Freie Universität Berlin

AG Software Engineering
Berlin, 26. Oktober 2009

Einladungsprozess in Saros

(In einem Werkzeug für verteilte Paarprogrammierung)

–

Bachelorarbeit

TAS SÓTI
4136556
tastnt at gmail com
www.tastnt.com

Betreuer: Christopher Oezbek und Stephan Salinger
Eingereicht bei: Prof. Dr. Lutz Prechelt

Zusammenfassung

Der Einladungsprozess in Saros ist der Vorgang, nach dessen Abschluss sich alle Eingeladenen in derselben Saros-Sitzung befinden und bereit sind, am zur gemeinsamen Nutzung ausgewählten Projekt zusammen zu arbeiten. Die wichtigsten Schritte sind das Auswählen des Projektes und der Sitzungsteilnehmer, Kompatibilitätsüberprüfung der Saros-Versionen, Synchronisation und Übertragung der Projektdateien sowie die Beendigung der Einladung. Das Ziel dieser Bachelorarbeit ist, den bereits vorhandenen Einladungsprozess zu verbessern und zu erweitern. Die Arbeit beschreibt den Einladungsprozess an sich, die Technik und die Konzepte; darüber hinaus wird ein kleines Testszenario mit einer geringen Anzahl an Benutzern durchgeführt.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Hausarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe; alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.[1]

Tas Sóti
Berlin, 26. Oktober 2009

[1] http://www.geisteswissenschaften.fu-berlin.de/we04/service/eidesstattliche_erklaerung.pdf

Inhaltsverzeichnis

0 Begriffe und Notationen **1**
0.1 Begriffe . . . 1
0.2 Notationen . . . 1

1 Einführung **1**
1.1 Was ist Paarprogrammierung? . . . 1
1.2 Was ist Saros? . . . 3
1.3 Der Einladungsprozess . . . 4

2 Zielsetzung **4**
2.1 Ausgangssituation . . . 4
2.2 Anforderungen . . . 5

3 Der Einladungsprozess **7**
3.1 Die Schritte im Detail . . . 7
3.2 Technik . . . 11
3.2.1 Smack Pakete . . . 11
3.2.2 XStreamExtensionProvider . . . 12
3.2.3 Collectors vs. Listeners . . . 13
3.2.4 SarosPacketCollector . . . 14
3.2.5 PacketFilters . . . 14
3.2.6 Die Info-Objekte . . . 15
3.3 VersionManager . . . 15
3.3.1 Version-Objekte und deren Vergleich . . . 16
3.3.2 Überprüfung des Versionskonfliktes . . . 17
3.3.3 Transitivität der Kompatibilität . . . 17
3.4 Verfolgung des Fortschrittes . . . 18
3.4.1 OutgoingInvitationProcess . . . 18
3.4.2 IncomingInvitationProcess . . . 20
3.5 Abbrechbarkeit . . . 21
3.5.1 Ein Abbruchsszenario . . . 22
3.5.2 Die zwei Phasen . . . 24
3.5.3 Abbruch über das Netz . . . 25
3.5.4 Kontrollstellen . . . 25
3.6 Nebenläufigkeit . . . 26
3.6.1 Globale Nebenläufigkeit . . . 27
3.6.2 Lokale Nebenläufigkeit . . . 29
3.6.3 Beschränkungen . . . 31

4 Outreach **31**
4.1 Die Mail . . . 33
4.2 Die Übung . . . 33

5 Testszenario **35**
5.1 Zielsetzung . . . 35
5.2 Testumgebung . . . 35
5.3 Ergebnis . . . 36

6 Methodik **37**
6.1 Arbeitsumgebung . . . 37
6.2 Das Team . . . 37
6.3 Softwaretechnische Aspekte . . . 38
6.3.1 Einarbeitung . . . 38
6.3.2 Wissensaustausch . . . 38
6.3.3 Prozesse . . . 39

7 Ausklang **41**

Literatur **43**

A Tests **45**
A.1 Testfälle . . . 45
A.1.1 TestProjekt1 . . . 45
A.1.2 TestProjekt2 . . . 45
A.1.3 Freier Test . . . 46
A.2 Fragebogen . . . 47
A.2.1 Fragen und Antworten zum TestProjekt1 . . . 47
A.2.2 Fragen und Antworten zum TestProjekt2 . . . 47
A.2.3 Allgemeine Fragen . . . 48

B Abbildungsverzeichnis **49**

Abbildungsverzeichnis

1 Saros im Einsatz . . . 3
2 Die Einladungsklassen . . . 7
3 Bestätigungsdialog bei fehlender Sarosunterstützung . . . 7
4 Bestätigungsdialog beim Versionskonflikt . . . 8
5 Anzeige des Versionskonfliktes beim Eingeladenen . . . 8
6 Ablauf der Einladung (die Parameter der Methoden werden nicht dargestellt) 10
7 Hierarchie der Info-Objekte für die Einladung . . . 15
8 Anzeige des Fortschrittes in der Fortschrittsansicht . . . 19
9 Beendete Einladungen in der Fortschrittsansicht . . . 20
10 Beendete Einladung mit Beanchrichtigung in der rechten unteren Ecke . . . 20
11 Die drei neuen Exception-Klassen . . . 21
12 Schaltflächen für den Abbruch . . . 22
13 Abbrechen während der Archivübertragung . . . 23
14 Während der Einladung wird in der Sitzungsansicht der Eingeladene (c_test) entsprechend gekennzeichnet . . . 30
15 Die Anzahl der Downloads in den letzten 12 Monaten von der Projektseite . . . 32
16 Eigenschaften der Internetverbindung . . . 36
17 Package Explorer . . . 49
18 Ein Projekt zur gemeinsamen Nutzung auswählen . . . 49
19 Einladungsassistent . . . 50
20 Fortschrittsansicht . . . 50
21 Benachrichtigung über die Einladung . . . 51
22 Auswahl des Zielprojektes . . . 52

23 Auswahl des Zielprojektes - automatische Suche nach einem ähnlichen Projekt 53
24 Anzeige des Fortschrittes 54
25 Die Einladungen wurden erfolgreich abgeschlossen 54

0 Begriffe und Notationen

0.1 Begriffe

Objekt Ein Java-Object.

Benutzer Der Anwender des Saros-Plugins.

Einladender ist die Person, die die Sitzung eröffnet (*Host*) und die anderen einlädt.

Eingeladene sind Personen, die in eine Sitzung von einem Einladenden eingeladen werden.

Teilnehmer Ein Benutzer, der sich in einer Einladung oder in einer Sitzung befindet.

OIP Die Klasse de.fu_berlin.inf.dpp.invitation.OutgoingInvitationProcess.

IIP Die Klasse de.fu_berlin.inf.dpp.invitation.IncomingInvitationProcess.

DTM Die Klasse de.fu_berlin.inf.dpp.invitation.DataTransferManager.

methode(..) Eine Java-Methode. Die Punkte bedeuten, dass die Parameter nicht aufgelistet werden, weil sie in dem gegebenen Kontext keine wichtige Rolle spielen.

Kontaktliste Der *Roster View* in Eclipse, Abb. 1 (1).

Sitzungsansicht Der *Shared Session View* in Eclipse, Abb. 1 (2).

Einladungsassistent Die Klasse de.fu_berlin.inf.dpp.ui.wizards.JoinSessionWizard bzw. deren Erscheinungsform als Dialogfenster für den Benutzer, Abb. 21, 22, 24.

0.2 Notationen

Neue Begriffe werden immer **fett geschrieben**, wenn sie das erste Mal in der Arbeit vorkommen.

Quellcode bekommt eine spezielle Formatierung, wie diese: beispielMethode(arg1, arg2).

Kursiv werden meistens *englische Begriffe* dargestellt, die sich schwer übersetzen lassen oder wegen des Kontexts auf Englisch geschrieben werden müssen (z.B. Schaltflächen in der Benutzeroberfläche).

1 Einführung

1.1 Was ist Paarprogrammierung?

Extremprogrammierung, oder auf Englisch *eXtreme Programming* (XP), ist eine agile Softwareentwicklungsmethode, die im Mittelpunkt die Kundenzufriedenheit und damit auch die sich ständig ändernden Anforderungen hat. Dabei wird viel Wert auf Teamarbeit, Kommunikation, Einfachheit und *Releases*, deren Auslieferung innerhalb von kurzen Zeitabschnitten erfolgen soll, gelegt. [XP]

> "The most surprising aspect of ExtremeProgramming is its simple rules. Extreme Programming is a lot like a jig saw puzzle. There are many small pieces. Individually the pieces make no sense, but when combined together a complete picture can be seen. The rules may seem awkward and perhaps even naive at first, but are based on sound values and principles.[2]"

Ein sehr wichtiges Konzept beim XP ist die **Paarprogrammierung**. Das heißt: zwei Entwickler sitzen an einem Rechner mit einer Tastatur und einer Maus und arbeiten gemeinsam an dem gleichen Code. Es gibt eine klar definierte Rollenverteilung, die sich in bestimmten Zeitabständen ändert:

Driver: der Entwickler, der gerade die Kontrolle (Maus und Tastatur) in der Hand hat und den Code schreibt.

Beobachter: der Entwickler, der dem Driver zuschaut, seine Arbeit verfolgt, mitdenkt und Vorschläge macht bzw. ihn auf Fehler hinweist.

Obwohl Studienergebnisse wie [AB] zeigen, dass Paarprogrammierung auch einige Schwierigkeiten birgt, zum Beispiel beim Vorliegen inkompatibler Persönlichkeiten oder unterschiedlicher Fachkenntnisse, so denkt die Mehrzahl der Entwickler, dass diese Praktik gut funktioniert und zur Verbesserung der Codequalität führt. Cockburn et al. untersützten diese Aussagen [AC] und sehen die wichtigsten Vorteile der Paarprogrammierung u.a. im geringeren Auftreten von Fehlern und Defekten, dem besseren Entwurf und Wissenstransfer und dem erhöhten Aufkommen an Freude während der Arbeit.
Das professionelle Interesse an der Paarprogrammierung ist seit der Erfindung und dem Erfolg von XP deutlich gestiegen [PB02]. Durch die Globalisierung und dem ständig zunehmenden wirtschaftlichen Druck gibt es immer mehr Unternehmen, die einen Teil der Programmierarbeit outsourcen. Und als Folge dieser Entwicklung entsteht gesteigerter Bedarf an verteilter Paarprogrammierung über das Internet. **Verteilte Paarprogrammierung** (oder auf English *Distributed Pair Programming* (DPP)) ist eine abgewandelte Variante der konventionellen Paarprogrammierung, bei der sich die Entwickler an zwei verschiedenen Orten befinden. Williams et al. haben untersucht, ob Paarprogrammierung ihre Effektivität beibehält, wenn die Paare nicht direkt nebeneinander sitzen. Die Untersuchungen zeigen, dass verteilte Paarprogrammierung eine brauchbare Art objektorientierter Softwareentwicklung darstellt und im Vergleich zur regulären Paarprogrammierung keine signifikanten Nachteile aufweist [PB02].
Die Wahl des Werkzeugs für Paarprogrammierung kann einen großen Einfluss auf die Effizienz und Effektivität der Entwicklung haben. Während ein *Screen-sharing* Programm wie RealVNC[3] sicherstellt, dass beide Entwickler genau wissen, was beim Partner am Bildschirm passiert (inklusive Mauszeiger bzw. Arbeitsplatz (*Desktop*)), bietet ein über das Internet synchronisierter Texteditor wie SubEthaEdit[4] mehr Interaktionsmöglichkeiten während der Programmiersitzung. Winkler et al. haben eine generische Vorgehensweise zur systematischen Evaluierung von Werkzeugen in einem definierten Kontext vorgestellt [DW], die bei der Suche eines Werkzeuges hilfreich sein kann.

[2] http://www.extremeprogramming.org/
[3] http://www.realvnc.com/
[4] http://www.codingmonkeys.de/subethaedit/

1.2 Was ist Saros?

Saros ist ein Werkzeug für verteile Paarprogrammierung. Es ist ein Plugin[5] für Eclipse[6], das die Möglichkeit eröffnet, ein Eclipse-Projekt über das Internet zu nutzen. Ein Host öffnet hierzu eine Sitzung und lädt andere Entwickler zum gemeinsamen Programmieren oder Begutachten von Code ein. Die erste Version von Saros ist durch die Diplomarbeit *Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierung verteilter Paarprogrammierung* [Dje06] von Riad Djemili entstanden. Mit dieser Version war es möglich zu zweit an einer **Saros-Sitzung** teilzunehmen und Dateien gemeinsam zu bearbeiten. Die Rollen Driver und Beobachter waren bereits vorhanden.

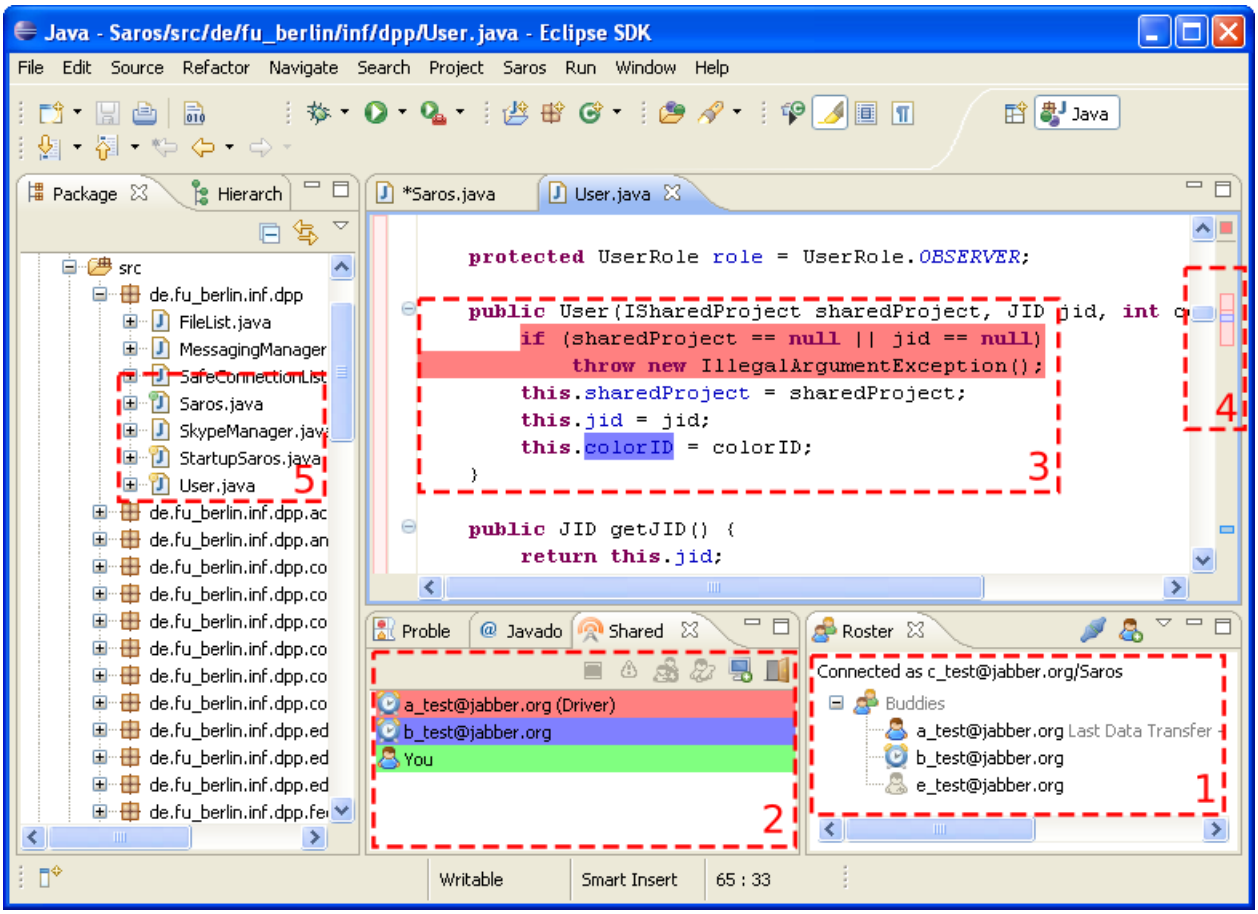

Abbildung 1: Saros im Einsatz

Die Abb. 1 stellt eine Sitzung aus der Sicht von *c_test* mit drei Teilnehmern dar (2). In der Kontaktliste (1) werden die Benutzer aufgelistet, die man in eine Saros-Sitzung einladen kann (falls sie auch mittels Saros angemeldet sind). *a_test* hat die Driver-Rechte, er kann also die Dateien bearbeiten. Im Editor können alle Benutzer Markierungen durchführen, die dann den anderen mit der entsprechen Farbe angezeigt werden (3). Der rosa Balken auf der rechten Seite zeigt den Sichtbereich des Drivers. Damit der Benutzer weiß, welche Dateien der Driver gerade auf hat, werden sie links im *Package Explorer* mit kleinen Kreisen markiert.
Die Entwicklung des Werkzeuges wurde im Rahmen einer Studienarbeit [Gus07] von Björn Gustavs fortgesetzt. Danach folgten die Arbeiten [Rie08], [Jac09], [Rin09a], [Doh09]. Zum Zeitpunkt des Schreibens sind noch die Arbeiten [Ros09], [Szu09], [Zil09], [Sta09b], [Sta09a],

[5] http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html
[6] http://eclipse.org/

[Kun09], [Lau09], [Rin09b] im Gange. Weitere Informationen zu Saros sind unter http://dpp.sourceforge.net/ zu finden.

1.3 Der Einladungsprozess

Der Einladungsprozess in Saros ist der Vorgang, nach dessen Abschluss sich alle Eingeladenen in derselben Saros-Sitzung befinden und bereit sind, an dem zur gemeinsamen Nutzung ausgewählten Projekt zusammen zu arbeiten. Im Folgenden wird ein Erfolgsszenario dargestellt und zwar aus der Sicht der Benutzer: Einladender und Eingeladene. Diese Form der Darstellung soll bloß einen kleinen Überblick verschaffen. Weitere Details sind im Abschnitt 3.1 zu finden.

1. Der Einladende wählt im **Package Explorer** (Abb. 17) von Eclipse ein Projekt zur gemeinsamen Nutzung aus und klickt auf *Share project...* im Rechts-Klick-Menü (Abb. 18).

2. Der **Einladungsassistent** (Abb. 19) erscheint, in welchem er die einzuladenden Personen auswählt und auf die Schaltfläche *Finish* klickt. Die Einladungen werden in der **Fortschrittsansicht** von Eclipse (Abb. 20) angezeigt. So kann der Einladende den Stand der Einladung verfolgen.

3. Die Eingeladenen bekommen eine Benachrichtigung über die Einladung. Sie können die Einladung akzeptieren oder ablehnen (Abb. 21).

4. Nachdem der Eingeladene *Next* geklickt hat, kann er entscheiden, ob er ein ganz neues oder bereits vorhandenes Projekt benutzen will (Abb. 22). Wenn er die Entscheidung gefällt hat, wird vom Eingeladenen das Projekt übertragen. Es erscheint eine Anzeige über den Stand der Übertragung (Abb. 24).

5. Wenn die Übertragung beendet ist, verschwindet der Einladungsassistent. Die Einladung ist abgeschlossen und der Eingeladene befindet sich in der Sitzung.

6. Der Einladende wird über die Beendigung der Einladung in der Fortschrittsansicht (Abb. 25) informiert .

2 Zielsetzung

2.1 Ausgangssituation

Der Einladungsprozess ist ein wichtiger Teil von Saros: funktioniert er nicht fehlerfrei, kann keine Sitzung aufgebaut werden und die ganze Applikation ist so nutzlos. Vor meiner Bachelorarbeit war bereits eine Implementierung des Einladungsprozesses vorhanden, aber sie hatte viele Schwachstellen, deren Behebung diese Arbeit zum Ziel hat. Ich gebe eine kurze Beschreibung des alten Einladungsprozesses und weise an jeder Stelle auf die Schwachtellen hin:

- Der ganze Ablauf basierte auf dem Konzept eines Zustandsautomaten und die Zustandsübergänge wurden technisch mittels *Listener* 3.2.3 verwirklicht. Die Zustände waren in Form eines Aufzählungstyps[7] vorhanden und der aktuelle Zustand änderte sich bei jedem Nachrichtenaustausch zwischen dem Einladenden und Eingeladenen.

[7] http://java.sun.com/javase/6/docs/api/java/lang/Enum.html

- Die Anfrage/Antwort-Architektur, die grundsätzlich mit *Collectors* 3.2.3 zu implementieren ist, wurde in ein technisches Konstrukt von *Listener* und Aufzählungstypen gezwungen. Das hatte an vielen Stellen Wettlaufsituationen (*Race Conditions*) zum Ergebnis. Um diese zu vermeiden waren an einigen Stellen Codeabschnitte wie

```
// HACK We need to sleep here, because if there are no files to
// wait for, we could finish the blockUntil... too fast.
try {
    Thread.sleep(3000);
}
```

Quellcode 1: Fehleranfällige Stelle im Code des alten Einladungsprozesses

  vorhanden.

- Abbrechbarkeit war nur sehr beschränkt möglich und die Dateiübertragung konnte gar nicht abgebrochen werden. Sie lief nach dem Abbrechen der Einladung im Hintergrund weiter.
- Der Fortschritt der Dateiübertragung wurde nicht angezeigt. Sonstige Fortschrittsanzeigen basierten auf den Zuständen. Monitore wurden nicht konsequent benutzt.
- Der Einladungsdialog war für die Einladung der Benutzer zuständig. Er hat auch den Fortschritt der Einladungen angezeigt, basierend auf den Zuständen. Das war eine technisch unsaubere Lösung und hat immer wieder zu Problemen geführt.
- Der Einladungsdialog war modal, so konnte der Einladender während der Einladung in Eclipse nicht weiterarbeiten.

Dieser Zustand musste verbessert werden. Die genauen Ziele sind im nächsten Abschnitt beschrieben.

2.2 Anforderungen

Einen großen Teil der Anforderungen habe ich aus den bisherigen Arbeiten extrahiert. Darüber hinaus wurden noch sowohl die Anforderungsdokumente[8] als auch der *Bugtracker*[9] berücksichtigt. Weiterhin habe ich der Liste neue Anforderungen hinzugefügt, die während meiner Bachelorarbeit entstanden sind. Ich strebe hier Vollständigkeit an, damit im Fall einer Weiterentwicklung alle Anforderungen des Einladungsprozesses an dieser Stelle zu finden sind.
Anforderungen, die ich selber implementiert oder deren Implementierung ich zumindest geändert habe, werden **fett** gekennzeichnet. Anforderungen, die noch zu erfüllen sind, werden unterstrichen.

1. **Der Einladende muss einen Benutzer aus der Kontaktliste aus einladen können.**
2. Ein Sitzungsteilnehmer muss die Sitzung verlassen können.
3. Wenn jemand die Sitzung verlässt, müssen darüber alle Sitzungsteilnehmer informiert werden.
4. Wenn der Einladende die Sitzung verlässt, müssen alle Sitzungsteilnehmer informiert und aus der Sitzung entfernt werden.

[8] https://svn.mi.fu-berlin.de/agse/sci/dpp/requirements
[9] http://sourceforge.net/tracker/?group_id=167540

5. Der Einladende muss einen Sitzungsteilnehmer aus der Sitzung entfernen können.
6. **Versionskonflikt 3.3.2: im Fall von unterschiedlichen Saros-Versionen muss sowohl der Einladende als auch der Eingeladene benachrichtigt werden.**
7. Auswahl des Zielprojektes beim Eingeladenen:
   (a) Der Eingeladene muss die Auswahl haben, ein neues Projekt zu erstellen oder ein bereits vorhandenes zu benutzen.
   (b) Es muss möglich sein, alle Projekte (den ganzen Arbeitsbereich) automatisch zu durchsuchen und das Projekt mit der höchsten Übereinstimmung als Zielprojekt zu benutzen. Die Stufe der Übereinstimmung muss dem Benutzer angezeigt werden. Das Durchsuchen muss abbrechbar sein.
   (c) Im Fall der Benutzung eines bereits vorhandenen Projektes als Zielprojekt muss anhand des Projektnamens eine automatische Vorentscheidung getroffen werden.[10]
8. **Abbrechbarkeit 3.5:**
   (a) **Der Einladende muss die Einladung in jedem Zustand abbrechen können.**
   (b) **Der Eingeladene muss die Einladung in jedem Zustand abbrechen können.**
   (c) **Bei einem technischen Fehler – egal ob auf Seite des Einladenden oder eines Eingeladenen – muss die Einladung abgebrochen werden.**
   (d) **Beim Abbruch der Einladung müssen die Beteiligten (Einladender und/oder Eingeladene) entsprechend informiert werden.**
   (e) Beim Abbruch der Einladung muss das Projekt beim Eingeladenen entweder gelöscht oder dessen Originalzustand wiederhergestellt werden.
9. **Fortschritt 3.4: sowohl der Einladende als auch der Eingeladene müssen den Fortschritt der Einladung verfolgen können: Fortschrittsbalken und eine kurze Beschreibung des Standes.**
10. **Nebenläufigkeit 3.6:**
    (a) **Der Einladende muss mehrere Benutzer gleichzeitig einladen können.**
    (b) **Der Einladende muss während der Einladung an dem Projekt weiterarbeiten können.**
    (c) Der Eingeladene muss während der Einladung in Eclipse weiterarbeiten können (nicht unbedingt an dem Projekt, das gerade übertragen wird).
    (d) **Sitzungsteilnehmer müssen während der Einladung von weiteren Teilnehmern am Projekt weiterarbeiten können.**
    (e) **Wenn die Einladung abgeschlossen ist, muss beim Eingeladenen alles konsistent sein.**
11. **Einladungs-ID: jede Einladung muss eine eindeutige Einladungs-ID haben.**
12. **Teilnehmerliste: das Hinzufügen eines Nutzers muss über alle bisherigen Sitzungsteilnehmer synchronisiert werden.**
13. **Einfachheit: die Implementierung muss konzeptuell einfach zu verstehen sein.**

[10] http://sourceforge.net/tracker/?func=detail&aid=2841965&group_id=167540&atid=843362

14. **Archive: Dateien müssen immer als Archiv versendet werden.**
15. **Das *Auto-Building* von Eclipse muss während des Erstellung des Projektes auf der Eingeladenenseite ausgeschaltet werden.**
16. **Die Erstellung des Projektes auf der Eingeladenenseite muss als atomare *Workspace-ModifyOperation* ausgeführt werden.**
17. Der Einladende muss das Projekt teilweise *sharen* können[11].
18. Wenn ein Sitzungsteilnehmer zu einer anderen Sitzung eingeladen wird, muss das bei ihm angezeigt werden. Er muss entscheiden können, ob er die Sitzung wechseln will.

3 Der Einladungsprozess

Der Einladungsprozess besteht aus zwei Hauptklassen: OutgoingInvitationProcess und Incoming−InvitationProcess. Die erste Klasse sorgt für den Ablauf an der Einladenderseite, die andere übernimmt die Kontrolle beim Eingeladenen.

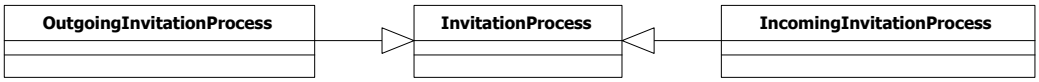

Abbildung 2: Die Einladungsklassen

Im folgenden Abschnitt werden alle wichtigen Schritte der Einladung beschrieben. Das Unterkapitel 3.1 beantwortet die Frage, *was* alles während der Einladung passiert. Auf die Frage *wie* geben die nächsten Unterkapitel Antwort. Die folgenden Schritte finden statt, nachdem der Einladende die Einladung einer Person initiiert hat.

3.1 Die Schritte im Detail

1. checkAvailability(): in der Kontaktliste werden alle Kontakte des eingeloggten Benutzers aufgelistet. Sollte ein Kontakt mit einem regulären Messenger-Klienten unterwegs sein, muss der Einladende informiert werden, dass die Klientsoftware der einzuladenden Person Saros nicht unterstützt (Abb. 3).

Abbildung 3: Bestätigungsdialog bei fehlender Sarosunterstützung

[11] https://svn.mi.fu-berlin.de/agse/sci/dpp/requirements/Requirements_from_Teles.doc

2. getSupportingPresence: die in Schritt 1 beschriebene Überprüfung wird vom DiscoveryManager erledigt.

3. RQ–JID: die sogenannte **Resource Qualified Jabber ID** liegt genau dann vor, wenn der Eingeladene über Saros verfügt. Sollte der DiscoveryManager doch ein falsches Ergebnis liefern, kann der Einladende den Hinweis außer acht lassen und mit der Einladung fortfahren. (Wenn der Eingeladene tatsächlich kein Saros hat, wird die Einladung weder akzeptiert noch abgelehnt, der Einladende muss die Einladung also selber abbrechen.)

4. checkVersion: um Konflikte zwischen inkompatiblen Saros-Versionen zu vermeiden, findet vor der Einladung eine Versionsüberprüfung 3.3.2 statt.

5. determineCompatibility: der VersionManager 3.3 übernimmt diese Aufgabe und fragt den Eingeladenen nach seiner Sarosversion.

6. versionInfo: das Ergebnis der Versionsüberprüfung wird dem Einladenden nur dann vorgelegt, wenn Probleme aufgetreten sind (Abb. 4).

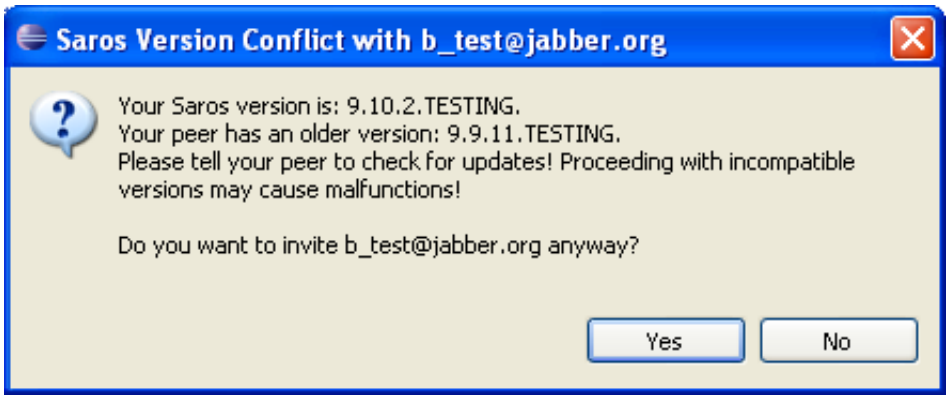

Abbildung 4: Bestätigungsdialog beim Versionskonflikt

7. sendInvitation: erst jetzt fängt die tatsächliche Einladung an.

8. invitationInfo: die Einladungsnachricht mit den nötigen Informationen (InvitationInfo 7) wird verschickt.

9. determineVersion: der Einladende trifft auch die Entscheidung über die Versionskompatibilität und benachrichtigt ggf. den Benutzer (Abb. 5).

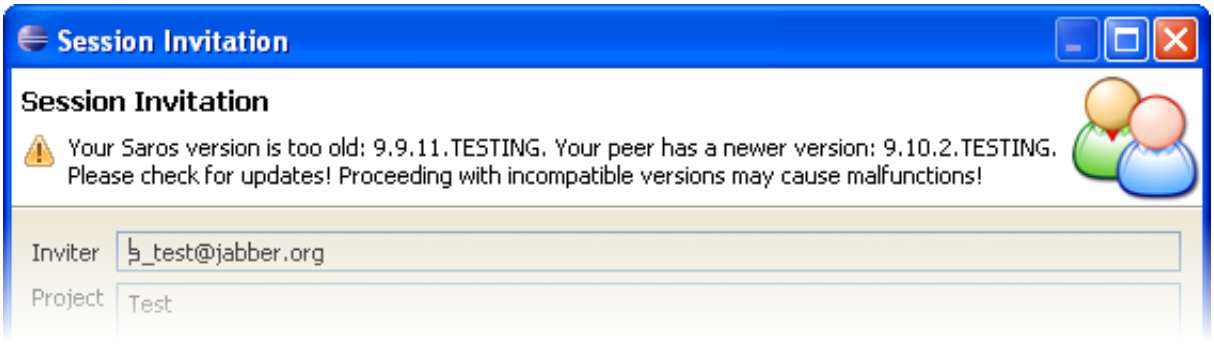

Abbildung 5: Anzeige des Versionskonfliktes beim Eingeladenen

10. getNextPage: wird aufgerufen, wenn der Eingeladene im Einladungsassistenten auf die Schaltfläche *Next* klickt (s. auch 1.3).

11. requestRemoteFileList: es wird eine Anfrage an den Einladenden geschickt, damit er die Fileliste des Projektes zusendet.

12. fileListRequest: diese Anfrage wird empfangen und verarbeitet.

13. getFileListDiff : schickt die Fileliste des Projektes dem Eingeladenen zu

14. Die nächste Seite des Einladungsassistenten wird angezeigt, wo man das Zielprojekt auswählen kann (s. auch 1.3). Dafür gibt es zwei Möglichkeiten: entweder wird ein neues Projekt erstellt oder der Benutzer sucht sich ein bereits vorhandenes Projekt aus, das Teile vom ankommenden Projekt des Einladenden enthält (das kann auch mittels *Scan Workspace* (Abb. 23) automatisch passieren). In diesem Fall werden die übereinstimmenden Dateien nicht übertragen und das kann die Einladung deutlich beschleunigen.

15. **return**: es erscheint eine Anzeige über den Stand der Übertragung (s. auch 1.3).

16. performFinish: wird aufgerufen, wenn der Eingeladene im Einladungsassistenten auf die Schaltfläche *Finish* klickt (s. auch 1.3).

17. accept: die Einladung wurde vom Benutzer akzeptiert, daher müssen die folgenden Schritte durchgeführt werden.

18. accepUnsafe: eine Wrappermethode.

19. createNewProject: ein lokales Eclipse-Projekt wird angelegt.

20. handleDiff: anhand des ausgewählten Zielprojektes und der Fileliste des Einladenden wird eine Fileliste erstellt, die alle Dateien enthält, die nicht im lokalen Zielprojekt vorhanden oder auf der Fileliste des Einladenden nicht zu finden sind.

21. fileListDiff : diese Liste wird dem Einladenden geschickt.

22. createArchive: der Einladende muss nun diese Dateien in ein Archiv packen.

23. stop: damit ein Projektarchiv erstellt werden kann, dürfen kurzfristig keine Änderungen am Projekt vorgenommen werden. Es kann natürlich sein, dass bereits mehrere Benutzer an dem Projekt arbeiten. Sie müssen alle angehalten werden. Das ist die Aufgabe des StopManagers. Weitere Informationen dazu finden Sie im Kapitel über Nebenläufigkeit 3.6).

24. allUsersStopped: der StopManager bestätigt, dass alle Sitzungsteilnehmer angehalten wurden. Jetzt kann das Projektarchiv erstellt werden. Diese Anhaltphase wird auch benutzt, um den Eingeladenen zur Sitzung hinzuzufügen (s. auch 3.6).

25. synchronizeUserList: da nun die Sitzung einen neuen Teilnehmer hat, müssen drüber auch die vorherigen Sitzungsteilnehmer informiert werden.

26. start: die Teilnehmer können nun ihre Arbeit am Projekt während der Übertragung des Archivs fortsetzen. (Das passiert nicht direkt durch den StopManager, wurde aber der Verständlichkeit halber so dargestellt. Weitere Details dazu im Kapitel 3.6).

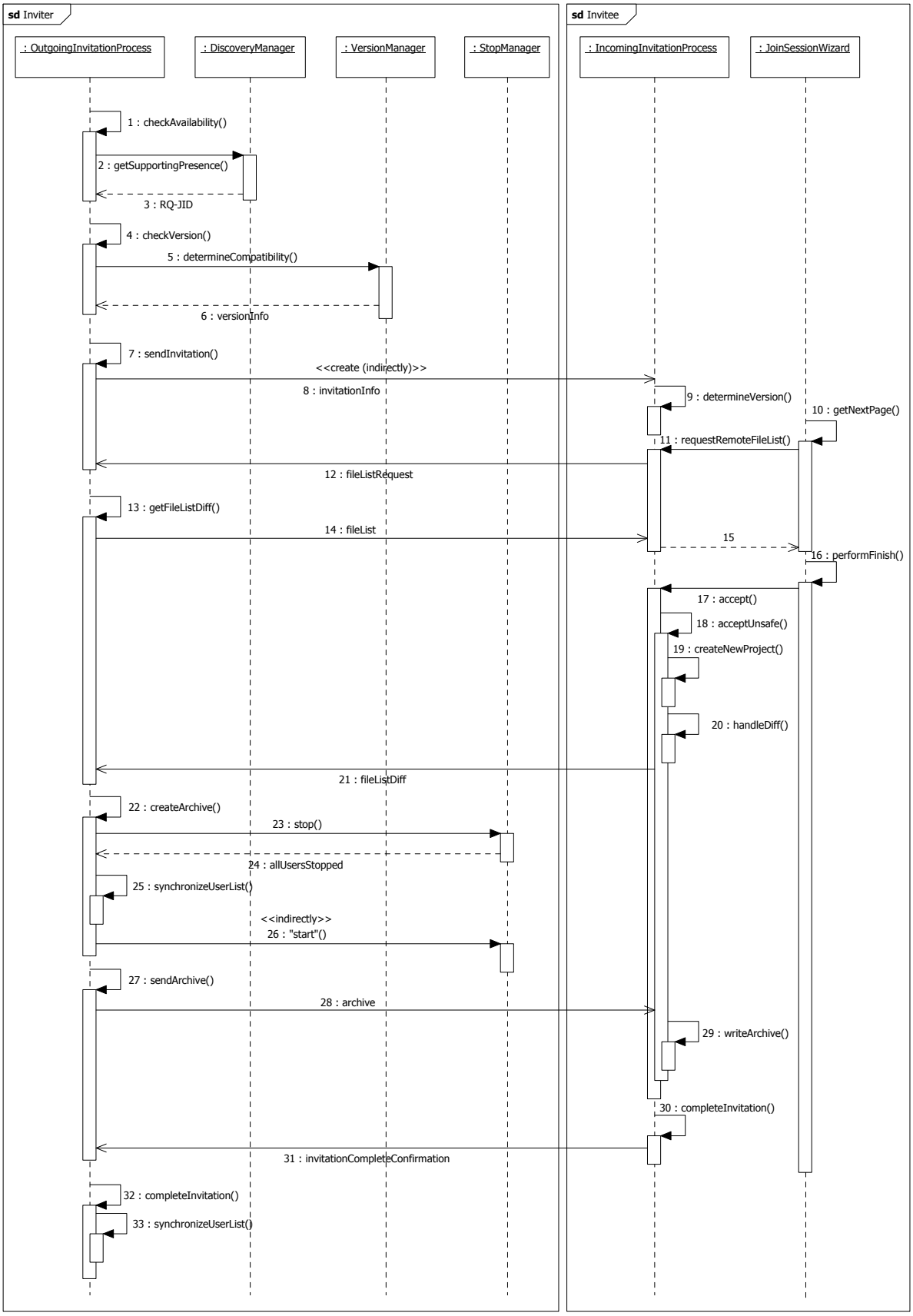

Abbildung 6: Ablauf der Einladung (die Parameter der Methoden werden nicht dargestellt)

27. sendArchive: das Archiv wird übertragen.

28. archive: das Archiv.

29. writeArchive: das angekommene Archiv wird entpackt und auf die Festplatte geschrieben.

30. completeInvitation: der Eingeladene führt die letzten Operationen zur Beendigung der Einladung durch.

31. invitationCompleteConfirmation: die Bestätigung darüber, dass die Einladung abgeschlossen wurde, wird dem Einladenden verschickt.

32. completeInvitation: Bestätigungsnachricht

33. synchronizeUserList: der Einladende informiert alle Sitzungsteilnehmer, dass der Eingeladene die Einladung abgeschlossen hat.

3.2 Technik

Der folgende Abschnitt gibt einen kurzen Überblick über Smack und die Benutzung der Klasse XStreamExtensionProvider[12]. Vollständigkeit wird hier nicht angestrebt. Das Ziel ist es, Basiswissen zu schaffen, anhand dessen die Arbeit (hauptsächlich der Quellcode) verständlich wird.

3.2.1 Smack Pakete

Smack ist [Sma] ist eine Java-Implementierung des XMPP[13]-Protokolls, die in Saros eingesetzt wird. Sie ermöglicht einen einfachen Verbindungsaufbau und die Kommunikation unter Benutzern. Die Kommunikation besteht darin, bestimmte Java-Objekte zu verschicken. Diese Objekte müssen in sogenannte **Pakete** eingebunden werden. Ein Paket ist ein Exemplar einer Klasse, die die abstrakte Packet-Klasse von Smack erweitert. Smack stellt grundsätzlich drei Klassen zur Verfügung, die Packet erweitern:

Message [14] Für das Verschicken von Nachrichten.

IQ [15] Für einfache Anfrage/Antwort-Kommunikation. Die IQ-Klasse ist abstrakt und kann daher nicht instanziiert werden

Presence [16] Für den Austausch der Anwesenheitsinformationen (online, offline usw.).

Die wichtigste Klasse für uns ist Message. Sie kann ohne Weiteres wie folgt benutzt werden:

```
Message myMessage = new Message();
myMessage.setBody("Hello Saros!");
myMessage.setProperty("myLocation", "Berlin");
```

Quellcode 2: Zusammenstellung einer Beispielnachricht

[12] de.fu_berlin.inf.dpp.net.internal
[13] http://xmpp.org/
[14] org.jivesoftware.smack.packet.Message
[15] org.jivesoftware.smack.packet.IQ
[16] org.jivesoftware.smack.packet.Presence

Die Informationen können aus der Nachricht auf die gleiche Art extrahiert werden. Das ist ein einfacher Fall für das Versenden von Informationen. Damit das Zusammenstellen von Paketen flexibel bleibt, bietet Smack eine ***Provider*architektur**. Diese ermöglicht individuelles *Parsen* von ***Extensions***. Zu einer Nachricht (Message) kann eine sogenannte *Extension* hinzugefügt werden, die die zu verschickenden Informationen enthält. Die dafür vorgesehene Methode ist addExtension(PacketExtension extension). Eine PacketExtension ist von Smack wie folgt definiert:

> "A packet extension is an XML subdocument with a root element name and namespace. Packet extensions are used to provide extended functionality beyond what is in the base XMPP specification.[17]"

Die Frage ist jetzt, wie man ein Exemplar von PacketExtension erstellt, das unser Objekt enthält, was wir verschicken möchten. Das passiert mittels eines ***Provider***s, der ein dafür vorgesehenes Konstrukt ist. Jeder *Provider* muss die Schnittstelle PacketExtensionProvider[18] implementieren. In unserem Fall tut das die Klasse XStreamExtensionProvider. Sie ist zuständig dafür, ein beliebiges Objekt entgegenzunehmen und es in eine PacketExtension zu kapseln.

3.2.2 XStreamExtensionProvider

Um ein Objekt über das Netwerk verschicken zu können, muss es in eine dafür geeignete Form gebracht werden. Diesen Vorgang nennt man **Serialisierung**. Für dieses Ziel wird die Bibliothek XStream [XSt] verwendet. Sie ermöglicht die Serialisierung beliebiger Java-Objekte.
Aus der Sicht des Einladungsprozesses ist bloß wichtig, wie man ein Objekt von einem zum anderen Sitzungsteilnehmer verschickt. Aus diesem Grund wird die interne Funktionsweise von XStream nicht weiter erläutert. Die Nutzung von XStream wird in der Klasse XStreamExtensionProvider gekapselt. Diese Klasse ermöglicht das Erstellen von PacketExtension s, die ein Objekt (myObject) enthalten, das wir verschicken möchten. Für jede Klasse, deren Objekte wir verschicken möchten, muss ein entsprechender *Provider* erstellt werden:

```
public static class InvitationExtensionProvider extends
    XStreamExtensionProvider<MyObject> {
    public InvitationExtensionProvider() {
        super("myObject", MyObject.class);
    }
}
```

Quellcode 3: Erstellung eines *Providers*

Nun liefert die create (..) -Methode dieses *Providers* die erwünschte *Extension*.

1. Erstellung einer PacketExtension, die myObject enthält

```
MyObject myObject = new MyObject(..);
XStreamPacketExtension<MyObject> myObjectExtension =
    myObjectExtensionProvider.create(myObject);
```

2. Extrahieren des myObjectes aus der Extension

```
MyObject myObject = myInfoExtensionProvider.getPayload(myObjectExtension);
```

Es ist wichtig, dass in beiden Fällen dasselbe Exemplar von XStreamExtensionProvider<MyObject > benutzt wird, sonst kann die Methode getPayload(..) zu unerwarteten Fehlern führen.

[17] org.jivesoftware.smack.provider.PacketExtension
[18] org.jivesoftware.smack.provider.PacketExtensionProvider

3.2.3 Collectors vs. Listeners

In den vorherigen zwei Kapiteln 3.2.1, 3.2.2 wurde erläutert, wie man Pakete zusammenstellt und gegebenfalls mit beliebigen Objekten füllt. Die nächste Frage ist: wie schickt man ein Paket an einen anderen Benutzer? (*In den folgenden Beispielen wird davon ausgegangen, dass bereits ein Exemplar **connection** von **XMPPConnection** vorliegt. Mehr Informationen dazu sind in der Dokumentation von Smack*[19] *zu finden).* Mit der setTo(String to) Methode kann definiert werden, an wen das Paket adressiert ist. Danach ruft man bloß sendPacket(Packet paket) auf und das Paket wird verschickt.

```
Message msg = new Message();
msg.setTo("b_test@jabber.org");
connection.sendPacket(msg);
```

Quellcode 4: Verschicken von Paketen (a_test@jabber.org)

Das versendete Paket muss beim Empfänger auch abgearbeitet werden. Smack bietet zwei Möglichkeiten dafür[20]:

PacketListener [21] Eine Klasse für Ereignis-basierten Paketaustausch.

```
PacketFilter myFilter = new FromContainsFilter("a_test@jabber.org");

PacketListener myListener = new PacketListener() {
  public void processPacket(Packet packet) {
    // Bearbeiten des ankommenden Pakets
  }
};
/* Listener registrieren. (Mit der Annahme, dass wir eine
 * XMPPConnection "connection" bereits erzeugt haben.)
 */
connection.addPacketListener(myListener, myFilter);
```

Quellcode 5: Benutzung von PacketListeners (b_test@jabber.org)

Diese Art von Versenden von Paketen wird meistens benutzt, wenn der Zeitpunkt des Empfangens eines Pakets nicht fest ist, zum Beispiel bei Einladungspaketen. Ein Benutzer, der eingeloggt ist, weiß nicht, wann genau eine Einladung von einem anderen Benutzer initiiert wird, deswegen wartet er durchgehend mittels eines Listeners auf ein Einladungspaket. Wenn das Paket eintrifft, wird die processPacket-Methode automatisch aufgerufen. Beim Registrieren eines Listeners muss ein sogenannter **Filter** angegeben werden. Das Filterkonzept wird im Kapitel 3.2.5 beschrieben.

PacketCollector [22] Eine Klasse für Anfrage/Antwort-basierten Paketaustausch.

```
PacketFilter myFilter = new FromContainsFilter("a_test@jabber.org");

/* Listener registrieren. (Mit der Annahme, dass wir eine
 * XMPPConnection "connection" bereits erzeugt haben.)
 */
PacketCollector myCollector = connection.createPacketCollector(myFilter);

//Timeout in Millisekunden.
long timeout = 500;
Packet result = myCollector.nextResult(timeout);
```

[19] http://www.igniterealtime.org/builds/smack/docs/latest/documentation/gettingstarted.html
[20] http://www.igniterealtime.org/builds/smack/docs/latest/documentation/processing.html
[21] org.jivesoftware.smack.PacketListener
[22] org.jivesoftware.smack.PacketCollector

Im Gegensatz zu einem Listener wird beim Ankommen eines Pakets, auf das ein Collector wartet, kein Ereignis ausgelöst. Ein Collector muss manuell gefragt werden, ob ein Paket angekommen ist. Das erledigt die Methode nextResult(**long** timeout), die solange blockiert, bis ein Paket da ist oder die angegebene Zeit (timeout) abgelaufen ist. Dieser Mechanismus wird eingesetzt, wenn die Kommunikation zwischen zwei Teilnehmern einen strikt sequentiellen Anfrage/Antwort-Ablauf hat. Bei den Schritten der Einladung 3.1 ist das genau der Fall.

3.2.4 SarosPacketCollector

Die *Listener* und *Collectors* werden standardmäßig auf einer XMPPConnection[23] registriert 3.2.3. Wenn die Verbindung aus irgendwelchem Grund abbricht, werden sie alle entfernt und müssten theoretisch alle neu registriert werden. Um dies zu vermeiden gibt es die Klasse XMPPChatReceiver, die über eine Liste aller *Listener* und *Collectors* verfügt. Sie selber registriert einen *Listener* auf der XMPPConnection und wenn ein Paket ankommt, benachrichtigt sie alle *Listener* und *Collectors* der Liste. Sollte die Verbindung abbrechen, muss bloß der einzige Listener des XMPPChatReceivers neu registriert werden.
Manchmal muss "simuliert" werden, dass ein Paket von einem anderen Benutzer ankommt (z.B. im DataTransferManager; weitere Informationen werden eventuell in der Arbeit [Szu09] zu finden sein.). "Simulieren" heißt nichts anderes als der manuelle Aufruf der Methode processPacket(myPacket) mit dem erwünschten Paket myPacket, damit alle *Listener* und *Collectors* benachrichtigt werden. Smack bietet grundsätzlich keine Möglichkeit dazu, weil er die Methode automatisch aufruft, wenn ein Paket ankommt und alle Filter passiert 3.2.5. Die Klasse SarosPacketCollector[24] ist nichts anderes als eine Kopie der Klasse PacketCollector von Smack. Der Unterschied ist einfach, dass dieser *Collector* nicht auf der XMPPConnection registriert wird, sondern er wird in die Liste des XMPPChatReceivers eingetragen (um so einen *Collector* zu erhalten ruft man XMPPChatReceiver.createCollector(..) auf). Möchte man das Ankommen eines Pakets "simulieren", ruft man XMPPChatReceiver.processPacket(myPacket) mit dem entsprechenden Paket auf.

3.2.5 PacketFilters

Beim Verschicken von Paketen 3.2.3 wird definiert, wer das Paket bekommen sollte. Damit weiß der XMPP-Server, wem er das Paket liefern muss. Wenn das Paket beim Adressaten ankommt, muss er entscheiden können, was er mit dem Paket macht. Erstmal muss er wissen, ob er das Paket überhaupt braucht und wenn ja, wer (welche Methode) das Paket weiter verarbeiten sollte . Zu diesem Zweck dienen die **PacketFilters**[25] von Smack.
Das Konzept ist einfach. Jeder PacketFilter muss die Methode **boolean** accept(Packet packet) implementieren. Gibt die Methode **true** zurück, wird das Paket angenommen, sonst wird es verworfen. Diese Überprüfung findet auf der Smack-Ebene statt. Passiert ein Paket die angegegebenen Filter nicht, wird weder processPacket (..) aufgerufen noch liefert nextResult (..) das Paket zurück (s. 3.2.3). Damit muss man sich bloß um einen richtig definierten Filter kümmern und Smack erledigt den Rest der Filterung.
Beim Einladungsprozess werden drei verschiedene Filter eingesetzt:

InvitationIDFilter Bei jeder Einladung wird auf Seite des Einladenden mittels Random.nextLong ()[26] eine zufällige Zahl generiert. Sie stellt sicher (mit sehr hoher Wahrscheinlichkeit),

[23] http://www.igniterealtime.org/builds/smack/docs/latest/documentation/gettingstarted.html
[24] de.fu_berlin.inf.dpp.net.internal
[25] org.jivesoftware.smack.filter.PacketFilter
[26] java.util.Random

dass der Nachrichtenaustausch während der Einladung nur zwischen dem Einladenden und Eingeladenen stattfindet und keine weitere Pakete empfangen werden, die den Einladungsprozess stören könnten. Die Einladungs-ID wird das erste Mal mit dem Einladungspaket (InvitationInfo 7) verschickt. Ab diesem Zeitpunkt wird sie noch solange benutzt, bis die Einladung abgeschlossen ist.

SessionIDFilter Diese ID bezieht sich auf eine Sitzung und wird beim Starten der Sitzung der Einladungs-ID ähnlich vergeben. Sie wird auch das erste Mal mit dem Einladungspaket TODO:Referenz. (InvitationInfo 7) verschickt, aber die Pakete müssen diese ID auch nach dem Abschluß der Einladung weiterhin enthalten, solange die Sitzung erhalten bleibt. Bei Beendung der Sitzung wird diese ID gelöscht.

PacketExtensionFilter Die vorherigen zwei Filter stellen sicher, dass wir nur aus der richtigen Einladung und Sitzung Pakete bekommen. Jetzt muss die Art der Pakete festgestellt werden. Ist das eine Versionsanfrage? Eine Abbruchsaufforderung? Oder möchte der Einladende gerade das Projektarchiv schicken? Diese Fragen beantwortet der PacketExtensionFilter, den man einfach vom entsprechenden PacketExtensionProvider 3.2.1 bekommen kann. Die Methode lautet: getPacketFilter().

3.2.6 Die Info-Objekte

Wie wir bei den Filtern gesehen haben, wird immer übeprüft, ob die ankommende Nachricht die entsprechenden IDs enthält. Grundsätzlich sollte jede Nachricht bzw. deren *Extension*, die innerhalb einer Sitzung verschickt wird, mindenstens über eine SitzungsID verfügen. Aus diesem Grund gibt es die Basisklasse DefaultSessionInfo, die immer als Grundlage benutzt werden soll, wenn neue Nachrichtentypen definiert werden. Bei der Einladung wird auch noch eine EinladungsID benötigt. Für diesen Zweck gibt es die Klasse DefaultInvitationInfo wie in der Abb. 7 zu sehen ist.

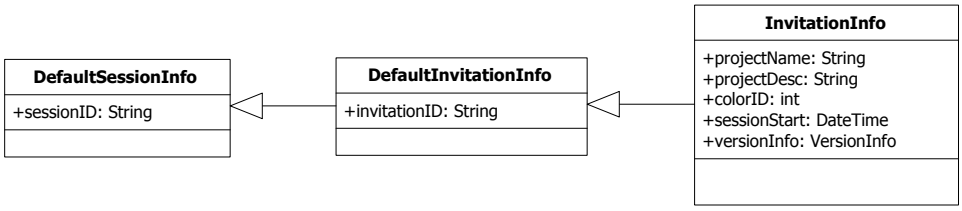

Abbildung 7: Hierarchie der Info-Objekte für die Einladung

Im Fall der Einladungsnachricht erweitern wir diese Klasse. Damit wird sichergestellt, dass die zwei IDs immer mitgeschickt werden. Nun wird ein Exemplar dieser Klasse (InvitationInfo) dem XStreamExtensionProvider übergeben und in die *Extension* gepackt.

3.3 VersionManager

Während der TELES-Sitzungen [Ros09] trat öfter mal der Fall auf, dass bei falscher Funktionsweise von Saros die zwei Sitzungsteilnehmer erst nach einiger Zeit bemerkten, worin der Fehler lag: sie benutzten zwei verschiedene, nicht kompatible Versionen von Saros. Folglich entstand der Bedarf für eine Versionsverwaltungskomponente, deren Rolle der VersionManager eingenommen hat.

3.3.1 Version-Objekte und deren Vergleich

Die aktuelle Version von Saros wird in der MANIFEST.MF-Datei[27] festgehalten. Der relevante Ausschnitt sieht zur Zeit wie folgt aus:

```
Bundle-Name: Saros Plug-in
Bundle-SymbolicName: de.fu_berlin.inf.dpp;singleton:=true
Bundle-Version: 9.10.2.DEVEL
Bundle-Activator: de.fu_berlin.inf.dpp.Saros
```

Quellcode 6: Versionsinformationen in der MANIFEST.MF-Datei

Mittels VersionManager.getVersion() kann diese Bundle-Version in Form eines Version-Objektes abgefragt werden. Nun kann das Vergleichen zweier solcher Version-Objekte folgende Ergebnisse liefern:

OK Die zwei Versionen sind miteinander kompatibel.

TOO_OLD Die Bezugsversion ist zu alt, um mit der anderen Version zusammenarbeiten zu können.

TOO_NEW Die Bezugsversion ist zu neu, um mit der anderen Version zusammenarbeiten zu können.

Der Vergleich kann auf zwei verschiedene Arten erfolgen:

Kompatibilitätstabelle: die wird im VersionManager definiert und heißt compatibilityChart. Diese Tabelle enthält die Kompatibilitätseinträge. Ein Eintrag besteht aus einem Version -Objekt und der Liste der zu dieser Version kompatiblen Version-Objekte. Bei jedem Release muss in die Tabelle die Release-Version eingetragen werden. Falls eine Abwärtskompatibilität zu älteren Versionen besteht, gehören diese Versionen ebenfalls in die Liste. Im folgenden Beispiel hier soll Release-Version 9.10.2 abwärtskompatibel zu 9.9.11 sein:

```
/**
 * Version 9.10.2
 */
compatibilityChart.put(new Version("9.10.2"), Arrays.asList(new Version("
    9.10.2"), new Version("9.9.11")));
```

Direkter Vergleich: der basiert auf dem Vergleich (compareTo) von Version-Objekten (s. org.osgi.framework.Version). Grundsätzlich ist es in Saros so, dass eine neuere Version "größer" als eine ältere bezüglich (compareTo)ist.

Beim Vergleich schaut der VersionManager nach, ob in der Kompatibilitätstabelle für die zu vergleichenden Versionen ein entsprechender Eintrag existiert. Sollte dies der Fall sein, sind die Versionen kompatibel, er gibt also OK zurück. Wenn kein Eintrag vorhanden ist, werden die Version-Objekte direkt verglichen. Dieser Vergleich liefert nur in dem Fall OK zurück, wenn die Versionen gleich sind. Das sollte aber vermieden werden: die Kompatibilitätstabelle muss über jede Release-Version einen Eintrag enthalten, bei dem mindestens die Release-Version selbst in der Liste der kompatiblen Versionen vorhanden ist.

[27] http://www.javaworld.com/javaworld/jw-03-2008/jw-03-osgi1.html

3.3.2 Überprüfung des Versionskonfliktes

Der Einladungsprozess 1.3 muss sicherstellen, dass sowohl der Einladende als auch der Eingeladene benachrichtigt werden, falls ein Versionskonflikt vorliegt. Ein Versionskonflikt besteht genau dann, wenn die Vergleichsfunktion determineCompatibility(Version localVersion, Version remoteVersion) vom VersionManager nicht OK liefert.
Die Schwierigkeit besteht darin, dass beide Teilnehmer über eine eigene Kompatibilitätstabelle 3.3.1 verfügen und falls die Saros-Versionen der Teilnehmer nicht gleich sind, sind die Kompatibilitätstabellen auch unterschiedlich. Ob ein Versionskonflikt vorliegt, kann lediglich derjenige Teilnehmer entscheiden, der die neuere Version hat – die alte Version von Saros weiß nämlich nicht, ob die neuere Version rückwärtskompatibel ist. Aus diesem Grund reicht es nicht aus, auf beiden Seiten einen lokalen Versionsvergleich mit der entfernten Version auszuführen; es muss auch ein Austausch des Ergebnisses vom Kompatibilitätsvergleich stattfinden. Dafür wird im VersionManager das folgende Objekt definiert:

```
* Data Object for sending version information
*/
public static class VersionInfo {
  public Version version;
  public Compatibility compatibility;
}
```

Quellcode 7: VersionInfo-Objekt für den Austausch von Versionsinformationen

Die Überprüfung eines Versionskonfliktes läuft bei der Einladung wie folgt ab:

1. Der Einladende schickt eine Versionsanfrage an den Eingeladenen mit einem VersionInfo -Objekt: version enthält die lokale Version und compatibility ist **null**, weil wir noch keine Vergleiche ausgeführt haben.

2. Der Eingeladene überprüft die Kombatibilität basierend auf seiner Kompatibilitätstabelle und schickt eine Antwort, also ein VersionInfo-Objekt mit diesem Vergleichsergebnis (compatibility) und der lokalen Version (version).

3. Der Einladende überprüft, wer die neuere Version hat. Falls er selbst diese haben sollte, führt er einen lokalen Kompatibilitätsvergleich anhand der empfangenen Version (version) aus, sonst übernimmt er die empfangene Kompatibilitätsinformation (compatibility) – das ist die endgültige Entscheidung. Sollte jetzt ein Versionskonflikt vorliegen, wird der lokale Benutzer benachrichtigt und er kann über den weiteren Ablauf entscheiden.

   Entscheidet er sich gegen die Fortsetzung, wird die Einladung abgebrochen. Sonst verschickt er eine Einladung, die unter anderem ein VersionInfo-Objekt mit dem Vergleichsergebnis (compatibility) und der lokalen Version (version)enthält.

4. Der Eingeladene bekommt die Einladung und muss über den Versionskonflikt eine endgültige Entscheidung treffen. Der Einladende hat aber bereits diese Entscheidung gefällt, diese kann nun also einfach übernommen werden. Sollte die ankommende „Entscheidung“ (compatibility) **null** sein (der Einladender konnte die Kompatibilitätsüberprüfung nicht ausführen), muss doch ein lokaler Vergleich stattfinden, der nun als endgültig zählt. Dieses Ergebnis wird dem Benutzer auch angezeigt.

3.3.3 Transitivität der Kompatibilität

Die oben beschriebene Versionsüberprüfung läuft zwischen dem Einladenden und dem Eingeladenen ab. Aber was passiert, wenn mindestens zwei Personen eingeladen werden, oder

wenn eine Person bereits in der Sitzung ist? Die Tatsache, dass die Version des Einladenden mit allen Eingeladenen kompatibel ist, bedeutet noch nicht, dass die Eingeladenen auch unter sich kompatible Versionen haben (die Kompatibilität ist also nicht transitiv.) Damit dies sichergestellt wird, sollte der Einladende eine sitzungsbezogene Versionstabelle erstellen und die Versionsüberprüfung über alle Teilnehmer durchführen. So könnte der Benutzer über die Gefahr von Kompatibilitätsproblemen benachrichtigt werden.

3.4 Verfolgung des Fortschrittes

Während der Einladung müssen mehrere Schritte 3.1 durchgeführt werden. Einige werden schnell erledigt, wie zum Beispiel die Versionsüberprüfung 3.3.2, aber andere brauchen eventuell längere Zeit, wie das Verschicken des Projektarchivs. Die Benutzer (sowohl der Einladende als auch der Eingeladene) müssen über den Fortschritt entsprechend informiert werden, um sichergehen zu können, den Prozess unter Kontrolle zu haben. Die wichtigsten Zeiger sind:

- der aktuelle Stand und
- die Restdauer.

Da dieses Problem bei vielen Anwendungen auftaucht, gibt es in Eclipse ein allgemeines Konzept dafür: **Monitore**. Monitore sind einfache Objekte, die den Fortschritt einer langlaufenden Operation respräsentieren. Sie haben eine direkte Verbindung sowohl zur Operation als auch zur Benutzeroberfläche. Sie sind also ein Kommunikationskanal zwischen diesen zwei Schichten: die Operation kann dem Monitor ständig berichten, wie weit sie ist und der Benutzer kann dadurch den Fortschritt sehen, bzw. eventuell den Vorgang abbrechen 3.5. Weitere Informationen und technische Details zur Benutzung von Monitoren findet man in der Eclipse Dokumentation[28] [Ecl].

3.4.1 OutgoingInvitationProcess

Es gibt also zwei wichtige Teilnehmer beim Monitorkonzept: die Operation, die die Arbeit erledigt und die Benutzeroberfläche, die den Fortschritt anzeigt. Zunächst wird auf die Operationen eingegangen.
Bei der ausgehenden Einladung wird während des ganzen Prozesses ein einziger Monitor benutzt. Der Einladungsprozess erhält diesen Monitor beim Aufruf der Methode start(SubMonitor monitor). Der Monitor hat insgesamt 100 Arbeitseinheiten und sie werden wie folgt aufgeteilt:

```
checkAvailability(monitor.newChild(1));
checkVersion(monitor.newChild(1));
sendInvitation(monitor.newChild(1));
getFileListDiff(monitor.newChild(1));
createArchive(monitor.newChild(3));
sendArchive(monitor.newChild(90));
completeInvitation(monitor.newChild(3));
```

Quellcode 8: Aufteilung der Arbeitseinheiten in OIP.start(SubMonitor monitor)

Wie zu erkennen ist, bekommt das Verschicken des Archivs 90% der Arbeitseinheiten. Das ist damit zu erklären, dass die Übertragung der Projektdateien die meiste Zeit beansprucht.

[28] http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/workbench_jobs.htm

Die anderen Schritte verschicken bloß kleine Pakete oder führen kleinere Operationen lokal durch. Im Fall eines sehr kleinen Projektes wäre diese Aufteilung nicht ganz richtig, aber wenn es so wenig Dateien zu übertragen gibt, verläuft der ganze Prozess relativ schnell und deswegen hat das keine negative Auswirkung auf die Verfolgung des Fortschrittes.
Wie bereits erwähnt, ist die wichtigste Idee bei den Monitoren, dass sie von der Benutzeroberfläche bis zu der tatsächlichen Operation, die durchgeführt werden muss, eine Brücke bilden. Der Einladungsprozess an sich (in diesem Fall ist die Klasse OutgoingInvitationProcess gemeint) hat keine langlaufenden Operationen. Er reicht also den Monitor in der Schicht eine Ebene tiefer, z.B. ist beim Erzeugen des Archivs die Klasse FileZipper diejenige, die die Arbeit macht. Deswegen muss auch sie dem Monitor den Fortschritt mitteilen. Eine wichtigere Rolle spielt der DataTransferManager, der für die Fortschrittsrückmeldung beim Verschicken des Archivs verantwortlich ist. Die Arbeit vom Sándor Szücs [Szu09] befasst sich unter anderem mit diesem Thema.

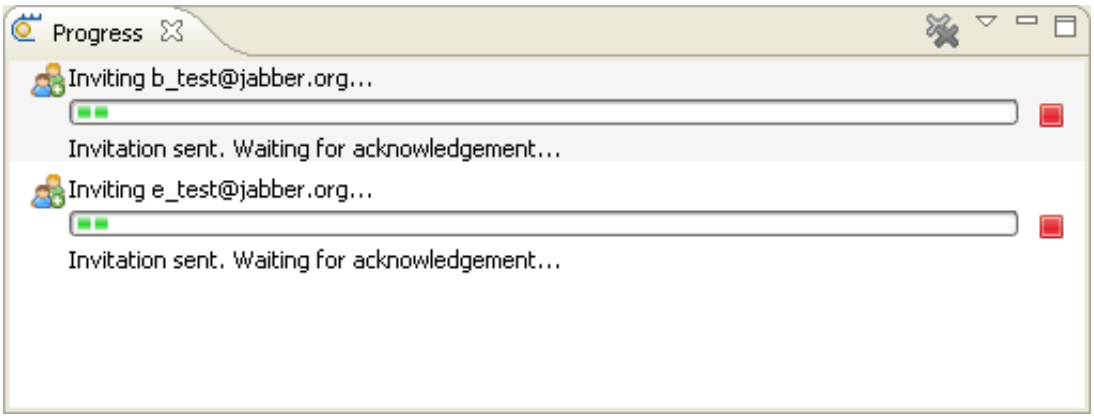

Abbildung 8: Anzeige des Fortschrittes in der Fortschrittsansicht

Nun kommen wir zum anderen Teilnehmer des Monitorkonzeptes: die Benutzeroberfläche. Seit Eclipse 3.0 gibt es die Jobs API [29] [30], die es ermöglicht, mehrere Operationen nebenläufig durchzuführen, ohne den ganzen Arbeitsbereich von Eclipse zu sperren. Jede langlaufende Operation, wie zum Beispiel die Einladung eines Benutzers, muss in ein Exemplar der Klasse Job[31] "eingebettet" werden. Wenn dieser Job gestartet wird, erscheint er in der Fortschrittsansicht (*Progress View*) in Eclipse. Es wird ein Fortschrittsbalken angezeigt sowie der aktuelle Stand des Monitors in Textform, wie in der Abb. 8 zu sehen ist.
Folgende Gründe sprechen für den Einsatz der Jobs API:

- Die oben gesetzten Ziele werden erreicht: sowohl der aktuelle Stand als auch die Restdauer sind erkennbar bzw. abschätzbar.
- Eine hohe Eclipse-Integration wird angestrebt: die Eclipse-Benutzer sind mit der Fortschrittsansicht vertraut. Das macht für sie die Benutzung von Saros intuitiver.
- Die API bietet eine technisch saubere Lösung.

Sollten mehrere Einladungen nebenläufig gestartet werden, werden sie ohne Priorisierung in der Fortschrittsansicht aufgelistet und abgearbeitet. Wenn eine Einladung fertig ist, wird

[29] http://www.eclipse.org/articles/Article-Concurrency/jobs-api.html
[30] http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/workbench_jobs.htm
[31] http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/api/org/eclipse/core/runtime/jobs/Job.html

sie entsprechend gekennzeichnet (s. Abb. 9), verschwindet aber nicht aus der Ansicht. Dadurch wird sichergestellt, dass der Benutzer bei Gelegenheit nachschauen kann, was mit seiner Einladung passiert ist. Bei Beendigung einer Einladung, egal ob erfolgreich oder er-

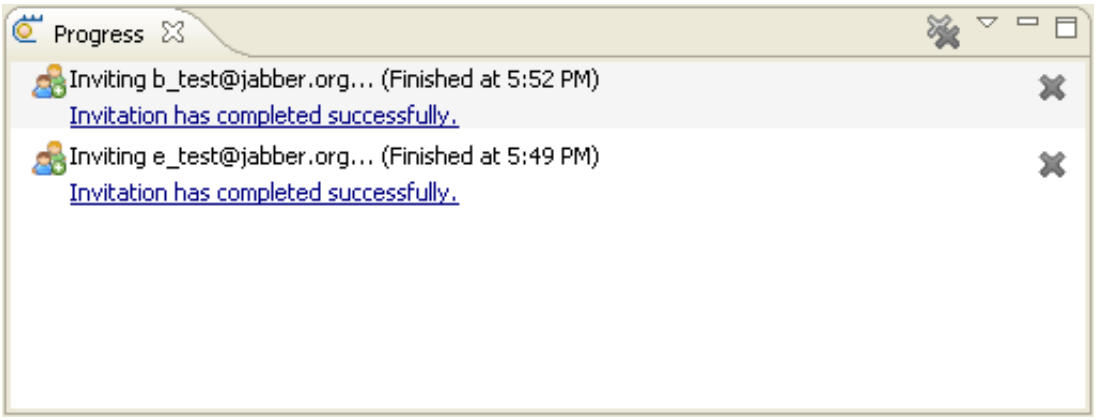

Abbildung 9: Beendete Einladungen in der Fortschrittsansicht

folglos, springt kein Dialog auf, der den Benutzer informiert, weil das seine Arbeit unterbrechen könnte. Anstatt dessen erscheint in dem sogenannten Fortschrittsbereich (*Workbench Progress Area*) in der rechten unteren Ecke von Eclipse ein dezenter Hinweis, dass der Job erledigt wurde. Wird dieses Zeichen angeklickt, öffnet sich ein Dialog und zeigt das Ergebnis der Einladung an. Dieses Zeichen wird unabhängig von der Fortschrittsansicht (*Progress*

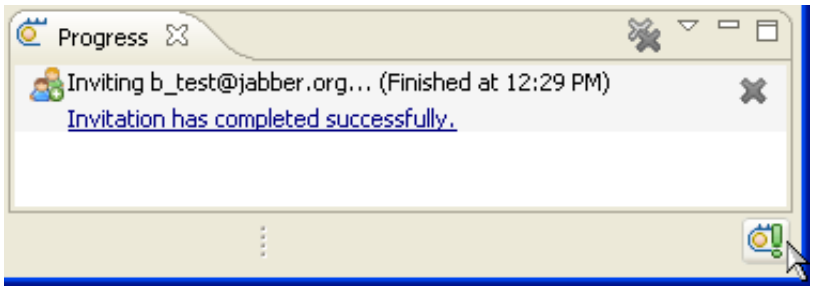

Abbildung 10: Beendete Einladung mit Beanchrichtigung in der rechten unteren Ecke

View) angezeigt. Selbst wenn die Ansicht geschlossen ist (im Gegensatz zur Abb. 10), erscheint dieser Hinweis in der rechten unteren Ecke.

3.4.2 IncomingInvitationProcess

Die Einladung wird auf der Seite des Eingeladenen mittels eines Assitenten (*Wizard*[32]) durchgeführt. Ein Assistent bietet die Möglichkeit an, einen Monitor in den unteren Bereich einer Assistentenseite[33] zu integrieren, wie in Abb. 24 zu sehen ist. Es gibt zwei Schritte 1.3 bei der Einladung, der Benutzer muss also einmal *Next* und danach *Finish* klicken. Aus diesem Grund wird der Prozess mit zwei Monitoren verfolgt. Der erste Monitor zeigt den Fortschritt des Dateilistenaustauschs an und wird dem Einladungsprozess bei der Methode requestRemoteFileList(SubMonitor monitor) übergeben.
Der zweite Monitor spielt eine wichtigere Rolle, weil er das Empfangen des Projektarchivs darstellt. Den bekommt der Einladungsprozess beim accept (..)), also wenn der Eingeladene

[32] http://help.eclipse.org/galileo/topic/org.eclipse.platform.doc.isv/guide/dialogs_wizards.htm
[33] org.eclipse.jface.wizard.WizardPage

das Zielprojekt ausgewählt hat und den *Finish*-Knopf betätigt. Aus den beim OutgoingInvitation −Process bereits erwähnten Gründen werden dem Empfangen des Archivs 90% der Arbeitseinheiten zugeordnet.
Die graphische Anzeige des Fortschrittes beinhaltet sowohl den Fortschrittsbalken als auch den Stand in Textform. Die Ziele sind also erreicht: der Benutzer wird über den Fortschritt ausreichend informiert.

3.5 Abbrechbarkeit

Neben dem Anzeigen des Fortschrittes 3.4 ist die Abbrechbarkeit einer langlaufenden Operation ein sehr wichtiger Teil der Benutzerfreundlichkeit. Diese zwei Konzepte spielen zusammen: sieht der Benutzer, dass ihm die Einladung doch zu lange dauern würde, bricht er sie ab. Das klingt simpel und plausibel. Die Frage ist: was passiert auf der anderen Seite? Damit sind wir beim Kern des Problems angekommen und zwar dass es sich um ein verteiltes System handelt. Das Abbrechen wird dadurch komplexer: beide Teilnehmer können das Abbrechen initiieren und das muss auf beiden Seiten zu einem frühzeitigen Abschluß der Einladung führen. Die verschiedenen Möglichkeiten aus der Sicht eines Teilnehmers sind:

1. Lokaler Abbruch auf Benutzeranfrage
2. Lokaler Abbruch wegen eines technischen Fehlers
3. Fernabbruch auf Benutzeranfrage
4. Fernabbruch wegen eines technischen Fehlers

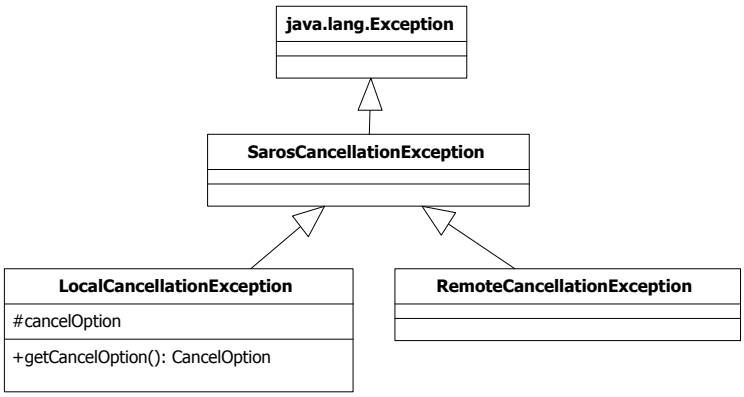

Abbildung 11: Die drei neuen Exception-Klassen

Diese Fälle müssen auf eine technische Lösung in Java abgebildet werden und als kanonische Lösung bieten sich die Exceptions an. Es werden drei neue definiert und eine bereits vorhandene Exception von Java benutzt:

LocalCancellationException wird im Fall 1. geworfen

IOException wird im Fall 2. geworfen

RemoteCancellationException wird in Fällen 3. und 4. geworfen

SarosCancellationException wird geworfen, wenn der Typ der Exception keine Rolle spielt, weil der Grund des Abbruches (cancellationCause) bereits feststeht

Wie oben erwähnt, kann eine Exception entweder durch einen technischen Fehler oder durch Abbruch vom Benutzer ausgelöst werden. Monitore dienen nicht nur zur Verfolgung des Fortschrittes 3.4, sie haben auch ein Attribut, das speichert, ob sie abgebrochen wurden oder nicht. Da sowohl die Benutzeroberfläche als auch die langlaufende Operation Zugriff auf den Monitor haben, kann der Monitor eine Abbruchsanforderung vom Benutzer an die Operation weiterreichen[34]. Die Benutzeroberfläche bietet meistens eine einfache Schaltfläche (s. Abb. 12), um den Abbruchswunsch entgegenzunehmen.

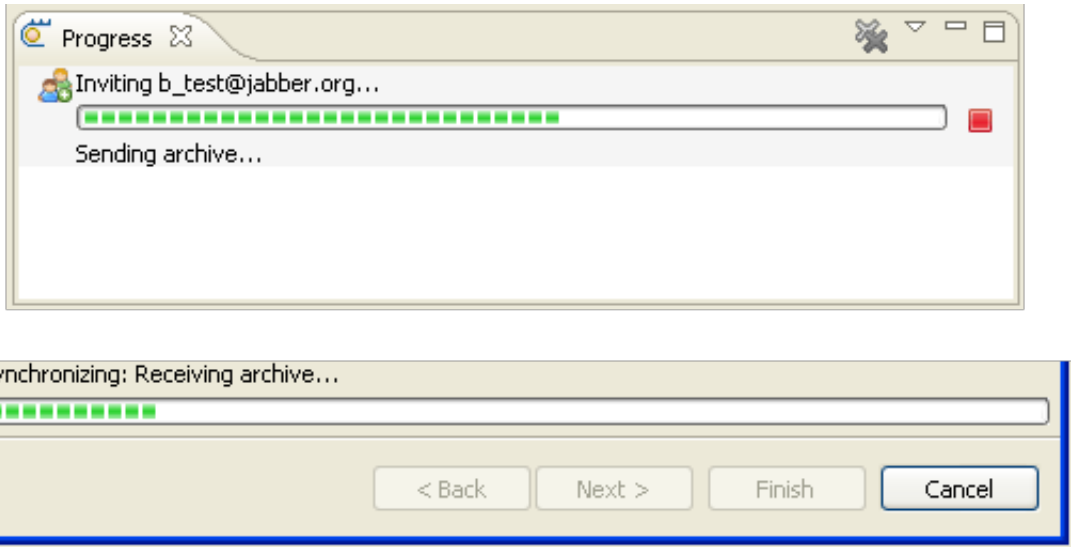

Abbildung 12: Schaltflächen für den Abbruch: rotes Rechteck für den Einladenden (oben), *Cancel*-Schaltfläche für den Eingeladenen (unten)

Die langlaufende Operation ist verantwortlich dafür, den Monitor in bestimmten Zeit- bzw. Arbeitsabständen nach dem Abbruchsstatus zu fragen. Sollte die Methode monitor.isCanceled () **true** zurückliefern, muss die Operation beendet und eine LocalCancellationException geworfen werden. Wird in einer Methode eine Exception geworfen, bricht der normale Ablauf des Programms ab und die Exception wandert im Aufrufstapel (Call Stack) hoch, bis sie abgefangen wird [Jav]. So lautet das Grundkonzept von Exceptions in Java[35]. Eine Abbruchexception hat zwei Folgen: erstens muss die Einladung lokal abgebrochen werden und zweitens muss (eventuell) der andere Teilnehmer über den Abbruch informiert werden (damit auch er einen Abbruch durchführt).

3.5.1 Ein Abbruchsszenario

Nehmen wir ein Beispiel (s. Abb. 13): während der Übertragung des Archivs bricht der Einladende die Einladung ab. Dies wird der Netzwerkschicht über den Monitor (1) bekannt gegeben. Sie bricht die Übertragung ab und räumt eventuell die nicht mehr benötigten Pakete auf. Sie muss weiterhin eine Exception werfen, um die höheren Schichten über den Abbruch zu informieren (2). Diese gelangt zur Einladungsklasse (in diesem Fall OutgoingInvitationProcess), die entscheidet, in welche Kategorie 3.5 der Abbruch gehört (rotes Dreieck links), führt alle weiteren Operationen durch, die zum Abschluss der Einladung benötigt werden (3) (unter anderem die Benachrichtigung des anderen Teilnehmers

[34] http://help.eclipse.org/galileo/topic/org.eclipse.platform.doc.isv/guide/runtime_jobs_progress.htm
[35] http://java.sun.com/docs/books/tutorial/essential/exceptions/definition.html

(5)) und wirft eine Exception an die Benutzeroberfläche weiter (4), die dann letztendlich den lokalen Benutzer über den Abbruch informiert. Jetzt betrachten wir den Abbruch von

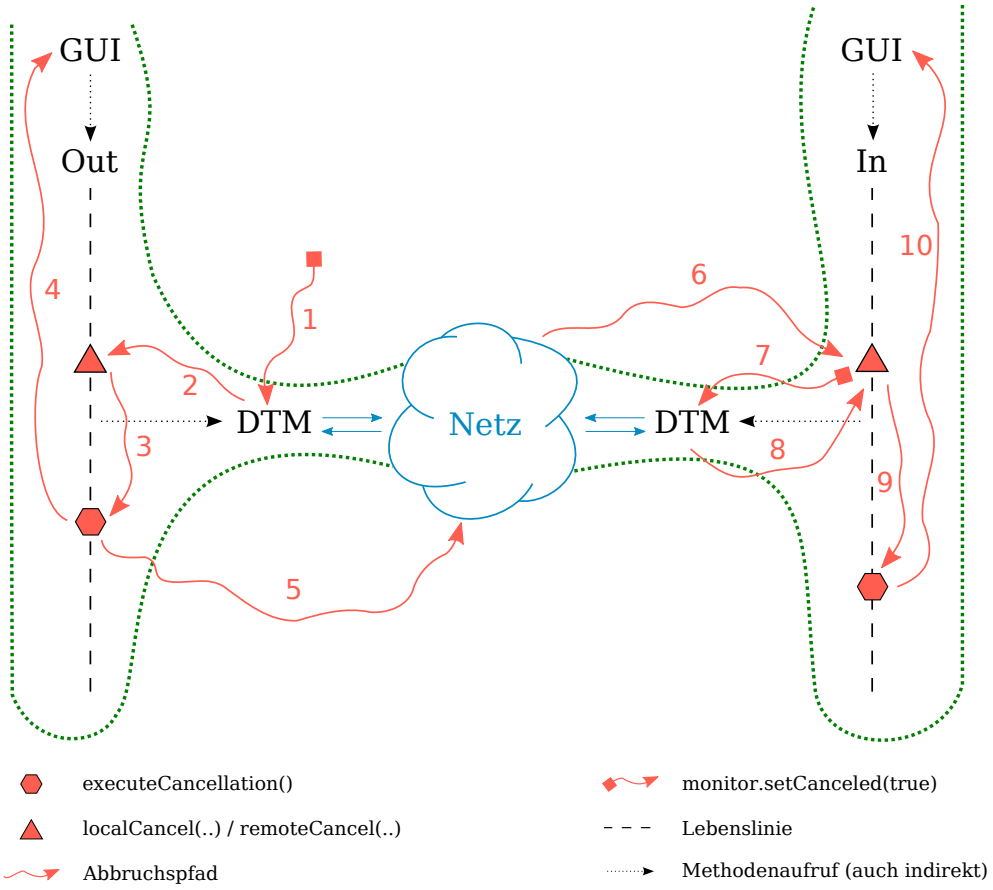

Abbildung 13: Abbrechen während der Archivübertragung, DTM - DataTransferManager, Out - OutgoingInvitationProcess, In - IncomingInvitationProcess

der anderen Seite (IncomingInvitationProcess). Hier kommt die Abbruchsnachricht an (6), die den Einladungsprozess zum Abbrechen auffordert. Er muss die Netzwerkschicht informieren, dies kann aber nur über den Monitor passieren. Er setzt also den Status des Monitors mittels monitor.setCanceled(**true**) auf "abgebrochen" (7). Die Netzwerkschicht kann anhand des Monitors nicht zwischen den verschiedenen Abbruchstypen 3.5 unterscheiden, sie nimmt an, dass der Monitor immer durch den Benutzer abgebrochen wird, wirft also eine LocalCancellationException (8). Das würde zu einem falschen Ergebnis führen, was die Benachrichtigung des Benutzers angeht. Die Lösung besteht darin, zwei Attribute für die Einladungsprozesse (OutgoingInvitationProcess und IncomingInvitationProcess) einzuführen:

AtomicBoolean cancelled besagt, ob die Einladung bereits abgebrochen wurde und darf während einer Einladung nur einmal gesetzt werden.

SarosCancellationException cancellationCause speichert den Grund des Abbruchs in Form von einer Exception.

Beim Empfangen der Abbruchsnachricht (6) werden also diese zwei Attribute entsprechend gesetzt (siehe rotes Dreieck rechts) und erst danach wird der Monitor abgebrochen (7). Kommt nun die LocalCancellationException von der Netzwerkschicht bei dem Einladungsprozess an (8), wird erstmal geprüft, ob bereits eine cancellationCause vorliegt (rotes Dreieck

rechts) . Ist das der Fall, wird diese gespeicherte Exception verarbeitet, andernfalls die ankommende LocalCancellationException (9). Die Benutzeroberfläche wird informiert (10) und damit ist die Einladung auf beiden Seiten beendet.

3.5.2 Die zwei Phasen

Das Abbrechen wird technisch in zwei Phasen aufgeteilt:

1. Der Aufruf der Methode localCancel (..) oder remoteCancel(..), bei der das Attribut cancelled gesetzt wird um sicherzustellen, dass das Abbrechen nur einmal stattfindet. Darüber hinaus wird der Grund des Abbrechens im Attribut cancellationCause in Form einer SarosCancellationException gespeichert. Sollte sie keine Nachricht enthalten(Exception.getMessage()[36]), so kann sie den Fällen (s. Kapitel 3.5) 1. bzw. 3., andernfalls den Fällen 2. bzw. 4. zugeordnet werden.

```
public void localCancel(String errorMsg, CancelOption cancelOption) {
    if (!cancelled.compareAndSet(false, true))
        return;
    log.debug("Inv" + Util.prefix(peer) + ": localCancel: " + errorMsg);
    if (monitor != null)
        monitor.setCanceled(true);
    cancellationCause = new LocalCancellationException(errorMsg,
        cancelOption);

public void remoteCancel(String errorMsg) {
    if (!cancelled.compareAndSet(false, true))
        return;
    log.debug("Inv" + Util.prefix(peer) + ": remoteCancel: " + errorMsg);
    if (monitor != null)
        monitor.setCanceled(true);
    cancellationCause = new RemoteCancellationException(errorMsg);
}
```

Quellcode 9: Die Methode für das lokale Abbrechen und für den Fernabbruch

2. Der Aufruf der Methode executeCancellation(), die nachschaut, was der Grund des Abbruchs ist und die Benutzeroberfläche dementsprechend informiert. Das passiert durch das Werfen der Exception, die in cancellationCause gespeichert ist. Davor werden noch alle nötigen "Aufräumarbeiten" erledigt, um die Einladung abzuschließen.

Es ist wichtig, die Reihenfolge einzuhalten: zuerst muss entweder ein localCancel (..) oder remoteCancel(..) stattfinden und erst danach darf executeCancellation() aufgerufen werden, sonst ist der cancellationCause noch nicht gesetzt.
Der Grund für die Auftrennung des Abbruchs in zwei Phasen liegt in der Natur des verteilten Systems. Das Abbrechen wird technisch mittels Exceptions verwirklicht. Die Abbruchsexception sollte immer direkt bei der langlaufenden Operation, ganz "tief" im Aufrufstapel (z.B. bei der Archivübertragung beim DTM) geworfen werden. So kann sie im Stapel hochwandern und auf jeder Ebene (z.B. Netzwerk → Einladungsprozess → Benutzeroberfläche) Bescheid geben. In der Abb. 13 wird der Aufrufstapel mit gepunkteter grüner Linie dargestellt. Wie erkennbar sein sollte, handelt es sich um ein "geschlossenes" System. Eine Exception kann nur dann im Aufrufstapel hochgereicht werden, wenn sie in der Kette der Methodenaufrufe geworfen wird. Im Fall eines Fernabbruches sieht das aber anders aus. Der eine Teilnehmer bekommt eine Nachricht, die das Abbrechen auslöst (6). Das kann

[36] http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Exception.html

aber nur dadurch passieren, dass eine Methode vom IIP oder OIP aufgerufen wird. In diesem Fall würde aber die geworfene Exception nicht im gewünschten Stapel hochwandern, sondern der externe Aufrufer würde sie bekommen. Mit den beschriebenen zwei Phasen wird dies umgangen. Die Abbruchsnachricht ruft bloß die Methode remoteCancel(..) auf, die keine Exceptions wirft, sondern nur den Grund des Abbruches speichert und den Monitor abbricht. Dadurch wird (im Fall der Archivübertragung) vom DTM eine Exception geworfen, die dann beim IIP oder OIP ankommt. Erst hier wird die Methode executeCancellation() aufgerufen, die basierend auf cancellationCause die Exception wirft. Die wird aber in den richtigen Stapel geworfen, d.h. die Benutzeroberfläche wird sie bekommen. Die Ursache des Abbruches wurde also mittels einen externen Aufrufs in den richtigen Aufrufstapel "eingeschmuggelt".

3.5.3 Abbruch über das Netz

Eine Alternative zur Lösung mit den zwei Phasen wäre ein Abbruch über das Netz und zwar mittels Nachrichtenaustausch zwischen den zwei DTMs (also ohne den Aufrufstapel zu verlassen, wie die Schritte (5) und (6) das tun). In diesem Fall wäre das Abbrechen der Datenübertragung nicht nur über den Monitor möglich, sondern der eine DTM könnte den anderen über den Abbruch informieren. Der benachrichtigte DTM würde eine RemoteCancellationException werfen und damit den ganzen Einladungsprozess abbrechen. Die Unterscheidung des Abbruchsgrundes – lokal (über den Monitor) oder fern (über die DTM-Nachricht) – wäre dadruch im DTM möglich. Diese Lösung hätte aber den Nachteil, dass sich die Netzkomponente (DTM) um das Abbrechen kümmern sollte. Es gibt verschiedene Arten von Datentransfers in Saros, wie z.B. **Jingle**[37] und **IBB**[38], die das Verschicken von Daten unterschiedlich implementieren (für Details siehe [Szu09]). Wenn wir das Abbrechen auf diese Art machen wollten, müssten dies alle Datentransferarten unterstützen, was zur Zeit nicht der Fall ist. Während beim Jingle der Abbruch des Transfers zwischen den DTMs "abgesprochen" wird, ist dies für IBB nicht vorgesehen. Mit den zwei Phasen ist diese Abhängigkeit aufgelöst.
Sollte der Abbruch des Transfers auch über den DTM stattfinden, tritt ein kritischer Wettlauf (*Race Condition*) der Abbruchsnachrichten auf. Einmal wird eine Nachricht über den DTM und einmal über den Einladungsprozess (Schritte (5) und (6)) verschickt. Das wird durch das Zwei-Phasen-Konzept automatisch behandelt. Dabei ist bloß zu beachten, dass der DTM in diesem Fall eine RemoteCancellationException wirft. Die beim Einladungsprozess (remoteCancel (..)) zuerst ankommende Abbruchsnachricht wird bearbeitet, die nächste wird verworfen. Da beide Nachrichten die gleiche Information enthalten, spielt es keine Rolle, welche zuerst eintrifft.

3.5.4 Kontrollstellen

Das Abbrechen findet technisch gesehen immer dadurch statt, dass eine Exception geworfen wird. Das kann durch einen technischen Fehler vorkommen, oder wegen eines benutzerinitiierten Abbruches. Klickt der Benutzer auf die Abbruchsschaltfläche (Abb. 12) oder kommt eine Abbruchsnachricht an, wird der Status des aktuellen Monitors 3.4 auf "abgebrochen" gesetzt. Nun hat die aktuelle langlaufende Operation die Aufgabe diesen Status in bestimmten Abständen zu prüfen und ggf. eine Exception zu werfen. Da der Einladungsprozess an sich auch als eine langlaufende Operation zu betrachten ist, sollte er den Status des Monitors verfolgen und eventuell einen Abbruch durchführen.

[37] http://xmpp.org/extensions/xep-0166.html
[38] http://xmpp.org/extensions/xep-0047.html

```
protected void checkCancellation() throws SarosCancellationException {
    if (cancelled.get()) {
        log.debug("Inv" + Util.prefix(peer) + ": Cancellation checkpoint");
        throw new SarosCancellationException();
    }

    if (monitor == null)
        return;

    if (monitor.isCanceled()) {
        log.debug("Inv" + Util.prefix(peer) + ": Cancellation checkpoint");
        localCancel(null, CancelOption.NOTIFY_PEER);
        throw new SarosCancellationException();
    }

    return;
}
```

Quellcode 10: Methode zur Überprüfung des Status des Monitors

Das erledigt die Methode checkCancellation(). Wenn der Monitor abgebrochen wurde, wird eine SarosCancellationException geworfen, weil es an dieser Stelle nicht entscheidbar ist, wo der Abbruch herkommt: vom lokalen oder entfernten Benutzer. Diese Exception wird sowohl im OIP als auch im IIP wie folgt abgearbeitet:

```
catch (SarosCancellationException e) {
    /**
     * If this exception is thrown because of a local cancellation, we
     * initiate a localCancel here.
     *
     * If this exception is thrown because of a remote cancellation, the
     * call of localCancel will be ignored.
     */
    localCancel(e.getMessage(), CancelOption.NOTIFY_PEER);
    executeCancellation();
}
```

Quellcode 11: Behandlung der SarosCancellationException in OIP und IIP

Falls die Einladung vom lokalen Benutzer abgebrochen wurde, wird das Attribut cancelled= **false** sein, also wird localCancel (..) durchlaufen. Wenn aber ein Fernabbruch vorliegt, wurde remoteCancel(..) bereits aufgerufen und localCancel (..) hat dadurch keine Auswirkungen.

3.6 Nebenläufigkeit

Hinsichtlich maximaler Produktivität ist die Nebenläufigkeit des Einladungsprozesses ein sehr wichtiger Aspekt. Aus Benutzersicht können wir die Nebenläufigkeit in die folgenden Stufen aufteilen:

Stufe 0: dem Benutzer wird ein modales Fenster mit den Einladungsinformationen angezeigt. Dadurch ist der ganze Eclipse Arbeitsbereich gesperrt, er kann weder auf dem aktuell übertragenen Projekt noch auf anderen Projekten in Eclipse arbeiten. Er muss warten, bis die Einladung abgeschlossen ist und das Fenster verschwindet.

Stufe 1: dem Benutzer wird ein nicht-modales Fenster mit den Einladungsinformationen angezeigt. Das Fenster kann in den Hintergrund verschoben werden, so dass der Eclipse Arbeitsbereich freigegeben wird. Der Benutzer kann in Eclipse ohne Beschränkungen weiterarbeiten, bis auf das aktuell übertragene Projekt. Das wird gesperrt.

Stufe 2: dem Benutzer wird ein nichtmodales Fenster mit den Einladungsinformationen angezeigt. Das Fenster kann in den Hintergrund verschoben werden, so dass der Eclipse Arbeitsbereich freigegeben wird. Der Benutzer kann in Eclipse ohne Beschränkungen weiterarbeiten, inklusive dem aktuell übertragenen Projekt.

Unter dem Aspekt "Nebenläufigkeit der Einladung" können die Benutzer wie folgt kategorisiert werden:

Einladender: der Benutzer, der die Einladung initiiert und das Projekt zur gemeinsamen Nutzung zur Verfügung stellt.

Eingeladenen: Benutzer, die vom Einladenden gerade eingeladen werden und sich in der Einladung befinden. Sie müssen das Projekt übertragen bekommen.

Sitzungsteilnehmer: Benutzer, die sich bereits in der Sitzung befinden und über das komplette Projekt verfügen.

Es wird natürlich bei allen drei Arten von Teilnehmern der Einladung angestrebt, die maximale Stufe an Nebenläufigkeit sicherzustellen, damit stößt man aber an bestimmten Stellen auf technische Schwierigkeiten, die später erläutert werden.
Da der Einladende mehrere Benutzer gleichzeitig einladen kann, führt das zu einer weiteren Aufteilung, die als Bezugspunkt die technische Implementierung im Vordergrund hat:

Lokale Nebenläufigkeit: der Einladende kann mehrere Benutzer gleichzeitig einladen. Diese parallelen Einladungen müssen einander berücksichtigen (gewöhnliche Aspekte der nebenläufigen Programmierung[39]).

Globale Nebenläufigkeit: da es sich um ein verteiltes System handelt, reicht es nicht aus, die Nebenläufigkeit lokal beim Einladenden zu handhaben. Alle Benutzer, die mit der aktuellen Sitzung in Berührung kommen (siehe Kategorisierung oben), müssen entsprechend kontrolliert werden.

Im Folgenden wird das Konzept unter Bezugnahme auf die oben aufgeführten Stufen diskutiert, Benutzerkategorien und lokale bzw. globale Nebenläufigkeit.

3.6.1 Globale Nebenläufigkeit

Eines der Hauptziele überhaupt von Saros ist es, das gemeinsam benutzte Projekt (also die Projektdateien) ständig über alle Sitzungsteilnehmer synchron zu halten. Alle Operationen (Erstellung, Änderung, Öffnen, Schließen von Dateien, Markierungen im Quellcode usw.), die an dem geteilten Projekt durchgeführt werden, werden in sogenannte **Aktivitäten** umgewandelt und an alle Teilnehmer verschickt. Eine Aktivität enthält meistens den Absender, eine Sequenznummer und den Typ bzw. den Inhalt der Änderung/Operation. Die empfangenen Aktivitäten müssen in die richtige Reihenfolge gebracht werden. Diese Aufgabe erledigt die Klasse ActivitySequencer[40](AS). Danach werden sie wieder in die entsprechenden Operationen zurückkonvertiert und auf den betroffenen Ressourcen ausgeführt. Mehr Informationen hierzu sind in den Arbeiten [Dje06, Rin09a] zu finden. Auf dieses Synchronisierungskonzept muss in der Einladung besondere Rücksicht genommen werden.
Die globale Nebenläufigkeit betrifft alle Benutzer, die mit der aktuellen Sitzung zu tun haben. Während einer Sitzung könnten am Projekt von allen Drivern zu einem beliebigen

[39] http://java.sun.com/docs/books/tutorial/essential/concurrency/
[40] de.fu_berlin.inf.dpp.net.internal

Zeitpunkt Änderungen vorgenommen werden. Das Ziel ist, der neu eingeladenen Person alle Projektdateien im aktuellen Zustand zur Verfügung zu stellen, damit das Konzept der synchronisierten Projekte nicht verletzt wird. Der kritische Abschnitt bei der Einladung ist das Erstellen, Versenden und Empfangen des Projektarchivs. Der erste Lösungsansatz wäre, alle Sitzungsteilnehmer anzuhalten (damit sie keine Änderungen am Projekt durchführen können), ein Projektarchiv zu erstellen und es dem Eingeladenen zuzuschicken. Nach dem Erhalt des Archivs hätte der neue Teilnehmer die Projektdateien im aktuellen Zustand. Jetzt könnten wieder alle Benutzer gestartet werden. Das einzige Problem ist, dass das Verschicken des Archivs auch längere Zeit beanspruchen kann und mit dieser Lösung könnten die Driver während der Archivübertragung nicht am Projekt weiterarbeiten. Aus diesem Grund muss die Archivübertragung aus dem kritischen Abschnitt herausgezogen werden.
Vor der Fortsetzung des globalen Nebenläufigkeitskonzeptes müssen die folgenden zwei Arten von Stopps eingeführt werden:

SM-Stopp: es gibt eine Klasse namens StopManager[41], die für das Anhalten der Benutzer zuständig ist [Zil09]. Wenn jemand mittels des Stopmanagers angehalten wird, wird sein Eclipse Arbeitsbereich nicht gesperrt, aber er kann an dem geteilten Projekt keine Änderungen mehr vornehmen. Das Öffnen von Dateien ist erlaubt.

AS-Stopp: nachdem ein Benutzer einer Sitzung beigetreten ist, muss sein ActivitySequencer gestartet werden. Wie oben beschrieben ist diese Klasse für das Abarbeiten der ankommenden und ausgehenden Aktivitäten zuständig. Solange der AS gestoppt ist, werden alle Aktivitäten in den lokalen Warteschlagen aufbewahrt und erst nach dem Starten verschickt bzw. verarbeitet.

Nun stellt sich die Frage: wie kann der Eingeladene nach dem Abschluss der Einladung ein konsistentes Projekt haben, wenn die Sitzungsteilnehmer während der Archivübertragung Änderungen an den Dateien vornehmen können? Der Ablauf sieht wie folgt aus:

1. Der Einladende stoppt alle Sitzungsteilnehmer mittels des Stopmanagers. Ab diesem Zeitpunkt können an dem Projekt keine Änderungen mehr vorgenommen werden.
2. Der Einladende erstellt das Projektarchiv.
3. Der Einladende fügt den Eingeladenen der Sitzung hinzu. Beim Eingeladenen ist der AS noch nicht gestartet, die ankommenden Aktivitäten werden also noch nicht abgearbeitet. Das würde auch noch keinen Sinn ergeben, weil er noch gar nicht über das Projekt verfügt.
4. Der Einladende sagt allen Sitzungsteilnehmern bescheid, dass der Eingeladene der Sitzung beigetreten ist. Ab diesem Zeitpunkt werden alle Aktivitäten aller Teilnehmer auch dem Eingeladenen verschickt.
5. Der Einladende startet alle Sitzungsteilnehmer mittels des Stopmanagers. Jetzt können an dem Projekt wieder Änderungen vorgenommen werden, also können alle wie gewohnt weiterarbeiten.
6. Der Einladende schickt das Archiv dem Eingeladenen.
7. Der Eingeladene empfängt das Archiv, während die ankommenden Aktivitäten in der Warteschlange gespeichert werden. Wenn das Archiv angekommen und entpackt wurde, startet er seinen AS. Da jetzt alle Projektdateien vorhanden sind, können die Aktivitäten abgearbeitet werden.

[41] de.fu_berlin.inf.dpp.synchronize

Auf diese Weise wird der kritische Abschnitt (Schritte 1 bis 5), während die Sitzungsteilnehmer am Projekt keine Änderungen vornehmen dürfen, auf die Erstellungszeit des Archivs reduziert. Dieser Abschnitt ist eine **globale atomare** Operation, wobei sich atomar auf die Nicht-Veränderbarkeit des Projektes bezieht. So werden also alle Aktivitäten, die nach diesem Abschnitt erfolgen, diesen Projektzustand als Referenzpunkt haben. Da der Eingeladene diese Aktivitäten dank Schritt 4 auch alle bekommt und im Archiv genau dieser Projektzustand gekapselt ist, ist somit sichergestellt, dass beim Eingeladenen keine Inkonsistenzen entstehen.

3.6.2 Lokale Nebenläufigkeit

Wie im Kapitel 3.1 dargelegt, ist der Einladungsprozess ein sequentieller Vorgang, der zwischen einem Einladenden und einem Eingeladenen abläuft. Auf der Seite des Einladenden wird der Prozess hauptsächlich von der Klasse OIP abgewickelt. Sie kommuniziert mit der Klasse IIP auf der Seite des Eingeladenen. Da sich der Eingeladene zur gleichen Zeit nur in einer einzigen Einladung (bzw. nur in einer Sitzung) befinden kann, ist die lokale Nebenläufigkeit für ihn irrevelant. Im Gegensatz dazu ist sie für den Einladenden von großer Bedeutung, weil er gleichzeitig mehrere Benutzer einladen kann. Das wird dadurch verwirklicht, dass der Einladende für jeden Eingeladenen ein Exemplar der Klasse OIP erstellt. Jedes Exemplar sorgt also für die Einladung eines Benutzers. Diese Exemplare werden in verschiedenen *Threads* ausgeführt und dadurch ensteht die lokale Nebenläufigkeit auf der Einladendenseite. Die anderen Sitzungsteilnehmer sind von der lokalen Nebenläufigkeit nicht betroffen.
Die Schritte 1 bis 21 (Abb. 6) der Einladung können ohne Probleme parallel ausgeführt werden, weil sie keine gemeinsamen Ressourcen verändern oder andere Operationen beinhalten, die aufeinander Auswirkungen haben. Der erste kritische Abschnitt folgt beim Schritt 23 des Einladungsprozesses, bei dem die Sitzungsteilnehmer durch den Stopmanager gestoppt werden.

```
List<StartHandle> startHandles;
synchronized (sharedProject) {
    startHandles = stopManager.stop(usersToStop,
        "Synchronizing invitation", subMonitor.newChild(25,
            SubMonitor.SUPPRESS_ALL_LABELS));
}
```

Quellcode 12: Der erste kritische Abschnitt bei den Schritten 23-24 im OIP

Die Synchronisation erfolgt über das Objekt sharedProject, das ein Exemplar der Klasse Shared −Project ist. Da bisher nur ein Eclipse-Projekt gleichzeitig geteilt werden kann, existiert nur dieses einzige Exemplar. Wenn dieser Abschnitt verlassen wird, sind alle Benutzer durch den Stopmanager gestoppt.
Der nächste kritische Abschnitt sorgt dafür, dass der Eingeladene der Sitzung hinzugefügt wird und dass dies auch alle anderen erfahren. Die Methode synchronizeUserList() schickt die aktuelle Teilnehmerliste an alle Teilnehmer und wartet auf eine Bestätitung, dass sie diese auch erhalten haben. Im Kontext der Synchronisation über sharedProject ist das Hinzufügen eines neuen Teilnehmers also – bei jedem Teilnehmer – atomar.

```
synchronized (sharedProject) {
    User newUser = new User(sharedProject, peer, colorID); this.sharedProject.
        addUser(newUser);
    log.debug(Util.prefix(peer) + " added to project, colorID: "
        + colorID);
    synchronizeUserList();
```

```
}
```

Quellcode 13: Der zweite kritische Abschnitt beim Schritt 25 im OIP

Der dritte kritische Abschnitt macht zwei Sachen: erstens wird der Status des Eingeladenen geändert, zweitens wird erneut die Teilnehmerliste synchronisiert.

```
synchronized (sharedProject) {
    sharedProject.userInvitationCompleted(sharedProject.getUser(peer));
    synchronizeUserList();
}
```

Quellcode 14: Der dritte kritische Abschnitt beim Schritt 33 im OIP

Jedes Benutzerobjekt (User[42]) hat ein Attribut **boolean** invitationComplete, das besagt, ob die Einladung des Benutzers bereits abgeschlossen ist. Am Anfang ist dieses Attribut immer **false**. Wenn der Benutzer mit der Einladung fertig ist, schickt er im Schritt 31 (Abb. 6) eine Bestätigung an den Einladenden. Jetzt ändert der Einladende diesen Status vom Eingeladenen auf **true** und synchronisiert die Teilnehmerliste.

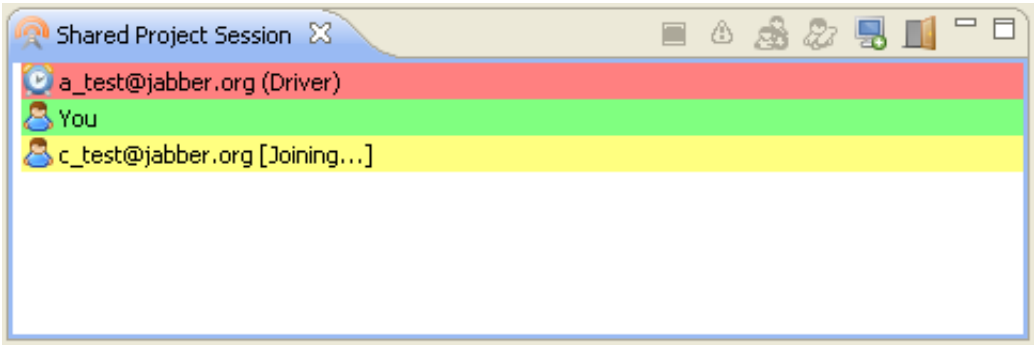

Abbildung 14: Während der Einladung wird in der Sitzungsansicht der Eingeladene (c_test) entsprechend gekennzeichnet

Mittels User.isInvitationComplete() ist immer abfragbar, ob sich jemand gerade in der Einladung befindet. Diese Methode wird auch dazu benutzt, den Benutzern in der Sitzungsansicht diesen Status anzuzeigen (Abb. 14).
Diese drei kritischen Abschnitte werden aus den folgenden Gründen über das gleiche Objekt synchronisiert:

- Abschnitt zwei und drei: in beiden Fällen wird die Teilnehmerliste synchronisiert. Sollte diese Synchronisation nebenläufig ausgeführt werden, könnte passieren, dass ein Benutzer am Ende nicht die aktuelle Liste bekommt. Z.B: *A* ist die aktuelle Liste, *B* ist nicht mehr aktuell. Beide werden vom Einladenden nebenläufig abgeschickt. Aus netzwerktechnischen Gründen kommt bei einem Teilnehmer zuerst die Liste *A* und erst danach die Liste *B* an. Der Teilnehmer wird also eine nichtaktuelle Liste haben, weil er immer annimt, dass die zuletzt ankommende Liste die aktuelle ist.

- Abschnitt eins und zwei (oder drei): hier besteht die Abhängigkeit darin, dass die Benutzer anhand der Teilnehmerliste gestoppt werden. Sollte sich die Liste während des Anhaltens der Benutzer ändern, könnte das zu unerwarteten Ergebnissen führen.

Damit ist sichergestellt, dass die verschiedenen Exemplare von OIP nebenläufig laufen können, ohne falsche Zustände oder unerwartetes Verhalten zu bewirken.

[42] de.fu_berlin.inf.dpp

3.6.3 Beschränkungen

Leider wurden im Rahmen dieser Arbeit nicht alle Probleme gelöst, die wegen der nebenläufigen Einladungen auftreten können. Das größte Problem ist auf die Funktionsweise des Stopmanagers [Zil09] zurückzuführen. Wenn er einen Benutzer anhalten will, schickt er ihm eine Stopp-Aktivität und blockiert solange, bis eine Bestätigung ankommt. Diese Aktivität wird vom ActivitySequencer des Benutzers abgearbeitet. Die Abarbeitung beinhaltet das Stoppen der lokalen Bearbeitung und das Verschicken der Bestätigung. Wenn sich nun ein Benutzer in der Einladung befindet, ist sein AS noch nicht gestartet, dadurch kann er auch keine Aktivitäten abarbeiten. Wenn ihn jemand stoppen will, wird der Stopmanager solange blockieren, bis die Bestätigung ankommt. Das kann aber nur dann passieren, wenn der Benutzer die Einladung beendet hat, der AS wird nämlich erst in Schritt 30 (Abb. 6) gestartet. Um diese lange Wartezeit zu umgehen, werden zur Zeit im OIP nur diejenigen Benutzer gestoppt, die sich gerade nicht in der Einladung befinden. Das könnte aber dazu führen, dass innerhalb eines Stoppblocks (zwischen den Schritten 23 und 26) der eben nicht angehaltene Benutzer seine Einladung beendet und an dem Projekt zu arbeiten anfängt. Die Lösung all dieser Probleme besteht darin, die Stopp-Aktivitäten beim Ankommen sofort abzuarbeiten, ohne auf die Warteschlange des ActivitySequencers zu warten. Das darf aber nur in den folgenden Fällen passieren:

- Der aktuelle Benutzer ist durch den Stopmanager bereits gestoppt.
- Der AS des Benutzers ist noch nicht gestartet.

Die zweite, nicht so schwerwiegende Beschränkung, wurde erst in der letzten Phase der Arbeit gefunden, deswegen konnte sie im Rahmen der Arbeit nicht mehr behoben werden. Sie hängt mit den Dateilisten (*Filelist*) zusammen. In den Schritten 14-21 wird die Differenz der Projekte berechnet, damit der Einladende weiß, welche Dateien er dem Eingeladenen zuschicken muss. Diese Differenzberechnung auf Seite des Eingeladenen basiert auf der Dateiliste, die er im Schritt 14 bekommt. Nun können zwischen den Schritten 14-23 weitere Dateien erstellt bzw. gelöscht werden. Da der Einladende das Archiv basierend auf der im Schritt 21 ankommenden Dateiliste berechnet, werden diese Änderungen (also Dateien) im Archiv nicht mehr enthalten sein (diese Inkonsistenz kann mittels Konsistenzwiederherstellung (s. [Zil09]) behoben werden). Die Lösung wäre, einfach noch eine Dateilistensynchronisation nach Schritt 23 durchzuführen. Da in diesem Block alle Benutzer SM-gestoppt sind, kann sich das Projekt nicht ändern.

4 Outreach

Die Verbreitung von Saros ist ein sehr wichtiges Ziel des ganzen Teams. Je mehr Leute Saros benutzen, desto öfter bekommen wir Feedback [Doh09] von den Benutzern und können das Plugin in die gewünschte Richtung weiterentwickeln bzw. Fehler korrigieren. Bei der Verbreitung spielt die Arbeit von Eike Starkmann [Sta09b] eine besonders wichtige Rolle. Er spricht Open Source Projekte über das Internet an und versucht unter anderem, Saros in deren Arbeitsprozesse zu integrieren bzw. dessen Nutzen für andere zu evaluieren.
In letzter Zeit hat das Team gute Fortschritte gemacht, was die Anzahl der Downloads von Saros angeht (Abb. 15), jedoch haben wir uns bisher bei den meisten Bemühungen auf eine Verbreitung im Internet konzentriert. Meine Idee ist es, Saros unter den Studenten unserer Universität, der Freien Universität Berlin, bekannt zu machen.

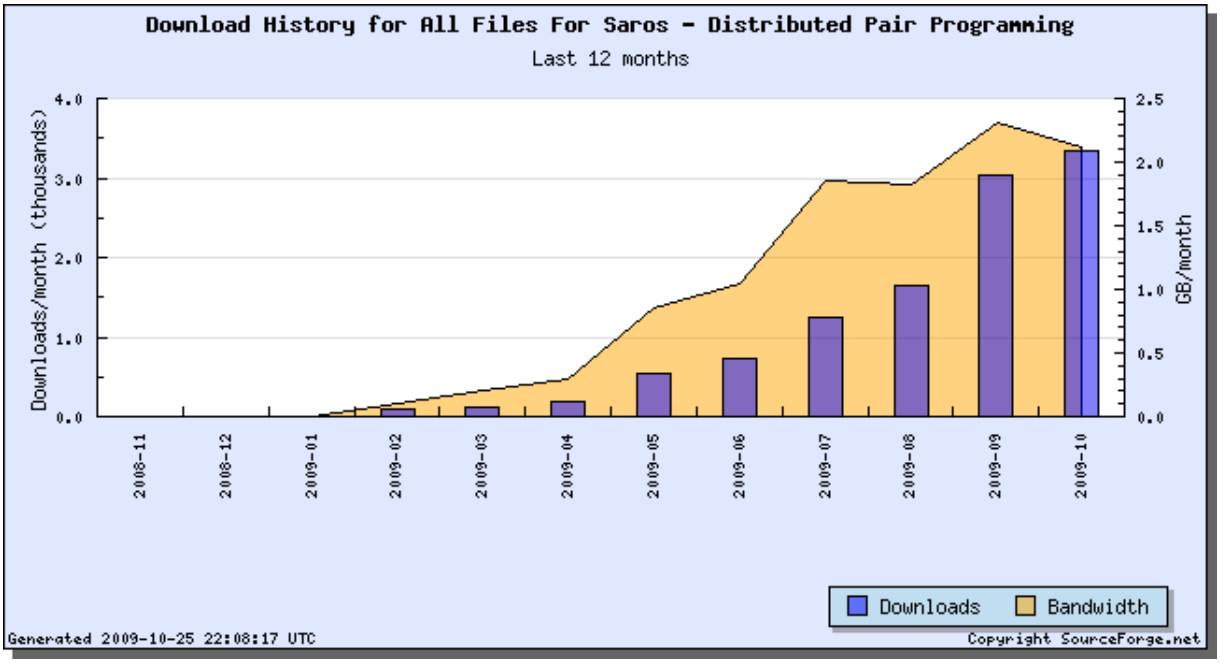

Abbildung 15: Die Anzahl der Downloads in den letzten 12 Monaten von der SourceForge Projektseite: http://sourceforge.net/project/stats/?group_id=167540&ugn=dpp (die Daten zum Oktober sind nicht ganz repräsentativ, weil diese zum Erstellungs-Zeitpunkt der Grafik noch nicht vollständig vorlagen)

Wie an vielen deutschen Universitäten, sind auch an der Freien Universität Berlin im Studiengang Informatik zwei grundsätzliche Lehr- und Lernformen[43] vorgesehen: Vorlesungen und Übungen. Der Stoff wird in den Vorlesungen vermittelt und in den Übungen werden Aufgaben gestellt und besprochen, um den Stoff zu verstehen, zu verarbeiten und zu vertiefen. In den meisten Veranstaltungen werden die Aufgaben in Zweier- oder Dreier Teams bearbeitet. Da Saros für Teamarbeit perfekt geeignet ist, könnten es die Studenten bei der Bearbeitung der Aufgaben einsetzen. Ich habe eine kleine Aufgabe 4.2 zusammengestellt, die man auf den Übungszetteln bestimmer Veranstaltungen unterbringen könnte, um Saros den Studenten näher zu bringen.
Bei der Auswahl der Veranstaltungen habe ich darauf geachtet, dass Paarprogrammierung entweder eine direkte Verbindung zum Stoff hat, wie z.B. bei der Vorlesung Softwareprozesse, oder dass die Übungen technisch genug sind, um von Saros zu profitieren. Die ausgewählten Veranstaltungen sind:

- Algorithmen und Programmierung III: Datenstrukturen und Datenabstraktion
- Algorithmen und Programmierung V: Netzprogrammierung
- Höhere Algorithmik
- Softwareprozesse

Die Dozenten und jeweils ein Tutor (ggf. der wissenschaftliche Mitarbeiter) der Veranstaltungen wurden von mir über eine Mail 4.1 angesprochen, in der ich meine Absichten darlege. Da das aktuelle Semester erst vor Kurzem angefangen hat, kann ich leider über die Ergebnisse des Outreaches nicht berichten. Die einzige zur Zeit vorliegende Information

[43] http://www.fu-berlin.de/service/zuvdocs/amtsblatt/2007/ab062007.pdf

ist, dass ich bei drei von vier Veranstaltungen eine positive Rückmeldung auf die E-Mail bekommen habe.

4.1 Die Mail

Sehr geehrter Prof. XY,

das neue Semester ist wieder da und Sie tun Ihr Bestes, um die Studenten zu motivieren und interessante Aufgabenstellungen zu finden. Ein sehr wichtiges Ziel neben der Vermittlung von fachspezifischen Inhalten ist die Förderung von Teamarbeit. Hierzu möchten wir hiermit einen Beitrag leisten.
Die Arbeitsgruppe Software Engineering arbeitet seit längerer Zeit an dem Eclipse-Plugin Saros zur verteilten Paarprogrammierung. Die Idee dahinter ist, dass verschiedene Entwickler ein Softwareprojekt zeitgleich gemeinsam bearbeiten können. Ein Host lädt hierzu über das Internet die gewünschten Teilnehmer in eine Saros-Sitzung ein, schickt ihnen die Projektdateien zu und verteilt Schreibrechte. Die Teilnehmer können dann Programmieraufgaben gemeinsam lösen, wobei die Kommunikation über ein Messenger-Programm (Skype, MSN, usw.) ablaufen sollte. Im Rahmen meiner Bachelorarbeit bin ich im Namen des Saros-Teams auch für die Verbreitung von Saros unter den FU-Studenten beauftragt. Eine solche Verbreitung ist sowohl für die Studenten als auch für das Saros-Team vorteilhaft. Die Studierenden erhalten ein Werkzeug, um auch von zu Hause aus zusammenarbeiten zu können und das Saros-Team bekommt Informationen, um die Qualität des Werkzeuges zu verbessern.
Meine Idee ist es, den Studierenden Saros über eine kleine Aufgabe näher zu bringen. Vielleicht könnten Sie diese in einen Ihrer ersten Übungszettel der Veranstaltung YZ aufnehmen. Die Bearbeitung nimmt mit Vorbereitung zwischen 30 und 60 Minuten in Anspruch. Die Aufgabe finden Sie im Anhang sowohl im Latex- als auch im OpenOffice-Format. Weitere Informationen zum Saros finden Sie unter: dpp.sourceforge.net/ Die Adresse der Saros-Mailingliste lautet: saros@lists.spline.de
Wir würden uns über Ihre Unterstützung freuen! Bei Fragen stehe ich unter der Adresse tastnt@gmail.com gerne zur Verfügung.

Mit bestem Dank,
Tas Sóti für das Saros-Team
AG Software Engineering

4.2 Die Übung

Aufgabe: Kollaborative Programmierung mit Saros

Saros ist ein Eclipse-Plugin für verteilte Paarprogrammierung, welches zur Zeit von der Arbeitgruppe Software Engineering entwickelt wird. Es eröffnet die Möglichkeit, ein Eclipse-Projekt über das Internet gemeinsam zu nutzen. Ein Host öffnet hierzu eine Sitzung und lädt andere Entwickler zum gemeinsamen Programmieren oder Begutachten von Code ein. Weitere Informationen finden Sie unter http://dpp.sf.net. Fragen und Anmerkungen können Sie an die Saros Mailingliste saros@lists.spline.de senden.

Zunächst müssen Sie Eclipse und das Saros-Plugin installieren:

1. Installieren Sie Eclipse Classic (http://www.eclipse.org/downloads/)

2. Installieren Sie Saros von der Eclipse Update-Site: http://dpp.sf.net/update
3. Loggen Sie sich in der *Roster View* ein (die Saros-Views sind in Eclipse unter *Window → Show View → Saros* zu finden):
   (a) Bereits vorhandenes Jabber-Account benutzen, z.B. testuser@jabber.org
   Server: jabber.org
   Benutzername: testuser
   (b) Bereits vorhandenes GMail-Account benutzen
   Server: gmail.com
   Benutzername: [username]@gmail.com oder [username]@googlemail.com
   (c) Account erstellen unter http://register.jabber.org/
4. Der Host wählt ein Projekt zur gemeinsamen Nutzung aus (es sollte mindestens 3 Dateien enthalten): *Rechts-Klick → Share project...*
   Nun laden Sie Ihren Übungspartner ein!

Unter *Windows → Preferences → Saros* finden Sie die Einstellungen zum Saros. Falls Sie Bemerkungen haben, helfen Sie uns und füllen Sie unsere kleine Umfrage unter *Feedback* aus!

In einer Saros-Sitzung wird ein Teilnehmer **Driver** genannt, wenn er die Rechte hat, die Projektdateien zu bearbeiten. Standardmäßig verfügt der Einladende über Driverrechte.

Widmen Sie sich nun gemeinsam folgenden Aufgaben:

1. Der Driver öffnet zwei Dateien. Die anderen Teilnehmer sollen rausfinden, welche Dateien beim Driver auf sind und welche gerade bearbeitet wird! Wie erkennt man das?
2. Es kommt oft vor, dass man wissen möchte, wo sich der Driver gerade befindet. Mit einem Doppelklick auf den Namen des Drivers in der *Shared Project Session*-Ansicht springt Eclipse zur aktuellen Stelle. Was macht man, wenn man dem Driver die ganze Zeit folgen möchte?
3. Es besteht die Möglichkeit, die gleiche Datei von mehreren Teilnehmern gleichzeitig zu bearbeiten. Wie macht man das?

Hinweis: sollten Sie Firewalls benutzen, könnten bei der Einladung Schwierigkeiten auftreten. Wir arbeiten an der Verbesserung. Netzwerkprofis sind im Saros-Team willkommen!

Lösungen für die Tutoren:

1. In der *Package Explorer*-Ansicht von Eclipse sind die Projektdateien zu finden. Alle dateien, die der Driver auf hat, sind bei den anderen Teilnehmern mit einem kleinen gelben Kreis am Dateiicon gekennzeichnet. Die Dateien, die auf sind und gerade bearbeitet werden, haben einen grünen Kreis.
2. Zur Verfolgung eines Drivers steht der sogenannte **Follow Mode** zur Verfügung. Dieser Modus kann in der *Shared Project Session*-Ansicht aktiviert und deaktiviert werden.
3. Diese Funktionalität heißt **Multi Driver Support**. Mit einem Rechtsklick in der *Roster*-Ansicht auf einen Benutzernamen können Driverrechte vergeben werden. Haben mehrere Teilnehmer die Rechte, können sie alle an einer gemeinsamen Datei gleichzeitig arbeiten.

5 Testszenario

5.1 Zielsetzung

Dieses kleine Testszenario hatte das Ziel, aus Benutzersicht einen Blick auf den Einladungsprozess zu werfen. Obwohl die Testfälle so organisiert wurden, dass möglichst viele Aspekte des Einladungsprozesses vorkommen, wurde wegen beschränkter Ressourcen (Anzahl der Testpersonen und Zeit) nicht angestrebt, alles vollständig durchzutesten bzw. alle Defekte zu finden.

5.2 Testumgebung

Es gab insgesamt drei Tester:

- a_test

```
Java Version: 1.6.0_12
Java Vendor: Sun Microsystems Inc.
Eclipse Runtime Version: 3.5.0.v20090525
Operating System: Windows XP (5.1)
Hardware Architecture: x86
```

Quellcode 15: Auszug aus der Logdatei.

- b_test

```
Java Version: 1.6.0_11
Java Vendor: Sun Microsystems Inc.
Eclipse Runtime Version: 3.5.0.v20090525
Operating System: Windows XP (5.1)
Hardware Architecture: x86
```

Quellcode 16: Auszug aus der Logdatei.

- c_test

```
Java Version: 1.6.0_16
Java Vendor: Sun Microsystems Inc.
Eclipse Runtime Version: 3.5.0.v20090525
Operating System: Linux (2.6.28-15-generic)
Hardware Architecture: i386
```

Quellcode 17: Auszug aus der Logdatei.

Zwei Windows- und ein Linuxrechner waren also im Einsatz. Die Rechner befanden sich im selben Netz hinter einem D-Link DIR-300[44] Router. Die Eigenschaften der Internetverbindung sind in Abb. 16 zu sehen.
Es gab zwei unterschiedliche Projekte, die bei den Tests benutzt wurden:

- TestProjekt1: 108 KB, 14 Dateien und 9 Ordner
- TestProjekt2: 126 MB, 1,236 Dateien und 134 Ordner

[44] ftp://ftp.dlink.de/dir/dir-300/documentation/DIR-300_reva_man_de_Handbuch.pdf

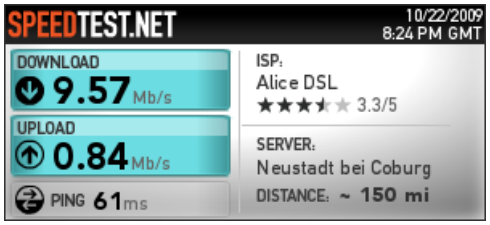

Abbildung 16: Eigenschaften der Internetverbindung

Die verschiedenen Größen waren wichtig, damit die Benutzer in beiden Fällen ein Gefühl bekamen, wie die Einladung funktioniert. Beim TestProjekt1 läuft sie sehr schnell ab und die Tester haben nicht die Möglichkeit, die Benutzerfreundlichkeit von Fortschrittsanzeigen oder Abbrechbarkeit einzuschätzen. Das Ziel war hier, erstmal einen allgemeinen Eindruck zu bekommen, wie eine Einladung abläuft, ohne mehrere Minuten lang auf die Datenübertragung zu warten. Auch der Versionskonflikt war in dieser Testphase zu testen. Nach den Tests zum TestProjekt1 A.1.1 bekamen die Tester die erste Hälfte der Fragen A.2.1 vorgelegt. Das Testen ging danach mit TestProjekt2 weiter. Da dieses Projekt wesentlich mehr Dateien enthält als das andere Projekt, hat hier die Datenübertragung mehr Zeit in Anspruch genommen. Dadurch waren Aspekte wie Fortschrittsanzeige, Abbrechbarkeit und Nebenläufigkeit gut zu beobachten. Nach diesen Testfällen A.1.2 beantworteten die Benutzer den Rest der Fragen A.2.2.
Die dritte Tesphase diente dazu, auf noch eventuell unbearbeitete Fragen Antworten zu finden. In dieser "freien" Testphase ging es darum, dass die Tester selber entscheiden konnten, wer wen einlädt, welche Einladung abbricht oder welches Projekt benutzt. Nach diesem selbständigen Testen waren die Tests abgeschlossen.

5.3 Ergebnis

Die Tests sind ohne größeren Probleme verlaufen. Der allgemeine Eindruck der Tester über den Einladungsprozess war eindeutig positiv. Sie fanden die Einladungen überschaubar und intuitiv. Es gab einen Wunsch, den Fortschrittsbalken in Abschnitte aufzuteilen, also jeweils ein Balken für die Erstellung des Archivs, Übertragung des Archivs usw. Diese Änderung würde ich ablehnen, weil die Fortschrittsanzeige dazu dienen soll, den Benutzer über den Zustand der **kompletten** Einladung zu informieren. Wenn man für jeden Schritt der Einladung einen Balken hätte, würde man den Gesamtüberblick verlieren.
Eine weitere Idee war, im Fall von Versionskonflikten die Fortsetzung der Einladung zu verbieten. Das ist auf jeden Fall eine Überlegung wert. Man sollte sich fragen, wie sinnvoll ist es, zwei unkompatible Saros-Versionen miteinander zu benutzen? Das könnte in bestimmten Fällen sogar zum Datenverlust führen. Die Frage, die wir hier stellen müssen, ist, wie viel Kontrolle Saros dem Benutzer geben will. Es handelt sich letztendlich um seine Daten und wenn er sich trotz hinreichender Benachrichtigung für die Fortsetzung der Einladung enscheidet, muss er die Verantwortung für die Konsequenzen tragen. Eine Kompromisslösung wäre, in den Einstellungen von Saros eine Option "Einladung im Fall von Versionskonflikten erlauben" einzuführen. Der Benutzer würde dann wissentlich, auf eigene Gefahr hin, diese Option wählen können, per default wäre sie ausgeschaltet.
Der Rest der Vorschläge wie z.B. Schriftgröße oder bestimmte Überschriften in der Benutzeroberfläche zu ändern, halte ich für personenbezogene Wünsche, die je nach Benutzer variieren könnten, deswegen werde ich sie hier nicht weiter diskutieren.
Die Tests haben also gezeigt, dass die Benutzer im Großen und Ganzen mit dem Einladungs-

prozess zufrieden waren.

6 Methodik

Dises Kapitel beschreibt den Prozess, der zu den dargestellten Ergebnissen geführt hat. Es soll einerseits als Selbstreflexion dienen, um die eigene Arbeitsweise zu analysieren, sie dadurch besser zu verstehen und bewusst verbessern zu können. Andererseits bekommt der Leser einen Einblick in meinen persönlichen Arbeitsstil und in die Ereignisse der letzten drei Monate aus meiner Sichtweise.

6.1 Arbeitsumgebung

Dem studentischen Saros-Team stand ein eigener Raum im Institut für Informatik der Freien Unversität Berlin, in der Takustrasse 9 zur Verfügung. Ein bloßer Blick auf den grünen Innenhof mit Glasdach[45] kann zum kurzfristigen Entleeren des codegefüllten Kopfes dienen, aber ich habe meistens lieber einen kurzen Spaziergang auf dem frisch gemähten Rasen vor dem Informatikgebäude[46] gemacht. Nicht dass mir das Büro nicht gefallen hätte, es ist für fünf bis sechs Personen bequem nutzbar. Obwohl die Hälfte der Arbeitsplätze mit Rechnern ausgestattet ist, habe ich meinen eigenen Laptop bevorzugt. Wenn man morgens vor 10 Uhr ankam, gab es gute Chancen, einen der Rollstühle zu bekommen, sonst blieb einem nur ein zwar regulärer, aber gepolsterter Stuhl übrig. Wer vor 9 Uhr kam, musste die Tür meistens selber aufschließen.

6.2 Das Team

Das lag natürlich nicht daran, dass die “Sarosianer” faul waren! Aber wer steht schon gerne früh auf? Sie sind lieber länger geblieben! Die Arbeitszeit lag so zwischen 10 und 18 Uhr. Die meiste Zeit habe ich mit Wojtek Polcwiartek, Marc Rintsch, Eike Starkmann und Sándor Szücs (nach Nachnamen sortiert!) verbracht. Am Anfang war auch noch Edna Rosen öfter mal dabei, danach hat aber bei ihr die Ausarbeitungsphase angefangen und sie hat die meiste Zeit nicht mehr im Büro verbracht. In der Endphase meiner Arbeit haben sich drei weitere Studenten der Arbeitsgruppe angeschlossen: Robert Kunze, Stephan Lau und Henning Staib. Mit denen habe ich aber auch wenig Zeit verbracht. Und die zwei wichtigsten Personen, ohne die es kein Saros-Team gäbe: **Christopher Özbek** und **Stephan Salinger**. Sie haben ihr Bestes gegeben, um uns “Schergen” (das war unser Deckname) zu unterstützen und die Entwicklung von Saros voranzutreiben. Sie waren immer kompetent, streng aber nett.
Die allgemeine Atmosphäre im Team und im Büro war locker und freundlich. Es gab eine gute Kommunikation unter den Mitgliedern und ich hatte nie das Gefühl, alleine dazustehen. Noch war ich jemals allein, alle waren stets anwesend und haben bei Gelegenheit Hilfestellung gegeben.

[45] http://www.flickr.com/photos/tastnt/525495979/in/set-72157600050315462/
[46] http://www.flickr.com/photos/tastnt/525494961/in/set-72157600050315462/

6.3 Softwaretechnische Aspekte

6.3.1 Einarbeitung

Saros besteht zur Zeit ungefähr aus 300 Klassen und Schnittstellen. Im Vergleich zu meiner bisherigen Erfahrung war das ein wesentlicher Schritt, ich habe früher noch nie mit so einem großen Projekt zu tun gehabt. Man öffnet eine Klasse, schaut rein und hat keine Ahnung, wo man anfangen soll. Es war ziemlich wenig Dokumentation vorhanden und selbst die ein-zwei Diplom- oder Studienarbeitung waren veraltet, was die Beschreibung der Architektur und vor allem des Codes angeht. Ich habe also angefangen mir ein paar relevante Klassen auszusuchen, wie z.B. OutgoingInvitationProcess oder IncomingInvitationProcess und sie durchzugehen, um zu verstehen, was sie machen. Die ersten Tage bestanden einfach daraus, mich durch eine Menge an Klassen zu klicken. Die Entdeckung der Suchfunktionen von Eclipse, wie z.B. alle Aufrufe einer Methode anzuzeigen (*Search → References → Project*), haben die Einarbeitung und auch die weitere Arbeit deutlich erleichtert. Obwohl ich mir jetzt Java-Entwicklung ohne diese Funktionalität nicht mehr vorstellen kann, habe ich sie während meines bisherigen Studiums gar nicht gebraucht. Diese Erkenntnisse zeigen den hohen Stellenwert von Softwareprojekten bzw. der Bachelor -oder Masterarbeiten im Studium.

6.3.2 Wissensaustausch

Ich merkte schnell: wenn ich etwas am Code nicht verstehe, muss ich andere fragen! Das Wissen ist schriftlich nicht festgehalten, ist aber im Team verteilt vorhanden. Eine fünfminütige Erklärung von einem Teammitglied spart nicht nur drei-vier Stunden an Nachdenkarbeit, sondern hilft auch mehr die Zusammenhänge zu verstehen. Christopher war immer der richtige Ansprechpartner, er verfügte im Team über das meiste Wissen. Oft haben mir aber auch die anderen Studenten sehr viel helfen können.
Die Paarprogrammierungssitzungen mit Christopher könnte ich als Meilensteine meiner Einarbeitung bezeichnen. Drei -oder viermal haben wir uns zu zweit oder zu dritt für drei-vier Stunden zusammengesetzt und Christopher hat (ohne Pause) Code geschrieben. Da wir nie Rollen getauscht haben, war das keine "richtige" Paarprogrammierung, aber das Ziel war auch ein bisschen anders: Lernen. Er hat nicht nur Code geschrieben, er hat auch versucht, uns "mitzunehmen". Er hat die Zusammenhänge erklärt und unsere Fragen beantwortet. Das war eine sehr wichtige Erkenntnis für mich. Das erste Mal während meines Studiums konnte ich jemandem bei der Arbeit zuschauen, der genügend Erfahrung hatte, um, ohne mir ständig die Frage "Weiß er überhaupt, was er macht?!" stellen zu müssen, etwas lernen zu können. Das ist eine Art des Lernens, die ich schon während des gesamten Studiums vermisst habe: jemand macht etwas vor, wo er sich **auskennt**, ich schaue zu und stelle Fragen; eine Art "persönlicher Kontakt zum Wissen".
Sándor Szücs war hauptsächlich für die Netzwerkschicht verantwortlich, die bei der Einladung in vielen meiner Themen eine wichtige Rolle spielte, wie z.B. Abbrechbarkeit 3.5, Verfolgung des Fortschrittes 3.4 oder allgemeine Kommunikation zwischen Einladendem und Eingeladenem. Sándor war die Person, mit der wir am meisten zusammen gearbeitet haben. Wir haben Konzepte erarbeitet, technische Details besprochen oder Architekturfragen geklärt. Wir waren ein perfektes Team.
11.30 war Mittagszeit. Zu dieser Zeit ging die ganze Arbeitsgruppe zusammen in die Mensa. Da ich meistens erst gegen 9 oder 10 im Büro ankam, war das ein bisschen früh für das Essen. Ich habe sehr oft mehr darauf gewartet, meine Fragen loszuwerden, als meinen Bauch zu füllen. Gemeinsames Essen ist nämlich viel mehr, als Essen. Es ist ein sehr wichti-

ger Bestandteil des Wissensaustausches. Mit Freude, Ärger oder Zufriedenheit geht man in die Mittagspause und hofft darauf, die negativen Gefühle aufzulockern und die positiven einzusammeln, aber vor allem: ein paar Fragen beantwortet zu bekommen. Sowohl die Studenten als auch die Betreuer sind dabei, so kann man gezielt Diskussionen initiieren und Lösungen finden. Hier hat man aber auch Zeit, die anderen persönlich besser kennenzulernen und Spaß zu haben.

6.3.3 Prozesse

Die wichtigsten Informationen um Saros herum haben wir in unserem Wiki[47] festgehalten. So auch die bedeutsamsten Infos für die Entwickler, die Entwicklungsrichtlinien[48]. Wir hatten jede dritte Woche ein Release. Der Releaseplan sah wie folgt aus (Originalfassung aus dem Wiki):

- Roles: RM Release Manager, TM Test Manager
- Tuesday until 16:00 both of the following tasks have to be done. Attention: The second tasks depends on the first one!
  - RM creates list of newly fixed bugs and features and sends it to the mailing-list
    - Use the Changelog from SVN as a starting point (use right click on the project in eclipse: Team - Show History, in history view: Generate Changelog)
    - Take a look at the bug tracker as well for bugs which are said to have been fixed in the release
    - Try to sort items by importance and write in a language which users can understand
  - TM and ATM review and update the Test Suite (ask developers for ways to reproduce newly fixed bugs and add appropriate test cases)
- Wednesday
  - 10:00-10:30 - RM creates a new release branch from trunk and runs the JUnit test cases (when finished RM sends an email to the mailing-list to inform that the branch is ready (including the problems occurred during the JUnit tests).)
  - 10:30-17:00 - TM runs the test suite
  - 17:00-19:00 - TM sends to the list
    - ...a list of critical bugs and regression
    - ...a list of comments on the change-log. In particular whether a [FIX] or actually worked
    - ...all log files generated during the test as one big archive
- Thursday: 09:00-16:00 - The whole team fixes critical bugs on the branch
- Friday
  - 09:00-14:00 - The whole team fixes critical bugs on the branch
  - 14:00-15:00 - RM + TM perform a last Sanity Test and run the JUnit test cases

[47] http://dpp.sourceforge.net/
[48] https://www.inf.fu-berlin.de/w/SE/DPPHowToDevel

- 15:00-16:00 - RM releases and merges fixes into Trunk, RM announces the new release to the Saros list and Saros Announce (TM closes all bugs found to be fixed)
- 16:00-17:00 - Next Release Plan

Die zwei Hauptrollen waren also: Release Manager (RM) und Testmanager (TM). ARM und ATM waren jeweils die Assistenten für die beiden Manager. Die endgültige Rollenverteilung fand meistens montags in der Releasewoche statt. Dieser Releaseplan war der einzige Hauptprozess. Sonst fand die Entwicklung zwischen den Releasewochen ohne festgelegte Regeln statt. Der einzige wichtige, die Entwicklung betreffende Prozess waren die Durchsichten oder auf Englisch *Peer Reviews*. Wenn jemand ein Stück Code fertig hatte, musste es in Form von eines Patches[49] an die Mailingliste geschickt werden. Der Code durfte nur dann in das Repository hochgeladen werden, wenn dessen Richtigkeit von aderen bestätigt wurde. Solange Christopher Özbek im Team aktiv war, hat er alle Patches durchgesehen, kommentiert und ggf. in das Repository hochgeladen. Da alle anderen wussten, dass Christopher das macht (und da alle mit ihren eigenen Arbeiten beschäftigt waren), kam es eher selten vor, dass die Studenten den Code von anderen durgesehen haben. In den letzten drei Wochen meiner Arbeit hat sich das geändert, weil Christopher sich vom Team verabschiedete. Wir hatten also keinen mehr, der die Durchsichten gemacht hat. Wir mussten uns darum selber kümmern und einen strengeren Durchsichtungsprozess einführen (Originalfassung aus dem Wiki):

1. Create your patch
   - Smaller patches are always better, because they are easier to review. So if possible split one large patch into independent ones.
   - Write a concise commit message following the CodeRules
2. Send to the DPP-devel mailing list on SourceForge asking for review (we use [Patch] as a message title prefix)
3. People will hopefully quickly start reviewing
   (a) While waiting it is a good idea to review at least two patches by others
   (b) If you find that your patch is complex and not easy to understand by the other project members, you should offer to conduct a pair programming review with anybody interested.
   - Let the reviewer be the driver, so that he or she proceeds at his or her own speed. If you as the author are the driver, you are probably doing things to fast.
   - Use Saros (let's try to eat our own dog food), because this gives us valuable experience with Saros and also tests the latest release
   - Even if you perform a review as a pair session, the reviewer should post his remarks to the list, so other can learn from mistakes/defects you found while pair reviewing the code
4. To get permission to commit to the repository you must get:
   - At least two +1

[49] http://svnbook.red-bean.com/nightly/de/svn-book.html

- or if after 48 hours you have not gotten two +1s, then one +1 and one +0 are sufficient
- No -1 which is a hard veto no matter how many +1s you got. You will have to address the problems describes in the -1 email.

5. Integrate the suggestions from the reviews. If these changes are more than trivial, you should resend the patch asking for a new review.
6. Let the patch be committed by one of the people with commit rights (they should not forget to sign-off the patch with the names of all people who reviewed the patch positively)

Durch diesen Prozess wurde die Codequalität eindeutig erhöht. Das kann ich persönlich bestätigen, weil ich in diesen letzten drei Wochen mit anderen zusammen ein paar Durchsichten gemacht habe. Darüber hinaus hat so eine Durchsicht (siehe *Peer Programming Review* im Durchsichtungsprozess) einen bedeutsamen Effekt auf den Austausch von Wissen. Derjenige, der den Code geschrieben hat, erklärt seine Implementierung, während die anderen versuchen diese nachzuvollziehen und Defekte zu erkennen. Diese Art von Durchsichten ähnelt der Paarprogrammierung 1.1, wobei in diesem Fall der Code bereits fertig geschrieben ist.

7 Ausklang

"Was macht denn nun eine gute Abschlussarbeit aus?" - lautet die Frage auf der Webseite der Software Engineering AG der Freien Universität Berlin. Und gleich die Anmerkung dazu:

> "Aus Sicht unserer Arbeitsgruppe ist eine solche Arbeit **Teil eines längeren Prozesses**, der meist vor ihr begann und nach ihr weitergeht. Ziel ist es daher nicht, eine isoliert gute Arbeit abzuliefern, sondern einen bestmöglichen Beitrag zu diesem Prozess zu leisten. [..][50]"

Aus der Sicht der Arbeitsgruppe... Teil eines längeren Prozesses.. Bestmöglicher Beitrag zu diesem Prozess. Bil..!? Ach, ne ne. Das ist bestimmt nicht gemeint. Von Bil.. pssst! Von Bildung spricht man selten an der Universtität... und da steht doch auch: "Aus – der – Sicht – der – Arbeitsgruppe". Hm... ich habe das bestimmt missverstanden. Einfach falsch interpretiert. Oder?

> "By education I am going to mean the ability to make more and more complex, more and more profound and extensive, the meanings that we attach to events and phenomena. When we are reading a text, we call this adducing of new meanings interpretation. When we are doing mathematics, we call this giving of meaning intuition and proof. When we are reading history, we call it a sense of historical context. When we are doing social science, we call it the sociological imagination. In all these areas, to be educated is to have the habit of finding many and diverse new meanings to attach to whatever events or phenomena we examine. We have lots of standard routines for doing this – interpretive paradigms, heuristic methods, theoretical schemes, investigative disciplines, and so on. But education is not these paradigms and methods and disciplines. Rather it is the

[50] https://www.inf.fu-berlin.de/w/SE/ThesisRules

> instinctive habit of looking for new meanings, of questioning old ones, of perpetually playing with and fighting about the meanings we assign to events and texts and phenomena." [..] [Abb02]

Hm... vielleicht ist die Frage doch berechtigt: was macht denn nun eine gute Abschlussarbeit aus der Sicht des **Studenten** aus? Es mag einem komisch vorkommen, aber ... ja! Das ist die Frage, auf die ich die Antwort suche. Was könnte denn für einen Studenten wichtig sein? Ich hab's! Das wird's bestimmt sein! Die Note! Das ist es! Eine gute Note zu bekommen. Die Note ist immer wichtig!

> "1. Grades tend to reduce students' interest in the learning itself. [..] 2. Grades tend to reduce students' preference for challenging tasks. [..] 3. Grades tend to reduce the quality of students' thinking. [..] 4. Grades aren't valid, reliable, or objective. [..] 5. Grades distort the curriculum. [..] 6. Grades waste a lot of time that could be spent on learning. [..] 7. Grades encourage cheating. [..] 8. Grades spoil teachers' relationships with students. [..] 9. Grades spoil students' relationships with each other. [..]
>
> Most of us are directly acquainted with at least some of these disturbing consequences of grades, yet we continue to reduce students to letters or numbers on a regular basis. Perhaps we've become inured to these effects and take them for granted. This is the way it's always been, we assume, and the way it has to be. It's rather like people who have spent all their lives in a terribly polluted city and have come to assume that this is just the way air looks – and that it's natural to be coughing all the time." [Koh99]

Ja! Ich muss zugeben... als 4136556 im Durchschnitt ca. 2.7 ist es schwer, aber...

> [..] "... you can seek education. It will not be easy. We have only helpful exercises for you. We can't give you the thing itself. And there will be extraordinary temptations – to spend whole months wallowing in a concentration that doesn't work for you because you have some myth about your future, to blow off intellectual effort in all but one area because you are too lazy to challenge yourself, to wander off to Europe for a year of enlightenment that rapidly turns into touristic self-indulgence. There will be the temptations of timidity, too, temptations to forgo all experimentation, to miss the glorious randomness of college, to give up the prodigal possibilities that—let me tell you—you will never find again; temptations to go rigidly through the motions and then wonder why education has eluded you. There are no aims of education. The aim is education. If – and only if – you seek it ... education will find you."[Abb02]

Literatur

[AB] ANDREW BEGEL, Nachiappan N.: *Pair Programming: What's in it for Me?*

[Abb02] ABBOTT, Andrew: *The Aims of Education Address*. http://www.inf.fu-berlin.de/lehre/pmo/eng/Abbott-Education.pdf. Version: 2002

[AC] ALISTAIR COCKBURN, Laurie W.: *The Costs and Benefits of Pair Programming*

[Dje06] DJEMILI, Riad: *Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierung verteilter Paarprogrammierung*, Freie Universität Berlin, Diplomarbeit, 2006. https://www.inf.fu-berlin.de/w/SE/ThesisDPPI

[Doh09] DOHRMANN, Lisa: *Erhebung von Benutzerfeedback aus der Nutzung eines Werkzeugs zur verteilten Paarprogrammierung*. https://www.inf.fu-berlin.de/w/SE/ThesisDPPVIII. Version: 2009

[DW] DIETMAR WINKLER, Stefan B.: *Evaluierung von Werkzeugen für Distributed Pair Programming: Eine Fallstudie*

[Ecl] *Eclipse Dokumentation*. http://help.eclipse.org/

[Gus07] GUSTAVS, Björn: *Weiterentwicklung eines Eclipse-Plug-Ins zur Verteilten Paarprogrammierung*. https://www.inf.fu-berlin.de/w/SE/ThesisDPPII. Version: 2007

[Jac09] JACOB, Christoph: *Weiterentwicklung eines Werkzeuges zur verteilten, kollaborativen Softwareentwicklung*. https://www.inf.fu-berlin.de/w/SE/ThesisDPPIV. Version: 2009

[Jav] *Java Documentation*. http://java.sun.com/docs/

[Koh99] KOHN, Alfie: *From Degrading to De-Grading*. http://www.alfiekohn.org/teaching/fdtd-g.htm. Version: 1999

[Kun09] KUNZE, Robert: *Unterstützung der Entwicklungsprozesse von Virtuellen Teams durch verteilte Paarprogrammierung*, Freie Universität Berlin, Diplomarbeit, 2009. https://www.inf.fu-berlin.de/w/SE/ThesisDPPXII

[Lau09] LAU, Stephan: *Verbesserte Präsenz durch Screensharing für ein Werkzeug zur Verteilten Paarprogrammierung*. https://www.inf.fu-berlin.de/w/SE/ThesisDPPXIII. Version: 2009

[PB02] PRASHANT BAHETI, Edward G. Laurie Williams W. Laurie Williams: Distributed Pair Programming: Empirical Studies and Supporting Environments / Department of Computer Science North Carolina State University. 2002. – Forschungsbericht

[Rie08] RIEGER, Oliver: *Weiterentwicklung einer Eclipse Erweiterung zur Realisierung und Protokollierung verteilten Paarprogrammierung im Hinblick auf Kollaboration und Kommunikation*. https://www.inf.fu-berlin.de/w/SE/ThesisDPPIII. Version: 2008

[Rin09a] RINTSCH, Marc: *Agile Weiterentwicklung eines Software-Werkzeuges zur verteilten Paarprogrammierung*. http://bj.spline.de/saros/studienarbeit.html. Version: 2009

[Rin09b] RINTSCH, Marc: *Technische Betreuung von Saros in einem betieblichen Umfeld*, Freie Universität Berlin, Diplomarbeit, 2009. https://www.inf.fu-berlin.de/w/SE/ThesisDPPXIV

[Ros09] ROSEN, Edna: *Bewertung verteilter Paarprogrammierung im betrieblichen Umfeld*, Freie Universität Berlin, Diplomarbeit, 2009. https://www.inf.fu-berlin.de/w/SE/ThesisDPPManagement

[Sma] *Smack Library*. http://www.igniterealtime.org/projects/smack/

[Sta09a] STAIB, Henning: *Verbesserung einer XMPP API für den Einsatz in Verteilter Paarprogrammierung*, Freie Universität Berlin, Diplomarbeit, 2009. https://www.inf.fu-berlin.de/w/SE/ThesisDPPX

[Sta09b] STARKMAN, Eike: *Verteilte Paarprogrammierung in Open Source Projekten*, Freie Universität Berlin, Diplomarbeit, 2009. https://www.inf.fu-berlin.de/w/SE/ThesisDPPIX

[Szu09] SZUCS, Sandor: *Behandlung von Netzwerk- und Sicherheitsaspekten in einem Werkzeug zur verteilten Paarprogrammierung*, Freie Universität Berlin, Diplomarbeit, 2009. https://www.inf.fu-berlin.de/w/SE/ThesisDPPVI

[XP] *Extreme Programming*. http://www.extremeprogramming.org/

[XSt] *XStream Library*. http://xstream.codehaus.org

[Zil09] ZILLER, Sebastian: *Behandlung von Nebenläufigkeitsaspekten in einem Werkzeug zur Verteilten Paarprogrammierung*, Freie Universität Berlin, Diplomarbeit, 2009. https://www.inf.fu-berlin.de/w/SE/ThesisDPPVII

A Tests

A.1 Testfälle

Einladung akzeptieren heißt: einmal *Next* drücken.
Projekt auswählen heißt: nach der Auswahl des Projektes *Finish* drücken.

A.1.1 TestProjekt1

Versionskonflikt wird eingerichtet, damit die Benutzer die Benachrichtigung über den Konflikt sehen.
Die Versionskonflikte sind nur simuliert. Achten Sie auf die Verständlichkeit und Darstellungsform. Wenn Fehler oder unerwartetes Verhalten auftritt, notieren Sie diese kurz, damit Sie danach den Fragebogen ausfüllen können!

1. (a) a_test lädt b_test und c_test gleichzeitig ein und wartet, bis die Einladungen abgeschlossen sind.
   (b) b_test und c_test akzeptieren die Einladung und wählen ein neues Projekt aus
   (c) a_test verlässt die Sitzung
2. (a) a_test ändert TestClass1 beliebig und lädt b_test ein
   (b) b_test akzeptiert die Einladung und wählt ein neues Projekt aus
   (c) Wenn die Einladung abgeschlosen ist, lädt a_test c_test aus der *Roster View* ein
   (d) c_test akzeptiert die Einladung und wählt das Projekt mittels *Scan workspace* aus
   (e) a_test verlässt die Sitzung
3. (a) a_test lädt b_test und c_test gleichzeitig ein und wartet
   (b) b_test lehnt die Einladung ab
   (c) c_test akzeptiert die Einladung und beim nächsten Schritt bricht er die Einladung ab
   (d) a_test verlässt die Sitzung
4. (a) c_test lädt a_test ein
   (b) c_test lädt b_test ein und bricht die Einladung von a_test ab
   (c) b_test akzeptiert die Einladung
   (d) c_test bricht die Einladung von b_test ab
   (e) c_test lädt a_test und b_test gleichzeitig ein
   (f) a_test und b_test akzeptieren die Einladung und wählen das Projekt mittels *Scan workspace* aus
   (g) c_test verlässt die Sitzung

A.1.2 TestProjekt2

Versionskonflikt wird ausgestellt.

1. (a) b_test lädt a_test und c_test gleichzeitig ein und wartet, bis die Einladungen abgeschlossen sind.

(b) a_test und c_test akzeptieren die Einladung und wählen ein neues Projekt aus

(c) b_test verlässt die Sitzung

2. (a) b_test ändert 3 Dateien und lädt a_test und c_test gleichzeitig ein und wartet, bis die Einladungen abgeschlossen sind.

   (b) a_test und c_test akzeptieren die Einladung und wählen das Projekt mittels *Scan workspace* aus

   (c) b_test verlässt die Sitzung

3. (a) c_test lädt a_test und b_test gleichzeitig ein und wartet, bis die Einladungen abgeschlossen sind.

   (b) a_test akzeptiert die Einladung und wählt ein neues Projekt aus. Nach dem *Next* bricht er die Einladung ab.

   (c) b_test akzeptiert die Einladung und wählt ein neues Projekt aus. Im Zustand *Receiving archive...* bricht er die Einladung ab.

   (d) c_test verlässt die Sitzung

4. (a) c_test lädt a_test und b_test gleichzeitig ein

   (b) a_test und b_test akzeptieren die Einladung und wählen ein neues Projekt aus

   (c) c_test bricht die Einladung von a_test im Zustand *Creating archive...* ab

   (d) c_test bricht die Einladung von b_test im Zustand *Sending archive...* ab

   (e) c_test verlässt die Sitzung

5. (a) a_test lädt b_test ein

   (b) b_test akzeptiert die Einladung und wählt das Projekt mittels *Scan workspace* aus

   (c) a_test bearbeitet eine beliebige Datei während der Einladung

   (d) Wenn die Einladung von b_test abgeschlossen ist, öffnet er die gleiche Datei und schaut nach, ob alles konsistent ist.

   (e) a_test gibt b_test Driver-Rechte

   (f) a_test lädt c_test ein

   (g) Während der Einladung bearbeiten a_test und b_test beliebige Dateien (und lassen sie offen)

   (h) Wenn die Einladung von c_test abgeschlossen ist, öffnet er die Dateien, die a_test und b_test auf haben und schaut nach, ob alles konsistent ist.

A.1.3 Freier Test

A.2 Fragebogen

Die Antworten aller Benutzer werden gleich nach den Fragen dargestellt.

A.2.1 Fragen und Antworten zum TestProjekt1

1. Awareness ist eine sehr wichtige Sache bei der verteilten Paar-Programmierung. Waren Sie mit den Informationen zufrieden, die Sie während der Einladung über die anderen erhalten haben?
   - Ja.
   - groesseres Schriftbild waere gut
   - Als Eingeladener wurden keine Informationen über Host oder anderen Teilnehmern mitgeteilt. Als Host, wurden genügend Informationen über die Clienten während der Einladung mitgeteilt.
2. Im Fall von Versionskonflikten soll der Benutzer darüber informiert werden. War dies der Fall? Wie könnte der Benachrichtigungshinweis verbessert werden?
   - Ja, eine ensprechende Fehlermeldung ist erschienen.
   - ja
   - Man wurde informiert, konnte aber den Konflikt einfach wegklicken und fortfahren.
3. Wie ist Ihr Eindruck über die Einladungen im Allgemeinen gewesen? Was hat Ihnen nicht gefallen und warum?
   - Der Einladungsprozess ist ueberschauber und intuitiv.
   - Das ganze sollte visuell und vielleicht akustisch stärker hervorgehoben werden.
   - Das Einladungssystem war etwas umständlich, die Einladung hätte vielleicht über einen einfachen ja-nein-Dialog geregelt werden können.

A.2.2 Fragen und Antworten zum TestProjekt2

1. Waren die Abbruchsnachrichten konsistent und verständlich?
   - Ja.
   - ja
   - Die Abbruchnachrichten waren eindeutig.
2. Wenn nicht, welche Probleme haben Sie erkannt?
   - Keine.
   - –
   - –
3. Das Abbrechen soll schnell funktionieren, damit der Benutzer bei seiner Arbeit nicht aufgehalten wird. War dies der Fall?
   - Ja.

- ja
- Das Abbrechen der Einladungen ging recht flott. Das Abbruchbenachrichtigung darüber, dass der Host die Session verlassen hat, hat etwas genervt.

4. Wenn nicht, in welchem Fall war die Wartezeit zu lang?

   - –
   - –
   - Das Verbinden nach der Einladung hat etwas lang gedauert.

5. Ein wichtiger Aspekt bei der Einladung ist, dass die Benutzer über den Fortschritt der Einladung informiert werden. Was hat Ihnen bei den Fortschrittsanzeigen nicht gefallen? Schreiben Sie eventuell Verbesserungsvorlschäge!

   - Der Progress-Balken war informativ.
   - Schrift war zu klein
   - Für die verschiedenen Phasen der Synchronisation, wie bei Creating archive... und sending archive... sollten nicht als einzeigen Ladenbalke dargestellt werden, sondern abschnittsweise.

6. Eine maximale Nebenläufigkeit (beim Einalder) während der Einladung ist angestrebt. Konnten Sie während der Einladung als Einladender an dem Projekt arbeiten?

   - Ja, hat funktioniert. Die Aenderungen waren bei dem Eigeladenen auch sichtbar.
   - ja, ging gut
   - Ja.

A.2.3 Allgemeine Fragen

1. Schreiben Sie drei Sachen auf, die Ihnen bei der Einladung gefallen hat!

   - Der Einladungsprozess ist intuitiv; man kann den immer abbrechen; man muss nicht auf die Bestätigung der Einladung warten bzw bis der Eingeladene die Sachen runtergeladen hat.
   - springt ins Auge, leicht anzunehmen
   - Anzeige der Genauigkeit der Projekte. Angabe des Namen des Einladenders mit Kontaktmöglichkeit.

2. Was würden Sie an der Einladung ändern? Schreiben Sie Verbesserungsvorschläge!

   - Vielleicht sollte man die Verbindung von unterschiedlichen Versionen unterbinden!
   - Text und Schriftbild, anstelle des Textes “Session Invitation” waere ein Satz a “User Soundso laedt Sie zu einer Session ein” gut
   - Eine Beschreibung zum Projekt wäre nett. Kleinere Dialogbox für den Projektnamen. Einfaches Unterbinden des Dialogs über Versionskonflikte des Plugins durch das Wegklicken könnte zu Fehlern führen.

B Abbildungsverzeichnis

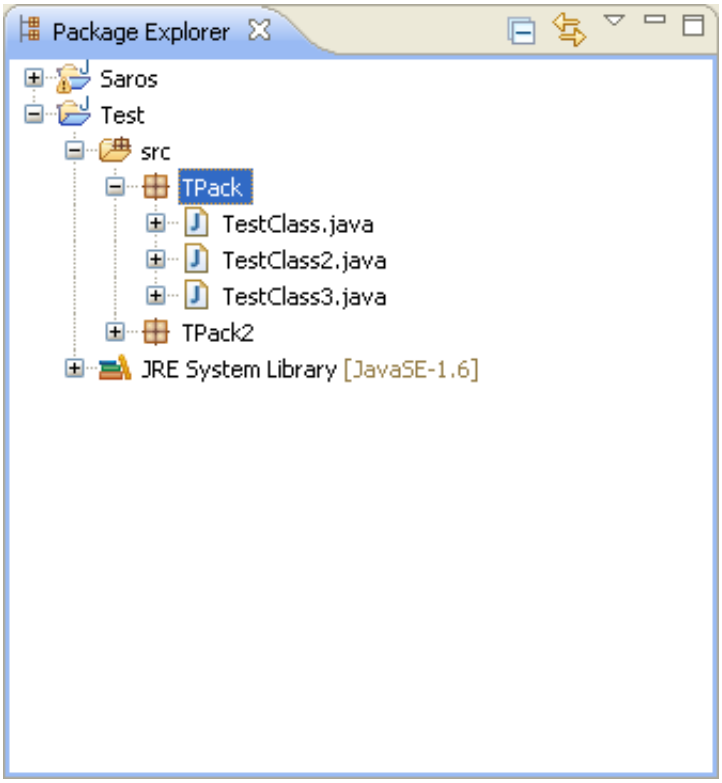

Abbildung 17: Package Explorer

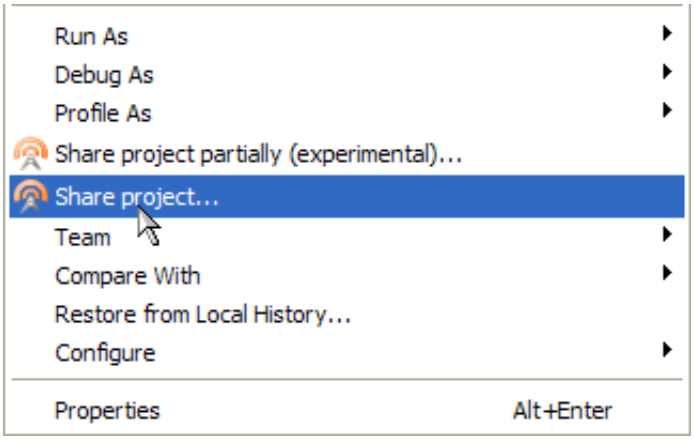

Abbildung 18: Ein Projekt zur gemeinsamen Nutzung auswählen

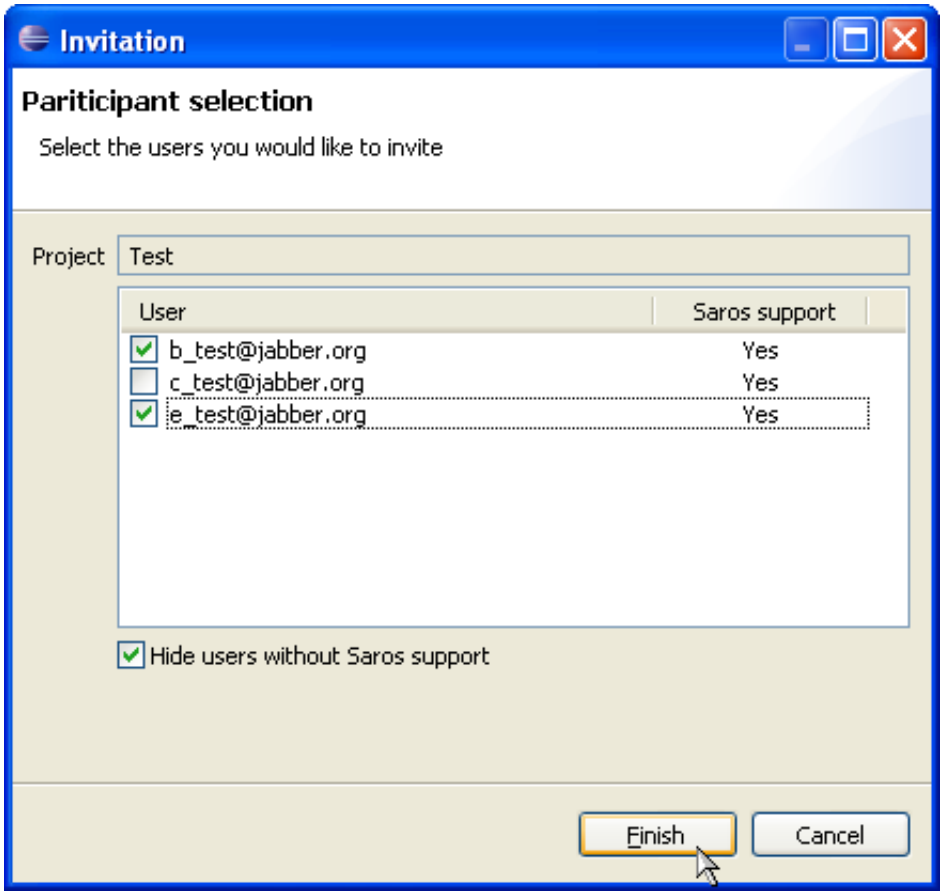

Abbildung 19: Einladungsassistent

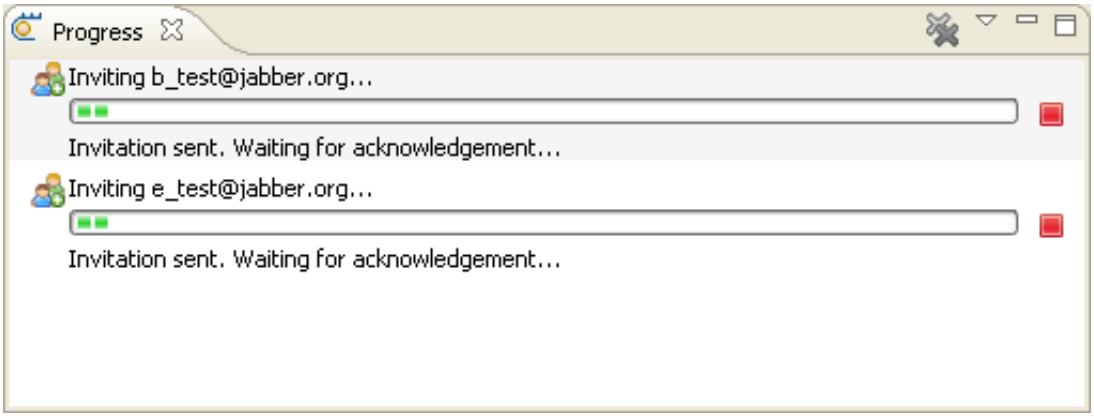

Abbildung 20: Fortschrittsansicht

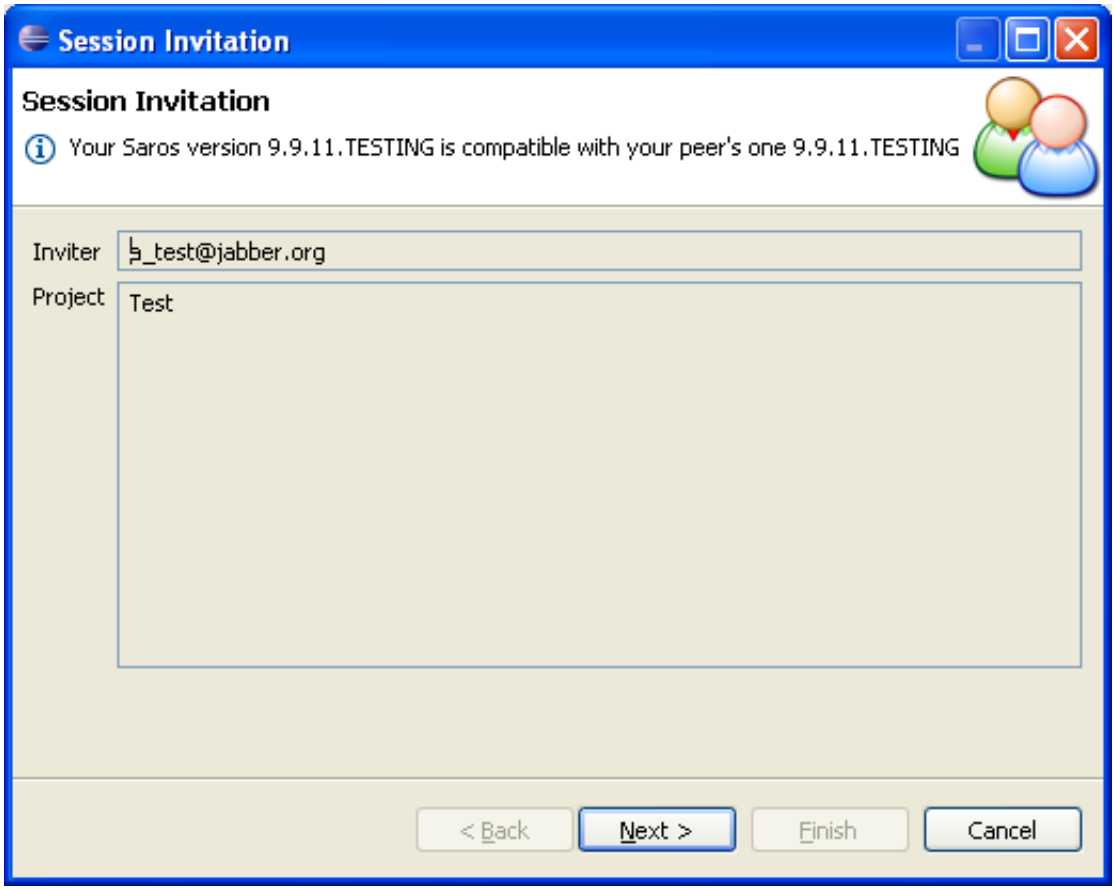

Abbildung 21: Benachrichtigung über die Einladung

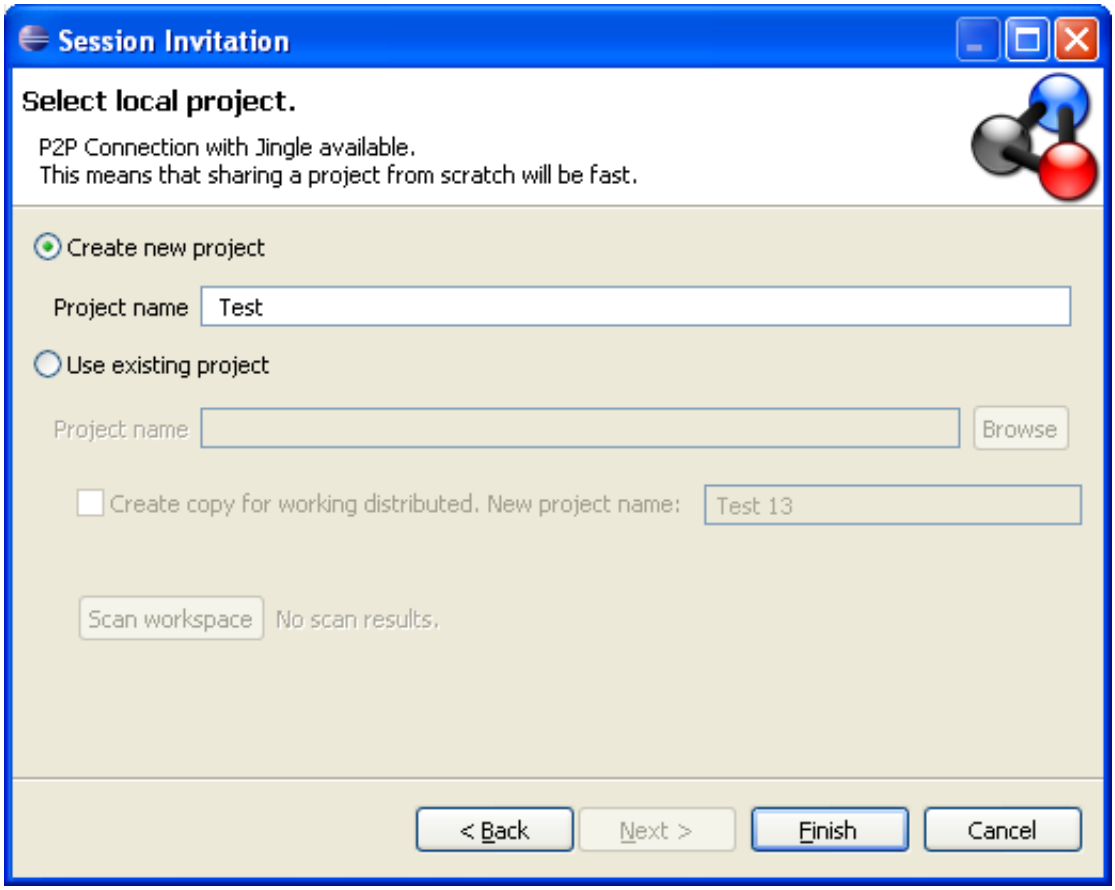

Abbildung 22: Auswahl des Zielprojektes

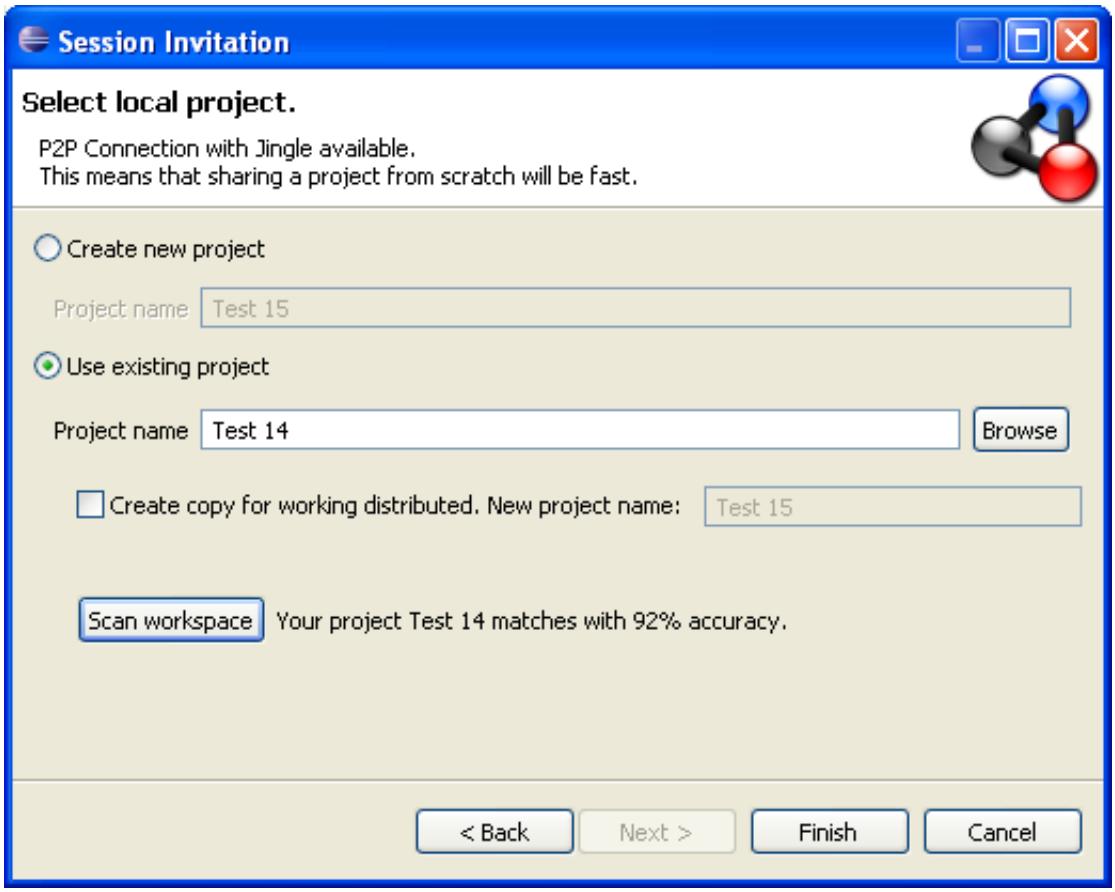

Abbildung 23: Auswahl des Zielprojektes - automatische Suche nach einem ähnlichen Projekt

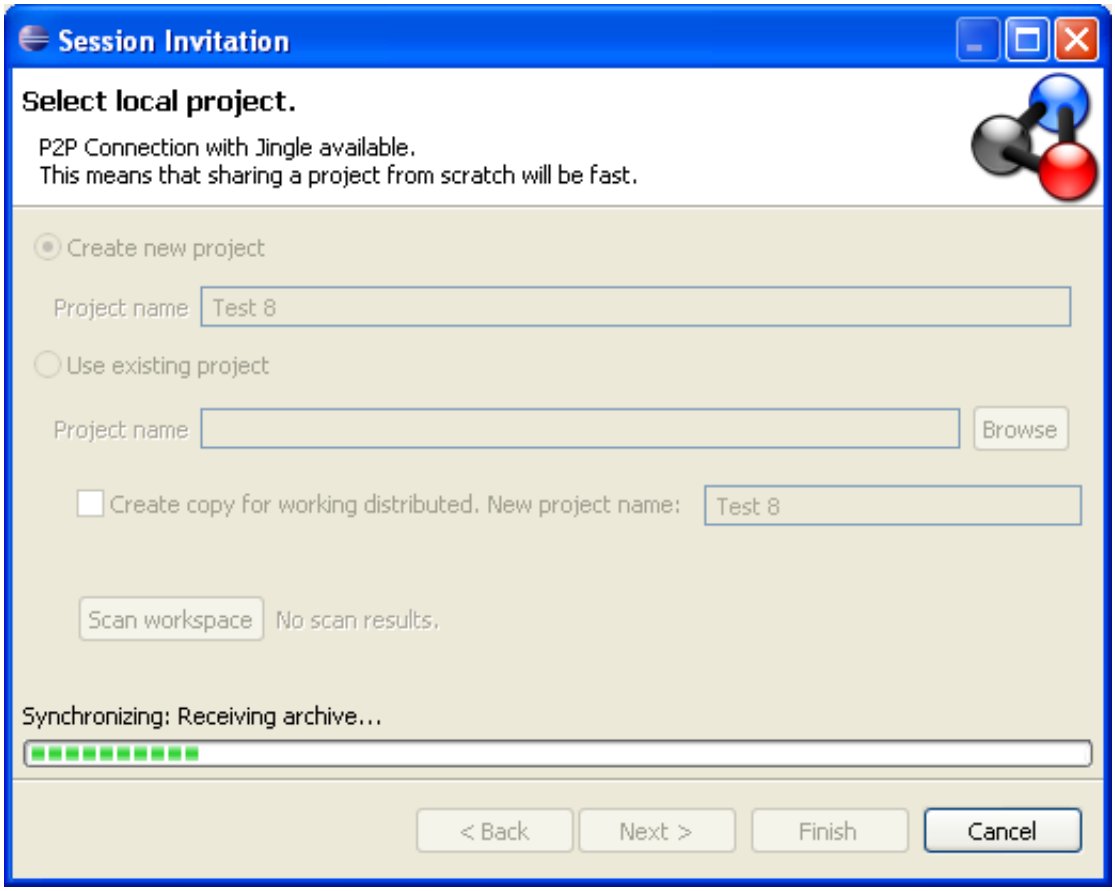

Abbildung 24: Anzeige des Fortschrittes

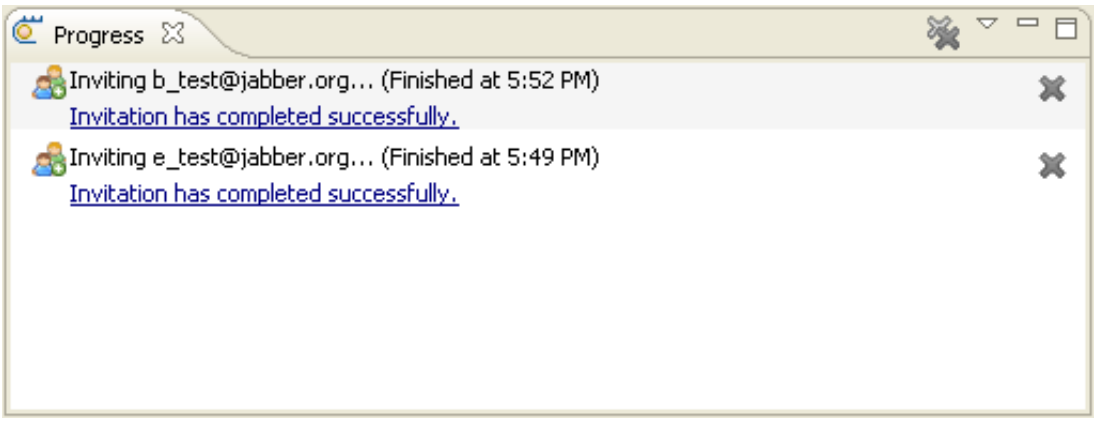

Abbildung 25: Die Einladungen wurden erfolgreich abgeschlossen