.

4.5.3 Tests ausführen

Tests können unter anderem lokal auf einem Rechner ausgeführt werden. Dazu benötigt man die Verbindung zum Internet und verschieden konfigurierte *Runconfigurations* zum Starten der zu steuernden Eclipseinstanzen. Beim Start muss Saros das vereinbarte Signal für den Testmodus bekommen. Die Vereinbarung ist der Parameter `de.fu_berlin.inf.dpp.testmode`. Es veranlasst Saros die *RMI Objekte* zu exportieren und eine *RMI Registry* zu starten. Die Verwendung von *RMI* benötigt Parameter, um die Codebasis festzulegen, den `SecurityManager` zu starten und die Verwendung bestimmter Richtlinien aus der übergebenen *Policy*-Datei einzustellen. Im Einzelnen lauten die Parameter:

Registry Port angeben: `-Dde.fu_berlin.inf.dpp.testmode=1099`

Codebasis festlegen: `-Djava.rmi.server.codebase=file:$PROJECT/bin`

SecurityManager starten: `-Djava.security.manager`

Policy Datei: `-Djava.security.policy=file:$PROJECT/bin/all.policy`

Objekt an Host binden `-Djava.rmi.server.hostname=$HOSTNAME`

Die Angabe eines *RMI Registry Ports* ermöglicht die lokale Ausführung mehrerer Instanzen, da jede Sarosinstanz seine eigene *Registry* hat. Sinnvoll ist das zur Entwicklung von Tests mit dem Framework oder für lokal automatisiert ausgeführte Tests. Im Emulator muss man zusätzlich das exportierte Objekt an die extern verfügbare Netzwerkschnittstelle binden, andernfalls ist das Objekt nur auf der lokalen Schnittstelle verfügbar.

Ausführen von Tests in der Emulation bedarf einer separaten Vorbereitung. Zur Automatisierung aller notwendigen Schritte gibt es eine Skriptsammlung[26], dessen Frontend *fap_deploy* heißt. Es ermöglicht die Wahl vorgefertigter Netzwerkemulationen, den Start eines Sniffers zur späteren Netzwerkanalyse auf dem Emulator, die Angabe am Test teilnehmender Rechner und die Synchronisierung eines lokalen Projekts auf die Rechnerauswahl. Die Sammlung beinhaltet eine Vielzahl kleiner Shellskripte, die entfernte Rechner vorbereiten. Zum Schluss starten sie auf den Testrechnern einen *VNC Server*. Es bewirkt den Start der grafische Oberfläche samt Eclipse im Testmodus. Der Tester muss sich per *VNC* mit dem *Dirigentenrechner* verbinden und die Tests aus dem bereitgestellten Eclipse starten.

4.6 Fazit

Ich denke das Testframework bietet eine gute Möglichkeit alle bekannten und zukünftigen Defekte in der Netzwerkkomponente zu analysieren. Ein Einzelner ist jetzt in der Lage ohne großen Aufwand verschiedene Netzwerkszenarien zu testen. Optimieren auf bestimmte Netzwerkeigenschaften ist durch die Testsuite unmsetzbar. Schwer lokalisierbare Schwächen auf Grund von Netzwerkeigenschaften sind jetzt wiederholbar.

Die fernsteuerbare Komponente des Testframeworks ist auch für andere Projekte interessant. Das zeigt die E-Mail Korrespondenz mit dem Entwickler des verwendeten Frameworks, *SWTBot*, im Anhang [C.2].

[26] Die Skriptsammlung ist unter `https://svn.mi.fu-berlin.de/agse/students/szuecs/src/vm_simulator` erhältlich.

Die aufwendige Arbeit war daher eine gute Investition.

4.6.1 Validierung

Die funktionale Anforderungen aus Abschnitt [4.2.1] sind erfüllt bis auf *NAT*. Ein *NAT* kann in der gebauten Lösung mit dem Softwarepaket *natd* aufgebaut werden und ist daher mit wenig Aufwand möglich. Nichtfunktionale Anforderungen aus Sektion [4.2.2] sind erfüllt. Die Benutzbarkeit ist durch eine parametrisierbare Automatisierung gegeben, die alle notwendigen Einstellungen im Emulator vornimmt und die Musikanten startet. Durch die Dokumentation innerhalb der Diplomarbeit und einem Vortrag im Forschungsmeeting der Arbeitsgruppe Software Engineering konnte

4.6.2 Verwendungsmöglichkeiten

Eingehends wurde das Ziel des Frameworks erklärt. Es wurde entwickelt, um Defekte in der Netzwerkkomponente messbar, testbar und wiederholbar zu machen. Durch die Flexibilität des Frameworks sind auch Anwendungsszenarien möglich, an die vorher nicht gedacht wurde.

SWTBot ermöglicht die Erstellung von Bildschirmfotos, d.h. das Framework ist in der Lage Bilder für Anleitungen automatisch zu erstellen.

Das Framework kann ohne Probleme lokal ausgeführt werden. Entwickler haben die Möglichkeit von Hand durchgeführte Tests zu automatisieren. Ein Vorteil dadurch ist die Wiederholbarkeit des Tests. Ein weiterer Aspekt ist das weniger Konzentration auf die Durchführung verloren geht. Daraus folgt das Auswirkungen implementierter Änderungen mehr Aufmerksamkeit bekommen können.

Das Framework könnte bei guter Testabdeckung den aktuell durchgeführten Testtag verändern. Manuelle Tests werden z.Z. in einer speziell dafür vorgesehenen Webapplikation namens *Testlink* gespeichert. Durch die Abdeckung dort aufgeführter Tests wäre es möglich *Testlink* abzuschaffen. Bisher ist das gesamte Saros-Team involviert, stattdessen könnten *TM*[27] und *ATM* einen automatisierten Testtag durchführen. Es wäre eine Prozessverbesserung, wenn nur noch maximal zwei Personen notwendig sind, um den Testtag durchzuführen. Wenn ausreichend Vertrauen in Tests und Testsuite vorhanden wäre, ließe sich der Testtag abschaffen.

Falls der Bau des Quellcodes auf den Zielrechnern möglich wird, könnte man innerhalb eines *Continious Integration* Prozesses die Testdurchführung automatisieren. Man hätte dadurch einen regelmäßigen Integrationstest geschaffen.

Die Klasse `SarosState` kann zur Implementierung des in Abschnitt [3.3.2] erwähnten *NetworkView* beitragen. Im Testlauf werden die Daten auswertbar. Schlüsselparameter wie zeitgesteuertes lesen aus Warteschlangen werden messbar und daher optimierbar.

[27] TM: Testmanager, ATM: Assistent TM

Kapitel 5

Ausblick

In diesem Abschnitt möchte ich auf zukünftige Enwicklungsschritte des Saros-Projektes eingehen, die aus meiner Sicht notwendig und sinnvoll für den Erfolg von Saros sind. Das Saros-Projekt sollte weiter an der Stabilität der Software arbeiten, um bald die Betaphase verlassen zu können. Dies würde aus meiner Sicht zu einer breiteren Benutzerbasis führen und daher dem Gesamtziel der AG Software Engineering dienen, mehr Einsicht in die Verwendung von *PP*, *DPP*, *SbS* und *DSbS* zu erhalten. Eine Unterstützung auf diesem Weg wäre der Einsatz von *continous integration*, an dessen Realisierung Florian Thiel[1] bereits arbeitet. Das präsentierte Testframework aus Kapitel [4] sollte in den *continous integration* Prozess aufgenommen werden, um regelmäßig ausgeführte Integrationstests zu erhalten. Insbesondere die Automatisierung der bisher in Dokumenten gesammelten Testfälle mit Hilfe des Frameworks sollte zur Stabilität von Saros beitragen.

Im Folgenden nenne ich einige Erweiterungswünsche.

- Das gleichzeitige Bearbeiten mehrerer Projekte wäre eine Fähigkeit, die den Einsatz von Saros in vielen Projekten erst ermöglichen würde. Viele Projekte sind in viele kleine Unterprojekte aufgeteilt, wobei z.Z. während einer Sitzung nur an einem gleichzeitig gearbeitet werden kann. Entwickler aus Projekten dieser Art sind bisher leider kaum in der Lage Saros effektiv einsetzen zu können.

- Eine Verbesserung des Replikationsansatzes, wie bereits in Abschnitt [3.4.2] beschrieben, könnte den Einstieg in Saros erleichtern. Eine *Lazy Replikation* während der Projektsynchronisierung, würde es ermöglichen, schneller mit der Arbeit am geteilten Projekt beginnen zu können. Daraus folgt, dass kurze Sitzungen besser unterstützt. Außerdem könnten Entwickler großer Projekte sofort mit einer Sitzung starten.

- In näherer Zukunft erhoffe ich mir, dass hohen Latenzen mit Hilfe von *Peer-to-Peer* Verbindungen gemindert werden. In Abschnitt [3.4.1] wurden von mir hierfür relevante Defekte erläutert. Bisher sind *Peer-to-Peer* Verbindungen nicht persistent und daher werden Benutzeraktivitäten über einen Umweg über XMPP-Server verschickt. Bei vorhandener Persistenz einer *Peer-to-Peer* Verbindung könnten leicht alle Aktivitäten über die bereits vorhandene *Peer-to-Peer* Verbindung geleitet werden. Daraus würde eine Latenzminderung folgen.

[1] Florian Thiel ist z.Z. des Schreibens Mitarbeiter der AG Software Engineering

Kapitel 6

Zusammenfassung

Diese Diplomarbeit baut auf die vorherigen Arbeiten [Dje06], [Gus07], [Rie08], [Jac09] und [Só09] auf und beschreibt Netzwerkarchitektur, Erweiterungen und Verbesserungen in Saros, sowie ein Testframework.

Es beginnt mit einer Beschreibung des wissenschaftlichen Kontexts, dem eingesetzten Softwareprozess und der Software Saros. Kapitel [3] erläutert anfangs eingesetzte Protokolle und die Netzwerkarchitektur in Saros. In den weiteren Abschnitten des Kapitels erkläre ich von mir getroffene Entscheidungen und entwickelte Lösungen. Es schließt mit einer Beschreibung ausstehender Arbeiten am Netzwerkmodul von Saros.

In Kapitel [4] ist das Unterprojekt Testframework beschrieben. Es zeigt von mir bestimmte Anforderungen an eine Testsuite für Saros. Die Grundlage für die bestimmten Anforderungen bildeten von mir gesammelte Erkenntnisse während der in Kapitel [3] beschriebener Arbeiten an Saros. Weiterhin schildert es getroffene Entscheidungen zur Wahl von unterstützten Frameworks und Softwarepaketen zum Bau des Testnetzwerks und der Teststeuerung. Ein anderer Abschnitt beschreibt die grundlegende Idee und den Bau. Es werden verschiedene Ansätze zur Verteilung und die Problematik des Pluginstarts beleuchtet. Die letzten Abschnitte behandeln die Anwendung des gesamten Frameworks und zeigt weitere Verwendungsmöglichkeiten auf.

Literaturverzeichnis

[AGDIS07] Erik Arisholm, Hans Gallis, Tore Dyba, and Dag I.K. Sjoberg. Evaluating pair programming with respect to system complexity and programmer expertise. *IEEE Trans. Softw. Eng.*, 33(2):65–86, 2007.

[BABM05] John Bicket, Daniel Aguayo, Sanjit Biswas, and Robert Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In *MobiCom '05: Proceedings of the 11th annual international conference on Mobile computing and networking*, pages 31–42, New York, NY, USA, 2005. ACM.

[Bec99] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, sep 1999.

[BRdB08] Sallyann Bryant, Pablo Romero, and Benedict du Boulay. Pair programming and the mysterious role of the navigator. *Int. J. Hum.-Comput. Stud.*, 66(7):519–529, 2008.

[CCG+07] Gerardo Canfora, Aniello Cimitile, Felix Garcia, Mario Piattini, and Corrado Aaron Visaggio. Evaluating performances of pair designing in industry. *J. Syst. Softw.*, 80(8):1317–1327, 2007.

[CH07] Jan Chong and Tom Hurlbutt. The social dynamics of pair programming. In *ICSE '07: Proceedings of the 29th international conference on Software Engineering*, pages 354–363, Washington, DC, USA, 2007. IEEE Computer Society.

[Coc04] Alistair Cockburn. *Crystal clear a human-powered methodology for small teams*. Addison-Wesley Professional, 2004.

[Cro08] Douglas Crockford. *JavaScript: The Good Parts*. O'Reilly Media, Inc., 2008.

[CW01] Alistair Cockburn and Laurie Williams. The costs and benefits of pair programming. pages 223–243, 2001.

[DASH09] Prasun Dewan, Puneet Agarwal, Gautam Shroff, and Rajesh Hegde. Distributed side-by-side programming. In *CHASE '09: Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, pages 48–55, Washington, DC, USA, 2009. IEEE Computer Society.

[Dje06] Riad Djemili. Entwicklung einer eclipse-erweiterung zur realisierung und protokollierung verteilter paarprogrammierung. Dilpomarbeit, Freie Universität Berlin, Dep. of CS., `https://www.inf.fu-berlin.de/w/SE/ThesisDPPI`, 2006.

[Fow99] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[Fow04] Martin Fowler. Inversion of control containers and the dependency injection pattern. `http://martinfowler.com/articles/injection.html`, Jan 2004. (Online abgerufen am 10. Januar 2010).

[Fow05] Martin Fowler. Fluent interface. `http://martinfowler.com/bliki/FluentInterface.html`, Dec 2005. (Online abgerufen am 20. Januar 2010).

[FRdB07] S. (nee Bryant) Freudenberg, P. Romero, and B. du Boulay. "talking the talk": Is intermediate-level conversation the key to the pair programming success story? In *AGILE '07: Proceedings of the AGILE 2007*, pages 84–91, Washington, DC, USA, 2007. IEEE Computer Society.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[Gus07] Björn Gustavs. Weiterentwicklung des eclipse-plug-ins saros zur verteilten paarprogrammierung. Studienarbeit, Freie Universität Berlin, Dep. of CS., `https://www.inf.fu-berlin.de/w/SE/ThesisDPPII`, 2007.

[HDAS09] Jo E. Hannay, Tore Dybå, Erik Arisholm, and Dag I. K. Sjøberg. The effectiveness of pair programming: A meta-analysis. *Inf. Softw. Technol.*, 51(7):1110–1122, 2009.

[HMDK04] Brian Hanks, Charlie McDowell, David Draper, and Milovan Krnjajic. Program quality with pair programming in cs1. In *ITiCSE '04: Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, pages 176–180, New York, NY, USA, 2004. ACM.

[HMESA08] Joe Hildebrand, Peter Millard, Ryan Eatmon, and Peter Saint-Andre. Service discovery xep-0030. `http://xmpp.org/extensions/xep-0030.html`, Juni 2008. (Online abgerufen am 10. Januar 2010).

[Hum97] Watts Humphrey. *Introduction to the Personal Software Process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[IET] Internet engineering task force. `http://www.ietf.org/`.

[Jac09] Christoph Jacob. Weiterentwicklung eines werkzeuges zur verteilten, kollaborativen softwareentwicklung. Dilpomarbeit, Freie Universität Berlin, Dep. of CS., `https://www.inf.fu-berlin.de/w/SE/ThesisDPPIV`, 2009.

[KSA09] Justin Karneges and Peter Saint-Andre. In-band bytestreams xep-0047. `http://xmpp.org/extensions/xep-0047.html`, March 2009. (Online abgerufen am 10. Januar 2010).

[LBSA+09] Scott Ludwig, Joe Beda, Peter Saint-Andre, Robert McQueen, Sean Egan, and Joe Hildebrand. Jingle xep-0166. `http://xmpp.org/extensions/xep-0166.html`, Juni 2009. (Online abgerufen am 10. Januar 2010).

[LC03] Kim Man Lui and Keith C. C. Chan. When does a pair outperform two individuals? In *Extreme Programming and Agile Processes in Software Engineering*, volume 2675 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003.

[Mad05] Lech Madeyski. *Software Engineering: Evolution and Emerging Technologies, Edited by: K. Zielinski and T. Szmuc*, volume 130 of *Frontiers in Artificial Intelligence and Applications*, chapter Preliminary Analysis of the Effects of Pair Programming and Test-Driven Development on the External Code Quality, pages 113–123. IOS Press, 2005.

[MC09] L. Rizzo M. Carbone. Dummynet revisited, May 2009.

[MWBF02] Charlie McDowell, Linda Werner, Heather Bullock, and Julian Fernald. The effects of pair-programming on performance in an introductory programming course. In *SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pages 38–42, New York, NY, USA, 2002. ACM.

[MWBF03] Charlie McDowell, Linda Werner, Heather E. Bullock, and Julian Fernald. The impact of pair programming on student performance, perception and persistence. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 602–607, Washington, DC, USA, 2003. IEEE Computer Society.

[NET] netem. `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`.

[Nis] Nist net. `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`.

[NJuOL05] Jerzy R. Nawrocki, Micha Jasiski, ukasz Olek, and Barbara Lange. Pair programming vs. side-by-side programming. In *Software Process Improvement*, volume 3792 of *Lecture Notes in Computer Science*, pages 28–38. Springer Berlin / Heidelberg, 2005.

[Nos98] John T. Nosek. The case for collaborative programming. *Commun. ACM*, 41(3):105–108, 1998.

[NW01] Jerzy Nawrocki and Adam Wojciechowski. Experimental evaluation of pair programming. In *Proceedings of the 12th European Software Control and Metrics Conference*, pages 269–276, London, UK, 2001. Shaker Publishing.

[NWF+03] Nachiappan Nagappan, Laurie Williams, Miriam Ferzli, Eric Wiebe, Kai Yang, Carol Miller, and Suzanne Balik. Improving the cs1 experience with pair programming. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 359–362, New York, NY, USA, 2003. ACM.

[NWW+03] Nachiappan Nagappan, Laurie A. Williams, Eric Wiebe, Carol Miller, Suzanne Balik, Miriam Ferzli, and Julie Petlick. Pair learning: With an eye toward future success. In Frank Maurer and Don Wells, editors, *Extreme Programming and Agile Methods - XP/Agile Universe 2003, Third XP*

and Second Agile Universe Conference, New Orleans, LA, USA, August 10-13, 2003, Proceedings, volume 2753 of *Lecture Notes in Computer Science*, pages 185–198. Springer, 2003.

[ORB] Orbit. `http://www.orbit-lab.org`. (Online abgerufen am 10 Januar 2010).

[PIC06] Picocontainer. `http://www.picocontainer.org/`, 2006.

[PSS08] Lutz Prechelt, Ulrich Stärk, and Stephan Salinger. 7 types of cooperation episodes in side-by-side programming. Technical report, Department of Computer Science, Freie Universität Berlin, 2008.

[PSS09] Lutz Prechelt, Ulrich Stärk, and Stephan Salinger. Types of cooperation episodes in side-by-side programming. In *Proceedings of the 21st Annual Workshop of Psychology of Programming Interest Group (PPIG '09)*. ppig.org, July 2009.

[Rie08] Oliver Rieger. Weiterentwicklung einer eclipse-erweiterung für verteilter paarprogrammierung im hinblick auf kollaboration und kommunikation. Dilpomarbeit, Freie Universität Berlin, Dep. of CS., `https://www.inf.fu-berlin.de/w/SE/ThesisDPPIII`, 2008.

[Riz] Luigi Rizzo. Dummynet. `http://info.iet.unipi.it/~luigi/dummynet/`.

[RMM09] J. Rosenberg, R. Mahy, and P. Matthews. Traversal using relays around nat (turn). `http://tools.ietf.org/html/draft-ietf-behave-turn-16`, July 2009. Stand: seit 4.1.2010 abgelaufener Draft.

[Ros07] J. Rosenberg. Interactive connectivity establishment (ice). `http://www.ietf.org/id/draft-ietf-mmusic-ice-19.txt`, October 2007. Stand: seit 1.5.2008 abgelaufener Draft.

[RWHM03] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. Simple traversal of user datagram protocol (udp) through network address translators (nats). `http://tools.ietf.org/html/rfc3489`, März 2003. RFC 3489.

[SAa] Peter Saint-Andre. Extensible messaging and presence protocol. `http://www.ietf.org/rfc/rfc3920.txt`. (Online abgerufen am 10. Januar 2010).

[SAb] Peter Saint-Andre. Xmpp extension protocols. `http://xmpp.org/extensions`. (Online abgerufen am 10. Januar 2010).

[Sma] Smack xmpp bibliothek. `http://www.igniterealtime.org/projects/smack/`.

[SMSA07] Dave Smith, Matthew Miller, and Peter Saint-Andre. Socks5 bytestreams xep-0065. `http://xmpp.org/extensions/xep-0065.html`, May 2007. (Online abgerufen am 10. Januar 2010).

[SWN+03] David Stotts, Laurie Williams, Nachiappan Nagappan, Prashant Baheti, Dennis Jen, and Anne Jackson. Virtual teaming: Experiments and experiences with distributed pair programming. Technical report, Dept. of Computer Science, University of North Carolina, Chapel Hill, NC 27599 Dept. of Computer Science, North Carolina State University, Raleigh, NC 27695, 2003.

[Só09] Tas Sóti. Einladungsprozess in saros. Bachelorarbeit, Freie Universität Berlin, Dep. of CS., `https://www.inf.fu-berlin.de/w/SE/ThesisDPPXI`, 2009.

[VIN] Vini - a virtual network infrastructure. `http://www.vini-veritas.net`. (Online abgerufen am 10 Januar 2010).

[VYW+02] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. Scalability and accuracy in a large-scale network emulator. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation Due to copyright restrictions we are not able to make the PDFs for this conference available for downloading*, pages 271–284, New York, NY, USA, 2002. ACM.

[Wil] Laurie Williams. *The Collaborative Software Process*. Phd dissertation at department of computer science, University of Utah, Salt Lake City, USA.

[WKCJ00] Laurie Williams, Robert R. Kessler, Ward Cunningham, and Ron Jeffries. Strengthening the case for pair programming. *IEEE Softw.*, 17(4):19–25, 2000.

[WU01] Laurie Williams and Richard L. Upchurch. In support of student pair-programming. *SIGCSE Bull.*, 33(1):327–331, 2001.

Anhang A

Messungen

A.1 Durchsatz Messung der Synchronisierung von Saros

Tabelle A.1
Durchsatzmessung bei der Synchronisierung. Suessapfel befand sich im WLAN Eduroam und brazzaville im Informatiknetz.

source	destination	network	connectiontype	datatype	sent as	datavol	time	throughput	Saros version
suessapfel	brazzaville	Internet	Jingle/TCP	Saros Projekt	Dateien	39 MB	8 s	4992 kB/s	r1395
suessapfel	brazzaville	Internet	Jingle/TCP	Saros Projekt	Archiv	39 MB	10 s	3994 kB/s	r1395
suessapfel	brazzaville	Internet	Jingle/UDP	Saros Projekt	Dateien	39 MB	41 s	974 kB/s	r1442
suessapfel	brazzaville	Internet	Jingle/UDP	Saros Projekt	Archiv	39 MB	29 s	1377 kB/s	r1442
suessapfel	brazzaville	Internet	Socks5	Saros Projekt	Archiv	39 MB	502 s	79 kB/s	r1442
suessapfel	brazzaville	Teles VPN	Jingle/TCP	Saros Projekt	Dateien	39 MB	31 s	1288 kB/s	9.5.29
suessapfel	brazzaville	Teles VPN	Jingle/TCP	Saros Projekt	Archiv	39 MB	24 s	1664 kB/s	9.5.29
suessapfel	brazzaville	Teles VPN	IBB Teles	random data	Archiv	10008 kB	2514 s	4 kB/s	9.5.29

source	destination	worked?
SarosA@jabber.ccc.de	SarosB@jabber.se	YES
SarosB@jabber.se	SarosA@jabber.ccc.de	NO
OtherIM@jabber.ccc.de	SarosB@jabber.se	YES
SarosB@jabber.se	OtherIM@jabber.ccc.de	NO
SarosA@jabber.ccc.de	SarosB@jabber.ccc.de	NO
OtherIM@jabber.ccc.de	SarosB@jabber.ccc.de	YES

A.2 Test verschiedener Jabber Server

A.2.1 Add buddy

Es wurden einige Jabber Server bzgl. des hinzufügens von JIDs zur 'Buddyliste' mit Saros und anderen IM Clients getestet. Als Jabber Client von OtherIM wurde PSI[1] verwendet.
Nach dem Fehlschlagen der Szenarien zu urteilen heißt das für mich:

- jabber.ccc.de vertraut NICHT jabber.se
- jabber.se vertraut jabber.ccc.de

A.2.2 Zertifikate von Jabber Servern

jabber.se und *jabber.cc* haben veraltete X509 Zertifikate. Die Anfragen aus Tabelle A.2.1 von den genannten Servern werden von anderen, wie *jabber.ccc.de*, wahrscheinlich deshalb abgelehnt.

A.2.3 Durchsatz bei IBB

IBB wird von jabber.ccc.de definitiv sehr stark gedrosselt. Das übertragen der Dateiliste vom Saros Projekt dauerte ca. 3 Minuten.

A.2.4 Verschiedenes

jabber.no hat mir die Nachricht geschickt, dass Dienste nur noch intern nutzbar sind. Hier die Message:

```
------------------8<-------------------
In an effort to better the jabber.no service we have concluded
that we need to remove external access for users of our transports.
This change will be done during this week (week 49) and will not
affect local users (you!), but should result in a snappier service. :)
Thank you.
------------------>8-------------------
```

Das erklärt dann auch zuverlässig warum Accounts von jabber.no nicht mit anderen jabber Accounts funktionieren.

[1] `http://psi-im.org/` (Online abgerufen am 23. Februar 2010)

Anhang B

Konfigurationen der Rechner

Tabelle B.1 zeigt die Aufgaben der virtuellen Rechner und die Zugriffsmöglichkeiten auf sie aus dem Informatiknetz. Es wurde pro Dienst und Rechner je ein Loch in die Windows 2003 Server Firewall gebohrt, sowie in VMware Server eine Portweiterleitung konfiguriert, die externe Ports auf interne abbildet.

Tabelle B.1
Die angegebenen Ports entsprechen den Portweiterleitungen für den Zugriff aus dem Informatiknetz.

Rechnernamen im VM Netz	SSH Port	VNC Port	Aufgabe
gw.transit.test	22		Dummynet,Paketfilter,Router
xmpp1.internet.test	2220		XMPP,STUN
xmpp2.internet.test	2221		XMPP,DNS
orchester.internet.test	2226	5926	Dirigent
rmiswtbot1.internet.test	2222	5922	Musikant
rmiswtbot2.internet.test	2223	5923	Musikant
rmiswtbot1.delay.test	2224	5924	Musikant
rmiswtbot1.fw.test	2225	5925	Musikant

B.1 FreeBSD 6.4

In der Abbildung B.1 ist die zentrale Konfigurationsdatei des FreeBSD Gateways zu sehen. Es enthält die aktivierten Dienste, Kernelmodule, die Netzwerkkonfiguration, weitere Netzwerkschnittstellen und statische Routen.

```
#
# loaded modules
#
sshd_enable="YES"
firewall_enable="yes"
firewall_type="open"
dummynet_enable="yes"
gateway_enable="yes"
#
```

```
# local network configuration
#
ifconfig_lnc0="inet 192.168.141.10  netmask 255.255.255.0"
hostname="gw.transit.test"
defaultrouter="192.168.141.2"
#
# virtual network - gateway addresses
#
ifconfig_lnc0_alias0="inet 192.168.23.1  netmask
    255.255.255.128"
ifconfig_lnc0_alias1="inet 192.168.23.129  netmask
    255.255.255.128"
ifconfig_lnc0_alias2="inet 192.168.24.1  netmask
    255.255.255.0"
ifconfig_lnc0_alias3="inet 192.168.25.1  netmask
    255.255.255.128"
ifconfig_lnc0_alias4="inet 192.168.25.129  netmask
    255.255.255.128"
ifconfig_lnc0_alias5="inet 192.168.26.1  netmask
    255.255.255.0"
ifconfig_lnc0_alias6="inet 192.168.27.1  netmask
    255.255.255.0"
#
# virtual network - static routes
#
# xmpp server
static_routes="xmpp"
route_xmpp="-net 192.168.23.0/25 192.168.23.1"
route_xmpp2="-net 192.168.23.128/25 192.168.23.129"
# saros clients internet
static_routes="internet"
route_internet="-net 192.168.24.0/24 192.168.24.1"
# saros clients NAT
static_routes="nat"
route_nat="-net 192.168.25.0/25 192.168.25.1"
route_nat="-net 192.168.25.128/25 192.168.25.129"
# saros clients delayed
static_routes="delay"
route_delay="-net 192.168.26.0/24 192.168.26.1"
# saros clients blocked
static_routes="fw"
route_fw="-net 192.168.27.0/24 192.168.27.1"
```

Die Konfiguration des Emulationsnetzes mit Dummynet und ipfw wird per Shellskript B.1 geladen. Kontrollkanäle wie SSH, VNC oder RMIRegistry, sowie die Verbindung zum Rechner `orchester.internet.test` werden von der Emulatorbehandlung ausgenommen.

```
#! /usr/bin/env bash

# uplink to the internet, internet DNS
INTERNET_GW=192.168.141.2

#
# emulator parts
#
CONDUCTOR=192.168.23.42
SERVER_NET=192.168.23.0/24
DELAY_NET=192.168.26.0/24
FIREWALLED_NET=192.168.27.0/24

# firewall command
PF_CMD="`which ipfw` -q"

# disable firewall before flush
$PF_CMD disable firewall
# flush current ruleset
$PF_CMD flush

#
# Choosable pipe configuration for the delay network
#
if [[ $1 == "--none" ]]; then
$PF_CMD pipe 1 config mask src-ip $DELAY_NET dst-ip
   $DELAY_NET
elif [[ $1 == "--802.11a" ]]; then
## pipe with bandwidth (802.11a), guessed delay and packet
   loss rate (1%)
$PF_CMD pipe 1 config bw 20Mbit/s delay 5ms mask src-ip
   $DELAY_NET dst-ip $DELAY_NET plr 0.01

elif [[ $1 == "--802.11b" ]]; then
## pipe with bandwidth (802.11b), guessed delay and packet
   loss rate (1%)
$PF_CMD pipe 1 config bw 3Mbit/s delay 5ms mask src-ip
   $DELAY_NET dst-ip $DELAY_NET plr 0.01

elif [[ $1 == "--802.11g" ]]; then
## pipe with bandwidth (802.11g), guessed delay and packet
   loss rate (1%)
$PF_CMD pipe 1 config bw 16Mbit/s delay 5ms mask src-ip
   $DELAY_NET dst-ip $DELAY_NET plr 0.01

elif [[ $1 == "--highpacketloss" ]]; then
## pipe with high packet loss rate (40%)
```

```
$PF_CMD pipe 1 config mask src-ip $DELAY_NET dst-ip
   $DELAY_NET plr 0.4

elif [[ $1 == "--highdelay" ]]; then
## pipe with high delay (TCP channel has 1.5s delay)
$PF_CMD pipe 1 config delay 750ms mask src-ip $DELAY_NET dst
   -ip $DELAY_NET

elif [[ $1 == "--smallbandwidth" ]]; then
## pipe with small bandwidth
$PF_CMD pipe 1 config bw 250kBit/s mask src-ip $DELAY_NET
   dst-ip $DELAY_NET

elif [[ $1 == "--adsl" ]]; then
$PF_CMD pipe 1 config bw 2Mbit/s delay 90ms mask src-ip
   $DELAY_NET dst-ip $DELAY_NET plr 0.005

elif [[ $1 == "--sdsl" ]]; then
$PF_CMD pipe 1 config bw 2Mbit/s delay 45ms mask src-ip
   $DELAY_NET dst-ip $DELAY_NET plr 0.005

elif [[ $1 == "--mixed" ]]; then
## add pipe with delay, small bandwidth and packet loss
## rate (5%) for inbound and outbound network
$PF_CMD pipe 1 config bw 1Mbit/s delay 250ms mask src-ip
   $DELAY_NET dst-ip $DELAY_NET plr 0.05

else
## nothing
$PF_CMD pipe 1 config mask src-ip $DELAY_NET dst-ip
   $DELAY_NET
fi

#
# static firewall ruleset
#
# localhost
$PF_CMD add 100 allow ip from any to any via lo0
$PF_CMD add 200 deny ip from any to 127.0.0.0/8
$PF_CMD add 300 deny ip from 127.0.0.0/8 to any
# _real_ internet
$PF_CMD add 500 allow ip from me to $INTERNET_GW
$PF_CMD add 600 allow ip from $INTERNET_GW to me

## orchester.internet.test: no delay and allow all
$PF_CMD add 1000 allow ip from any to $CONDUCTOR
$PF_CMD add 1000 allow ip from $CONDUCTOR to any
```

```
## use the pipe 1 as rule nummber 2000 for $DELAY_NET
   incoming and outgoing
## for all besides ssh,vnc,rmi
$PF_CMD add 2000 pipe 1 in dst-ip $DELAY_NET not dst-port
   22,5901,1099
$PF_CMD add 2000 pipe 1 out src-ip $DELAY_NET not src-port
   22,5901,1099

## add rules that allow communication via server network
## (Socks5 is allowed, decisision on xmpp server)
$PF_CMD add 10000 allow tcp from $FIREWALLED_NET to
   $SERVER_NET
$PF_CMD add 10000 allow tcp from $SERVER_NET to
   $FIREWALLED_NET

## add rules dropping all tcp packets besides ssh, rmi and
   vnc
$PF_CMD add 10003 allow tcp from any to $FIREWALLED_NET
   22,1099,5901
$PF_CMD add 10003 allow tcp from $FIREWALLED_NET
   22,1099,5901 to any
$PF_CMD add 20000 drop tcp from $FIREWALLED_NET to any
$PF_CMD add 20000 drop tcp from any to $FIREWALLED_NET

# else allow any 2 any
$PF_CMD add 60000 allow ip from any to any

$PF_CMD enable firewall
```

B.2 Debian Lenny

Debian Lenny wurde als Serverbetriebsystem gewählt. Es gibt zwei virtuelle Maschinen auf denen je einmal die XMPP Server Software Openfire in Version 3.6.4 läuft. Auf einem läuft der VM-Netz lokale DNS und auf dem anderen der für Jingle notwendige STUN Dienst.

B.2.1 XMPP1

Ein STUN Server wird über init auf `xmpp1.internet.test` gestartet. Der STUN Dienst ist unter `stun1.internet.test:3478` und `stun2.internet.test:3479` erreichbar. Die Netzwerkkonfigration B.1 von `xmpp1.internet.test`.

```
# The loopback network interface
auto lo
iface lo inet loopback
#
# virtual internet
auto eth0
iface eth0 inet static
  address 192.168.23.5
  netmask 255.255.255.0
  network 192.168.23.0
  broadcast 192.168.23.255
  gateway 192.168.23.1
# stun needs a 2nd ip
auto eth0:0
iface eth0:0 inet static
  address 192.168.23.6
  netmask 255.255.255.0
  network 192.168.23.0
  broadcast 192.168.23.255
  gateway 192.168.23.1
```

Abbildung B.1: Netzwerkkonfigration von `xmpp1.internet.test`

B.2.2 XMPP2

Auf `xmpp2.internet.test` laufen der zweite XMPP und der DNS Dienst. Als DNS Serversoftware DJBDNS [1]. Ein DNS Server ist für die Auflösung von Rechnernamen innerhalb von XMPP zwingend notwendig. Eine statische Auflösung von Rechnernamen mittels Einträgen in der Datei `/etc/hosts`, hatte mit der Kombination XMPP Server 'Openfire' und XMPP-Client 'Spark' nicht funktioniert.
Viele XMPP Dienstanbieter wollen ihren Dienst hochverfügbar anbieten. Für Hochverfügbarkeit läßt man den aktuell zuständigen Servernamen zu einem Dienst über DNS SRV nach RFC 2782 lokalisieren. Die mitgelieferten Werkzeuge von DJBDNS unterstützen nicht direkt SRV Einträge. Mit Hilfe einer Webapplikation [2] kann man sich die Einträge bauen lassen und zur Konfiguration B.2 hinzufügen. Die Konfiguration wird aktiv, nachdem man das Makefile zum neuladen des Dienstes startet. Testen der SRV Einträge verdeutlicht Listing B.3. Die Netzwerkkonfigration B.4 von `xmpp2.internet.test`.

B.3 Kubuntu

Die Benutzersysteme, auf denen Eclipse, Saros, Spark und eine schlanke grafische Benutzeroberfläche namens 'fluxbox' installiert wurden, laufen unter Kubuntu 9.0.4 Codename 'jaunty'. Zusätzlich zum administrativen Dienst SSH wurde der VNC Server

[1] `http://cr.yp.to/djbdns.html` und `http://tinydns.org/` (Online abgerufen am 23. Februar 2010)

[2] `http://www.anders.com/projects/sysadmin/djbdnsRecordBuilder/#SRV` (Online abgerufen am 23. Februar 2010)

```
.test:127.0.0.1:a:259200
.transit.test:127.0.0.1:a:259200
.141.168.192.in-addr.arpa:127.0.0.1:a:259200
=gw.transit.test:192.168.141.10:86400
=gw1.internet.test:192.168.23.1:86400
=gw2.internet.test:192.168.23.129:86400
=gw3.internet.test:192.168.24.1:86400
=gw4.internet.test:192.168.25.1:86400
=gw5.internet.test:192.168.25.129:86400
=gw6.internet.test:192.168.26.1:86400
=gw7.internet.test:192.168.27.1:86400
.internet.test:127.0.0.1:a:259200
.23.168.192.in-addr.arpa:127.0.0.1:a:259200
.24.168.192.in-addr.arpa:127.0.0.1:a:259200
.25.168.192.in-addr.arpa:127.0.0.1:a:259200
=xmpp2.internet.test:192.168.23.130:86400
=xmpp1.internet.test:192.168.23.5:86400
:_xmpp-server._tcp.xmpp2.internet.test
    :33:\000\012\000\144\024\225\005xmpp2\010internet\004test
    \000:86400
:_xmpp-server._tcp.xmpp1.internet.test
    :33:\000\012\000\144\024\225\005xmpp1\010internet\004test
    \000:86400
+stun1.internet.test:192.168.23.5:86400
+stun2.internet.test:192.168.23.6:86400
=orchester.internet.test:192.168.23.42:86400
=rmiswtbot1.internet.test:192.168.24.21:86400
=rmiswtbot2.internet.test:192.168.24.22:86400
.delay.test:127.0.0.1:a:259200
.26.168.192.in-addr.arpa:127.0.0.1:a:259200
=rmiswtbot1.delay.test:192.168.26.21:86400
.fw.test:127.0.0.1:a:259200
.27.168.192.in-addr.arpa:127.0.0.1:a:259200
=rmiswtbot1.fw.test:192.168.27.21:86400
```

Abbildung B.2: DJBDNS (aka tinydns) Konfigurationsdatei

```
[root@rmiswtbot1.internet.test:~]$ host -t SRV _xmpp-server.
    _tcp.xmpp2.internet.test
_xmpp-server._tcp.xmpp2.internet.test has SRV record 10 100
    5269 xmpp2.internet.test.
[root@rmiswtbot1.internet.test:~]$ host -t SRV _xmpp-server.
    _tcp.xmpp1.internet.test
_xmpp-server._tcp.xmpp1.internet.test has SRV record 10 100
    5269 xmpp1.internet.test.
```

Abbildung B.3: Erfolgreiche Tests der SRV Einträge für XMPP

```
# The loopback network interface
auto lo
iface lo inet loopback

# virtual internet
auto eth0
iface eth0 inet static
  address 192.168.23.130
  netmask 255.255.255.0
  network 192.168.23.0
  broadcast 192.168.23.255
  gateway 192.168.23.1
```

Abbildung B.4: Netzwerkkonfigration von `xmpp2.internet.test`

'tighvnc' installiert. Er ermöglicht einen schnelleren grafischen Zugriff auf die Endsysteme, im Vergleich zu X über SSH. Die Authentifizierung von VNC basiert auf ein geteiltes Geheimnis und kann mit Hilfe des Programms 'tightvncpasswd' geändert werden. Die Entwicklerwerkzeuge `subversion` und Sun Java 6 wurden installiert, um Saros verwalten und testen zu können.

Wie in Abschnitt 4.4.1 erwähnt gibt es zwei verschiedene Arten von Klientenrechnern. Einmal den Dirigenten und mehrere Saros-Musikanten.
Der Dirigent ist im Emulationsnetzwerk unter `orchester.internet.test` erreichbar und bekommt einen Bypass durch Dummynet und Firewall. Das Loch durch den Paketfilter soll die direkte Steuerung der Saroseinheiten ohne Verzögerung oder gar Unterbrechung ermöglichen.
Saros-Musikanten starten, bei Login des Benutzer 'saros', automatisch ein Eclipse im Sarostestmodus. Dafür wurde ein Startskript B.5 geschrieben, dass beim Start der X-Sitzung B.6 mitgestartet wird. Die Kommandozeile zum Starten einer Sarosinstanz, wie in B.5 gezeigt, läßt aus dem 'Debug View' von Eclipse kopieren. Dafür muss man die Eigenschaften des Zielprozesses über das Kontextmenü anzeigen lassen. Um ein Problem solcher Art zu lösen, hat es sich als effizient erwiesen synchrone Kommunikation über öffentliche Kanäle, wie im folgenden Beispiel in #eclipse auf freenode.net, zu benutzen.

> 14:42 <spawnyd> hi! Any idea to generate/export/.. an eclipse run configuration to an Ant/shell/.. script?
>
> 14:46 <paulweb515> spawnyd: there isn't, AFAIK. You can go to the debug view and pick your process properties. That will give you the full command line of the last launch
>
> 14:47 <spawnyd> paulweb515: ty that's enough for my use case.

```
#! /bin/sh

LOGDIR=$HOME/rmibotlog
LOGFILE=$LOGDIR/`hostname`_`date "+%y.%m.%d.%H:%M:%S.log"`
PROJECT="$HOME/rmibotproject/Saros"
RUNTIME=$PROJECT/../../runtime-EclipseApplication
ECLIPSE_DIR="$HOME/eclipse-swtbot"
ECLIPSE_PLUGIN_DIR="$ECLIPSE_DIR/plugins"
HOSTNAME=`hostname`
JAVA=`/usr/bin/which java`

if [ ! -x $JAVA ]; then
  echo "java is not installed" >> $LOGFILE
else
$JAVA -Dosgi.requiredJavaVersion=1.5 \
-XX:MaxPermSize=256m -Xms512m -Xmx512m -ea \
-Djava.rmi.server.codebase=file:$PROJECT/bin \
-Djava.security.manager \
-Djava.security.policy=file:$PROJECT/bin/all.policy \
-Djava.rmi.server.hostname=$HOSTNAME \
-Dde.fu_berlin.inf.dpp.testmode=1099 \
-Declipse.pde.launch=true \
-Dfile.encoding=UTF-8 \
-classpath $PROJECT/../org.eclipse.equinox.launcher_1.0.201.R35x_v20090715.
    jar \
org.eclipse.equinox.launcher.Main \
-launcher /home/saros/eclipse-swtbot/eclipse \
-name Eclipse -showsplash 600 -product org.eclipse.platform.ide \
-data /home/saros/workspace-swtbot/../runtime-EclipseApplication \
-configuration "file:$PROJECT/../Eclipse Application (RMI swtbot_project)/" \
-dev "file:$PROJECT/../Eclipse Application (RMI swtbot_project)/dev.
    properties" \
-os linux -ws gtk -arch x86 -nl en_US
#2>&1 >> $LOGFILE
fi
```

Abbildung B.5: `start-rmiswtbot-project.sh`: Skript zum Starten von Saros im Testmodus

```
exec bin/start-rmiswtbot.sh &
exec fluxbox
```

Abbildung B.6: Veränderter Start der X-Sitzung für für Musikanten

Anhang C

Testframework

C.1 Verwendung des Testframeworks

C.1.1 Framework Testlauf

Der Start eines JUnit Test auf der Teststeuerung `orchester.internet.test` beginnt einen automatischen vom JUnit Test gesteuerten Testlauf, der in den Bildern von Abbildung C.1 zu sehen ist. Zur Veranschaulichung befinden sich im Internet zwei Quicktime Videos. Auf dem ersten sieht man einen Testlauf im Emulator, das andere zeigt die lokale Ausführung auf einem Entwicklungsrechner.

- Emulator: `http://userpage.fu-berlin.de/sszuecs/emulator_video.html`
- Lokal: `http://userpage.fu-berlin.de/sszuecs/lokal_video.html`

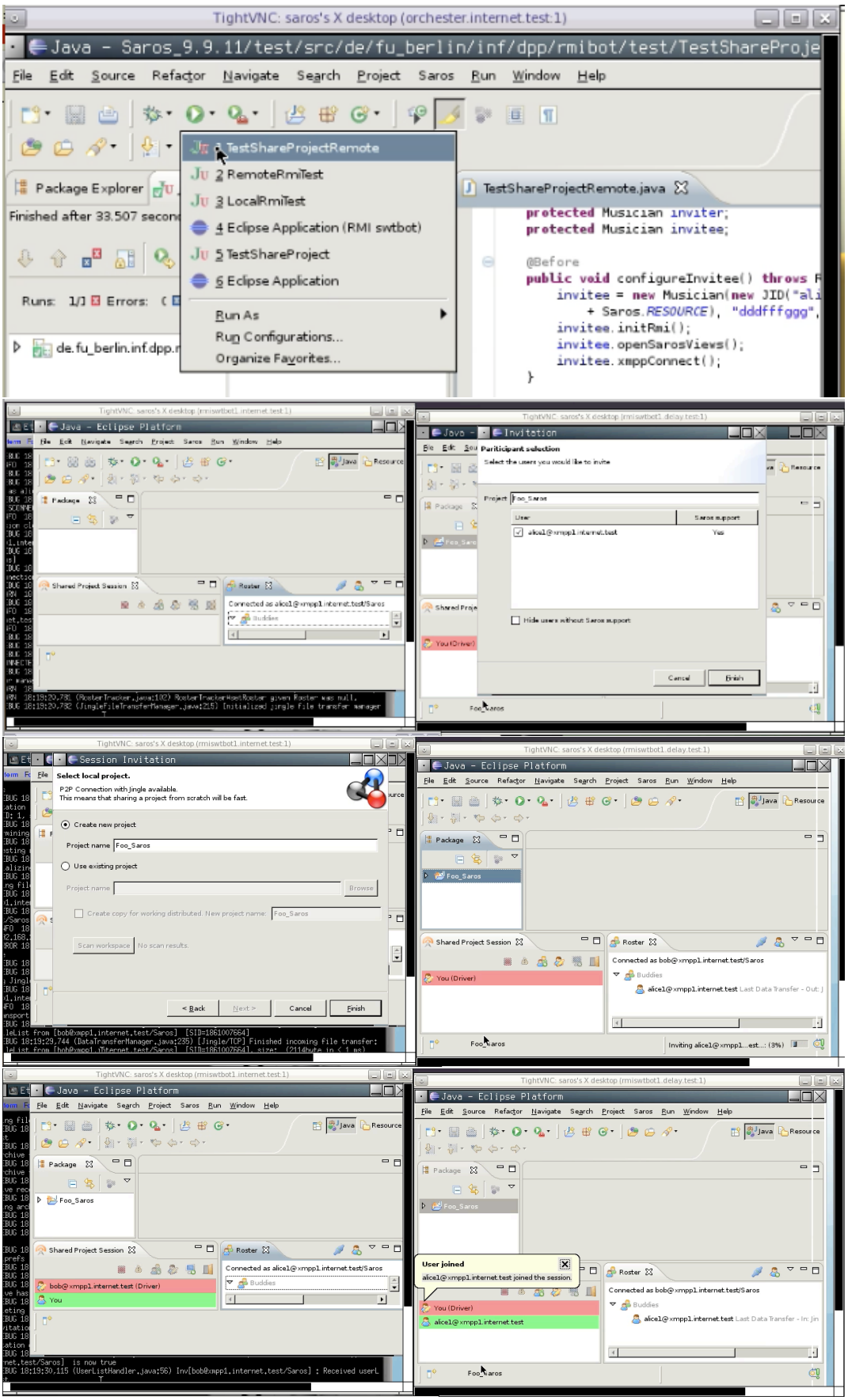

Abbildung C.1: Automatisch gesteuerte Situation einer Einladung im Testemulator

C.1.2 Automatismen

Verteilen und konfigurieren

Die zentrale Verteilung und Steuerung der Skriptsammlung wird vom Skript C.1.2 übernommen.

```
#!/usr/bin/env ruby
#
#= fap_deploy - Fully Automated Patch Deployment
#== desciption
# This is a tool for automatic deployment and configuration
# of the testemulator. It uses UNIX tools like ssh, rsync,
# svn and patch to deploy and patch a source tree on remote
# vm hosts. It uses the fully_automated_deployment shell-
# script collection to work on the remote machines after
# sync.
#
#== installation
#   You need the following packages on your system:
#* ssh
#* rsync
#* ruby
#* rubygems
#* gems
#  * net-ssh
#  * threadify (optional for faster deployment)
#
# You have to use ssh with key authentication, because of
   usability and
# security. If you don't want to use keys then you have to
   patch this file.
#
#== usage
#$ ./fap_deploy --help
#Usage: fap_deploy [options]
#       --sshprivkey <keyfile>        use given private key
   to authenticate on hosts
#       --sniffer                     Start a sniffer on the
   gateway. You find the file /tmp/packet.pcap on the
   gateway
#       --skipsync <type>             skip project
   synchronization
#       --hosts <host:port,host:port,..>
#                                     use given host port
   combinations to prepare them
#       --branch <branchname>         This is used only by
   patch deploy. Use given branchname to switch remote
   working copies, otherwise we force TRUNK
```

```
#       --project <projectdir>        use given project for
   auto deploy
#       --patch <patchfile>           use given patchfile for
    auto deploy
#       --dropin <zipfile>            use given dropin for
   auto deploy
#       --emulator <name>             Use given emulator
   configuration.
#Currently known emulator configurations:
#< 802.11a | 802.11b | 802.11g | adsl | sdsl |
   highpacketloss | highdelay | smallbandwidth | mixed >
#

base = "#{File.dirname(File.expand_path(__FILE__))}"
$:.unshift(base)

require 'optparse'
require 'rubygems'
require 'net/ssh'
THREADIFY = require 'threadify'
require 'lib/runcommand'
include Runcommand

RSYNC = `which rsync`.strip
# remote patch filename
PATCHFILE="FAD.patch"
CLEAN_PATCHFILE_CMD = "rm #{PATCHFILE}"
# remote DROPIN name
DROPIN="saros-dropin.zip"
# remote PROJECT path
PROJECT_ROOT="rmibotproject"
PROJECT_NAME="Saros"

$options = {}
# defaults
$options[:cmd] = nil
$flags=""
$options[:user] = 'root'
$options[:sshprivkey] = "/Users/sszuecs/.ssh/suess_mb_dsa"
$options[:uris] = []
[2222,2224,2226].each do |port|
  $options[:uris] << ["ibadan.imp.fu-berlin.de",port]
end
$options[:emulator_uri] = URI.parse("ssh://root@ibadan.imp.
   fu-berlin.de:22/root")

opts = OptionParser.new do |opts|
```

```
opts.on("--sshprivkey <keyfile>", "use given private key
    to authenticate on hosts") do |file|
  $options[:sshprivkey] = file.strip if File.exists?(file.
      strip)
end
opts.on("--sniffer", "Start a sniffer on the gateway. You
    find the file /tmp/packet.pcap on the gateway") do
  $options[:sniffer] = "/usr/sbin/tcpdump -s0 -w /tmp/
      packet.pcap 'not ( port 22 or port 5901 or port 1099
      )'"
end
opts.on("--skipsync <type>", "skip synchronization, type
    can be < project |\
patch | dropin >. It is used to choose remote scripts to
    run for machine \
configuration in order how to run Saros.") do |type|
  $options[:skipsync] = type.chomp
  $options[:cmd] = case $options[:skipsync]
  when "project"
    "bin/fap_project_deploy.sh"
  when "patch"
    "bin/fap_patch_deploy.sh"
  when "dropin"
    "bin/fap_dropin_deploy.sh"
  else
    raise OptionParser::InvalidOption.new("unknown type
        '#{$options[:skipsync]}'")
  end
end
opts.on("--hosts <host:port,host:port,..>", "use given
    host port combinations to prepare them") do |s|
  $options[:uris] = []

  s.split(",").map do |socket|
    host,port = socket.split(":")
    $options[:uris] << [host, port.to_i]
  end
end
opts.on("--branch <branchname>", "This is used only by
    patch deploy. Use \
given branchname to switch remote working copies,
    otherwise we force TRUNK".
squeeze(" ")) do |name|
  $flags = " --branch #{name.strip}"
end
opts.on("--project <projectdir>", "use given project for
    auto deploy") do |file|
```

```
    $options[:project] = file.strip if File.directory?(file.
        strip)
    $options[:cmd] = "bin/fap_project_deploy.sh"
  end
  opts.on("--patch <patchfile>", "use given patchfile for
      auto deploy") do |file|
    $options[:patch] = file.strip if File.exists?(file.strip
        )
    $options[:cmd] = "bin/fap_patch_deploy.sh"
    $options[:cmd] << $flags
  end
  opts.on("--dropin <zipfile>", "use given dropin for auto
      deploy") do |file|
    $options[:dropin] = file.strip if File.exists?(file.
        strip)
    $options[:cmd] = "bin/fap_dropin_deploy.sh"
  end
  opts.on("--emulator <name>", "Use given emulator
      configuration.\nCurrently \
  known emulator configurations: \
  \n< none | 802.11a | 802.11b | 802.11g | adsl | sdsl |
      highpacketloss | \
  highdelay | smallbandwidth | mixed >".squeeze(" ")) do |
      name|

    if name.match(/^802\.11[abg]$/) or name.match(/^[as]dsl$
        /) or
      name.match(/^[as]dsl$/) or name == "highpacketloss" or
      name == "highdelay" or name == "smallbandwidth" or
          name == "none"

      $options[:emulator] = "--#{name}"
    else
      raise OptionParser::InvalidOption.new("unknown
          emulator configuration: #{name}")
    end
  end

  if $options[:help]
    puts opts.banner
  end
end
begin
  opts.parse!(ARGV)
rescue OptionParser::InvalidOption => e
  STDERR.puts "#{e}\n#{'-'*70}\n"
  STDERR.puts opts
```

```
  exit(1)
end

# Block to synchronize all files from root/ to remote
   machines.
# clean remote patchfile
blk_root = lambda do |pair|
  host,port = pair
  system("#{RSYNC} -e \"ssh -p #{port}\" -a root/* root@#{
     host}:bin/")
  run_command_on_host($options[:user], $options[:sshprivkey
     ], host, port, CLEAN_PATCHFILE_CMD)
end

# Block to synchronize all files from saros/ to remote
   machines.
blk_saros = lambda do |pair|
  host,port = pair
  system("#{RSYNC} -e \"ssh -p #{port}\" saros/* saros@#{
     host}:bin/")
  system("#{RSYNC} -e \"ssh -p #{port}\" #{$options[:patch]}
      saros@#{host}:patch/#{PATCHFILE}") if $options[:patch]
  system("#{RSYNC} -e \"ssh -p #{port}\" #{$options[:dropin
     ]} saros@#{host}:#{DROPIN}") if $options[:dropin]
  system("#{RSYNC} -e \"ssh -p #{port}\" -av #{$options[:
     project]}/* saros@#{host}:#{PROJECT_ROOT}/#{
     PROJECT_NAME}/") if $options[:project]
end

blk_cmd = lambda do |pair|
  host,port = pair
  run_command_on_host($options[:user], $options[:sshprivkey
     ], host, port, $options[:cmd])
end

STDERR.puts <<HERE
+-------------------------------------------------------------+
|          fad - Fully Automated Deployment                   |
|                                                             |
|                 by Sandor Szuecs 01/2010                    |
|          mailto:sandor.szuecs@fu-berlin.de                  |
+-------------------------------------------------------------+
HERE

# Choose message each() or threadify() for running blocks.
sym = THREADIFY ? :threadify : :each
```

```
unless $options[:skipsync]
  $options[:uris].send(sym, &blk_root)
  $options[:uris].send(sym, &blk_saros)
  puts "====== synced hosts ======"
end
$options[:uris].send(sym, &blk_cmd)
puts "====== scriptrunner end ======"
if $options[:emulator]
  system("scp -P #{$options[:emulator_uri].port} gw/ipfw.sh
      \
  #{$options[:emulator_uri].user}@#{$options[:emulator_uri].
      host}:#{$options[:emulator_uri].path}/")
  run_command_on_host($options[:emulator_uri].user, $options
      [:sshprivkey],
    $options[:emulator_uri].host, $options[:emulator_uri].
        port,
    "#{$options[:emulator_uri].path}/ipfw.sh #{$options[:
        emulator]}")
  puts "====== emulator configured ======"
end
if $options[:sniffer]
  run_command_on_host($options[:emulator_uri].user, $options
      [:sshprivkey],
    $options[:emulator_uri].host, $options[:emulator_uri].
        port, $options[:sniffer])
  puts "====== sniffer started ======"
end
```

C.2 E-Mail Konversation

Die E-Mail Konversation zeigt das Interesse am Kern des Testframeworks für die Weiterentwicklung des *SWTBots*. Die Klasse `RmiSWTWorkbenchBot` möchte ich nachträglich integrieren.

```
From:    sandor.szuecs@fu-berlin.de
To:    KetanPadegaonkar@gmail.com
Date:    17. November 2009 18:25:22 MEZ
Subject:  forked swtbot on github to implement a remote interface

Hi Ketan,

I am one of the developers of dpp-Saros [1] an eclipse plugin that
provides distributed pair and distributed side-by-side programming. I
write my master thesis on that project and have to provide a test-
framework to automate tests for our plugin. The plugin is XMPP-Client
based and we have an architecture that lack of testability. I tested
swtbot, but we need an RMI interface for swtbot. I forked the project
from your github repository to implement what we need.

I think I should delegate SWTWorkbenchBot and all returned Objects to
```

```
an RMI interface. For each class like SWTWorkbenchBot it will result in
one Interface and a delegation class:
- class name: RemoteSWTWorkbenchBot
- interface name: IRemoteSWTWorkbenchBot

Have you an idea for a reasonable package hierarchy?

something like:
org.eclipse.swtbot.eclipse.finder.SWTWorkbenchBot
org.eclipse.swtbot.eclipse.finder.RemoteSWTWorkbenchBot
org.eclipse.swtbot.eclipse.finder.IRemoteSWTWorkbenchBot

or something like:
org.eclipse.swtbot.eclipse.finder.SWTWorkbenchBot
org.eclipse.swtbot.remote.eclipse.finder.RemoteSWTWorkbenchBot
org.eclipse.swtbot.remote.eclipse.finder.IRemoteSWTWorkbenchBot


If you have hints or questions I am glad to read and/or answer these.

If you like the idea, I send you a pull request when it's done.

[1] http://dpp.sourceforge.net/

Regards, Sandor Szuecs
--

From:    sandor.szuecs@fu-berlin.de
To:    ketanpadegaonkar@gmail.com
Date:    18. November 2009 06:39:40 MEZ
Subject:  Re: forked swtbot on github to implement a remote interface

I would say two plugins new plugins that look something like these
names would be good:

org.eclipse.swtbot.swt.finder.rmi
org.eclipse.swtbot.eclipse.finder.rmi

From the description of it, it seems like you plan to extend existing
classes to implement the RMI interface

I'm not too clear with the implementation idea that you are coming up
from what you describe here. Feel free to make a few tiny changes and
ask for comments. I'm surely interested in where this goes :)

Cheers!

--
Ketan
studios.thoughtworks.com | twitter.com/ketanpkr

From:    sandor.szuecs@fu-berlin.de
To:    ketanpadegaonkar@gmail.com
Date:    19. November 2009 11:01:34 MEZ
Subject:  Re: forked swtbot on github to implement a remote interface

On 18.11.2009, at 06:39, Ketan Padegaonkar wrote:

> I would say two plugins new plugins that look something like these
> names would be good:
>
```

```
> org.eclipse.swtbot.swt.finder.rmi
> org.eclipse.swtbot.eclipse.finder.rmi

Ok I will do this.

> From the description of it, it seems like you plan to extend existing
> classes to implement the RMI interface

I tried to extend SWTWorkbenchBot by inheritance and I did not get rid
of compile errors that were not solvable for me.
I ended up using delegation at this point.
The Classes from org.eclipse.swtbot.swt.finder.rmi that are returned by
SWTWorkbenchBot are Serializable now.

The major problem I got has something to do with SWT. A Testapp that
use the SWTBotTestrunner and use RmiSWTWorkbenchBot instead of
SWTWorkbenchBot throws Exceptions that a Widget is null (SWTBotMenu
during call to super() ).
As I understand, Scott Lewis pointed out [1] that SWT Controls are not
possible to RMIfy natively.
I have to ideas to get around:
1) A Proxy pattern that hides SWT Controls and don't Serialize them.
2) Use a different approach then RMI. Maybe "ECF Remote Services" are
the way to go.

Do you have opinions?

> I'm not too clear with the implementation idea that you are coming up
> from what you describe here.

Maybe I should clarify my needs. The Plugin which I want to test is an
XMPP Client. The problem with testing it that we have to control two
or more instances.
simple Example:
'A@jabber.org' writes in a local Sourcefile that 'A' shares with 'B'.
The result must be proofed on B's instance.

I searched the internet for testframeworks that can do this, but there
are no that I can use out of the box. There are plenty frameworks that
can run xUnit Tests on different nodes, but they just run, no s
ynchronization or start,choose,stop,wait mechanisms.

Our architecture is not testable like services or simple unit tests.
That's the basic problem that I want to address.

> Feel free to make a few tiny changes and
> ask for comments. I'm surely interested in where this goes :)


Thanks!
:)

[1] http://eclipsesource.com/blogs/2009/05/06/those-leaky-networks/

Regards, Sandor Szuecs
--

From:    sandor.szuecs@fu-berlin.de
To:    ketanpadegaonkar@gmail.com
Date:    19. November 2009 13:06:50 MEZ
Subject:  Re: forked swtbot on github to implement a remote interface
```

```
How comfortable would you be to use a dynamic language like jruby to achieve
this?

This is a very trivial problem to solve because of something that ruby
provides — method_missing.

Let me know you thoughts, and I could tell you more.

Cheers!

—
Ketan
studios.thoughtworks.com | twitter.com/ketanpkr

From:    sandor.szuecs@fu-berlin.de
To:    ketanpadegaonkar@gmail.com
Date:    19. November 2009 13:17:16 MEZ
Subject:  Re: forked swtbot on github to implement a remote interface

On 19.11.2009, at 13:06, Ketan Padegaonkar wrote:

> How comfortable would you be to use a dynamic language like jruby to
    achieve
> this?

Yeah ruby is so nice and I would really love to do all this in ruby.
:)

> This is a very trivial problem to solve because of something that ruby
> provides — method_missing.

Yes I explained it my colleagues that this would be nice to have this in java
    .

> Let me know you thoughts, and I could tell you more.


It would be really nice if you can tell me more!

Regards, Sandor Szuecs
—

From:    sandor.szuecs@fu-berlin.de
To:    ketanpadegaonkar@gmail.com
Date:    20. November 2009 13:21:24 MEZ
Subject:  Re: forked swtbot on github to implement a remote interface

Hi Sandor,

The ruby scripts are self documented. You'll need to figure out how to embed
    jruby in eclipse but that's trivial to do.

More about embedding it within eclipse on my blog[1]. There's a plugin that
    has the necessary jar to embed jruby within eclipse.

Let me know what you think of this approach :)

[1] - http://ketan.padegaonkar.name/2009/04/10/cucumber-on-jruby-inside-
    eclipse.html
```

—
Ketan
studios.thoughtworks.com | twitter.com/ketanpkr

Anhang abgeschnitten, weil die Idee nicht weiter verfolgt wurde.

From: sandor.szuecs@fu-berlin.de
To: ketanpadegaonkar@gmail.com
Date: 21. Februar 2010 17:46:15 MEZ
Subject: Re: forked swtbot on github to implement a remote interface

Hi Ketan,

On 18.11.2009, at 06:39, Ketan Padegaonkar wrote:

> I would say two plugins new plugins that look something like these
> names would be good:
>
> org.eclipse.swtbot.swt.finder.rmi
> org.eclipse.swtbot.eclipse.finder.rmi
>
> From the description of it, it seems like you plan to extend existing
> classes to implement the RMI interface
>
> I'm not too clear with the implementation idea that you are coming up
> from what you describe here. Feel free to make a few tiny changes and
> ask for comments. I'm surely interested in where this goes :)

there are two videos that show working szenarios with RMIfied
SWTWorkbenchBot. :-)

http://userpage.fu-berlin.de/sszuecs/emulator_video.html
http://userpage.fu-berlin.de/sszuecs/lokal_video.html

The first is showing a fully remote test running on different VMs.
The second is a local testrun.

I think I will have time in march to integrate to swtbot and push it to
my github.

Regards, Sandor Szuecs
—

From: sandor.szuecs@fu-berlin.de
To: ketanpadegaonkar@gmail.com
Date: 21. Februar 2010 22:00:51 MEZ
Subject: Re: forked swtbot on github to implement a remote interface

Quite neat :) Looking forward to see the sources!

Cheers!

— Ketan

From: sandor.szuecs@fu-berlin.de
To: ketanpadegaonkar@gmail.com
Date: 22. Februar 2010 11:53:39 MEZ
Subject: cite email, was Re: forked swtbot on github to implement a remote
interface

```
Can I use our private e-mail for my diploma thesis?
I want to show the relevance of this part of my thesis.
If you like I delete your e-mail address from the cited e-mails.

Regards, Sandor Szuecs
--

From:    sandor.szuecs@fu-berlin.de
To:    ketanpadegaonkar@gmail.com
Date:    22. Februar 2010 12:47:52 MEZ
Subject:  Re: cite email, was Re: forked swtbot on github to implement a
    remote interface

Sure, go ahead with my email address if that helps.

--
Ketan
studios.thoughtworks.com | twitter.com/ketanpkr
```