Diplomarbeit zur Erlangung des akademischen Grades
Diplom-Informatiker

Verbesserung einer XMPP-Bibliothek für den Einsatz in verteilter Paarprogrammierung

Henning Staib

Matrikelnummer: 3684796
henning.staib@fu-berlin.de

Berlin, 25. Mai 2010

Fachbereich Mathematik und Informatik
Institut für Informatik
Arbeitsgruppe Software Engineering

Eingereicht bei Prof. Dr. Lutz Prechelt
Zweitkorrektor: Prof. Dr. Heinz Schweppe
Betreuer: Karl Beecher und Stephan Salinger

Die von der Freien Universität Berlin am Fachbereich Mathematik und Informatik entwickelte Software Saros, die den softwaregestützen Einsatz der Entwicklungsmethode Distributed Pair Programming ermöglicht, nutzt zur Kommunikation über das Netzwerk die Open-Source-Bibliothek Smack, welche den Internetstandard XMPP zur Nachrichtenübertragung zwischen entfernten Klienten implementiert. Im Rahmen dieser Arbeit wird die Bibliothek Smack gezielt für den Einsatz in Saros verbessert und zudem werden Maßnahmen erörtert und umgesetzt, um die Open-Source-Community des Smack-Projektes zu reaktivieren. Durch die Entwicklung von Patches konnte die Grundlage für den Umbau des Netzwerkmoduls von Saros geschaffen werden, um dessen Zuverlässigkeit zu erhöhen und die Komplexität der Implementierung zu reduzieren. Die Maßnahmen zur Reaktivierung der Smack-Open-Source-Community beinhalten das Etablieren von Qualitätskriterien für Entwicklungsbeiträge sowie die Erstellung von Richtlinien für Smack-Entwickler, um den Einstieg in die Entwicklung und die Aufnahme in die Entwickler-Community zu erleichtern. Im zeitlichen Rahmen dieser Arbeit konnte dadurch jedoch keine Steigerung der Aktivität der Community erzielt werden.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

Die von der Arbeitsgruppe Software Engineering am Fachbereich für Mathematik und Informatik der Freien Universität Berlin entwickelte Software Saros ermöglicht die Umsetzung der Entwicklungsmethoden Pair Programming beziehungsweise Distributed Pair Programming, wodurch zwei oder mehr Entwickler kollaborativ am selben Software-Projekt von verschiedenen Orten über das Netzwerk zusammenarbeiten können.

Die Kommunikation zwischen den Teilnehmern wird durch die Nutzung der Open-Source-Bibliothek Smack, einer Implementierung der freien Protokollspezifikation XMPP umgesetzt. Das XMP-Protokoll ist ein Internetstandard zur Nachrichtenübermittlung zwischen entfernten Klienten ähnlich zu bekannten Instant-Messaging-Diensten, geht aber aufgrund seiner Erweiterbarkeit über diese Funktionalität hinaus und ermöglicht so den Einsatz in einer Vielzahl verschiedener netzbasierter Anwendungen.

Ein Ziel dieser Arbeit ist die gezielte Verbesserung der Bibliothek Smack durch das Beseitigen bekannter Fehler, dem Überprüfen der Konformität entsprechend der XMPP-Spezifikationen und dem Sicherstellen der korrekten Verwendung der Smack-Schnittstelle in Saros.

Ein weiterer Schwerpunkt der Arbeit liegt darin, Maßnahmen zu suchen um der stillstehenden Weiterentwicklung von Smack neue Impulse zu geben und so die Open-Source-Community um das Smack-Projekt zu reaktivieren.

2 Grundlagen

Zum Verständnis des Kontextes dieser Arbeit folgt eine Erläuterung der verwendeten Softwarekomponenten Saros und Smack sowie eine Einführung in das XMP-Protokoll.

2.1 Saros

Saros ist eine Erweiterung der integrierten Entwicklungsumgebung Eclipse, die es zwei oder mehr Entwicklern ermöglicht, gleichzeitig am selben Softwareprojekt zu arbeiten. Grundlage dieses als Distributed Pair Programming (DPP) beziehungsweise Distributed Party Programming bezeichneten Verfahrens ist das Konzept des Pair Programmings (PP).

2.1.1 Pair Programming

Pair Programming ist eine Methode in der Softwareentwicklung, die häufig im Zusammenhang mit agilen Softwareentwicklungsprozessen – wie beispielsweise dem Extreme Programming (XP) [Bec99] – angewendet wird. Dabei arbeiten zwei Entwickler an derselben Aufgabe an einem Rechner zusammen. Während einer der Entwickler (der Driver) die Eingabegeräte bedient und den Quellcode bearbeitet, überwacht der andere Entwickler (der Observer oder Navigator) diese Tätigkeiten, weist frühzeitig auf potentielle Fehler im Quellcode hin, denkt über die Problemlösung im gesamten Kontext nach und unterstützt somit den Driver. Die beiden Rollen sollen regelmäßig getauscht werden, damit die Entwickler das Problem aus unterschiedlichen Blickwinkeln betrachten können.

Zahlreiche Untersuchungen und Experimente zum Pair Programming [WKCJ00; AGDS07; DAS+07; Nos98; HMDK04; LC03; MWBF02] zeigen, dass im Vergleich zur Solo-Programmierung eine höhere Produktivität, bessere Code-Qualität und eine geringere Fehleranfälligkeit erreicht werden kann. Weitere Vorteile sind der

Wissenstransfer zwischen beiden Entwicklern, die schnellere und bessere Teambildung, die gemeinsame Verantwortung für den geschriebenen Code sowie die stärkere Fokussierung auf die zu lösende Aufgabe durch die gegenseitige Kontrolle.

Die Steigerung der Code-Qualität wird vor allem dadurch erreicht, dass Driver und Observer auf verschiedenen Abstraktionsebenen arbeiten, beziehungsweise regelmäßig die Abstraktionsebene wechseln, um sich gemeinsam über ein Problem und dessen Lösung unterhalten zu können. Der Observer bewegt sich beim Kontrollieren des Quellcodes des Drivers eher auf einer unteren Abstraktionsebene (Syntax) und beim Nachdenken über Lösungsansätze für das Problem auf einer hohen Abstraktionsebene. Die Abstraktionsebene des Drivers liegt meist zwischen denen des Observers [BRB08; CH07; FRB07].

Größter Kritikpunkt am Pair Programming sind die Kosten, die durch die doppelte Besetzung zur Lösung einer Aufgabe entstehen, im Vergleich zur höheren Produktivität, wenn beide Entwickler an unterschiedlichen Aufgaben arbeiten. Dabei wird jedoch häufig vernachlässigt, dass der in Einzelarbeit entstandene Code mehr Fehler enthält, deren nachträgliche Behebung zusätzlich Zeit und Ressourcen benötigt und somit insgesamt höhere Kosten verursacht, als durch den Einsatz von Pair Programming entstanden wären [PM03; Hum97].

Experimente, in denen Pair Programming mit Solo-Programmierung verglichen wurden [Nos98; Wil00; WKCJ00; NW01], kommen zu unterschiedlichen Ergebnissen bezüglich des durch Pair Programming verursachten Mehraufwandes. Entscheidende Faktoren für diese Schwankungen sind Erfahrungs- und Wissensunterschiede sowie die Persönlichkeit der Teilnehmer und die Bereitschaft zur Kooperation.

Der Einsatz von Pair Programming setzt die räumliche Nähe der beiden Teilnehmer voraus. Ist diese nicht gegeben oder nicht gewünscht, bietet sich die Anwendung des Distributed Pair Programmings an, sofern die Entwickler diese Entwicklungsmethode dennoch nutzen möchten.

2.1.2 Distributed Pair/Party Programming

Um DPP zu ermöglichen, bedarf es einer softwaregestützten Lösung, die es beiden Teilnehmern ermöglicht eine gemeinsame Sicht auf den zu bearbeitenden Quellcode zu erhalten. Die Rechner beider Teilnehmer müssen dazu über ein Netzwerk miteinander verbunden sein.

Grundsätzlich existieren zwei Ansätze, um den Teilnehmern eine gemeinsame Sicht zu ermöglichen. Beim Collaboration Transparency genannten Ansatz wird die

gesamte Sicht des Drivers auf den Bildschirm des Observers übertragen. Dazu werden spezielle Desktop Sharing Programme wie beispielsweise Microsofts Remote Desktop oder die Open-Source-Software VNC (Virtual Network Computing) genutzt, die den Bildschirminhalt des einen Rechners auf den anderen übertragen und meist auch eine Interaktion mit dem entfernten Bildschirm ermöglichen. Nachteil beim Einsatz dieser Software ist die benötigte hohe Bandbreite zwischen den Rechnern und dass für die Entwicklung weiterhin Programme und Werkzeuge verwendet werden, die nur von einem Benutzer zur Zeit bedient werden können.

Den alternativen Ansatz stellen Werkzeuge dar, die für die gleichzeitige Verwendung und Bedienung von mehreren Benutzern konzipiert worden sind. Programme, die dem so genannten Collaboration Awareness Ansatz folgen [DB92; LL90; Han04], bieten meist eine spezielle Benutzeroberfläche, welche die Aktivitäten der anderen Teilnehmer visualisiert und den Inhalt der bearbeiteten Dokumente synchronisiert. Beispiele für derartige Echtzeit-Gruppeneditoren sind SubEthaEdit [PW03], Gobby [KGHB05] und Saros [ST06].

Vorteile des DPP im Vergleich zum Pair Programming entstehen durch die Möglichkeit des parallelen Arbeitens an zwei Rechnern aber dennoch an derselben Aufgabe. So kann der Observer beispielsweise die Dokumentation einer Bibliothek nachschlagen und so dem Driver schnell Hilfestellungen zur Verwendung dieser Bibliothek geben [SWN+03]. Zudem kann jeder Teilnehmer seinen Rechner personalisiert einrichten, so dass sich der Entwickler bei einem Wechsel der Rollen nicht mehr an die Eigenheiten des verwendeten Systems gewöhnen muss [Bec99], z. B. Mausgeschwindigkeit, Tastaturwiederholrate, etc.

Nachteilig ist die Abhängigkeit des DPP von einem funktionierenden, schnellen Netzwerk sowie der korrekten Funktionsweise des verwendeten Werkzeugs. Je nach Funktionsumfang der genutzter Software können weitere Nachteile entstehen, wenn beispielsweise Sprachkommunikation oder Videokommunikation nicht unterstützt werden und somit wichtige soziale Aspekte, die bei der Zusammenarbeit am selben Ort bestünden, wegfallen [SWN+03].

2.1.3 Side-By-Side Programming

Eine etwas losere Variante des Pair Programmings ist das von Alistair Cockburn beschriebene Side-by-Side Programming [Coc04]. Im Gegensatz zum Pair Programming stehen beiden Entwicklern eigene Arbeitsplätze mit Rechner zur Verfügung, die sich aber sehr nahe beieinander befinden müssen. Die Entwickler arbeiten da-

bei weitestgehend eigenständig, können aber zur Lösung eines Problems bei Bedarf zusammenarbeiten, wobei ein Entwickler ohne Aufwand sofort Einsicht auf den Bildschirm des anderen Entwicklers haben kann. Laut den Untersuchungen von Nawrocki et al. ist der Mehraufwand bei dieser Entwicklungsmethode geringer als beim Pair Programming [NJOL05].

Ein Nachteil dieses Verfahrens ist die Beschränkung der teilnehmenden Entwickler auf zwei bis drei Personen, da die nötige räumliche Nähe der Entwickler zueinander inklusive eines Arbeitsplatzes nicht mehr möglich ist.

Beim Distributed Side-by-Side Programming [DASH09] können die Entwickler unter Zuhilfenahme softwaregestützter Lösungen über das Netzwerk zusammenarbeiten. Die sich daraus ergebenden Nachteile sind analog zu denen des DPP.

2.1.4 Entwicklung Saros

Saros wurde 2006 von Riad Djemili im Rahmen einer Diplomarbeit [Dje06] entwickelt und anschließend durch weitere Studien-, Bachelor- und Diplomarbeiten erweitert und verbessert.

Zu der Basis-Implementierung von Riad Djemili, die das Initiieren einer Saros-Sitzung und das Synchronisieren eines Projektes auf mehreren Rechnern ermöglicht, hat Björn Gustavs [Gus07] den Mehr-Observer-Modus sowie die Visualisierung von Markierungen durch den Observer hinzugefügt. Oliver Rieger [Rie08] implementierte die Jupiter-basierte Nebenläufigkeitskontrolle, um einen Mehr-Driver-Modus umzusetzen, und integrierte Punkt-zu-Punkt-Verbindungen zwischen den Teilnehmern, um große Projekte schnell synchronisieren zu können.

Die anschließenden Arbeiten von Christoph Jacob [Jac09] und Marc Rintsch [Rin09] dienten vor allem der Verbesserung der bestehenden Implementierung, da diese eine Reihe gravierender Fehler aufwies, die den Produktiveinsatz der Software verhinderten. Zudem wurden weitere Präsenzinformationen über die geöffnete Dateien oder die Position innerhalb einer Datei der entfernten Teilnehmer in die Oberfläche integriert. Durch die Entwicklungen von Sebastian Ziller [Zil09] und Sandor Szücs [Sz9] wurden weitere Probleme bezüglich der nebenläufigen Bearbeitung von Dokumenten und der Netzwerk- und Sicherheitsaspekte behoben.

Auf dieser Grundlage konnte daraufhin neue Funktionalität wie der nebenläufige Einladungsprozess von Tas Sóti [SÓ9], der Multi-User-Chat und Sprach-Chat von Olaf Loga [Log10] und das Screensharing von Stephan Lau [Lau10] implementiert werden.

2.1.5 Funktionsweise von Saros

Um eine Saros-Sitzung zu initiieren, müssen alle Teilnehmer zunächst Eclipse mit der installierten Saros-Erweiterung auf ihrem Rechner starten und mit ihrem XMPP-Nutzerkonto an einem XMPP-Server angemeldet sein (näheres in Abschnitt 2.4). Der Initiator einer Saros-Sitzung kann nun ein existierendes Projekt teilen und einen oder mehrere seiner Kontakte zur Mitarbeit an diesem Projekt einladen. Nach Abschluss des Einladungsprozesses haben alle eingeladenen Teilnehmer eine vollständige Kopie des Projektes und befinden sich in der Rolle des Observers im so genannten Verfolger-Modus. Der Initiator fällt automatisch in die Rolle des Drivers.

Vom Driver geöffnete Dateien werden im Verfolger-Modus auch bei den Observern geöffnet, und es wird an die entsprechende Position innerhalb der Datei gesprungen, an der sich der Driver mit seinem Cursor befindet. Durch Annotationen im *Package-Explorer* und am rechten Rand des Editor-Fensters kann der Observer immer erkennen, welche Dateien gerade geöffnet sind und welcher Abschnitt einer Datei gerade vom Driver bearbeitet wird.

Markiert ein Observer eine Stelle im Quellcode mit der Maus, so wird dies durch farbliche Hinterlegung des markierten Bereiches auch für den Driver und weitere Observer sichtbar. Verlässt der Observer den Verfolger-Modus kann er sich frei innerhalb des Projektes bewegen, sieht aber weiterhin durch die Annotationen, wo sich alle anderen Teilnehmer gerade befinden.

Alle vom Driver durchgeführten Änderungen an den Dateien werden sofort für die weiteren Teilnehmer sichtbar und zudem farblich hinterlegt (Abb.: 2.1).

Als Kommunikationsmittel stehen den Teilnehmern ein Multi-User-Chat sowie zwischen jeweils zwei Teilnehmern ein Sprach-Chat zur Verfügung. Werden für die Entwicklung auch Werkzeuge außerhalb von Eclipse benötigt, so können diese den anderen Teilnehmern über das integrierte Screensharing sichtbar gemacht werden.

Durch diese technische Umsetzung ermöglicht Saros alle wesentlichen Aspekte des Pair Programmings beziehungsweise des Distributed Pair Programmings im Rahmen des Collaboration Awareness Ansatzes.

Darüber hinaus hat der Initiator die Möglichkeit weiteren Teilnehmern die Driver-Rolle zuzuordnen, so dass auch die parallele Bearbeitung derselben oder unterschiedlicher Dateien im Projekt durchführbar ist. Mit der in Saros implementierten Nebenläufigkeitskontrolle wird in diesem Fall sichergestellt, dass die gleichzeitigen Änderungen mehrerer Driver in der richtigen Reihenfolge und konsistent auf die

bearbeiteten Dateien angewendet werden.

Das parallele Arbeiten mehrerer Driver an einem Projekt ist eine Umsetzung des in Abschnitt 2.1.3 beschriebenen Side-by-Side Programming beziehungsweise des Distributed Side-by-Side Programming.

Für die Übertragung der Projekte, der Übermittlung der Präsenzinformationen und Aktivitäten sowie zur Bereitstellung der Kommunikationsmöglichkeiten nutzt Saros das im Folgenden beschriebene XMP-Protokoll.

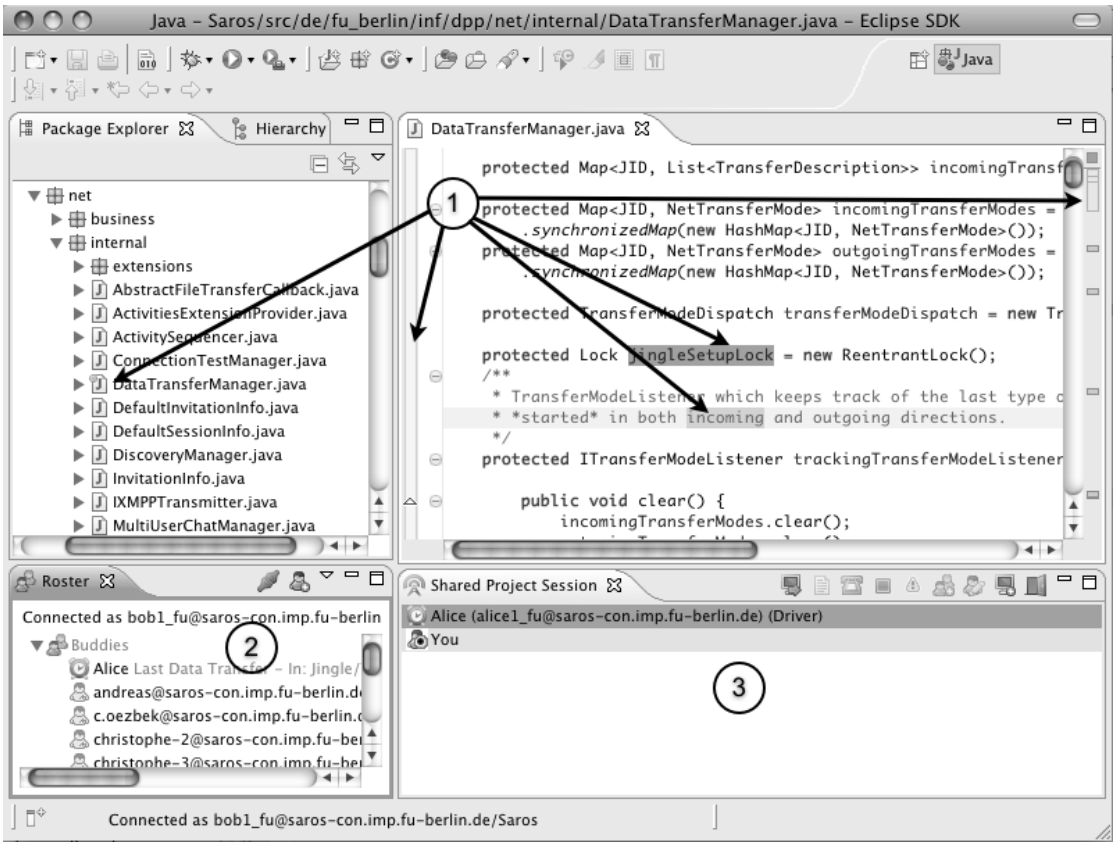

Abbildung 2.1: Eclipse-Fenster mit integriertem Saros. (1) zeigt die Präsenzinformationen anderer Nutzer durch Markierungen im Quellcode, Annotationen am Dokumentrand sowie an geöffneten Dateien im *Package-Explorer*. (2) zeigt die Kontaktliste und welche der Kontakte online sind. Im Session-View (3) werden Teilnehmer einer Saros-Sitzung mit deren Rollen angezeigt.

2.2 XMPP

Jeremie Miller begann 1998 mit der Entwicklung der ersten Jabber-Server-Implementierung, und damit der ersten Implementierung einer rudimentären Version des späteren XMP-Protokolls, aus dem Bedürfnis heraus, eine freie Spezifikation für einen Instant-Messaging-Dienst zu Verfügung zu stellen. Zum damaligen Zeitpunkt gab es bereits eine Reihe kostenloser Instant-Messaging-Dienste, die aber alle auf proprietären Protokollen der jeweiligen Entwicklerfirmen basierten. Zu den bekanntesten Diensten zählen ICQ von Mirabilis (heute Digital Sky Technologies) [Mir96], AIM von AOL [AOL96], MSN Messenger (heute Windows Live Messenger, WLM) von Microsoft [Mic99] und Yahoo Messenger von Yahoo! [Yah98]. Benutzer eines dieser Netzwerke hatten zudem keine Möglichkeit mit den Teilnehmern eines anderen Netzwerkes Kontakt aufzunehmen, so dass häufig mehrere Instant-Messaging-Klienten gleichzeitig genutzt werden mussten oder der Klient eine Vielzahl an Protokollen zu implementieren hatte. Bei diesen Multi-Protokoll-Klienten forderten Änderungen an einem der Protokolle immer wieder eine Aktualisierung der Software, ohne die ein Teil der Netzwerke nicht nutzbar war. Allen Protokollen war eine monolithische Infrastruktur gemeinsam, was zum einen zu Skalierungsproblemen und zum anderen zu Sicherheitsproblemen für Firmen, die einen internen Instant-Messaging-Dienst nutzen wollten, führte [Ada02].

Um den 1999 erschienenen Jabber-Server *jabberd* entstand schnell eine Open-Source-Community, die Klienten für Linux, Mac OS und Windows entwickelte und gemeinsam die Spezifikation des Protokolls verfeinerte. Nachdem sich auch Firmen an der Entwicklung von Jabber-kompatibler Software beteiligten, wurde 2001 die Jabber Software Foundation von der Firma Jabber, Inc. gegründet, um die weitere Entwicklung zu koordinieren.

2004 wurde das nunmehr weit entwickelte und bereits bewährte Protokoll in Zusammenarbeit mit der Internet Engineering Task Force (IETF) als Teil der Request for Comments (RFC) Reihe standardisiert und reiht sich seitdem in die Liste der Internet-Kern-Technologien neben TCP/IP, HTTP, SMTP, POP und IMAP ein [Int04a; Int04b]. Im Zuge der Standardisierung wurde das Protokoll von Jabber (deutsch: plappern) in Extensible Messaging and Presence Protocol (XMPP) umbenannt.

Anschließend setzten viele große Firmen wie Google, Apple, Cisco, IBM, Nokia und Sun XMPP als Basis vieler Produkte und Dienste ein. Die Anfang 2007 in

XMPP Standards Foundation (XFS) umbenannte Jabber Software Foundation, die von vielen dieser Firmen finanziell unterstützt wird, entwickelt die Spezifikation in Form der XMPP Extension Protocols (XEP) weiter [SAST09].

2.2.1 Infrastruktur

XMPP verwendet eine dezentralisierte Klient-Server-Architektur ähnlich der des E-Mail-Netzwerks [SAST09]. Um eine Nachricht von Teilnehmer A zu Teilnehmer B zu senden, müssen sich beide zunächst gegenüber einem XMPP-Server authentifizieren, wobei dies für A und B unterschiedliche XMPP-Server sein können. Zum Senden der Nachricht wird diese von A an den mit ihm verbundenen XMPP-Server übertragen. Dieser leitet die Nachricht weiter an den XMPP-Server von B, der sie anschließend an den Empfänger B ausliefert. Im Gegensatz zum E-Mail-Netzwerk werden Nachrichten in diesem XMPP-Federation genannten Verbund von XMPP-Servern immer direkt und nicht über mehrere Zwischenstationen an den XMPP-Server des Empfängers gesandt (Abb.: 2.2).

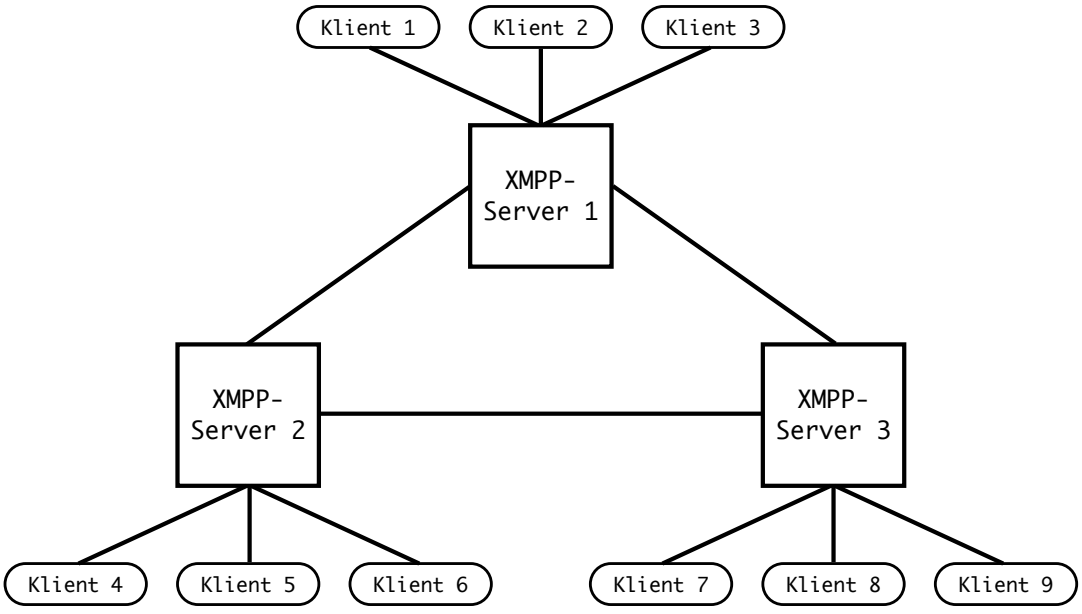

Abbildung 2.2: Jeder Klient ist mit einem XMPP-Server verbunden. Nachrichten können über die Verbindungen zwischen den XMPP-Servern zwischen allen Klienten ausgetauscht werden. Innerhalb der XMPP-Federation sind alle XMPP-Server untereinander verbunden.

Diese Architektur bietet – verglichen mit reinen Punkt-zu-Punkt-Netzwerken – eine klare Trennung der Aufgabenbereiche (separation of concerns), so dass sich Entwickler von Klienten-Software mehr auf die Benutzbarkeit und Server-Entwickler auf die Zuverlässigkeit und Skalierbarkeit des Systems fokussieren können. Außerdem gibt es keinen Single-Point-of-Failure wie bei proprietären Instant-Messaging-Diensten, da jeder seinen eigenen XMPP-Server betreiben kann und ein Ausfall eines Servers nicht zum Ausfall des ganzen Netzwerkes führt.

Das Konzept der Transports oder auch Gateways ermöglicht eine Anbindung des XMPP-Netzwerkes an andere Netzwerke wie ICQ oder Yahoo!IM, ohne dass die notwendigen Protokolle auf Klientenseite unterstützt werden müssen. Der XMPP-Server leitet Nachrichten dabei an einen XMPP-Transport, der diese dann über das Protokoll des anderen Netzwerkes zum Empfänger schickt. Dies geschieht für Nutzer beider Netzwerke transparent.

2.2.2 Protokoll

Jeder Teilnehmer am XMPP-Netzwerk hat eine eindeutige Adresse, die so genannte JabberID (JID), mit welcher der Empfänger einer Nachricht angegeben werden kann. Diese setzt sich aus einem Benutzernamen, dem Servernamen und einer Ressource (z.B. „henning.staib@jabber.org/Saros“) zusammen. Die Ressource ist eine beliebige Zeichenkette, um zwischen mehrfachen Verbindungen desselben Nutzers von verschiedenen Standorten oder mehreren Klienten unterscheiden zu können.

Der Datenaustausch zwischen dem Klienten und dem XMPP-Server findet über einen XML-Datenstrom statt. Über eine langlebige TCP-Verbindung werden zwei beliebig lange XML-Dokumente für die eingehende und ausgehende Kommunikation zwischen Klient und Server ausgetauscht. Solch ein XML-Dokument kann drei Arten von Unterelementen enthalten: `<message>`-Tags zum Senden und Empfangen von Nachrichten, `<presence>`-Tags zum Bekanntmachen, ob ein Kontakt online oder offline ist, und `<iq>`-Tags (Info/Query), die eine Anfrage-Antwort-Interaktion erlauben und zum Austausch von Informationen zwischen Klienten oder zwischen Server und Klient dienen.

```
<stream:stream>
  ...
  <presence/>
  <iq type="get">
    <query xmlns="jabber:iq:roster"/>
  </iq>




  <message from="alice@jabber.org"
           to="bob@jabber.org">
    <body>How are you?</body>
  </message>
  <presence type="unavailable"/>
</stream:stream>
```

Listing 2.1: XML-Strom Klient – Server

```
<stream:stream>
  ...

  <iq type="result">
    <query xmlns="jabber:iq:roster">
      <item jid="bob@jabber.org"/>
      <item jid="carl@jabber.org"/>
    </query>
  </iq>

  <message from="bob@jabber.org"
           to="alice@jabber.org">
    <body>I'm fine</body>
  </message>

</stream:stream>
```

Listing 2.2: XML-Strom Server – Klient

Listing 2.1 zeigt den XML-Datenstrom vom Klienten zum Server. Der Klient sendet nach dem Öffnen des Stroms ein `<presence>`-Tag welches signalisiert, dass er online ist. Anschließend folgt eine Anfrage an den XMPP-Server nach der Kontaktliste. Nach Erhalt der Antwort sendet er eine Textnachricht, geht wieder offline und schließt den Datenstrom. In Listing 2.2 ist ein Ausschnitt des XML-Dokumentes,

welches vom XMPP-Server zum Klienten gesendet wird, abgebildet. Es enthält zunächst das Ergebnis der Anfrage nach der Kontaktliste und danach die Antwort auf die Textnachricht, die von einem anderen Klienten gesendet wurde.

Die weiteren Kindelemente dieser drei als XMPP-Stanzas bezeichneten Tags sowie deren Semantik werden durch die Kernspezifikation als auch durch die Erweiterungen (XEPs) definiert. Um zwischen verschiedenen Erweiterungen unterscheiden zu können, sind die Kindelemente jeweils mit eigenen XML-Namespaces versehen (z.B. `<jingle xmlns='urn:xmpp:jingle:1'>`). Listing 2.3 zeigt eine XMPP-Nachricht mit einer beispielhaften Erweiterung.

```
<message from="alice@jabber.org"
         to="bob@jabber.org">
  <body>How are you?</body>
  <foo xmlns="urn:xmpp:examle:foobar">
    <bar/>
  </foo>
</message>
```

Listing 2.3: XMPP-Nachricht mit der Beispielerweiterung „foobar“

Die Kernspezifikation umfasst die Definitionen zur Netzwerkarchitektur, zu Authentisierungsmechanismen zwischen Klient und Server sowie zwischen zwei Servern, zur Internationalisierung (I18N) wie auch zur Verwaltung der Kontaktliste. Um über einen anderen XMPP-Teilnehmer Statusinformationen durch `<presence>`-Pakete erhalten zu können, ist in der Kernspezifikation weiterhin ein Protokoll angegeben, in dem beide Teilnehmer das Hinzufügen des jeweils anderen zu ihrer als Roster bezeichneten, serverseitig gespeicherten Kontaktliste authorisieren müssen.

2.2.3 XMPP Extension Protocols

Es folgt ein kurzer Überblick über die im Rahmen dieser Diplomarbeit wichtigen XMPP-Erweiterungen.

Service Discovery XEP-0030 *Service Discovery* [HMESA08] spezifiziert ein Protokoll, um Informationen zum XMPP-Server oder anderen Teilnehmern am XMPP-Netzwerk zu erhalten. Die Informationen umfassen die Identität in Form einer Kategorie und eines Typs (z. B. Chat-Räume), die unterstützen Erweiterungen und Protokolle sowie weitere dem Angefragten zugehörige Dienste wie beispielsweise Proxy-Server oder Übersetzungsdienste.

SOCKS5 Bytestreams Mit XEP-0065 *SOCKS5 Bytestreams* [SMSAK10] wird ein Protokoll definiert, um einen Binärdatenstrom außerhalb des XMPP-XML-Datenstroms zwischen zwei Teilnehmern einzurichten. Diese Verbindung kann entweder über eine Punkt-zu-Punkt-Verbindung oder über eine vermittelte Verbindung durch einen SOCKS5-Proxy-Server geschehen. Das in RFC1928 [Int96] beschriebene SOCKS5-Protokoll dient dabei der Zuordnung der beiden Verbindungen der Teilnehmer zum Proxy-Server, um Daten austauschen zu können. Eine genauere Erläuterung des Protokolls findet sich in Kapitel 5.1.

In-Band Bytestreams Die Spezifikation XEP-0047 *In-Band Bytestreams* [KSA09] beschreibt ein Protokoll, um einen Binärdatenstrom innerhalb des XMPP-XML-Datenstroms aufzubauen. Die zu sendenden Daten werden in Blöcke fester Größe geteilt und als Nutzlast eines XMPP-Stanzas verschickt. Kapitel 6.1 geht näher auf die Details des Protokolls ein.

Stream Initiation Die Erweiterung XEP-0095 *Stream Initiation* [MME04a] stellt ein allgemeines Protokoll für die Einrichtung eines Datenstroms zwischen zwei XMPP-Teilnehmern dar. Dabei können zunächst die Art der Daten sowie die zu nutzenden Transportmethoden ausgehandelt werden.

Stream Initiation File Transfer Durch XEP-0096 *Stream Initiation File Transfer* [MME04b] wird die allgemeine Erweiterung XEP-0095 *Stream Initiation* für den Austausch von Dateien präzisiert. Das Protokoll erlaubt dabei die Angabe von Informationen über die zu sendende Datei sowie über die zu verwendende Transportmethode. Derzeit unterstützte Transportmethoden sind *SOCKS5 Bytestreams* als Standardtransport und *In-Band Bytestreams* als Ausweichlösung, falls erstere fehlschlägt.

Jingle Die Erweiterung XEP-0166 *Jingle* [LBSA+09] stellt ein Protokoll zur Verfügung, um Punkt-zu-Punkt-Multimedia-Sitzungen zwischen zwei XMPP-Teilnehmern zu ermöglichen. In Anlehnung an das in der IP-Telefonie häufig verwendete *Session Initiation Protocol* (SIP) können Typ und Parameter der Sitzung sowie die möglichen Transportmethoden ausgehandelt werden. Das Protokoll bietet – verglichen mit dem *Stream Initiation*-Protokoll – mehr Flexibilität beim Aushandeln der Parameter und ermöglicht zudem diese während einer laufenden Sitzung zu modifizieren.

Auf dieser allgemeinen Spezifikation aufbauend gibt es Erweiterungen zum Initiieren von Sprach- und Video-Chat-Sitzungen (XEP-0167 *Jingle RTP Sessions* [LSAE+09]) wie auch zum Austausch von Dateien (XEP-0234 *Jingle File Transfer* [SA10a]).

Daneben gibt es Erweiterungen, welche die unterschiedlichen Transportmethoden beschreiben. Darunter XEP-0177 *Jingle Raw UDP Transport Method* [BSAL+09] zum Einrichten eines Datenstroms über ein verbindungsloses Protokoll, XEP-0176 *Jingle ICE-UDP Transport Method* [BLSA+09], das zusätzlich Network Address Translator (NAT) Traversel integriert, XEP-0260 *Jingle SOCKS5 Bytestreams Transport Method* [SAMK+10], das eine erweiterte Variante der bereits beschriebenen XEP-0065 zur Verwendung im *Jingle*-Kontext definiert, und XEP-0261 *Jingle In-Band Bytestreams Transport Method* [SA10b], das analog zur XEP-0047 die Daten innerhalb des XMPP-XML-Stroms überträgt.

2.3 Smack

Smack ist eine Open-Source-XMPP-Klienten-Bibliothek für Java, deren Entwicklung 2006 von der Firma *Jive Software* begonnen und unter der Apache 2.0 Lizenz veröffentlicht wurde. Die Bibliothek bietet eine abstrahierte, objektorientierte Schnittstelle, um als Klient mit dem XMPP-Netzwerk zu kommunizieren. Dabei übernimmt Smack das Serialisieren und Deserialisieren zwischen den XML-Fragmenten und den als `Packet` bezeichneten Java-Objekten.

Neben der fast vollständigen Implementierung der XMPP-Kernspezifikationen (siehe Kapitel 7.4 und 7.5) existieren auch Schnittstellen für einige der XEPs, darunter *Service Discovery*, Multi-User-Chat, XHTML-Nachrichten, Dateitransfer, Ad-Hoc-Commands, um Funktionen bei einem anderen Klienten aufzurufen, Teilnehmersuche und Publish-Subscribe, um Informationen und Ereignisse für viele XMPP-Teilnehmer bekannt zu machen. Darüber hinaus bietet Smack eine Schnittstelle, um eigene Erweiterungen des XMP-Protokolls zu implementieren.

Näheres zur Entwicklungsgeschichte und der Open-Source-Community findet sich in Kapitel 4.1.

2.4 Nutzung von Smack in Saros

Saros stützt sich auf Smack, um die Netzwerkschicht, die für den Austausch von Daten zwischen den Saros-Nutzern zuständig ist, zu implementieren. So basiert die Kontaktliste und deren Verwaltung auf der `Roster`-Schnittstelle von Smack, welche Methoden zum Hinzufügen und Entfernen von Kontakten anbietet sowie Abfragen zu den Statusmeldungen der Kontakte ermöglicht. Ob ein Kontakt Saros unterstützt, wird über die *Service Discovery* ermittelt.

Die Synchronisation eines Projektes zu Beginn einer Saros-Sitzung erfolgt mit Hilfe der *Jingle*-Schnittstelle, über die eine Punkt-zu-Punkt-Verbindung hergestellt und darüber alle benötigten Daten übertragen werden können. Tritt beim Initiieren dieser *Jingle*-Verbindung ein Fehler auf, so wird anschließend versucht die Daten über Smacks *File Transfer*-Schnittstelle zu versenden. Diese Funktionalität zum Austausch größerer Datenmengen wird vom `DataTransferManager` gekapselt, welcher ebenfalls Basis der `StreamService`-Schnittstelle ist, die für die Sprach-Chat- und Screensharing-Funktionen von Saros verwendet wird.

Grundlage des Multi-User-Chats von Saros ist die `MultiUserChat`-Schnittstelle

von Smack, die um die Funktionalität, in einem Eclipse-Fenster angezeigt werden zu können, erweitert wurde.

Zum Austausch der als Aktivitäten bezeichneten Änderungen und Markierungen der Saros-Nutzer in einer Datei sowie für die Präsenzinformationen verwendet Saros eine eigene XMPP-Erweiterung. Die XML-Elemente dieser Erweiterung unter dem XML-Namespace „de.fu_berlin.inf.dpp“ werden dazu in ein `<message>`-Stanza eingebettet und zwischen den Teilnehmern einer Saros-Sitzung versandt. Die Nutzlast dieser XML-Elemente stellen mittels der `XStream`-Bibliothek zu XML serialisierte Java-Objekte dar. Überschreitet die XML-Repräsentation einer Liste von Aktivitäten die maximal zulässige Größe einer XMPP-Nachricht so kann diese auch über den `DataTransferManager` verschickt werden.

In Kapitel 4.3 wird näher erläutert, in welchen Fällen sich durch die Nutzung von Smack Probleme ergeben.

3 Aufgabenstellungen

Die Aufgabenstellungen dieser Diplomarbeit gliedern sich in drei Teile. Der erste Teil ist die zielgerichtete Verbesserung der XMPP-Bibliothek Smack, um deren Einsatz im Projekt Saros insbesondere bei der Projektsynchronisation zu verbessern. Im zweiten Teil sollen Vorschläge für Maßnahmen erarbeitet und umgesetzt werden, um die Community des Open-Source-Projektes Smack zu reaktivieren und somit die Weiterentwicklung voranzutreiben. Der dritte Teil umfasst das Mitwirken an den regelmäßigen Arbeiten im Projekt Saros sowie der Unterstützung weiterer am Projekt beteiligten Studenten.

Verbesserung XMPP-Bibliothek Im Rahmen der Verbesserung der Bibliothek Smack sind Arbeitspakete definiert worden, um seit längerem bestehende Probleme bei der Verwendung von Smack zu lösen.

Um Dateien zwischen zwei XMPP-Teilnehmern zu übertragen, wird das unter anderem das Protokoll *SOCKS5 Bytestreams* genutzt. Aufgrund von Fehlern in der Implementierung schlägt der Aufbau dieser Verbindung jedoch häufig fehl. Des Weiteren ist es nicht möglich, bestimmte Parameter des Protokolls zu konfigurieren oder es gegebenenfalls ganz zu deaktivieren. Kapitel 5 beschreibt diese und weitere Probleme, die bei der Analyse der Implementierung entdeckt wurden und die entsprechenden Lösungen.

Weitere Arbeitspakete behandeln Probleme mit dem Registrieren des Beobachters für Änderungen an der Kontaktliste, dem Trennen der Verbindung vom XMPP-Server aufgrund von Fehlern beim Parsen der XML-Pakete sowie mit der Authentifizierung gegenüber dem XMPP-Server. Die Kapitel 7.1 und 7.4 geben genaue Erläuterungen über die Probleme und Lösungen der ersten beiden Arbeitspakete. Das Arbeitspaket Authentifizierung wurde obsolet, da das Problem bereits von der Smack-Community gelöst wurde.

Im letzten Arbeitspaket wurde evaluiert, warum das Initialisieren des zum Aufbau einer *Jingle*-Sitzung benötigten `JingleFileTransferManager`s eine Dauer von

über 20 Sekunden hat, warum teilweise ungeeignete IP-Adressen für den Verbindungsaufbau ermittelt werden und ob die Unterstützung von *Jingle* für verbindungsorientierte Protokolle verbessert werden kann. Kapitel 8.3 fasst die Ergebnisse dieser Analyse zusammen.

Durch die hohe Komplexität sowohl des *Jingle*-Protokolls als auch dessen Implementierung, lag es nicht mehr im zeitlichen Rahmen dieser Arbeit, Lösungen für die zuletzt genannten Probleme zu implementieren. Stattdessen wurde ein neues Arbeitspaket festgelegt, welches das ebenfalls im Zusammenhang mit der Dateiübertragung verwendete *In-Band Bytestream*-Protokoll analysiert und es analog zur Umsetzung des SOCKS5-Arbeitspaketes als eigenständige Schnittstelle extrahiert.

Smack-Community Unter dem Aspekt der folgenden Fragestellungen sollte erörtert werden, inwiefern eine Weiterentwicklung des Projektes Smack möglich ist:

- Wie kann die Community eines durch eine Firma kontrollierten Open-Source-Projektes reaktiviert werden?
- Welche Erfolgschancen hat eine Abspaltung des Projektes?
- Wie kann die Qualität der Entwicklerbeiträge einer Open-Source-Community gesteigert werden?

Dazu werden in Kapitel 4 zunächst die Probleme des Projektes Smack und seiner Community beleuchtet und anschließend im Abschnitt 4.4 Lösungsansätze aufgeführt. Die Ergebnisse der umgesetzten Vorschläge werden in Kapitel 9 beschrieben.

Saros In der Regel wird alle vier Wochen eine neue Version von Saros veröffentlicht, welche die Verbesserungen und Fehlerbehebungen, die seit dem letzten Release hinzugefügt wurden, enthält. Jedem Release geht eine Testphase voraus, in der die Mitglieder des Saros-Teams eine festgelegte Anzahl definierter Anwendungsfälle, wie beispielsweise den Einladungsprozess oder das gleichzeitige Bearbeiten einer Datei testen. Die Auswahl der Anwendungsfälle ist dabei abhängig von den hinzugefügten Änderungen. Probleme und Fehler, die bei diesen Tests festgestellt werden, müssen dann innerhalb der folgenden zwei Tage berichtigt werden. Anschließend wird ein neues Release erstellt und die Änderung auf der Saros-Mailingliste und der Saros-Projektwebsite veröffentlicht.

Um diesen Prozess zu organisieren, werden den Saros-Team-Mitgliedern vor jedem Release Rollen zugeteilt. Der Release-Manager erstellt vor Beginn der Tests eine Liste aller Neuerungen und Änderungen und sendet diese an die Mailingliste. Zudem ist er für das Durchführen der Schritte zum Erstellen eines Releases innerhalb der Versionsverwaltung sowie für die Veröffentlichung des Releases verantwortlich. Unterstützt wird er dabei durch den Assistent-Release-Manager.

Der Test-Manager wählt anhand der vom Release-Manager zusammengefassten Änderungsliste Anwendungsfälle für die Tests aus. Nach Durchführung der Tests sendet er die Ergebnisse sowie die sich daraus ergebenden Aufgaben an die Saros-Mailingliste. Nach dem Release schließt der Test-Manager die Tickets zu den behobenen Fehlern im Bugtracker. Der Assistent-Test-Manager unterstützt den Test-Manager bei diesen Aufgaben.

Alle weiteren Team-Mitglieder, denen keine dieser Rollen zugeteilt wurde, helfen beim Durchführen der Tests und bei der Behebung von aufgetretenen Fehlern.

Damit eine Änderung am Saros-Quellcode der Versionsverwaltung hinzugefügt werden kann, ist eine Durchsicht dieser Änderung von mindestens einem weiteren Mitglied des Teams notwendig. Der Autor der Änderung veröffentlicht dazu den Patch auf der Mailingliste und bittet um Code-Durchsicht. Der Gutachter eines Patches kommentiert diesen und gibt eine Bewertung ab. Die Bewertung +1 bedeutet, dass keine Mängel an den Änderungen gefunden wurden. Mit +0 wird bewertet, wenn zwar keine Mängel gefunden wurden, der Gutachter sich aber mit dem Kontext des Patches nicht ausreichend auskennt, um dessen Richtigkeit umfassend beurteilen zu können. Werden Mängel bei der Durchsicht festgestellt, wird die Bewertung -1 vergeben, der Patch darf der Versionsverwaltung nicht hinzugefügt werden und muss nach Behebung der Mängel erneut gesichtet werden.

Eine Änderung darf dann hinzugefügt werden, wenn eine +1 Bewertung und eine weitere +1 oder +0 Bewertung vorliegt. Vergeht nach der ersten Bewertung ein längerer Zeitraum, ohne dass eine zweite Code-Durchsicht durchgeführt werden konnte, darf der Patch ebenfalls hinzugefügt werden.

Jeder Student im Saros-Team steht als Ansprechpartner für einen bestimmten Teilbereich von Saros zur Verfügung, um so insbesondere neuen Team-Mitgliedern den Einstieg in das Projekt zu erleichtern und sich gegenseitig zu unterstützen. Durch konstruktive Kritik und Verbesserungsvorschläge soll das Projekt darüber hinaus weiter vorangetrieben werden. Kapitel 8 beschreibt die Rahmen dieser Arbeit durchgeführten Tätigkeiten in diesem Bereich.

4 Probleme von Smack

Ursachen für die Probleme die den Einsatz von Smack in Saros erschweren, liegen sowohl in der Entwicklungsgeschichte der Open-Source-Community um das Smack-Projekt als auch in der Vielzahl bekannter und nicht behobener technischer Probleme welche im Folgenden näher erläutert werden.

4.1 Community

Das Open-Source-Projekt Smack wurde 2002 von Matt Tucker, dem Mitbegründer und technischen Leiter der Firma *Jive Software* initiiert. Seit 2006 wird Smack und die Community um das Projekt über die Plattform *Ignite Realtime* von *Jive Software* organisiert. Neben Smack finden sich auf *Ignite Realtime* weitere Open-Source-Projekte, die eine auf XMPP basierende Echtzeit-Kollaboration ermöglichen, darunter Openfire (eine Java-basierte XMPP-Server-Implementierung), Spark (ein auf Smack basierender Instant-Messaging-Klient), XIFF (eine Flash XMPP-Bibliothek) sowie SparkWeb (ein auf XIFF basierender Instant-Messaging-Klient für Flash).

Bis 2008 wurden diese Projekte hauptsächlich von *Jive Software* weiterentwickelt, da sie Teil der vom Unternehmen verkauften Kommunikationslösungen für Firmenkunden waren. Seit 2008 konzentriert sich das Geschäft vor allem auf den Vertrieb der auf Java Enterprise Edition basierenden Kollaborations- und Wissensmanagement-Software Jive SBS (Social Business Software). Die Weiterentwicklung der *Ignite Realtime*-Projekte wird seitdem fast vollständig von der Community vorangetrieben.

Da die Entwicklung der Projekte zunächst maßgeblich von *Jive Software* umgesetzt wurde, hat sich keine entsprechend große und aktive Community um die Projekte entwickeln können. So wird die Organisation der Community und der Projekte innerhalb der nicht öffentlichen Gruppe „Community Planning“ verwaltet, die derzeit 23 Mitglieder hat, von denen aber nur etwa ein Drittel aktiv ist.

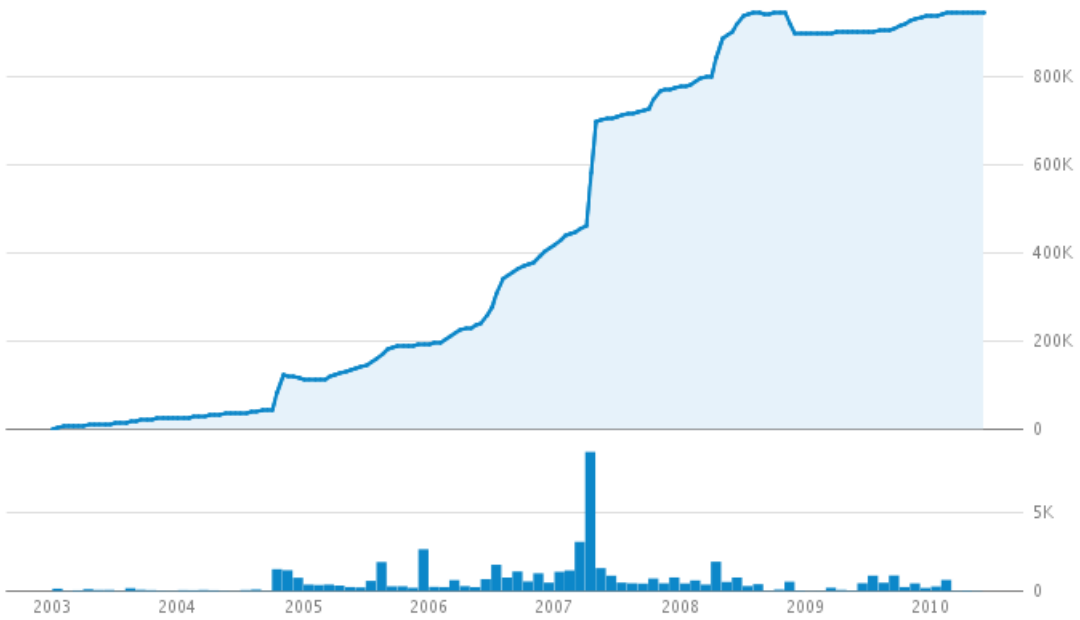

Abbildung 4.1: Das obere Diagramm zeigt die Entwicklung der Anzahl der Code-Zeilen in der Versionsverwaltung für alle *Ignite Realtime*-Projekte. Das untere Diagramm zeigt die Anzahl der *Commits* in die Versionsverwaltung. In beiden ist eine Stagnation seit Mitte 2008 zu erkennen.

Während der Großteil der Aktivitäten im Projekt Openfire stattfindet, stagniert die Weiterentwicklung der anderen Projekte. Das letzte offizielle Release von Smack war im Oktober 2008, von Spark im November 2007, von XIFF im April 2008, von SparkWeb im Juli 2008 und von Openfire im Mai 2009 (Abb.: 4.1).

Gründe für diesen Stillstand sind zum einen das Fehlen konkreter Zielsetzungen und Planungen (Roadmap) zur Weiterentwicklung der Projekte, zum anderen die mangelnde Beachtung von Forumsbeiträgen und Entwicklerbeiträgen von Mitgliedern außerhalb des Kernteams. Als Konsequenz werden Patches oft nicht durchgesehen und finden somit selten Eingang in den Hauptentwicklungszweig (trunk) der Quellcode-Basis.

Zudem gibt es keine Anleitung oder Dokumentation, wie und inwiefern ein Beitritt zum Entwickler-Team eines Projektes möglich ist, welche Vorgaben und Kriterien ein Entwicklerbeitrag erfüllen muss, um in die Quellcode-Basis aufgenommen zu werden, und welcher Aufgabenbereich den einzelnen Mitgliedern des Entwickler-Teams zugeteilt ist. Infolgedessen gibt es große Hürden für neue Entwickler sich an dem Projekt zu beteiligen, was sich wiederum negativ auf die Motivation, über-

haupt einen Beitrag zu leisten, auswirkt. Abbildung 4.2 zeigt das sinkende Interesse am Smack-Projekt.

Da sich die *Jive Software*-Entwickler aus den Projekten zurückgezogen haben und somit nicht mehr als Ansprechpartner zur Verfügung stehen, fehlt den aktiven Mitgliedern häufig auch das entsprechende Wissen über bestimmte Komponenten der Software, so dass die Durchsicht eines Patches viel Zeit in Anspruch nimmt, da zunächst der Kontext und die bestehende Implementierung verstanden werden müssen.

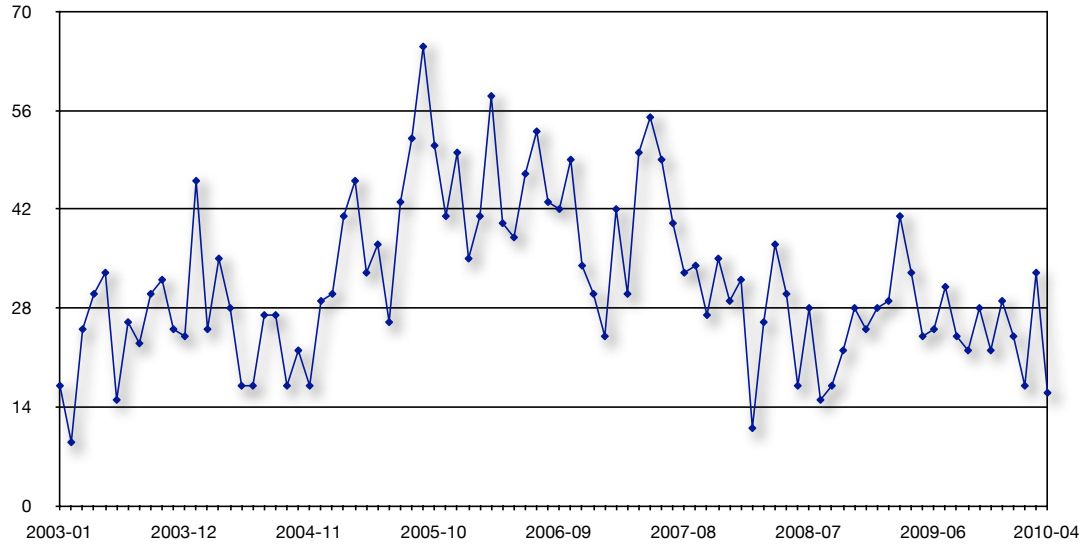

Abbildung 4.2: Dargestellt ist die Anzahl neuer Beiträge im Smack-Forum pro Monat. Für die letzten Jahre ist eine kontinuierliche Abnahme der Aktivität zu sehen.

4.2 Technische Probleme

Obwohl Smack die derzeit am weitesten fortgeschrittene freie Java-Implementierung für XMPP-Klienten ist, gibt es eine Vielzahl offener technischer Probleme. Aufgrund der geringen Aktivität bei der Weiterentwicklung haben sich bereits über 70 offene Probleme im Bugtracker von *Ignite Realtime* für Smack angesammelt. Diese umfassen Fehler und Speicherlecks im Multi-User-Chat, Mängel beim Parsen des XML-Datenstroms, Probleme bei der Authentifizierung mit dem XMPP-Server, Verbesserungsvorschläge für die API sowie Anliegen zur Implementierung weiterer XMPP-Erweiterungen.

Die Spezifikation einiger der bereits implementierten Erweiterungen wurden mittlerweile von der XSF aktualisiert und geändert, so dass Smack nun eine veraltete oder unvollständige und teilweise inkorrekte Umsetzung dieser Erweiterungen beinhaltet. Beispiele dafür sind die Spezifikationen für *In-Band Bytestreams* (XEP-0047) und *Jingle* (XEP-0166).

Problematisch beim Beheben von Fehlern ist die mangelnde Code-Dokumentation, die inkonsistente Strukturierung von Klassen und Packages, Code-Wiederholungen, unterschiedliche Code-Formatierungen sowie uneinheitliche Verwendung von Sichtbarkeitsmodifikatoren.

Zwar wird die Implementierung durch ca. 250 Unit-Tests überprüft, den Großteil davon stellen jedoch die in Kapitel 5.4 näher beschriebenen Tests mit einer Abhängigkeit zu einer XMPP-Server-Implementierung dar. So schlagen ca. 10 % dieser Test in Kombination mit einem lokal installierten Openfire XMPP-Server fehl.

4.3 Auswirkungen der Probleme auf Saros

Ein Teil der offenen Probleme von Smack wirkt sich auch auf die Funktionalität von Saros aus. Beim Hinzufügen und Entfernen von Kontakten zur Kontaktliste werden zur Bestätigung dieser Anfragen zwischen den Teilnehmern XMPP-Nachrichten ausgetauscht. Ist einer der Teilnehmer nicht online, werden diese auf dem XMPP-Server zwischengespeichert. Verbindet sich nun dieser Teilnehmer mit dem XMPP-Server, kann es vorkommen, dass der Klient über diese Nachrichten nicht informiert wird und sie somit verloren gehen. In Kapitel 7.1 wird näher auf dieses Problem und dessen Lösung eingegangen.

Schwerwiegender sind jedoch die Probleme im Zusammenhang mit dem Aus-

tausch von Binärdaten zwischen den Teilnehmern einer Saros-Sitzung. Saros nutzt dazu die *Jingle*-API und die *File Transfer*-API von Smack. Obwohl in der Spezifikation vorgesehen, unterstützt die Smack-Implementierung von *Jingle* keine verbindungsorientierten Protokolle wie zum Austausch von Dateien nötig. Daher findet sich in Saros eine eigenständige Implementierung zum Herstellen einer Verbindung zum Datenaustausch. Aufgrund der schwierigen Handhabung der *Jingle*-API hat diese Implementierung eine recht hohe Komplexität, ohne dabei die Möglichkeiten des *Jingle*-Protokolls vollständig auszunutzen. Ein weiteres Problem entsteht durch die von Smack im Rahmen der *Jingle*-Implementierung verwendete Bibliothek JSTUN. Diese dient zum Bestimmen von Netzwerkadressen, die für den Verbindungsaufbau verwendet werden können. Zum einen sind die dabei ermittelten IP-Adressen nicht immer die richtigen, zum anderen benötigt JSTUN über 20 Sekunden zum Initialisieren.

Obwohl die im Zusammenhang mit der *File Transfer*-API verwendeten Übertragungsmethoden *SOCKS5 Bytestream* und *In-Band Bytestream* das Herstellen eines beliebig langen Datenstroms ermöglichen, wird diese Funktionalität von Smack nicht zur Verfügung gestellt. Um einen solchen Datenstrom zu simulieren, baut Saros mit Hilfe der *File Transfer*-API immer wieder neue Sitzungen auf, da diese Schnittstelle nur einen Datenstrom begrenzter Länge zulässt. Zudem treten beim Herstellen dieser Dateiübertragungsverbindungen häufig Fehler auf, die zur Folge haben, dass in Saros der Einladungsprozess fehlschlägt. Die Kapitel 5 und 6 beschreiben, wie dieses Problem gelöst worden ist.

4.4 Lösungsansätze

Obwohl Smack seit seinem Beginn ein Open-Source-Projekt ist, stellen sich ähnliche Herausforderungen an die Community wie für Spinout Projekte [WG04]. Dabei handelt es sich um ein von einer Firma entwickeltes und später unter einer Open-Source-Lizenz veröffentlichtes Projekt wie beispielsweise Mozilla von Netscape oder der Java-Compiler Jikes von IBM. Nachdem sich 2008 die *Jive Software*-Entwickler weitestgehend aus dem Smack-Projekt zurückgezogen haben, ohne dass eine aktive Community entstanden ist, stellen sich ähnliche Probleme dar wie von West et al. beschrieben [WO05].

Da die Community nicht wie bei klassischen Open-Source-Projekten zusammen mit dem Projekt gewachsen ist, entstehen Schwierigkeiten neue Mitglieder zu wer-

ben, zu motivieren und bei ihnen ein Gefühl für Verantwortung und Anteil am Projekt zu entwickeln. Erschwerend kommt hinzu, dass die bestehende, große und komplexe Code-Basis neuen Entwicklern erklärt und die verwendeten Designprinzipen und Entscheidungen transparent gemacht werden müssen [WO05]. Des Weiteren sollte es ein Konzept geben, wie neue Entwickler der Community beitreten können, welche Aufgaben sie wahrnehmen können und in welche Richtung sich das Projekt entwickeln soll.

Zu Beginn der Bearbeitung der vorliegenden Diplomarbeit waren diese Probleme weder von *Jive Software* noch von der Community gelöst. Um Impulse zur Bewältigung dieser Aufgaben geben zu können, sollte zunächst erreicht werden Mitglied in der Entwickler-Community zu werden. Zu diesem Zweck wurden Patches für Smack geschrieben, die höheren Qualitätskriterien genügen als solche, die bereits seit längerem von den aktiven Mitgliedern der Community unbeachtet blieben.

Code-Durchsichten der existierenden Patches zeigten, dass diese zwar häufig ein spezielles Problem lösten, dabei aber nicht den umgebenden Kontext beachteten oder die Ursache des Problems behoben. Somit konnte nicht sichergestellt werden, ob durch den Patch nicht neue Fehler oder Regressionen auftreten würden. Zudem waren keine Unit-Tests vorhanden, um die Verbesserungen zu testen.

Unter der Annahme, dass dies die Gründe für die geringe Beachtung existierender Patches sind, habe ich den Patch für *SOCKS5 Bytestreams* entwickelt (siehe Kapitel 5), der den gesamten Kontext der XEP-0065 Spezifikation beachtet und eine ausführliche Test-Suite enthält.

Als Mitglied der Community habe ich anschließend einen Entwurf für ein Dokument, das Richtlinien für Smack-Entwickler enthält, erarbeitet und diesen mit den bestehenden Community-Mitgliedern abgestimmt. Das Dokument umfasst einheitliche Formatierungsregeln für den Quellcode, definiert Kriterien für Patches, dokumentiert die Package-Hierarchie, listet bewährte Entwicklungsmethoden (Best Practice) auf und gibt Hilfe beim Erstellen von Unit-Tests und beim Erzeugen von Patch-Dateien. Teile dieses Dokuments lehnen sich an die Richtlinien, die auch für das Projekt Saros gelten, an. Anhang A.1 enthält den Entwurf dieses Dokuments.

Ursprünglich geplant war zudem die Erarbeitung eines Dokuments mit formalen Kriterien zur Aufnahme neuer Mitglieder in die Smack-Community und zur Erlangung von Schreibrechten auf dem Versionsverwaltungssystem für den Quellcode. Ein entsprechender Vorschlag existierte bereits, wurde aber noch nicht veröffentlicht. Um Schreibrechte für die Versionsverwaltung zu erhalten, gibt es ein „Men-

torship Program“ demzufolge ein Mentor die Entwicklungen eines neuen Mitglieds begutachtet und sie in seinem Namen der Versionsverwaltung hinzufügt. Nach einer bestimmten Anzahl eingereichter Patches erhält das neue Mitglied dann auch Schreibrechte.

Diesem Vorgehen entsprechend wurden die weiteren in den Kapiteln 6 und 7 beschriebenen Verbesserungen entwickelt.

Falls die im Rahmen dieser Diplomarbeit erstellten Patches dennoch nicht dem Smack-Projekt hinzugefügt würden, sollte evaluiert werden, inwiefern eine Abspaltung (Fork) des Projektes möglich ist. Ein bereits bestehender Fork [SFI09], der einige der existierenden, aber nicht zu den offiziellen Smack-Quellen hinzugefügten Patches enthält, wurde seit Juli 2009 nicht mehr weiterentwickelt. Bei einem weiteren im Februar 2010 angelegten Fork [SFI10], der ebenfalls viele der ausstehenden Patches sowie einige eigene Änderungen enthält, konnte sich bisher auch keine aktive Community für die Weiterentwicklung finden.

Da sich im Zeitraum vom Oktober 2009 bis April 2010 im Smack-Community-Forum sowie bei den bestehenden Forks keine weiteren Entwickler, die Beiträge zu Smack leisten wollen, gefunden haben, wurde gemeinsam mit meinen Betreuern entschieden von einer weiteren Abspaltung des Projektes abzusehen und die verbleibende Zeit zu nutzen, um Schreibrechte für die offiziellen Smack-Quellen zu erlangen.

Kapitel 9 beschreibt die entsprechend der hier dargestellten Vorgehensweise erreichten Ziele.

5 Verbesserung der SOCKS5 Bytestream Implementierung

Im Rahmen des ersten Arbeitspaketes wurde die bestehende Implementierung des *SOCKS5 Bytestreams* analysiert, es wurden Fehler behoben und weitere Funktionen ergänzt. Dieses Kapitel beschreibt das dazugehörige Protokoll und wie die Verbesserungen umgesetzt wurden.

5.1 Das Protokoll SOCKS5 Bytestream

Mit dem XEP-0065 *SOCKS5 Bytestream*-Protokoll wird die Möglichkeit bereitgestellt, einen Datenstrom außerhalb des normalen XMPP-Datenstroms auszuhandeln, um größere Mengen Binärdaten (z. B. Dateien) zwischen den Teilnehmern direkt oder über einen speziellen Proxy auszutauschen. Das Protokoll baut dabei auf Teilen des von der IETF spezifizierten SOCKS5 Protokolls (RFC1928) auf und erweitert dieses, um den Einsatz über XMPP zu ermöglichen.

Die folgenden Teilnehmer sind an der Etablierung einer Verbindung beteiligt:

Initiator ist der Teilnehmer, der eine Verbindung mit dem Ziel aufbauen möchte.

Ziel ist der Teilnehmer, mit dem der Initiator eine Verbindung aufbauen möchte.

Proxy ist ein Server, der als möglicher Vermittler des Bytestroms zwischen Initiator und Ziel dienen kann.

StreamHost ist der Proxy, mit dem sich das Ziel verbindet und über den der Bytestrom zwischen Initiator und Ziel versendet wird.

Das *SOCKS5 Bytestream*-Protokoll definiert zwei Varianten, wie eine Verbindung zwischen dem Initiator und dem Ziel aufgebaut werden kann.

Bei der direkten Verbindung (Abb.: 5.1) fungiert der Initiator des *SOCKS5 Bytestreams* gleichzeitig als StreamHost, kennt somit dessen Adressinformationen und sendet diese an das Ziel, welches daraufhin versucht, eine Direktverbindung mit dem Initiator über das SOCKS5 Protokoll aufzubauen.

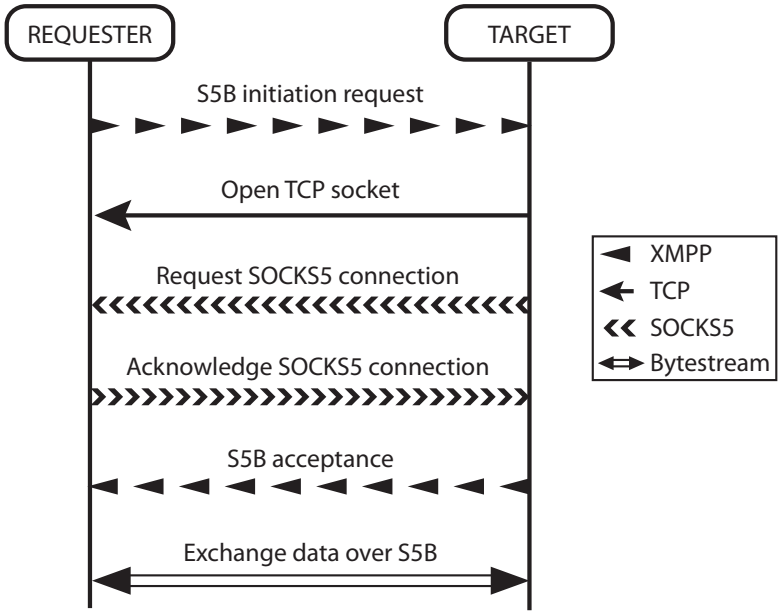

Abbildung 5.1: Ablauf des *SOCKS5 Bytestream*-Protokolls bei direkter Verbindung. Der Initiator sendet die Anfrage über XMPP an das Ziel. Das Ziel öffnet eine TCP-Verbindung zum lokalen SOCKS5-Proxy des Initiators. Anschließend wird über das SOCKS5-Protokoll die Verbindung ausgehandelt. Nach erfolgreicher Verbindung bestätigt das Ziel die Verbindung über XMPP. Danach können Binärdaten über den SOCKS5-Kanal ausgetauscht werden.

Bei der vermittelten Verbindung (Abb.: 5.2) ist der StreamHost nicht der lokale Proxy des Initiators, sondern ein Proxy, der sich im XMPP-Netzwerk befindet. Um die Adressinformationen eines Proxy dem Ziel bekanntmachen zu können, muss der Initiator zunächst über eine *Service Discovery*-Anfrage an seinen XMPP-Server einen Proxy finden und dessen Adressinformationen abrufen. Anschließend handeln sowohl Ziel als auch Initiator auf dieselbe Weise über das SOCKS5-Protokoll eine Verbindung mit diesem StreamHost aus. Bevor aber Daten über den StreamHost versendet werden können, muss der Initiator den Bytestrom noch durch das Senden einer XMPP-Nachricht an den Proxy aktivieren.

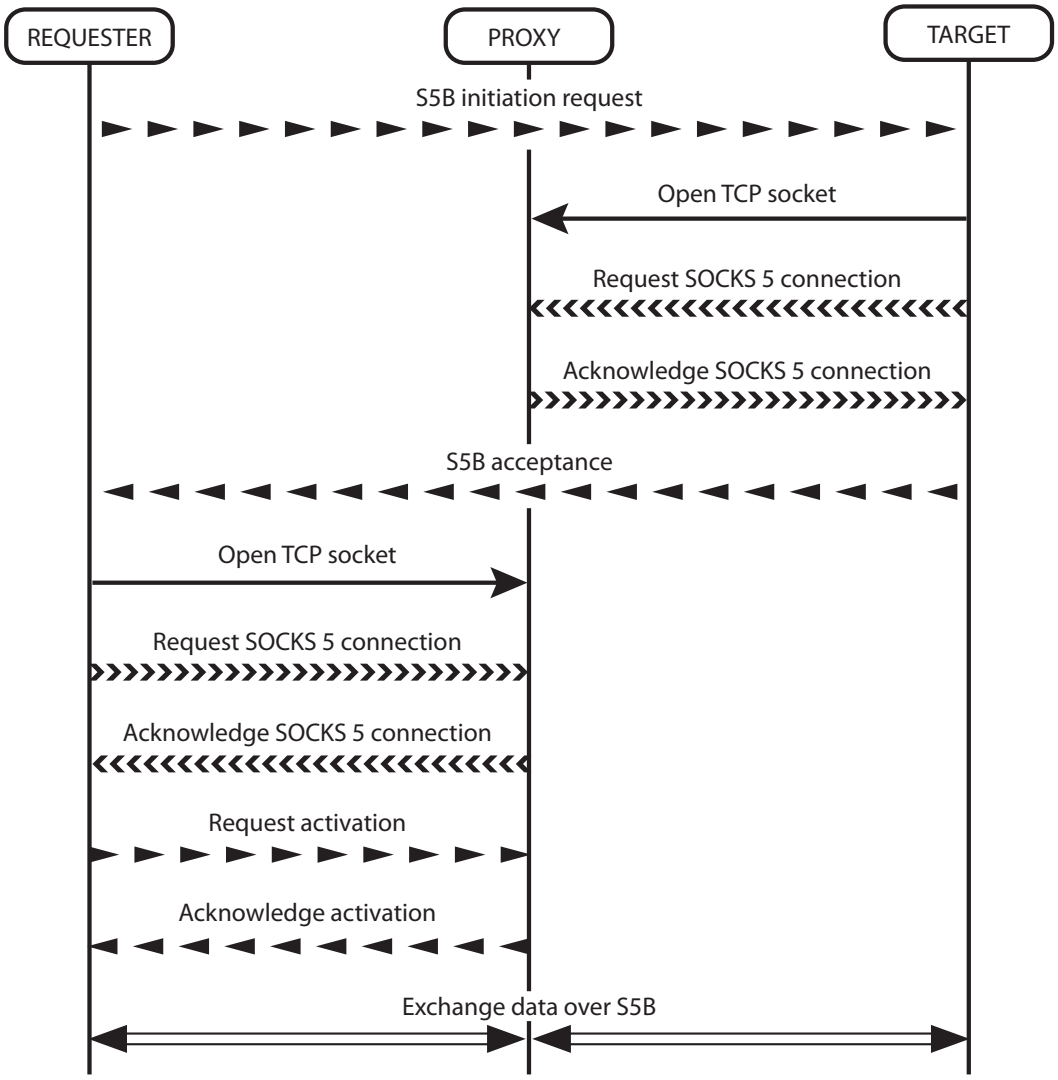

Abbildung 5.2: Ablauf des *SOCKS5 Bytestream*-Protokolls bei vermittelter Verbindung. Der Initiator sendet die Anfrage über XMPP an das Ziel. Das Ziel öffnet eine TCP-Verbindung zum SOCKS5-Proxy. Anschließend wird über das SOCKS5-Protokoll die Verbindung ausgehandelt. Nach erfolgreicher Verbindung bestätigt das Ziel die Verbindung über XMPP. Der Initiator verbindet sich ebenfalls zum SOCKS5-Proxy und aktiviert den Datenstrom durch eine XMPP-Nachricht. Legende siehe Abbildung 5.1.

5.2 Probleme der bisherigen Umsetzung

Smack hat das *SOCKS5 Bytestream*-Protokoll nur im Rahmen der XEP-0096 *Stream Initiation File Transfer*-Spezifikation implementiert. *Stream Initiation File Transfer* ist eine Erweiterung der XEP-0095 *Stream Initiation*-Spezifikation und definiert ein Protokoll speziell für den Austausch von Dateien zwischen den Teilnehmern im XMPP-Netzwerk. Neben der Möglichkeit Informationen wie Größe und Beschreibung der zu übertragenden Dateien dem Ziel bereitzustellen, werden im *Stream Initiation File Transfer*-Protokoll auch die Übertragungsmethoden ausgehandelt. Derzeit sind zwei Übertragungsmethoden durch die XMPP Standards Foundation spezifiziert: *SOCKS5 Bytestream* und *In-Band Bytestream*. Smack bietet aber keine Möglichkeit, das an sich eigenständige *SOCKS5 Bytestream*-Protokoll direkt zum Aufbauen eines Bytestroms zu verwenden, sondern lässt dies nur im Rahmen eines Dateitransfers mit festgelegter Größe der zu übertragenden Daten zu.

Des Weiteren bietet die Smack *File Transfer*-API keine Informationen darüber, welche der beiden Übertragungsmethoden für einen Dateitransfer gerade verwendet wird. Saros nutzt die *File Transfer*-API um einen Bytestrom durch aufeinanderfolgendes Aufbauen einer Dateiübertragung zu simulieren. Hier ist diese Information wichtig, da sich die maximale Übertragungsgeschwindigkeit beider Methoden stark unterscheidet und somit bestimmte von Saros bereitgestellte Dienste wie Sprach-Chat oder Screensharing unter Umständen nicht sinnvoll nutzbar sind. Die maximale Übertragungsgeschwindigkeit für *In-Band Bytestreams* ist zwar abhängig von der XMPP-Serverkonfiguration, wird aber in der Regel auf $4\,\mathrm{kB/s}$ begrenzt. Bei *SOCKS5 Bytestreams* kann die Übertragungsgeschwindigkeit bei einer direkten Verbindung der maximalen möglichen Bandbreite zwischen Initiator und Ziel entsprechen.

Die Implementierung des *SOCKS5 Bytestream*-Protokolls ist zudem nicht ganz vollständig und teilweise fehlerhaft umgesetzt. Mögliche Ursachen dafür sind zum einen die mehrfachen Änderungen an der XEP-0065 Spezifikation [SMSAK10], zum anderen musste die ursprüngliche Implementierung vor allem mit dem XMPP-Server von *Jive Software* zusammenarbeiten wobei bestimmte Teile des Protokolls nicht relevant waren.

Das *SOCKS5 Bytestream*-Protokoll spezifiziert, dass ein Teilnehmer, der anzeigt das Protokoll zu unterstützen, auf eine Initiierungsanfrage entweder mit einer Bestätigung oder einem *not-acceptable*-Fehler antworten muss. Die Smack-Implementierung ignoriert alle Initiierungsanfragen, sofern sie nicht im Rahmen des *Stream*

Initiation File Transfer-Protokolls eintreffen. Ähnlich verhält sich bis heute der XMPP-Server Openfire in der Standardkonfiguration, dessen Implementierung des SOCKS5-Proxy-Servers keine Verbindungen zulässt, wenn diese nicht zuvor durch das XEP-0096 Protokoll ausgehandelt wurden.

Ein Fehler in der Umsetzung des Protokolls ist das Antworten mit einer falschen Fehlermeldung nachdem alle Verbindungsversuche des Ziels zu den Proxy-Servern fehlgeschlagen sind. Damit ist für den Initiator nicht mehr ersichtlich, ob die Verbindung nicht zustande gekommen ist, weil die Initiierungsanfrage abgelehnt wurde oder das Ziel sich mit keinem Proxy verbinden konnte. Diese Information kann für den Benutzer einer auf Smack basierenden Anwendung relevant sein.

Weitere Probleme entstehen direkt durch die Implementierung. So führt das umgesetzte Timeout-Management häufig dazu, dass Verbindungsversuche zu früh abgebrochen werden, obwohl noch nicht alle Möglichkeiten ausgeschöpft worden sind. Der Initiator wartet maximal 10 Sekunden bis der Verbindungsversuch als gescheitert angesehen wird. Das Ziel versucht währenddessen sich nacheinander mit den vom Initiator bekanntgegebenen Proxy-Servern zu verbinden. Für das Verbinden des Ziels zum Proxy ist aber kein Timeout spezifiziert, so dass der vom Betriebssystem vorgegebene Timeout verwendet wird, der in der Regel weit über 10 Sekunden liegt. Da der erste Proxy, den das Ziel probiert, meist der Proxy des Initiators zum Aufbau einer Direktverbindung ist, führt dieses Timeout-Verhalten zwangsweise zu einem frühzeitigen Abbruch, sofern sich der Initiator hinter einem Network Address Translator (NAT) oder einer Firewall befindet.

Durch dieses Timeout-Verhalten kann es anschließend aufgrund einer Wettlaufsituation sogar zu einem Abbruch des gesamten Dateitransfers kommen. Dies sollte eigentlich verhindert werden, indem bei der Dateiübertragung im Falle, dass die SOCKS5-Verbindung nicht aufgebaut werden kann, auf den *In-Band Bytestream* zurückgegriffen wird. Das Ziel wartet aber nur 10 Sekunden ab dem Beginn der Initiierung der Dateiübertragung auf eingehende *In-Band Bytestream*-Anfragen, und der Initiator sendet die entsprechende Initiierungsanfrage erst, nachdem der SOCKS5-Verbindungsversuch durch Timeout abgebrochen wurde.

Gemäß der Spezifikation sollten SOCKS5-Proxy-Server das Erstellen bidirektionaler Verbindungen unterstützen, gefordert werden allerdings nur unidirektionale Verbindungen. Obwohl in gängigen XMPP-Server-Implementierungen die komplette Spezifikation umgesetzt ist, bietet die Smack-Implementierung keine über die Mindestforderung hinausgehenden bidirektionalen Verbindungen.

Da es keine API für die Erstellung von *SOCKS5 Bytestreams* gibt, fehlen der Implementierung auch einige Konfigurationsmöglichkeiten, um das Verhalten zu beeinflussen. So ist es wünschenswert, den Port, den der lokale SOCKS5 Proxy öffnet, einstellen zu können, falls dieser beispielsweise schon belegt oder durch eine Firewall gesperrt ist. Ferner sollte wählbar sein, ob der lokale SOCKS5-Proxy überhaupt gestartet und verwendet werden soll. In vielen Fällen, in denen der Benutzer einen NAT-Router oder eine Firewall verwendet, wird der lokale SOCKS5-Proxy nicht erreichbar sein und belegt somit unnötig Systemressourcen. Um das Timeout-Problem zu lösen, sollten die Antwortzeit auf eine Initiierungsanfrage, das Timeout beim Verbinden zu einem SOCKS5 Proxy sowie das Timeout für die maximale Dauer über alle Verbindungsversuche zu den Proxy-Servern konfigurierbar sein.

Schwierigkeiten beim Beheben dieser Probleme ergeben sich vor allem daraus, dass es keine Unit-Tests für die *SOCKS5 Bytestream*-Implementierung gibt und die Wartbarkeit des Codes nicht unbedingt gegeben ist. (Siehe Tabelle 5.1)

5.3 Umsetzung

Die XEP-0065 *SOCKS5 Bytestream*-Spezifikation stellt ein eigenständiges Protokoll innerhalb der XMPP-Spezifikation dar, deren Verwendungsmöglichkeiten nicht nur auf den Austausch von Dateien eingeschränkt ist. Daher ist es sinnvoll die bisherige Implementierung in Form einer neuen Schnittstelle zu extrahieren. Zudem ließ sich das Problem, dass auf eine Initiierungsanfrage immer mit einer Bestätigung oder einer Ablehnung geantwortet werden muss, im Rahmen der bestehenden Implementierung nicht lösen. Eine Übersicht zur neuen Schnittstelle ist in Abbildung 5.3 dargestellt.

Diese neue Schnittstelle ist analog zur *File Transfer*-API aufgebaut. Einstiegspunkt ist die Klasse `Socks5BytestreamManager`, die Methoden zu Verfügung stellt, um neue *SOCKS5 Bytestreams* zu initiieren und um Beobachter für eingehende *SOCKS5 Bytestream*-Anfragen zu registrieren. Außerdem kann hier die maximale Wartezeit für die Antwort auf alle ausgehenden *SOCKS5 Bytestream*-Anfragen festgelegt werden. Bei erfolgreicher Herstellung einer Verbindung wird ein Exemplar der Klasse `Socks5BytestreamSession` zurückgegeben mit dessen `InputStream` und `OutputStream` bidirektional Daten ausgetauscht werden können.

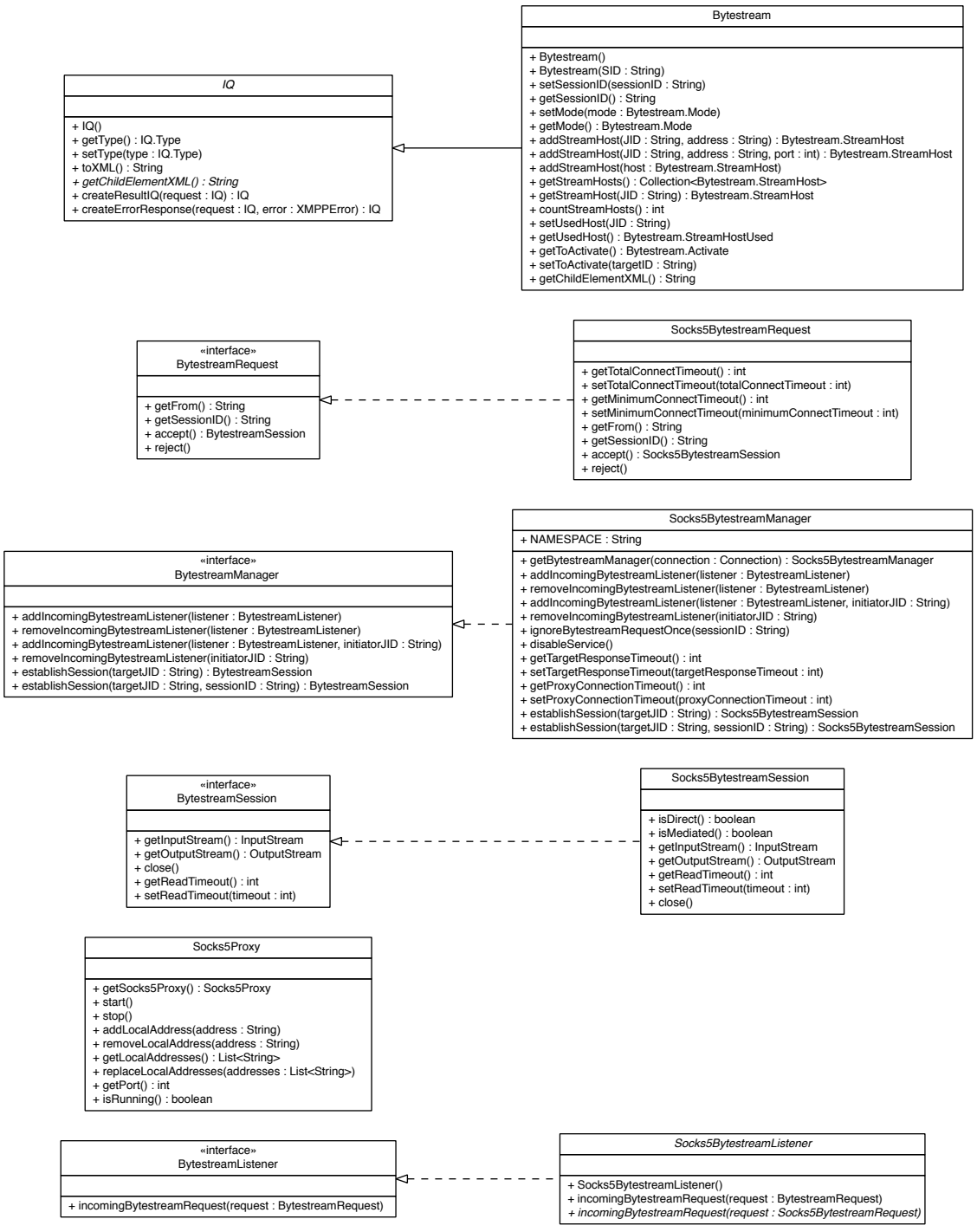

Abbildung 5.3: UML-Diagramm der *SOCKS5 Bytestream*-Schnittstelle

Über eingehende *SOCKS5 Bytestream*-Anfragen werden die zuvor registrierten `Socks5BytestreamListener` informiert und bekommen zur weiteren Behandlung ein Objekt der Klasse `Socks5BytestreamRequest` übergeben. Dieses bietet Methoden, um die Anfrage anzunehmen oder abzulehnen. Soll die Anfrage angenommen werden, kann zuvor das Timeout für die Verbindungsversuche zu den vom Initiator angegebenen SOCKS5-Proxy-Servern eingestellt werden. Wenn eine Verbindung zustande gekommen ist, wird ebenfalls eine `Socks5BytestreamSession` für den bidirektionalen Datenaustausch zurückgegeben.

Um auf alle eingehenden *SOCKS5 Bytestream*-Anfragen entsprechend der Spezifikation reagieren zu können, wird sofort nach der Herstellung einer authentifizierten Verbindung zum XMPP-Server ein spezieller Beobachter registriert. Dieser bearbeitet zunächst die Anfragen und abhängig davon, ob Beobachter für diese Anfrage über den `Socks5BytestreamManager` registriert wurden, reicht er sie weiter oder lehnt sie automatisch ab.

Ob und auf welchem Port ein lokaler SOCKS5-Proxy-Server eingerichtet werden soll, lässt sich nun zum Programmstart über statische Methoden der Klasse `SmackConfiguration` oder über das Bearbeiten der `smack-config.xml` Konfigurationsdatei festlegen.

5.3.1 Verbesserungen

Beim Herstellen einer neuen SOCKS5-Verbindung muss der Initiator zunächst über die *Service Discovery* nach SOCKS5-Proxy-Servern suchen. Dazu wird an den verbundenen XMPP-Server eine unspezifische Anfrage nach allen Servern bzw. Diensten, die dem XMPP-Server bekannt sind, gestellt. Anschließend muss für jeden dieser Dienste mit einer neuen Anfrage überprüft werden, ob es sich um einen SOCKS5-Proxy handelt. Durch Überspringen der Anfrage an Dienste von denen bereits bekannt ist, dass es sich nicht um SOCKS5-Proxy-Server handelt, können nachfolgende SOCKS5-Verbindungsversuche etwas schneller hergestellt werden.

Jede SOCKS5-Initiierungsanfrage enthält eine Liste von Adressen zu SOCKS5-Proxy-Servern. Das Ziel sollte entsprechend der Spezifikation versuchen in derselben Reihenfolge, wie vom Initiator angegeben, eine Verbindung zu den SOCKS5-Proxy-Servern aufzubauen. Um eine Direktverbindung zu ermöglichen war der lokale SOCKS5-Proxy in der bisherigen Implementierung immer der erste Eintrag dieser Liste. Stellte sich aber heraus, dass eine direkte Verbindung aufgrund eines NAT-Routers oder einer Firewall nicht möglich war, so wurde diese Informa-

tion ignoriert und in folgenden *SOCKS5 Bytestream*-Anfragen wieder eine Liste in derselben Sortierung angegeben. In der verbesserten Implementierung wird der SOCKS5-Proxy-Server, mit dem zuletzt eine Verbindung erfolgreich hergestellt werden konnte, priorisiert. Dies verkürzt die Zeit zum Aufbau nachfolgender *SOCKS5 Bytestream*-Verbindungen um einige Sekunden, abhängig von den konfigurierten Timeouts.

Die IP-Adressen unter denen der lokale SOCKS5-Proxy erreichbar ist, können nun vor dem Versenden einer SOCKS5-Initiierungsanfrage konfiguriert werden. Während die bestehende Implementierung nur die Netzwerkadresse der lokalen Netzwerkschnittstelle an das Ziel übermittelt hat, können nun die Adressen weiterer Netzwerkschnittstellen sowie die öffentliche IP-Adresse des NAT-Routers angegeben werden. Dadurch werden die Chancen zur Herstellung einer Direktverbindung verbessert.

Der Verbindungsaufbau zum SOCKS5-Proxy-Server geschieht über das SOCKS5-Protokoll. Dabei wird zunächst die zu verwendende Authentisierungsmethode ausgehandelt und anschließend ein Befehl mit Parametern übertragen. Die Implementierung des lokalen SOCKS5-Proxy unterstützt nur einen Teil des Protokolls. So ist die einzig zulässige Authentisierungsmethode die „no-authentication“- Methode, bei der keine Authentifizierung stattfindet. Von den drei im Protokoll spezifizierten Befehlen wird nur der CONNECT Befehl und für diesen nur der Parameter-Adresstyp „domain address“ erkannt. Während die bisherige Implementierung bei unerwarteten, nicht unterstützten Eingaben einfach den `Socket` geschlossen hat, wird nun zunächst mit den dafür vorgesehenen Fehlermeldungen geantwortet. So kann der Anfragende besser auf diese Fehler reagieren und die Verbindung ordentlich trennen.

Durch das Extrahieren der Schnittstelle, das Umstrukturieren in mehrere Klassen sowie zusätzliche Dokumentation ist die Wartbarkeit des Codes zudem deutlich erhöht worden, so dass zukünftige Erweiterungen oder Fehlerbehebungen einfacher und schneller umgesetzt werden können.

	bisherige Implementierung	verbesserte Implementierung
Zeilen insgesamt	1538	2639
Code-Zeilen	898	1174
Kommentarzeilen	229	631
Kommentare (%)	20,32	34,96
Komplexität	189	254
Klassen	11	16
Komplexität / Klasse	17,18	15,88
Code-Abdeckung durch Unit-Tests (%)	68,5	90,35

Tabelle 5.1: Code-Statistik *SOCKS5 Bytestreams*. Die Statistik wurde mit den Werkzeugen Sonar von SonarSource und EclEmma erstellt. Die angegebene Komplexität ist die zyklomatische Komplexität oder auch McCabe-Metrik [McC76].

5.4 Tests

Um frühzeitig Fehler der Implementierung aufzudecken, deren Funktionalität zu überprüfen und damit die Qualität des aus der neuen Implementierung resultierenden Patches zu erhöhen, wurden Unit-Tests erstellt. Für die bisherige Implementierung des *SOCKS5 Bytestream*-Protokolls im Rahmen der *File Transfer*-API existierten keine solche Funktionsprüfungen und für die gesamte *File Transfer*-API gab es zwei Unit-Tests, die nur den idealen Verlauf eines Dateiaustausches abgebildet haben.

Das Erstellen von Unit-Tests für die Komponenten der Smack-API, die ein Protokoll implementieren, ist schwierig, da die Kommunikation oft zwischen mehr als zwei Kommunikationspartnern stattfindet. So sind an einem Testfall zum *SOCKS5 Bytestream*-Protokoll der Initiator, das Ziel, der XMPP-Server für die Übertragung der Daten und die *Service Discovery* sowie ein oder mehrere SOCKS5-Proxy-Server beteiligt. Insbesondere der XMPP-Server lässt sich nicht ohne erheblichen Aufwand durch ein Stellvertreterobjekt ersetzen, da sehr viel der Server-Logik implementiert werden müsste und die Komplexität des Unit-Tests so sehr hoch wird.

Die meisten für Smack existierenden Unit-Tests erben daher von der abstrakten Klasse `SmackTestCase`. Diese erstellt vor jedem Testfall eine einstellbare Anzahl von Verbindungen zu einem – über eine Konfigurationsdatei spezifizierten –

XMPP-Server her. Der verwendete XMPP-Server sollte dabei ein lokal installierter Server sein, bei dem bestimmte Funktionen wie beispielsweise die *In-Band Registration* aktiviert sind, um die für einen Test benötigten Nutzerkonten automatisch zu erzeugen.

Durch die Verwendung der Klasse `SmackTestCase` sind die Unit-Tests aber abhängig von der jeweiligen Server-Implementierung. Das heißt je nach verwendetem XMPP-Server, dessen Version und Funktionsumfang, können Testfälle fehlschlagen, ohne dass dabei ersichtlich ist, ob die Ursache dafür bei der Smack-Implementierung oder dem XMPP-Server liegt.

Ein weiteres Problem entsteht dadurch, dass innerhalb eines Testfalles keine oder nur unzureichende Kontrolle sowohl über die vom XMPP-Server als auch von den Teilnehmern versendeten und empfangenen Pakete möglich ist. Beispielsweise können bestimmte Antworten der *Service Discovery* des XMPP-Servers nicht ohne weiteres verändern werden. Somit lassen sich viele Fehlerfälle und Probleme nicht nachstellen und infolgedessen kann keine vollständige Abdeckung des zu testenden Codes durch Unit-Tests erreicht werden.

Zur Lösung dieser Probleme wurde eine Schnittstelle implementiert, die es erlaubt, Unit-Tests unabhängig von einem laufenden XMPP-Server und mit voller Kontrolle über die versendeten Pakete aller nicht für den Test relevanten Kommunikationspartner zu entwickeln. Diese Schnittstelle, im folgenden `Protocol`-API genannt, verwendet die Java-Bibliotheken *mockito* und *PowerMock*, um ein Stellvertreterobjekt bzw. eine Attrappe der Klasse `Connection` zu erstellen, welche der zentrale Punkt ist, über den alle XMPP-Pakete gesendet und empfangen werden.

Mit Hilfe von *mockito* lässt sich sehr einfach eine Attrappe (englisch: mock) einer beliebigen Klasse erstellen (Listing: 5.1). Für eine solche Attrappe kann nun zum einen der Rückgabewert der Methoden dieser Klasse festgelegt werden und zum anderen die übergebenen Parameter beim Aufruf dieser Methoden erfasst werden. Wird also vom zu testenden Code eine Methode einer Attrappe aufgerufen, so können die übergebenen Parameter, z. B. ein ausgehendes XMPP-Paket, erfasst und eine zuvor definierte Antwort auf dieses XMPP-Paket zurückgegeben werden. Der Vorteil dieser Bibliothek besteht darin, dass nur für die relevanten Methoden das Verhalten spezifiziert werden muss. Alle weiteren Methoden der Attrappe führen keinen weiteren Code aus und geben automatisch sinnvolle Werte zurück. *PowerMock* erweitert *mockito*, um auch das Verhalten für statische Methoden festlegen zu können, und bietet Hilfsmittel um einfach auf normalerweise nicht zugängliche

Attribute von Objekten zuzugreifen.

```
// Atrappe einer LinkedList erstellen
LinkedList mockedList = mock(LinkedList.class);

// beim Aufruf von get(0) soll die Zeichenkette "first" zurueckgegeben werden
when(mockedList.get(0)).thenReturn("first");

// beim Aufruf von get(1) soll eine Ausnahme geworfen werden
when(mockedList.get(1)).thenThrow(new RuntimeException());

// gibt "first" zurueck
mockedList.get(0);

// gibt null zurueck
mockedList.get(999);

// wirft eine Ausnahme
mockedList.get(1);
```

Listing 5.1: Beispiel einer Atrappe für eine Liste

Die `Protocol`-API besteht aus drei Teilen. Mit Hilfe der Klasse `ConnectionUtils` kann eine Attrappe für die Klasse `Connection` erzeugt werden. Diese nutzt ein Exemplar der Klasse `Protocol`, um zu versendende XMPP-Pakete zu sichern und Antworten auf diese Pakete zurückzugeben. Bevor der zu testende Code ausgeführt wird, müssen zunächst alle für den Testfall benötigten Antwortpakete vorbereitet und an das `Protocol` übergeben werden. Zu jeder Antwort können zudem beliebig viele Implementierungen des Interfaces `Verification` angegeben werden. Eine `Verification` muss die Methode `verify()` mit dem Anfrage- und Antwort-Paket als Parameter zur Überprüfung der Korrektheit, implementieren. Es gibt eine Reihe vordefinierter `Verification`s, die beispielsweise überprüfen, ob der Sender der Anfrage auch der Empfänger der Antwort ist oder ob das Anfrage-Paket einem bestimmten Typ (GET, SET RESULT, ERROR) entspricht. Nachdem der zu testende Code ausgeführt worden ist, kann auf dem `Protocol` die Methode `verifyAll()` aufgerufen werden (Listing: 5.2). Diese überprüft, ob die Anzahl der angegebenen Antworten mit der der Anfragen übereinstimmt und führt anschließend die in `Verification` implementierte Überprüfung für jedes Anfrage-Antwort Paar aus.

```
@Test
public void shouldNegotiateSocks5BytestreamAndTransferData() throws Exception {

    // get Socks5BytestreamManager for connection
    Socks5BytestreamManager byteStreamManager = Socks5BytestreamManager.
        getBytestreamManager(connection);

    // create responses in the order they should be queried according to XEP-0065

    // build discover info that supports the SOCKS5 feature
    DiscoverInfo discoverInfo = createDiscoverInfo(targetJID, initiatorJID);
    discoverInfo.addFeature(Socks5BytestreamManager.NAMESPACE);

    // return that SOCKS5 is supported if target is queried
    protocol.addResponse(discoverInfo, Verification.correspondingSenderReceiver,
                    Verification.requestTypeGET);
    ...
    // build used stream host response
    Bytestream streamHostUsedPacket = createBytestreamResponse(targetJID,
        initiatorJID);
    streamHostUsedPacket.setSessionID(sessionID);
    streamHostUsedPacket.setUsedHost(proxyJID);

    // return used stream host info as response to the bytestream initiation
    protocol.addResponse(streamHostUsedPacket, new Verification<Bytestream,
        Bytestream>() {

        public void verify(Bytestream request, Bytestream response) {
            assertEquals(response.getSessionID(), request.getSessionID());
            assertEquals(1, request.getStreamHosts().size());
            StreamHost streamHost = request.getStreamHost();
            assertEquals(response.getUsedHost().getJID(), streamHost.getJID());
        }

    }, Verification.correspondingSenderReceiver, Verification.requestTypeSET);
    ...
    // call the method that should be tested
    BytestreamSession session = byteStreamManager.establishSession(targetJID,
        sessionID);
    ...
    // verify protocol flow
    protocol.verifyAll();
}
```

Listing 5.2: Beispiel eines Unit-Tests mit Nutzung der `Protocol`-API (gekürzt). Es werden zunächst Antworten erstellt und diese einem Exemplar von `Protocol` hinzugefügt. Zusätzlich wird für jede Antwort eine Liste von `Verification`s übergeben. Nach Aufruf der zu testenden Methode wird der Ablauf verifiziert.

Um die bei einem Testfall versendeten und empfangenen Pakete manuell überprüfen zu können, kann der Schalter `printProtocol` des `Protocol`s gesetzt werden. Daraufhin wird der Verlauf des Protokolls beim Aufruf von `verifyAll()` in übersichtlicher Form auf der Konsole ausgegeben (Listing: 5.3).

```
================== Start ==================

---------------------- Request ----------------

<?xml version="1.0" encoding="UTF-8"?>
<iq id="0Qo55-54" to="target@xmpp-server/Smack" type="get">
  <query xmlns="http://jabber.org/protocol/disco#info"/>
</iq>

---------------------- Response ----------------

<?xml version="1.0" encoding="UTF-8"?>
<iq id="0Qo55-48" to="initiator@xmpp-server/Smack" from="target@xmpp-server/Smack"
     type="result">
  <query xmlns="http://jabber.org/protocol/disco#info">
    <feature var="http://jabber.org/protocol/bytestreams"/>
  </query>
</iq>

...
---------------------- Request ----------------

<?xml version="1.0" encoding="UTF-8"?>
<iq id="0Qo55-58" to="target@xmpp-server/Smack" type="set">
  <query xmlns="http://jabber.org/protocol/bytestreams" sid="session_id" mode="tcp
      ">
    <streamhost jid="proxy.xmpp-server" host="127.0.0.1" port="7778"/>
  </query>
</iq>

---------------------- Response ----------------

<?xml version="1.0" encoding="UTF-8"?>
<iq id="0Qo55-52" to="initiator@xmpp-server/Smack" from="target@xmpp-server/Smack"
     type="result">
  <query xmlns="http://jabber.org/protocol/bytestreams">
    <streamhost-used jid="proxy.xmpp-server"/>
  </query>
</iq>

...
================== End ==================
```

Listing 5.3: Ausgabe des Protokolls für den in Listing 5.2 dargestellten Unit-Test

Mit Hilfe der `Protocol`-API lassen sich alle Szenarien – bis auf die eines abrupten Verbindungsabbruches – innerhalb eines Unit-Tests nachstellen. Selbst Testfälle, in denen sehr viele Antworten vorbereitet werden müssen, können durch geschicktes Auslagern dieser Vorbereitungen in Hilfsmethoden übersichtlich und kurz gehalten werden.

Für die *SOCKS5 Bytestream*-Implementierung wurden 43 Unit-Tests entwickelt, die zusammen eine Code-Abdeckung von ca. 90% erreichen.

5.5 Verbesserungen der Spezifikation

Während der Umsetzung des *SOCKS5 Bytestream*-Protokolls sind einige Fehler und Inkonsistenzen in der zugrunde liegenden Spezifikation aufgefallen. So enthielten einige Beispiele ungültiges XML, in denen ein Start-Tag mehrere schließende spitze Klammern („>“) enthielt. Die weiteren Probleme betrafen direkt das Protokoll.

In der Spezifikation wird gefordert, dass das `<query>`-Tag immer das Attribut `sid` mit dem Bezeichner für die *SOCKS5 Bytestream*-Sitzung als Wert enthält. Das `<query>`-Tag ist Teil der Initiierungsanfrage sowie der Antwort darauf. Außerdem wird es im Rahmen der *Service Discovery* genutzt. Entsprechend der Spezifikation sollte auch die Anfrage an den SOCKS5 Proxy nach seiner Netzwerkadresse das `sid` Attribute enthalten, obwohl zu diesem Zeitpunkt noch nicht bekannt sein kann, ob dieser SOCKS5-Proxy überhaupt für die Sitzung verwendet wird. Die `sid` stellt somit keine sinnvolle Information für den SOCKS5-Proxy zu diesem Zeitpunkt dar. Gängige XMPP-Server-Implementierungen ignorieren dieses Attribut für die Netzwerkadressanfrage und fügen es auch nicht der Antwort hinzu.

Der Ablauf des Protokolls ist in der Spezifikation durch einen Hauptablauf (primary flow) und einen Nebenablauf (alternate flow) beschrieben, wobei der Hauptablauf bei Punkten, die vom idealen Ablauf abweichen, auf einen Punkt des Nebenablaufes verweist und dieser nach einigen weiteren Schritten wieder auf einen Punkt des Hauptablaufes zurück verweist. Diese Verweise sind mehrfach nicht korrekt, da einige nicht mit dem zuvor beispielhaft geschilderten Ablauf des Protokolls konsistent sind und andere nicht mehr auf die korrekten Stellen im Hauptablauf zurück verweisen. Zudem fehlt ein Schritt im Nebenablauf der den Fall behandelt, dass das Ziel sich zu einem der angegebenen Proxy-Server nicht verbinden konnte.

Eine weitere Inkonsistenz betrifft die Fehlermeldung, die das Ziel im Falle, dass es sich mit keinem der angegebenen SOCKS5-Proxy-Servern verbinden kann, zu-

rück schicken soll. Im beispielhaften Ablauf wird der `<item-not-found>`-Fehler, im formalen Ablauf der `<remote-server-not-found>`-Fehler angegeben.

Die E-Mail an die Bearbeiter der XEP Spezifikationen, die auf diese Fehler aufmerksam macht, ist im Anhang A.2 zu finden. Die Reaktion auf diese E-Mail ist in Abschnitt 9.3 beschrieben.

6 Verbesserung der In-Band Bytestream Implementierung

Entsprechend der in Kapitel 3 aufgeführten Gründe wurde die Schnittstelle für *In-Band Bytestream* analog zur *SOCKS5 Bytestream*-Schnittstelle aus der bestehenden *File Transfer*-API extrahiert.

6.1 Das Protokoll In-Band Bytestream

Die XEP-0047 *In-Band Bytestream*-Spezifikation definiert ein Protokoll, um einen Datenstrom zwischen zwei Teilnehmern innerhalb des XMPP-Datenstroms aufzubauen. Dabei wird der zu sendende Datenstrom in einzelne Blöcke fester Größe unterteilt, welche in XMPP-Nachrichten eingebettet und über das XMPP-Netzwerk verschickt werden.

Um die Binärdaten innerhalb des XMPP-XML-Stroms übertragen zu können, müssen sie in eine textuelle Repräsentation überführt werden, die keine der für XML reservierten Zeichen enthält. Daher werden alle Blöcke bei der Einbettung in XMPP-Nachrichten in Base64 kodiert.

Das Protokoll bietet zwei Arten an, die Daten in XMPP-Nachrichten einzufügen: über den `<iq>`-Stanza oder über den `<message>`-Stanza. Listing 6.1 zeigt ein Datenpaket in beiden Varianten.

Empfohlen wird die Verwendung des `<iq>`-Stanzas, da diese Nachrichten entsprechend der XMPP-Spezifikation immer bestätigt werden müssen und so implizit eine Flusskontrolle für den Datenstrom existiert. Zudem werden die XMPP-Server weniger belastet, da sie keine Nachrichten zwischenspeichern müssen, und der Empfänger der Daten kann im Fehlerfall sofort reagieren.

```
<iq id="pC3O8-308" type="set"
    to="target@xmpp-server/Smack" from="initiator@xmpp-server/Smack">
  <data xmlns="http://jabber.org/protocol/ibb" seq="0" sid="Rq5vgBYlqaNj">
    5Gah69ph6T1P27bhAgy7XWtcnuJvITkRduCzMyzLIDzOQn8d6A
    pP0AMd8h7Jli7W53T8cqxMacII0nBUHQMdjUDX4wWJ23qxM7E9
    iXT4AQX8mO1Kw2s+FVPiwM9TpNME9p6hgnBnVxUbXYUK8ckM
  </data>
</iq>

<message id="nk11e-1"
         to="target@xmpp-server/Smack" from="initiator@xmpp-server/Smack">
  <data xmlns="http://jabber.org/protocol/ibb" seq="0" sid="Rq5vgBYlqaNj">
    5Gah69ph6T1P27bhAgy7XWtcnuJvITkRduCzMyzLIDzOQn8d6A
    pP0AMd8h7Jli7W53T8cqxMacII0nBUHQMdjUDX4wWJ23qxM7E9
    iXT4AQX8mO1Kw2s+FVPiwM9TpNME9p6hgnBnVxUbXYUK8ckM
  </data>
</message>
```

Listing 6.1: Ein *In-Band Bytestream*-Datenpaket in einem `<iq>`-Stanza (oben) und in einem `<message>`-Stanza (unten).

Aufgrund der Abwärtskompatibilität zu vorherigen Versionen der XEP-0047-Spezifikation können weiterhin auch `<message>`-Stanzas ohne die implizite Flusskontrolle verwendet werden. Um dennoch eine Flusskontrolle einzuführen, existiert die XEP-0079 *Advanced Message Processing*-Spezifikation, die es erlaubt, `<message>`-Nachrichten Anweisungen hinzuzufügen, wie der XMPP-Server mit einer Nachricht verfahren soll. Allerdings wird diese Spezifikation nur von sehr wenigen XMPP-Server-Implementierungen unterstützt [WIK10].

Abbildung 6.1 zeigt den Ablauf eines Datenaustausches über das *In-Band Bytestream*-Protokoll. Der Initiator sendet eine Initiierungsanfrage mit der Sitzungskennung, der Größe der zu sendenden Blöcke und dem zu verwendenden Stanza an das Ziel. Das Ziel kann diese Anfrage nun bestätigen oder eine Fehlermeldung zurücksenden, falls der *In-Band Bytestream* nicht unterstützt oder nicht erwünscht wird. Anschließend können unter Verwendung der Sitzungskennung bidirektional Daten zwischen dem Initiator und dem Ziel ausgetauscht werden. Jedes Datenpaket enthält eine fortlaufende Nummer, um die Reihenfolge der Pakete sicherzustellen und den Verlust eines Paketes auf dem Übertragungsweg feststellen zu können. Soll der *In-Band Bytestream* beendet werden, so kann einer der Beteiligten eine Beendigungsanfrage senden. Diese muss immer bestätigt werden und der Bytestrom gilt anschließend als geschlossen.

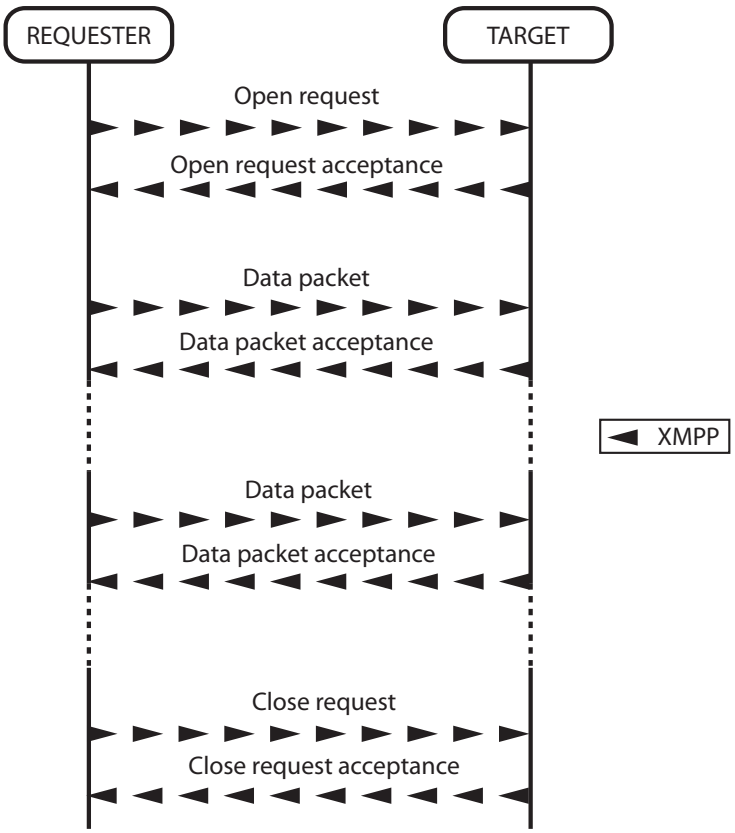

Abbildung 6.1: Ablauf des *In-Band Bytestream*-Protokolls bei Verwendung von `<iq>`-Stanzas. Der Initiatior sendet eine Initiierungsanfrage an das Ziel. Das Ziel bestätigt diese Anfrage und anschließend können Daten in beide Richtungen zwischen Initiator und Ziel ausgetauscht werden. Möchte einer der Beteiligten den Datenstrom beenden, wird eine Beendigungsanfrage verschickt, welche anschließend bestätigt werden muss.

6.2 Probleme der bisherigen Umsetzung

Die im Rahmen der *File Transfer*-API vorhandene Implementierung des *In-Band Bytestreams* unterstützt nur `<message>`-Stanzas, um die Daten zu versenden. Da bei dieser Variante die Datenpakete nicht bestätigt werden, tritt es häufig auf, dass der Initiator in kurzer Zeit eine große Anzahl an Paketen verschicken kann, während diese nur langsam und verzögert beim Ziel ankommen. Somit wird beim Initiator und beim Ziel ein sehr unterschiedlicher Fortschritt einer Übertragung angezeigt.

Wie auch bei der *SOCKS5 Bytestream*-Implementierung wird auf eingehende *In-Band Bytestream*-Anfragen außerhalb eines Dateitransfers nicht reagiert, obwohl die Spezifikation mindestens eine Fehlermeldung dafür fordert.

Da der *In-Band Bytestream* nur innerhalb der unidirektionalen Dateiübertragung verwendet wird, unterstützt die Implementierung auch keine bidirektionalen Datenströme.

Bei der Implementierung kann es außerdem zu einem Wettlauf kommen, durch den die Datenübertragung zwar nicht gestört aber der *In-Band Bytestream* irregulär geschlossen und somit das Protokoll verletzt wird. So kann es vorkommen, dass die Verarbeitung der Beendigungsanfrage ausgeführt wird, bevor die Auswertung des letzten Datenpaketes abgeschlossen ist. Infolgedessen wird die Beendigungsanfrage mit einem `<remote-server-timeout>`-Fehler beantwortet, da Smack fälschlicherweise annimmt, das letzte Datenpaket sei aufgrund eines Server Timeouts nicht eingetroffen. Das Protokoll wird insofern verletzt, als dass die einzig zulässigen Antworten auf eine Beendigungsanfrage eine Bestätigung oder – im Falle, dass die *In-Band Bytestream*-Sitzung dem Empfänger unbekannt ist – eine `<item-not-found>`-Fehlermeldung sein dürfen.

Die Größe der Blöcke eines Datenpaketes muss der Spezifikation nach zwischen 1 und 65535 liegen. Die empfohlene Größe von 4096 ist in Smack durch eine Konstante definiert und kann somit nicht konfiguriert werden.

Die Übersichtlichkeit und Wartbarkeit der Implementierung wird dadurch beeinträchtigt, dass alle der fünf beteiligten Klassen in einer Java-Datei enthalten sind und diese fast vollständig undokumentiert ist. Wie für die *SOCKS5 Bytestream*-Implementierung gibt es auch für die *In-Band Bytestream*-Implementierung keine eigenständigen Unit-Tests.

6.3 Umsetzung

Analog zur neuen *SOCKS5 Bytestream*-Schnittstelle wurde auch die *In-Band Bytestream*-Schnittstelle aus der bestehenden *File Transfer*-API extrahiert. Abbildung 6.2 gibt eine Übersicht der Klassen der neuen Schnittstelle.

Über den `InBandBytestreamManager` kann zunächst der für die Datenpakete zu verwendende Stanza, die Standardblockgröße sowie die maximal erlaubte Blockgröße für eingehende Anfragen konfiguriert werden. Um auf Initiierungsanfragen zu reagieren, muss die Implementierung eines `InBandBytestreamListener`s registriert werden. Durch Aufruf der Methode `establishSession(String)` wird eine *In-Band Bytestream*-Sitzung mit dem übergebenden Ziel erzeugt.

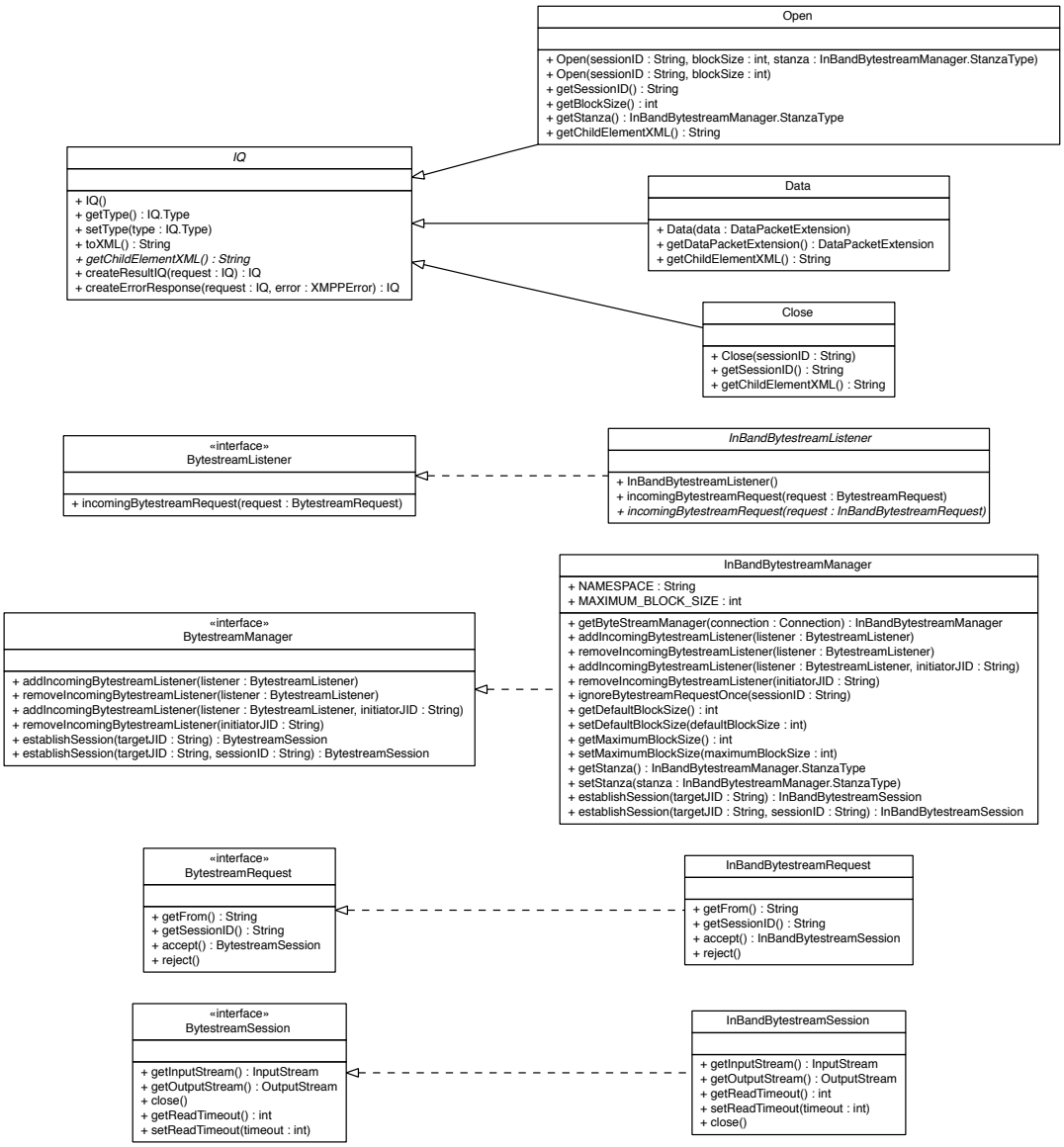

Abbildung 6.2: UML-Diagramm der *In-Band Bytestream*-Schnittstelle

Bei eingehenden *In-Band Bytestream*-Anfragen wird ein Exemplar der Klasse `InBandBytestreamRequest` erzeugt, das Methoden zum Annehmen und Ablehnen der Sitzung zur Verfügung stellt.

Konnte eine *In-Band Bytestream*-Sitzung erfolgreich hergestellt werden, kann nun über die zurückgegebene `InBandBytestreamSession` auf den `InputStream` bzw. `OutputStream` zugegriffen werden, um Daten zu übertragen. Abhängig vom verwendeten Stanza für die Datenpakete kommen dabei Unterklassen der Klassen `IBBInputStream` bzw. `IBBOutputStream` zum Einsatz. Die *In-Band Bytestream*-Sitzung wird beendet, wenn entweder beide Streams geschlossen werden, die `close()`-Methode der `InBandBytestreamSession` aufgerufen wird oder ein Fehler auftritt.

Wie bei der *SOCKS5 Bytestream*-Implementierung wird ein Beobachter registriert, der auf alle eingehenden *In-Band Bytestream*-Anfragen reagiert sobald eine authentifizierte Verbindung zum XMPP-Server hergestellt wird. Dieser `InitiationListener` informiert die registrierten `InBandBytestreamListener` oder lehnt die Anfrage ab.

6.3.1 Verbesserungen

Die neue *In-Band Bytestream*-Implementierung ist nun vollständig kompatibel zur XEP-0047 Spezifikation. Daten können bidirektional zwischen Initiator und Ziel ausgetauscht werden und alle variablen Aspekte des Protokolls sind konfigurierbar.

Zudem werden die in der Spezifikation erwähnten Sicherheitshinweise nun beachtet. Um einen versteckten Informationskanal und Angriffe auf die Implementierung zu verhindern, muss für jedes Datenpaket überprüft werden, ob nur die in der Base64-Kodierung erlaubten Zeichen vorkommen. Weiterhin muss sichergestellt sein, dass das Füllzeichen „=“ nur am Ende der Base64-Zeichenkette auftritt.

Für die Erstellung der Unit-Tests wurde die in Kapitel 5.4 beschriebene `Protocol`-API genutzt. Mit insgesamt 72 Unit-Tests konnte eine Code-Abdeckung von über 85 % erreicht werden (Tabelle: 6.1).

	bisherige Implementierung	verbesserte Implementierung
Zeilen insgesamt	786	2247
Code-Zeilen	462	959
Kommentarzeilen	62	504
Kommentare (%)	11,83	34,45
Komplexität	105	237
Klassen	14	22
Komplexität / Klasse	7,5	10,77
Code-Abdeckung durch Unit-Tests (%)	82,98	86,94

Tabelle 6.1: Code-Statistik *In-Band Bytestreams*. Die Statistik wurde mit den Werkzeugen Sonar von SonarSource und EclEmma erstellt. Die angegebene Komplexität ist die zyklomatische Komplexität oder auch McCabe-Metrik [McC76].

7 Weitere Patches

Die Verbesserungen der Implementierungen für *SOCKS5 Bytestreams* und *In-Band Bytestreams* sind vor allem an Anforderungen von Saros an Smack ausgerichtet und haben eine relativ hohe Komplexität und Größe. Dies bedeutet einen hohen Aufwand beim Review der Patches durch die bestehende Smack-Community und erschwert wohlmöglich die Akzeptanz, da nur wenige der Anforderungen von Smack – definiert durch die Liste der offenen Tickets – durch diese Patches behandelt werden. Daher wurden eine Reihe weiterer kleiner Patches erstellt, die einige der im Bugtracker aufgeführten Probleme lösen. Einige der Patches dienen auch der Lösung von Problemen in Saros, andere behandeln für die Smack-Comunity relevante und seit langer Zeit offen stehende Probleme.

7.1 Frühzeitiges Registrieren des RosterListeners

In der XMPP-Spezifikation wird die Liste aller bestätigten Kontakte, die jedem Teilnehmer zugeordnet ist, als Roster (deutsch: Mitgliedsverzeichnis) bezeichnet. Wenn ein neuer Kontakt dem Roster hinzugefügt oder ein alter gelöscht werden soll, müssen entsprechende XMPP-Nachrichten zwischen den Teilnehmern ausgetauscht werden, um die Aktion zu bestätigen oder abzulehnen.

In Smack kann dazu ein `RosterListener` registriert werden, der bei Eingang einer solchen Nachricht informiert wird, um beispielsweise einen Dialog mit einer Nachfrage für den Benutzer anzuzeigen. Ist aber einer der beteiligten Teilnehmer nicht mit dem XMPP-Netzwerk verbunden, so werden die Nachrichten auf dem XMPP-Server aufbewahrt und erst ausgeliefert, wenn der Teilnehmer wieder verbunden ist.

Smack ermöglicht das Registrieren des `RosterListener`s aber erst, nachdem bereits erfolgreich eine authentifizierte Verbindung zum XMPP-Server hergestellt wurde. Der XMPP-Server beginnt sofort nach Aufbau der Verbindung mit dem Ausliefern der vorgehaltenen Nachrichten. So kann es vorkommen, dass diese Nach-

richten eintreffen, bevor der `Rosterlistener` registriert werden konnte, und diese Informationen somit verloren gehen.

Der `RosterListener` kann nur auf einem Exemplar der Klasse `Roster` registriert werden, welches erst bei der Authentifizierung der Verbindung erstellt wird. Dieser `Roster` wird dann beim Aufruf der Methode `Connection.getRoster()` zurückgegeben. Ist der `Roster` noch nicht erstellt, so wird `null` zurückgegeben.

Der Patch ändert dieses Verhalten, so dass die Methode `getRoster()` immer entweder das bestehende oder, falls keines existiert, ein neu erstelltes Exemplar eines `Roster`s zurückgibt. Somit kann nun jederzeit ein `RosterListener` registriert werden. Alle modifizierenden Methoden des `Roster`s, beispielsweise zum Anlegen eines neuen Eintrags oder einer neuen Gruppe, werfen nun eine `IllegalStateException`, falls noch keine authentifizierte Verbindung zum XMPP-Server besteht. Dieses Verhalten ist analog zu bestimmten Methoden der `Connection`-Klasse, die auch eine bestehende Verbindung zum XMPP-Server voraussetzen.

Im Rahmen dieses Patches wurden sieben Unit-Tests erstellt, die die Funktionsweise des geänderten Verhaltens überprüfen. Zudem wurden die Dokumentation und die JavaDocs entsprechend angepasst.

7.2 RosterListener nur bei Änderungen informieren

Ein weiteres Problem im Zusammenhang mit dem `RosterListener` ist, dass dieser häufiger als nötig über die Aktualisierung von Einträgen des `Roster`s informiert wird. Im Verlauf einer XMPP-Sitzung sendet der XMPP-Server mehrfach alle Einträge des `Roster`s an die Teilnehmer, obwohl nur einige oder gar keine aktualisierte Informationen enthalten.

Durch den Patch wird nun, vor dem Einfügen eines Eintrags in die Liste der aktualisierten Einträge, überprüft, ob zwischen dem bereits bekannten und dem vom XMPP-Server gesendeten Eintrag ein Unterschied besteht. Nur wenn dies der Fall ist, wird der Eintag in die Liste eingefügt und der `RosterListener` informiert.

7.3 Verbesserung des Delayed Delivery Parser

Ein seit Dezember 2008 offenes Problem betrifft den Parser für die in XEP-0203 *Delayed Delivery* beschriebene Erweiterung. Diese ermöglicht es einer `<message>`-

Nachricht Informationen anzufügen, wie lange sie auf dem XMPP-Server aufbewahrt wurde, bis sie ausgeliefert werden konnte.

Vorgänger der XEP-0203-Spezifikation ist die mittlerweile obsolete XEP-0091 *Legacy Delayed Delivery*-Spezifikation. Beide Erweiterungen sind funktional gleich, unterscheiden sich aber in den verwendeten Tags und dem genutzten Datumsformat. Da die obsolete Erweiterung immer noch von vielen XMPP-Server-Implementierungen verwendet wird, unterstützt Smack beide Varianten.

Während für XEP-0203 das in der XEP-0082 *XMPP Date and Time Profiles* spezifizierte Datumsformat („`CCYY-MM-DDThh:mm:ss[.sss]TZD`“) vorgeschrieben ist, wird für XEP-0091 ein eigenes Format („`CCYYMMDDThh:mm:ss`“) verwendet. Bei letzterem gibt es keine Trennzeichen zwischen Jahr, Monat und Tag, so dass bei Weglassung der führenden Nullen bei Monat und Tag mehrdeutige Datumsangaben möglich sind.

Einige XMPP-Server-Implementierungen liefern solch mehrdeutige Datumsangaben aus und der in Smack implementierte Parser für *Delayed Delivery* wirft dann eine `ParseException`. Diese Ausnahme bleibt unbehandelt und führt zu einem Abbruch der Verbindung zum XMPP-Server.

Der Patch verbessert die Parser-Implementierung dahingehend, dass nun auch versucht wird mehrdeutige Angaben zu analysieren. Kann die Datumsangabe dennoch nicht erkannt werden, wird das aktuelle Datum als Zeitstempel verwendet. Dies ist entsprechend der Sicherheitshinweise aus XEP-0203 möglich, da es auf dem Übertragungsweg einer Nachricht mit *Delayed Delivery*-Information vom Absender über einen oder mehrere XMPP-Server bis zum Empfänger ohnehin mehrere Schwachstellen geben kann, an denen das Datum manipuliert werden könnte. Dieser Angabe sollte daher nur unter Vorbehalt vertraut werden.

Um das Datum zu analysieren, wird nun zunächst mit Hilfe eines regulären Ausdrucks festgestellt, welches Datumsformat vorliegt und anschließend der entsprechende Parser verwendet. Bei mehrdeutigen Angaben wird das Datum auf verschiedene Arten interpretiert und anschließend das gewählt, welches in der Vergangenheit liegt und den kleinsten Abstand zur aktuellen Zeit hat.

Weitere Verbesserungen sind das Entfernen mehrfacher Deklarationen der gleichen Konstanten für Datumsformate, die Threadsicherheit beim Zugriff auf Datumsformate sowie das Erstellen von Unit-Tests für den *Delayed Delivery*-Parser.

7.4 Verbesserung der Robustheit des Message-Parsers und I18N

Im Smack-Community-Forum wurde im Januar 2010 ein Problem mit dem Parser für `<message>`-Tags veröffentlicht, das in einem kurzen Zeitraum sehr häufig gelesen wurde, jedoch von den Smack-Entwicklern weitestgehend ignoriert wurde.

In der XMPP-Spezifikation RFC3921 wird der Aufbau einer `<message>`-Nachricht festgelegt. Eine `<message>`-Nachricht kann ein oder mehrere `<body>`-Kindelemente haben, die den für Menschen lesbaren Inhalt einer Nachricht in ein oder mehreren Sprachen enthalten. Innerhalb des `<body>`-Tags sind keine weiteren XML Elemente mehr erlaubt.

Es existieren jedoch noch viele XMPP-Klienten bzw. XMPP-Klientenbibliotheken, die den Inhalt des `<body>`-Tags mit HTML-Elementen formatieren und damit ungültiges XMPP-XML ausliefern. Der Smack `Message`-Parser wirft beim Analysieren einer Nachricht, die weitere Tags im `<body>`-Tag enthält, eine `XmlPullParserException`, die wiederum zum Abbruch der Verbindung zum XMPP-Server führt. Daher ist es wünschenswert die Robustheit des Message-Parsers gegenüber diesen Nachrichten zu verbessern, damit es nicht anderen Teilnehmern am XMPP-Netzwerk möglich ist, die Serververbindung von auf Smack basierenden Klienten zu trennen.

Beim Lösen dieses Problems wurde zudem festgestellt, dass die Zuordnung der `<body>`-Elemente einer Nachricht zu ihrer Sprache nicht korrekt implementiert ist. So wird das im `<message>`-Tag angegebene `xml:lang`-Attribut, welches die Standardsprache der Nachricht angibt, ignoriert und als Standardsprache immer die des Betriebssystems, auf dem Smack ausgeführt wird, verwendet. Infolgedessen können `<body>`-Tags übergangen werden, da Smack fälschlicherweise annimmt, es gäbe bereits einen `<body>`-Tag zu dieser Sprache.

Mit dem Patch wird das `xml:lang`-Attribut des `<message>`-Tags nun ausgewertet und jedem `Body` einer `Message` die richtige Sprache zugeordnet. Dies erforderte weitere Änderungen an der Klasse `Message`, da sich der Standard-`Body` einer `Message` nun anders bestimmt und nicht mehr der Sprache `null` zugeordnet ist.

Die Robustheit des `Message`-Parsers wurde verbessert, indem alle im `<body>`-Tag enthaltenen XML-Elemente als einfache Zeichenketten ohne weitere Semantik interpretiert werden. Dazu werden in einer Schleife alle Tags durchlaufen und deren Zeichenkettenrepräsentation an das Ergebnis konkateniert.

Für den Patch wurden 15 Unit-Tests erstellt, welche die Funktionen des Parsers testen und die Zuordnung der <body>-Tags zu den Sprachen überprüfen. Um für die Tests die entsprechenden XML-Kontrollzeichenketten zu erzeugen wurde die Bibliothek *java-xmlbuilder* verwendet. Diese ermöglicht es XML-Dokumente deutlich übersichtlicher und syntaktisch schöner als mit herkömmlicher Zeichenkettenkonkatenation zu erstellen (Listing: 7.1).

```
/*
 * <message from="romeo@montague.lit/orchard"
 *          id="zid615d9"
 *          to="juliet@capulet.lit/balcony"
 *          type="chat"
 *          xml:lang="de">
 *   <body>Wie geht es dir?</body>
 *   <body xml:lang="en">How are you?</body>
 * </message>
 */
String xmppMessage = XMLBuilder.create("message")
    .a("from", "romeo@montague.lit/orchard")
    .a("to", "juliet@capulet.lit/balcony")
    .a("id", "zid615d9")
    .a("type", "chat")
    .a("xml:lang", "de")
    .e("body")
        .t("Wie geht es dir?")
        .up()
    .e("body")
        .a("xml:lang", "en")
        .t("How are you?")
    .asString(outputProperties);
```

Listing 7.1: Beispiel für XML-Erzeugung mit *java-xmlbuilder*

7.5 I18N für Message Subjects

Neben den <body>-Elementen kann eine <message>-Nachricht auch ein oder mehrere <subject>-Elemente enthalten, die das Thema der Nachricht in verschiedenen Sprachen beschreiben. Der Message-Parser von Smack liest allerdings nur das erste <subject>-Tag und ignoriert alle folgenden.

Um die Standardkonformität der Implementierung entsprechend RFC3921 zu verbessern und weil die in Kapitel 7.4 beschriebenen Probleme auch für <subject>-Tags zutreffen, wurde ein Patch erstellt, der analog zur Implementierung der internationalisierten <body>-Tags diese Funktionalität auch für das Thema einer Nachricht bereitstellt.

8 Beitrag zu Saros

Entsprechend der in Kapitel 3 beschriebenen Aufgabenstellungen folgt eine Übersicht der im Projekt Saros durchgeführten Tätigkeiten. Da der Fokus dieser Diplomarbeit auf der Entwicklung von Patches für Smack lag, umfassen die Patches für Saros nur kleinere Fehlerbehebungen in Bezug auf die Verwendung von Smack und das Korrigieren von Fehlern, die bei den regelmäßigen Tests festgestellt wurden.

8.1 Patches

Zu einem Problem beim Umbenennen, Löschen und Bewegen von Dateien in einem über Saros verteilten Projekt existierte bereits seit längerem ein Patch von Wojciech Polcwiartek. Meine Aufgabe bestand darin, diesen zu überprüfen und so zu ändern, dass er sich auf die aktuellen Quelle innerhalb der Versionsverwaltung anwenden lässt, da Teile des Problems bereits durch andere Patches gelöst wurden.

Beim Testen dieses Patches bemerkte ich einen weiterer Fehler, der beim Umbenennen einer Datei im Stammverzeichnis eines Projekts zu einer Ausnahme führte. Durch einen Patch sowie dem Hinzufügen des dazugehörigen Testfalls im Test-Managementsystem wurde der Fehler behoben.

Die weiteren Patches betrafen die Verwendung von Smack in Saros. So wurde das Saros- und das *Jingle*-Feature zu spät über die *Service Discovery* bekannt gemacht, was dazu führen konnte, dass fälschlicherweise angenommen wurde, ein Teilnehmer unterstütze weder Saros noch *Jingle*.

Um die *Service Discovery* von Smack zu nutzen, wird ein Exemplar der Klasse `ServiceDiscoveryManager` benötigt, welches für jede XMPP-Verbindung ein Singleton ist und über eine statische Methode erstellt beziehungsweise zurückgegeben wird. Smack lässt jedoch auch das Erstellen weiterer Exemplare dieser Klasse über den Konstruktor zu, was im Anschluss zu einem fehlerhaften Verhalten bei Anfragen an die *Service Discovery* führt. Wurde in Saros ein neuer Kontakt zur Kontaktliste hinzugefügt, trat genau dieser Fall ein, der dazu führte, dass das die

Service Discovery falsche Antworten auf Anfragen nach den unterstützen Features verschickte und ein Saros-Klient nicht mehr als solcher erkannt werden konnte.

Ein weiterer behobener Fehler hing nur indirekt mit der Verwendung von Smack zusammen. So wurde der Stand der Fortschrittsanzeige beim Übertragen des Projektes zwischen den Teilnehmern falsch berechnet und führte bei sehr langsamen Übertragungsgeschwindigkeiten, wie beim *In-Band Bytestream*, zur irrtümlichen Annahme der Nutzer, der Übertragungsvorgang sei stehengeblieben.

8.2 Code-Durchsichten

Da die durchgeführten Code-Durchsichten viele verschiedene Bereiche von Saros betreffen, deren genauere Erläuterung nicht Teil dieser Diplomarbeit ist, folgt eine tabellarische Auflistung der Durchsichten mit Informationen über das Problem, den Autor, der Bewertung sowie Anmerkungen.

Patch	Autor	Bewertung	Anmerkungen
InvitationWizard, Jobs, Cancellation	Tas Sóti	-1; +0	Patch, der Bachelorarbeit zur Implementierung der nebenläufigen Einladung sowie einer verbesserten Abbruchbehandlung.
[FIX]Avoid IllegalStateException when SessionManager stopSharedProject() called before the SharedProject has ever been started	Tas Sóti	+0	Patch der einen Fehler beim Beenden einer Saros-Sitzung behebt.
[INTERNAL]Changing JID to User in activities	Marc Rintsch	-1; +1	Patch zur Änderung der Schnittstelle zur Erzeugung von Aktivitäten von `JID`-Objekten zu `User`-Objekten.

Patch	Autor	Bewertung	Anmerkungen
SPath	Marc Rintsch	+1	Patch der Saros darauf vorbereitet mehrere Projekte in einer Sitzung zu teilen. Ersetzt den zuvor verwendeten `IPath` zum referenzieren einer Datei durch `SPath` der zusätzlich Projekt-Informationen enthält.
[FIX]BinaryChannel getData() returned byte Array padded with NULL bytes	Sandor Szücs	+1	Patch der verhindert, dass das von der Methode zurückgegebene Byte-Array am Ende mit Nullen aufgefüllt ist und somit nur noch die tatsächlichen Nutzdaten enthält.
Google Protobuf für BinaryChannel for 20-40% speed-up	Christopher Özbeck	-1; +1	Patch der anstelle der Java-Object-Serialisierung die Bibliothek Google Protobuf verwendet und so zu einer deutlichen Verbesserung der Übertragungsgeschwindigkeit beim Senden von Daten über den `BinaryChannel` ermöglicht.
[FEATURE]Improve abilities to select output file	Christopher Özbeck	+0	Patch der einen Dateiauswahldialog beim Speichern der Zusammenfassung von Saros-Statistikdateien einfügt.

Patch	Autor	Bewertung	Anmerkungen
StatisticsCollector	Christopher Özbeck	+1	Patch der weitere angemerkte Fehler, die bei der Durchsicht des Patches zum neuen Dateiauswahldialog beim Zusammenfassen von Saros-Statistiken entdeckt wurden.
Migration to JUnit 4	Florian Thiel	+1	Patch der alle Unit-Tests in Saros von JUnit 3 auf JUnit 4 umstellt.
VoIP	Olaf Loga	-1	Patch, der die Umsetzung der Bachelorarbeit zur Integration des Sprach-Chat Features enthält.
StreamService	Stephan Lau	-1; -1	Einführung einer Schnittstelle für Dienste, die Daten nicht über Pakete sondern über `InputStream`s und `OutputStream`s versenden. Grundlage für Srach-Chat und Screensharing.
[FIX]Failing service discovery should not crash adding contacts. User is asked instead	Christopher Özbeck	+1	Tritt ein Fehler beim Hinzufügen eines Nutzers zur Kontaktliste auf, soll anstelle einer Fehlermeldung der Nutzer gefragt werden ob er den Kontakt dennoch hinzufügen möchte.
[FIX]Consistency recovery	Christopher Özbeck	+1	Patch, der den Abbruch der Konsistenzwiederherstellung eines Projektes ermöglicht und den Fortschritt der Wiederherstellung anzeigt.

Patch	Autor	Bewertung	Anmerkungen
Archive Streaming	Karl Beecher	-1	Bei der Projektsynchronisation soll nicht mehr das komplette komprimierte Archiv des Projektes in den Arbeitsspeicher geladen und anschließend verschickt werden, sondern über die `StreamService`-Schnittstelle beim Auslesen von der Festplatte direkt versendet werden.
STF - Sandors Test Framework	Sandor Szücs	-1; +1	Einführung eines Test-Frameworks, welches das Testen von Saros in unterschiedlichen Netzwerkkonfigurationen ermöglicht.
Extension of SharedEditor listening	Moritz v. Hoffen	+1	Erweiterung des Beobachters des Texteditors um über Textmarkierungen und Änderungen des Darstellungsfeldes zu informieren.

Tabelle 8.1: Code-Durchsichten im Projekt Saros. Mehrfache Bewertungen bedeuten das der Patch mehrmals durchgesehen wurde.

8.3 Analyse der Jingle Probleme

Zum Aufbauen von Punkt-zu-Punkt-Verbindungen zwischen den Saros-Teilnehmern müssen zunächst die IP-Adressen, unter denen jeder Teilnehmer erreichbar ist, ermittelt werden. Zu diesen IP-Adressen gehören neben den Adressen der auf dem einem Rechner vorhandenen Netzwerkschnittstellen auch die öffentliche IP-Adresse, falls sich der Rechner hinter einem NAT befindet. Saros verwendet zur Bestimmung dieser Adressen die Klasse `ICETransportManager` aus Smacks *Jingle*-Schnittstelle, deren Funktionalität sich wiederum auf der Klasse `ICENegociator` der JSTUN-Bibliothek abstützt [Kin05].

JSTUN implementiert das *Session Traversal Utilities for NAT* (STUN) Protokoll [Int03] mit dessen Hilfe das Vorhandensein und die Art von Firewalls und NAT-Routern ermittelt werden kann. Das STUN-Protokoll wiederum ist Teil der Interactive Connectivity Establishment (ICE) Methode [Int10a] zur Überwindung von NATs, welches ebenfalls von JSTUN implementiert wird.

Zur Bestimmung der besten IP-Adressen, über die eine Direktverbindung hergestellt werden kann, führt JSTUN mit einem speziellen STUN-Server für jede lokale Netzwerkschnittstelle einen Verbindungstest aus, bei dem mehrere Anfragen an den STUN-Server gestellt werden und auf deren Antwort gewartet werden muss. Dies geschieht einmalig bei der ersten Nutzung von *Jingle* und hat zur Folge, dass die Initialisierung über 20 Sekunden in Anspruch nehmen kann.

Befindet sich der Klient tatsächlich hinter einem NAT-Router, ist unter den durch JSTUN ermittelten IP-Adressen meist nicht die öffentliche IP-Adresse, da die Verbindungstests für einen nicht konfigurierbaren, zufälligen, lokalen Port durchgeführt werden, der in der Regel auf dem NAT-Router nicht freigegeben und somit blockiert wird.

Um dieses Problem zu lösen, sollten von Saros daher vor dem Initiieren einer *Jingle*-Sitzung bestimmte Ports auf dem NAT-Router mit Hilfe der Protokolle *Universal Plug and Play* (UPnP) und *NAT Port Mapping Protocol* (NAT-PMP) freigegeben werden.

Ein weiteres Problem ist, dass die von Smacks *Jingle*-Schnittstelle bereitgestellte ICE-Methode nur für den Einsatz von UDP-basierten Sitzungen konzipiert ist, während Saros für den Datenaustausch ein verbindungsorientiertes Protokoll wie TCP benötigt. Derzeit existiert nur ein Entwurf für eine ICE-TCP-Methode [Int10b], für die es weder eine Server-Implementierung noch eine standardisierte Spezifikation (XEP) gibt, wie diese im Rahmen von *Jingle* verwendet werden soll.

Für den Umbau des Netzwerksmoduls von Saros habe ich einen Prototypen implementiert, der auf die nicht-performante Verwendung der JSTUN-Bibliothek verzichtet. Stattdessen wird der STUN-Server nur noch genutzt um die öffentliche IP-Adresse zu bestimmen. Mit Hilfe von UPnP- und NAT-PMP-Bibliotheken können darüber hinaus Ports auf einem NAT-Router geöffnet werden.

8.4 Sonstiges

Im Rahmen der regelmäßigen Releases von Saros übernahm ich jeweils einmal die Rolle des Test-Managers, des Assistent-Test-Managers sowie des Assistent-Release-Managers und wirkte zweimal als Tester bei der Durchführung der Anwendungsfälle mit.

Die Anwendungsfälle, welche die einzelnen Schritte der Teilnehmer spezifizieren und die Erwartungen auflisten, lagen zunächst in Form einer Reihe von Microsoft Word-Dokumenten vor. Nach Kritik an der umständlichen Verwaltung dieser Dokumente sowie deren Inkompatibilität zu anderen Office-Programmen wurde auf Vorschlag von mir und weiteren Saros-Mitgliedern die Web-Application TestLink zur Organisierung der Anwendungsfälle eingesetzt und die bestehenden Anwendungsfälle in das neue System übertragen.

Als Vorbereitung für die Arbeiten zur Integration von Sprach-Chat und Screensharing haben Sandor Szücs und ich in einer Pair-Programming-Sitzung einen Prototypen für eine Schnittstelle zur Übertragung von Datenströmen auf Grundlage des `DataTransferManager`s implementiert.

Die daraus hervorgegangene `StreamService`-Schnittstelle sowie das darauf basierende Screensharing-Feature von Stephan Lau und das Sprach-Chat-Feature von Olaf Loga wurde gemeinsam mit den Autoren mehrfach getestet und durchgesehen.

Für die Arbeit von Julia Schenk zur Verbesserung der Benutzbarkeit von Saros haben ich sowie weitere Teilnehmer einer Test-Sitzung eine Liste bestehender Mängel und Vorschläge zu Verbesserungen zusammengestellt.

9 Auswertung

Um die Ergebnisse aus den Aufgabenstellungen für diese Arbeit zusammenzufassen, folgt eine Beschreibung der Reaktion der Smack-Community auf die erstellten Patches sowie eine Analyse, inwiefern eine Reaktivierung der Community erreicht werden konnte. Abschließend wird darauf eingegangen, welche Auswirkungen die Verbesserungen auf das Saros-Projekt haben.

9.1 Reaktion auf die Patches

Im Februar 2010 habe ich den ersten Patch zur Verbesserung der *SOCKS5 Bytestream*-Implementierung im Smack-Community-Forum auf der *Ignite Realtime*-Plattform veröffentlicht. Als Reaktion habe ich ein Benutzerkonto für die Bugtracker-Software JIRA erhalten, und der Smack-Entwickler Günther Nieß, welcher mir später als Mentor im Rahmen des Mentorship-Programs zugeteilt wurde, hat sich bereit erklärt eine Code-Durchsicht durchzuführen. Um die Durchsicht zu erleichtern, wurde der Patch noch einmal bearbeitet, in kleine, leichter zu verstehende Patches geteilt und unnötige Formatierungsänderungen der bestehenden Smack-Quellen entfernt. Nach der Durchsicht des ersten dieser Teil-Patches und der sofortigen Behebung der angemerkten Probleme, stoppte der Durchsichtprozess. Trotz mehrmaliger Nachfragen und Bitten, den Prozess fortzusetzen, konnte der Patch innerhalb des zeitlichen Rahmens dieser Diplomarbeit nicht zu Ende durchgesehen werden, da der Smack-Entwickler zum einen ein schwerwiegendes Hardware-Problem mit seinem Rechner hatte und zum anderen durch sein Studium mit anderen Aufgaben belastet war.

Das Verfahren, um Patches von Community-Mitgliedern der Smack-Versionsverwaltung hinzuzufügen, sieht vor, dass das Mitglied ein Formular zur Abtretung des Copyrights am Quellcode an *Jive Software* unterschreibt und dieses an den *Jive Software*-Firmensitz in Portland, Oregon, USA faxt. Dieser Bitte bin ich nach Veröffentlichung des *SOCKS5 Bytestream*-Patches nachgekommen, habe aber nie eine

Reaktion darauf erhalten.

Der Patch zum frühzeitigen Registrieren des `RosterListener`s wurde Mitte März 2010 veröffentlicht, führte aber zunächst zu keiner Reaktion der Community. Einige Tage später wurde meine Aufnahme in die nicht-öffentliche Gruppe „Community Planning“ bestätigt. Nachdem ich in dieser Gruppe weitere Smack-Entwickler gebeten habe, den `RosterListener`-Patch durchzusehen, erhielt ich Mitte April Anmerkungen vom Smack-Entwickler Guus der Kinderen nach deren Umsetzung der Patch allerdings dennoch nicht den Smack-Quellen hinzugefügt wurde.

Auf die Veröffentlichung der weiteren Patches zur Verbesserung des `Message`-Parsers, des *Delayed Delivery*-Parsers, der Internationalisierung der `Message`-Subjects sowie zum *In-Band Bytestream* erfolgte trotz weiterer Bitten um Code-Durchsichten keine weitere Reaktion der Smack-Entwickler. Gründe dafür sind, dass sowohl Günther Nieß als auch Guus der Kinderen neben ihren Tätigkeiten außerhalb des Smack-Projektes wenig Zeit fanden und es derzeit keine weiteren aktiven Smack-Entwickler in der Community gibt.

9.2 Reaktivierung der Community

Anfang März 2010 habe ich einen ersten Entwurf für das in Abschnitt 4.4 beschriebene Dokument „Richtlinien für Smack-Entwickler“ erstellt und später in der Gruppe „Community Planning“ veröffentlicht. Ziel war es zunächst gemeinsam mit den Mitgliedern dieser Gruppe die Details für diese Richtlinien abzustimmen und anschließend das Dokument, welches dann dem Konsens der bestehenden Entwickler-Community entspricht, im Smack-Community-Forum zu veröffentlichen.

Das Dokument wurde von den Smack-Entwicklern sehr begrüßt und als sinnvoll erachtet und in der anschließenden Diskussion konnten bestehende Punkte konkretisiert und verfeinert sowie neue Punkte hinzugefügt werden. Leider kam die Diskussion nach einem Monat zum Erliegen und mehrfache Anfragen, ob ich den aktuellen Stand des Dokuments veröffentlichen könne, wurden nicht beantwortet. Da ich die Veröffentlichung nicht ohne die Zustimmung wenigstens eines weiteren Smack-Entwicklers durchführen wollte, lassen sich keine Aussagen darüber treffen, inwiefern diese Richtlinien und Hilfestellungen weiteren Entwicklern und neuen Community-Mitgliedern bei der Erstellung von Patches geholfen haben.

Erschwerend kommt hinzu, dass während der sechs Monate, in denen ich das Smack-Community-Forum beobachtet und aktiv durch die Beantwortung von Fra-

gen teilgenommen habe, sich nicht ein weiterer Entwickler zur Mitarbeit an Smack gefunden hat und in diesem Zeitraum auch keine neuen Patches von der Community in das Forum eingestellt wurden.

9.3 Verbesserung der XEPs

Als Reaktion der in Kapitel 5.5 dargelegten Verbesserungsvorschläge der Spezifikation für *SOCKS5 Bytestreams* erhielt ich eine E-Mail von Peter Saint-Andre, dem Geschäftsführer der XMPP Standards Foundation und Mitentwickler zahlreicher XEPs, mit der Auskunft, dass diese Spezifikation gerade einer Bearbeitung unterzogen wird, um die angemerkten sowie weitere Probleme zu beheben.

Seit März 2010 ist eine neue Version der Spezifikation veröffentlicht, deren Aufbau nun konsistent zu anderen XEPs auf die formale Beschreibung des Protokolls durch einen Hauptablauf und einen Nebenablauf verzichtet und stattdessen Diagramme verwendet. Der Großteil der von mir angemerkten Kritikpunkte ist in diesem, sich noch in Bearbeitung befindlichen und als Realease Candidate bezeichneten Dokument bereits beachtet worden.

9.4 Verbesserungen in Saros

Neben den bereits in Kapitel 8.1 geschilderten Verbesserungen bezüglich der Verwendung von Smack in Saros ermöglichen die für Smack entwickelten Patches weitere Fehlerbehebungen und Optimierungen insbesondere des Netzwerkmoduls von Saros.

Entsprechend der ursprüngliche Planung durch meine Betreuer und mich war zunächst vorgesehen, dass meine Entwicklungen für Smack erst zum Einsatz kommen sollten, wenn diese Eingang in die offizielle Versionsverwaltung des Smack-Projektes gefunden haben. Im Sommersemester 2010 sollte Saros als Entwicklungswerkzeug von den an der Vorlesung „Algorithmen und Programmierung 2“ teilnehmenden Studenten verwendet werden. Diesem Einsatz vorausgegangene Tests zeigten eine Reihe von Problemen mit dem Netzwerkmodul von Saros auf. In verschiedenen Netzwerkkonfigurationen und unter Verwendung unterschiedlicher Betriebssysteme konnten nur sehr unzuverlässig Punkt-zu-Punkt-Verbindungen zwischen den Teilnehmern hergestellt werden. Teilweise schlug selbst die Ausweichlösung, das Versenden der Daten über das XMPP-Netzwerk, fehl. Infolgedessen konnten daher häufig

keine Saros-Sitzungen aufgebaut werden.

Um dem zu entgegnen, wurde ein größerer Umbau des Netzwerkmoduls begonnen, der sich auf die durch die Patches neu eingeführten Schnittstellen für *SOCKS5 Bytestreams* und *In-Band Bytestreams* abstützt. Damit müssen zum einen die Datenströme zwischen den Saros-Nutzern nicht mehr durch wiederholtes Aufbauen eines Dateitransfers umgesetzt werden und zum anderen kann die Komplexität der Klassen zum Aufbau von Punkt-zu-Punkt-Verbindungen durch den Wegfall der ohnehin nicht für die Verwendung von verbindungsorientierten Protokollen geeigneten *Jingle*-Implementierung von Smack stark reduziert werden. Zum Zeitpunkt der Erstellung dieser Ausarbeitung war der Umbau noch nicht abgeschlossen und es lassen sich somit keine Aussagen über die tatsächlich erreichten Verbesserungen treffen. Bei der Verwendung der neuen Schnittstellen sind den an diesem Umbau beteiligten Saros-Entwicklern Michael Jurke, Sebastian Bauch und Christopher Özbeck weitere Verbesserungsmöglichkeiten aufgefallen deren Umsetzung ich in Form aktualisierter Versionen der Patches im Smack-Community-Form veröffentlicht habe.

Die Integration der Verbesserungen bezüglich der Registrierung des `RosterListener`s ist geplant, aber noch nicht umgesetzt.

10 Ausblick

Um den durch diese Arbeit begonnenen Prozess der Reaktivierung der Smack-Community fortzuführen, ist es zunächst nötig, die sich noch immer in Abstimmung befindlichen Dokumente mit „Richtlinien für Smack-Entwickler“ und dem „Mentorship Program“ zu veröffentlichen, um neuen Mitgliedern den Einstieg als Entwickler zu erleichtern.

Als weitere Maßnahme sollte eine Roadmap aufgestellt werden, die künftige Entwicklungsziele definiert und priorisiert sowie Meilensteine zur Veröffentlichung neuer Smack-Versionen enthält. Somit haben bestehende Entwickler ein Ziel, auf das sie hinarbeiten können, und neuen Entwicklern könnten zunächst kleinere Aufgaben aus dieser Roadmap zugeteilt werden.

Neuen und bestehenden Entwicklern sollten zudem bestimmte Verantwortlichkeiten für Teile des Projektes zugewiesen werden, so dass es für Probleme oder Fragen immer einen Ansprechpartner gibt und es nicht mehr nötig ist, jedes Mitglied einzeln nach der Durchsicht eines Patches oder dem Anlegen eines Nutzerkontos für den Bugtracker zu fragen.

Voraussetzung für das Reaktivieren der Community ist aber entweder ein höheres Engagement der bestehenden Smack-Entwickler bei der Unterstützung neuer Entwickler oder das Senken der Hürden für den Schreibzugriff auf die Smack-Versionsverwaltung, um so eine neue, aktivere Entwicklerbasis aufzubauen.

Zu den technischen Entwicklungszielen zählen neben dem Beheben der seit langem offenen Fehlern aus dem Bugtracker und der Verbesserung der Code-Abdeckung durch Unit-Tests eine verbesserte Implementierung der *Jingle*-Spezifikationen. Eine einfach zu bedienende und zuverlässig funktionierende Umsetzung der darin enthaltenen Features könnte die Popularität von Smack steigern und somit auch neue Entwickler zur Mitarbeit motivieren.

Für Saros könnte die Komplexität des Netzwerkmoduls durch die Verwendung einer verbesserten *Jingle*-Schnittstelle deutlich reduziert und die Zuverlässigkeit weiter erhöht werden. Zudem könnten Funktionen wie der Sprach-Chat und das

Screensharing die für diesen Zweck effizienteren UDP-basierten Sitzungen verwenden. Können zwischen den Saros-Teilnehmern einer Sitzung zuverlässig Direktverbindungen hergestellt werden, wäre es auch möglich, die Änderungs- und Präsenzinformationen über diesen Kanal zu senden und so verbesserte Latenzzeiten bei der Übertragung dieser Daten zu erreichen.

Mein Ziel bleibt es weiterhin die bereits entwickelten Patches der Smack-Code-Basis hinzuzufügen, als Smack-Entwickler weiter am Projekt mitzuarbeiten und somit dem Saros-Team als Ansprechpartner bei Problemen und Weiterentwicklungen bezüglich Smack zur Verfügung zu stehen.

11 Zusammenfassung

Ziel dieser Arbeit ist die Verbesserung der XMPP-Bibliothek Smack, welche die Basis der Netzwerkfunktionalität von Saros, einer Software zur kollaborativen Programmierung, ist. Kapitel 2 beschriebt die Funktionsweise und Entstehungsgeschichte der Software Saros sowie die ihr zugrunde liegenden Konzepte des Pair Programming und des Distributed Pair Programming.

Um das gleichzeitige, verteilte Arbeiten mehrerer Teilnehmer an einem Projekt zu ermöglichen, müssen die Projektdateien sowie Änderungs- und Präsenzinformationen zwischen den Teilnehmern über das Netzwerk ausgetauscht werden. Saros verwendet dazu das in Abschnitt 2.2 erläuterte XMP-Protokoll. Das Open-Source-Projekt Smack implementiert XMPP und bietet eine abstrahierte Schnittstelle zur Nutzung der in diesem Protokoll spezifizierten Funkionen.

Kapitel 4 gibt einen Überblick zu den technischen Problemen aufgrund von Fehlern in der Implementierung von Smack und deren Auswirkungen auf Saros. Des Weiteren wird die Entwicklungsgeschichte dieses Open-Source-Projektes, welches zunächst hauptsächlich von der Firma *Jive Software* vorangetrieben aber seit 2008 sich selbst überlassen wurde, sowie die daraus resultierende Inaktivität der Community näher beleuchtet.

Ansätze, um die Community zu reaktivieren und damit die weitere Entwicklung an Smack fortzusetzen, sind in Abschnitt 4.4 aufgeführt. Einer der Ansätze ist das Erstellen von Patches, die höheren Qualitätskriterien genügen als solche, die bisher von den Smack-Entwicklern nicht beachtet wurden. Die Kapitel 5, 6 und 7 beschreiben detailliert die im Rahmen dieser Arbeit erstellten Verbesserungen an den Protokollen zur Übertragung von Binärdaten sowie die Behebung seit Langem offener Fehler.

Wie in Kapitel 9 näher ausgeführt, konnten durch diese und weitere Maßnahmen – wie dem Definieren von Richtlinien für Smack-Entwickler sowie dem Festlegen von Rahmenbedingungen für die Aufnahme neuer Entwickler für das Smack-Projekt – bisher weder eine Reaktivierung der Community noch die Integration der entwickel-

ten Patches in Smack-Versionsionsverwaltung erreicht werden. Gründe dafür liegen vor allem im Mangel zeitlicher Ressourcen der bestehenden Smack-Entwickler, die Aufnahme neuer Mitglieder zu unterstützen, sowie fehlender Anreize für neue Mitglieder, sich an dem Projekt zu beteiligen.

Dennoch konnte durch die Entwicklung der Patches eine Grundlage für den Umbau des Netzwerkmoduls von Saros geschaffen werden, durch den der Austausch von Binärdaten zur Projektsynchronisation in unterschiedlichen Netzwerkkonfigurationen deutlich verbessert und die Komplexität des Netzwerkmoduls reduziert werden kann. Kapitel 8 beschreibt die weiteren Tätigkeiten, die im Kontext dieser Arbeit im Projekt Saros durchgeführt wurden.

Abschließend wird in Kapitel 10 ein Ausblick auf zukünftige Maßnahmen zur Reaktivierung der Smack-Community, Vorschläge zur Weiterentwicklung von Smack sowie deren Nutzen für das Projekt Saros gegeben.

12 Literatur

12.1 Zitierte Quellen

Ada02 Adams, D. J.: *Programming Jabber: Extending XML Messaging*. 1. O'Reilly, 2002 http://commons.oreilly.com/wiki/index.php/Programming_Jabber. – ISBN 978–0–596–00202–2

AGDS07 Arisholm, Erik ; Gallis, Hans ; Dybå, Tore ; Sjøberg, Dag I. K.: Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. In: *IEEE Trans. Softw. Eng.* 33 (2007), Nr. 2, S. 65–86. http://dx.doi.org/http://dx.doi.org/10.1109/TSE.2007.17. – DOI http://dx.doi.org/10.1109/TSE.2007.17. – ISSN 0098–5589

Bec99 Beck, Kent: *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999 http://www.mip.sdu.dk/~brianj/Extreme%20Programming%20Explained%20-%20Kent%20Beck%3B%20Addison-Wesley,%201999.pdf. – ISBN 0201616416

BRB08 Bryant, Sallyann ; Romero, Pablo ; Boulay, Benedict du: Pair programming and the mysterious role of the navigator. In: *Int. J. Hum.-Comput. Stud.* 66 (2008), Nr. 7, S. 519–529. http://dx.doi.org/http://dx.doi.org/10.1016/j.ijhcs.2007.03.005. – DOI http://dx.doi.org/10.1016/j.ijhcs.2007.03.005. – ISSN 1071–5819

CH07 Chong, Jan ; Hurlbutt, Tom: The Social Dynamics of Pair Programming. In: *ICSE '07: Proceedings of the 29th international conference on Software Engineering*. Washington, DC, USA : IEEE Computer Society, 2007. – ISBN 0–7695–2828–7, S. 354–363

Coc04 Cockburn, Alistair: *Crystal clear a human-powered methodology for small teams*. Addison-Wesley Professional, 2004. – ISBN 0201699478

DAS+07 Dybå, Tore ; Arisholm, Erik ; Sjøberg, Dag I. K. ; Hannay, Jo E. ; Shull, Forrest: Are Two Heads Better than One? On the Effectiveness of Pair Programming. In: *IEEE Softw.* 24 (2007), Nr. 6, S. 12–15. http://dx.doi.org/http://dx.doi.org/10.1109/MS.2007.158. – DOI http://dx.doi.org/10.1109/MS.2007.158. – ISSN 0740–7459

DASH09 Dewan, Prasun ; Agarwal, Puneet ; Shroff, Gautam ; Hegde, Rajesh: Distributed side-by-side programming. In: *CHASE '09: Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*. Washington, DC, USA : IEEE Computer Society, 2009. – ISBN 978–1–4244–3712–2, S. 48–55

DB92 Dourish, Paul ; Bellotti, Victoria: Awareness and coordination in shared workspaces. In: *CSCW '92: Proceedings of the 1992 ACM conference on Computer-supported cooperative work*. New York, NY, USA : ACM, 1992. – ISBN 0–89791–542–9, S. 107–114

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`

FRB07 Freudenberg, S. (nee B. ; Romero, P. ; Boulay, B. du: "Talking the talk": Is intermediate-level conversation the key to the pair programming success story? In: *AGILE '07: Proceedings of the AGILE 2007*. Washington, DC, USA : IEEE Computer Society, 2007. – ISBN 0–7695–2872–4, S. 84–91

Gus07 Gustavs, Björn: *Weiterentwicklung des Eclipse-Plug-Ins Saros zur Verteilten Paarprogrammierung*, Freie Universität Berlin, Studienarbeit, 2007. `https://www.inf.fu-berlin.de/w/SE/ThesisDPPII`

Han04 Hanks, Brian: Distributed Pair Programming: An Empirical Study. In: *Extreme Programming and Agile Methods - XP/Agile Universe 2004*, 2004, 81–91

HMDK04 Hanks, Brian ; McDowell, Charlie ; Draper, David ; Krnjajic, Milovan: Program quality with pair programming in CS1. In: *ITiCSE '04: Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*. New York, NY, USA : ACM, 2004. – ISBN 1–58113–836–9, S. 176–180

Hum97 Humphrey, Watts: *Introduction to the Personal Software Process*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1997. – ISBN 0–201–54809–7

Jac09 Jacob, Christoph: *Weiterentwicklung eines Werkzeuges zur verteilten, kollaborativen Softwareentwicklung*, Freie Universität Berlin, Diplomarbeit, 2009. `https://www.inf.fu-berlin.de/w/SE/ThesisDPPIV`

Lau10 Lau, Stephan: *Verbesserte Präsenz durch Screensharing für ein Werkzeug zur verteilten Paarprogrammierung*, Freie Universität Berlin, Bachelorarbeit, 2010. `https://www.inf.fu-berlin.de/w/SE/ThesisDPPXIII`

LC03 Lui, Kim M. ; Chan, Keith C. C.: When Does a Pair Outperform Two Individuals? In: *Extreme Programming and Agile Processes in Software Engineering* Bd. 2675, Springer Berlin / Heidelberg, 2003 (Lecture Notes in Computer Science). – ISBN 978–3–540–40215–2, 1011

LL90 Lauwers, J. C. ; Lantz, Keith A.: Collaboration awareness in support of collaboration transparency: requirements for the next generation of shared window systems. In: *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM, 1990. – ISBN 0–201–50932–6, S. 303–311

Log10 Loga, Olaf: *Verbesserung der Kommunikationsmöglichkeiten in Saros*, Freie Universität Berlin, Bachelorarbeit, 2010. `https://www.inf.fu-berlin.de/w/SE/ThesisDPPXV`

McC76 McCabe, Thomas J.: A complexity measure. In: *ICSE '76: Proceedings of the 2nd international conference on Software engineering*. Los Alamitos, CA, USA : IEEE Computer Society Press, 1976, S. 407

MWBF02 McDowell, Charlie ; Werner, Linda ; Bullock, Heather ; Fernald, Julian: The effects of pair-programming on performance in an introductory programming course. In: *SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education*. New York, NY, USA : ACM, 2002. – ISBN 1–58113–473–8, S. 38–42

NJOL05 Nawrocki, Jerzy R. ; Jasiński, Michal ; Olek, Lukasz ; Lange, Barbara: Pair Programming vs. Side-by-Side Programming. In: *Software Process Improvement* Bd. 3792, Springer Berlin / Heidelberg, 2005 (Lecture Notes in Computer Science). – ISBN 978–3–540–30286–5, 28–38

Nos98 Nosek, John T.: The case for collaborative programming. In: *Commun. ACM* 41 (1998), Nr. 3, S. 105–108. `http://dx.doi.org/http://doi.acm.org/10.1145/272287.272333`. – DOI http://doi.acm.org/10.1145/272287.272333. – ISSN 0001–0782

NW01 Nawrocki, Jerzy ; Wojciechowski, Adam: Experimental Evaluation of Pair Programming. In: *Proceedings of the 12th European Software Control and Metrics Conference*. London, UK : Shaker Publishing, 2001, S. 269–276

PM03 Padberg, Frank ; Müller, Matthias M.: Analyzing the Cost and Benefit of Pair Programming. In: *Software Metrics, IEEE International Symposium on* 0 (2003), S. 166. `http://dx.doi.org/http://doi.`

ieeecomputersociety.org/10.1109/METRIC.2003.1232465. – DOI http://doi.ieeecomputersociety.org/10.1109/METRIC.2003.1232465. – ISSN 1530–1435

Rie08 Rieger, Oliver: *Weiterentwicklung einer Eclipse Erweiterung zur Realisierung und Protokollierung Verteilter Paarprogrammierung im Hinblick auf Kollaboration und Kommunikation*, Freie Universität Berlin, Diplomarbeit, 2008. https://www.inf.fu-berlin.de/inst/ag-se/theses/D_OliverRieger.pdf

Rin09 Rintsch, Marc: *Agile Weiterentwicklung eines Software-Werkzeuges zur verteilten, kollaborativen Programmierung in Echtzeit*, Freie Universität Berlin, Studienarbeit, 2009. https://www.inf.fu-berlin.de/w/SE/ThesisDPPV

Só9 Sóti, Tas: *Einladungsprozess in Saros*, Freie Universität Berlin, Bachelorarbeit, 2009. https://www.inf.fu-berlin.de/w/SE/ThesisDPPXI

SAST09 Saint-Andre, Peter ; Smith, Kevin ; Tronon, Remko: *XMPP: The Definitive Guide Building Real-Time Applications with Jabber Technologies*. O'Reilly Media, Inc., 2009. – ISBN 059652126X, 9780596521264

SWN+03 Stotts, David ; Williams, Laurie ; Nagappan, Nachiappan ; Baheti, Prashant ; Jen, Dennis ; Jackson, Anne: Virtual Teaming: Experiments and Experiences with Distributed Pair Programming / Dept. of Computer Science, University of North Carolina, Chapel Hill, NC 27599 Dept. of Computer Science, North Carolina State University, Raleigh, NC 27695. 2003 (2753). – Forschungsbericht. – 129 – 141 S.

Sz9 Szücs, 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

WG04 West, Joel ; Gallagher, Scott: *Key Challenges of Open Innovation*. 2004

Wil00 Williams, Laurie: *The Collaborative Software Process*, University of Utah, Salt Lake City, USA, PhD Dissertation, 2000

WKCJ00 Williams, Laurie ; Kessler, Robert R. ; Cunningham, Ward ; Jeffries, Ron: Strengthening the Case for Pair Programming. In: *IEEE Softw.* 17 (2000), Nr. 4, S. 19–25. http://dx.doi.org/http://dx.doi.org/10.1109/52.854064. – DOI http://dx.doi.org/10.1109/52.854064. – ISSN 0740–7459

WO05 West, Joel ; O'Mahony, Siobhan: Contrasting Community Building in Sponsored and Community Founded Open Source Projects. In: *HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences*. Washington, DC, USA : IEEE Computer Society, 2005. – ISBN 0–7695–2268–8–7, S. 196.3

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`

12.2 Online Quellen

AOL96 AOL: *AIM*. Online Referenz: `http://products.aim.com/`, 1996

BLSA+09 Beda, Joe ; Ludwig, Scott ; Saint-Andre, Peter ; Hildebrand, Joe ; Egan, Sean ; McQueen, Robert: *XEP-0176: Jingle ICE-UDP Transport Method*. Online Referenz: `http://xmpp.org/extensions/xep-0176.html`, 2009

BSAL+09 Beda, Joe ; Saint-Andre, Peter ; Ludwig, Scott ; Hildebrand, Joe ; Egan, Sean: *XEP-0177: Jingle Raw UDP Transport Method*. Online Referenz: `http://xmpp.org/extensions/xep-0177.html`, 2009

HMESA08 Hildebrand, Joe ; Millard, Peter ; Eatmon, Ryan ; Saint-Andre, Peter: *XEP-0030: Service Discovery*. Online Referenz: `http://xmpp.org/extensions/xep-0030.html`, 2008

Int96 Internet Engineering Task Force: *SOCKS Protocol Version 5*. Online Referenz: `http://tools.ietf.org/html/rfc1928`, 1996

Int03 Internet Engineering Task Force: *STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)*. Online Referenz: `http://tools.ietf.org/html/rfc3489`, 2003

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

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

Int10a Internet Engineering Task Force: *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols*. Online Referenz: `http://ietfreport.isoc.org/idref/rfc5245/`, 2010

Int10b Internet Engineering Task Force: *TCP Candidates with Interactive Connectivity Establishment (ICE)*. Online Referenz: `http://tools.ietf.org/html/draft-ietf-mmusic-ice-tcp-08`, 2010

KGHB05 Kern, Philipp ; Glatt, Thomas ; Herr, Benjamin ; Burgmeier, Armin: *Gobby*. Online Referenz: `http://gobby.0x539.de/trac/`, 2005

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

KSA09 Karneges, Justin ; Saint-Andre, Peter: *XEP-0047: In-Band Bytestreams*. Online Referenz: `http://xmpp.org/extensions/xep-0047.html`, 2009

LBSA+09 Ludwig, Scott ; Beda, Joe ; Saint-Andre, Peter ; McQueen, Robert ; Egan, Sean ; Hildebrand, Joe: *XEP-0166: Jingle*. Online Referenz: `http://xmpp.org/extensions/xep-0166.html`, 2009

LSAE+09 Ludwig, Scott ; Saint-Andre, Peter ; Egan, Sean ; McQueen, Robert ; Cionoiu, Diana: *XEP-0167: Jingle RTP Sessions*. Online Referenz: `http://xmpp.org/extensions/xep-0167.html`, 2009

Mic99 Microsoft: *Windows Live Messenger*. Online Referenz: `http://messenger.live.de/`, 1999

Mir96 Mirabilis/AOL: *ICQ*. Online Referenz: `http://www.icq.com/`, 1996

MME04a Muldowney, Thomas ; Miller, Matthew ; Eatmon, Ryan: *XEP-0095: Stream Initiation*. Online Referenz: `http://xmpp.org/extensions/xep-0095.html`, 2004

MME04b Muldowney, Thomas ; Miller, Matthew ; Eatmon, Ryan: *XEP-0096: SI File Transfer*. Online Referenz: `http://xmpp.org/extensions/xep-0096.html`, 2004

PW03 Pittenauer, Martin ; Wagner, Dominik: *SubEthaEdit*. Online Referenz: `http://www.subethaedit.net/`, 2003

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

SA10b Saint-Andre, Peter: *XEP-0261: Jingle In-Band Bytestreams Transport Method*. Online Referenz: `http://xmpp.org/extensions/xep-0261.html`, 2010

SAMK+10 Saint-Andre, Peter ; Meyer, Dirk ; Karneges, Justin ; Lundblad, Marcus ; Hartke, Klaus: *XEP-0260: Jingle SOCKS5 Bytestreams Transport Method*. Online Referenz: `http://xmpp.org/extensions/xep-0260.html`, 2010

SFI09 *Smack XMPP API Community Patch Repository*. Online Referenz: `http://repo.or.cz/w/Smack.git?h=refs/heads/smack-3.1.1`, 2009

SFI10 *github Smack Fork*. Online Referenz: `http://github.com/dereulenspiegel/smack`, 2010

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

ST06 Saros-Team: *Saros - Distributed Collaborative Editing and Distributed Party Programming*. Online Referenz: `https://www.inf.fu-berlin.de/w/SE/DPP`, 2006

WIK10 *Comparison of XMPP server software*. Online Referenz: `http://en.wikipedia.org/w/index.php?title=Comparison_of_XMPP_server_software&oldid=354310750`, 2010

Yah98 Yahoo!: *Yahoo Messenger*. Online Referenz: `http://messenger.yahoo.com/`, 1998

Selbständigkeitserklärung

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

Berlin, den 25. Mai 2010
Henning Staib

A Anhang

A.1 Guidelines for Smack Contributors

A.1.1 Introduction

If you are planning on writing patches for Smack to add new features, improve existing features or fix bugs these guidelines should help you create patches that can be reviewed fast by the community and then be applied to the source repository. This document covers guidelines about the code style, some code rules you should be aware of, how to test your code and how to create and publish a patch.

A.1.2 Code Style

Many developers use an IDE like Eclipse which comes with some handy tools to format the code automatically. If the formatting rules of the existing Smack source code and of the IDE used by the developer differ it may lead to patches that contain lots of unnecessary formatting changes which makes the code hard to review.

This is a proposal how all Smack source code should be formatted. Code contributions will not be declined if they don't match this styleguide but you are encouraged to follow this guide in order to create/preserve a maintainable code base that is easy to read for all developers.

A good set of rules is defined by the Java Code Conventions from Sun.

Indentation

4 white spaces should be used as the unit of indentation. Not using tabs has the advantage that the code looks the same on every editor and it avoids mixed indentation with tabs and white spaces.

```
/**
 * Indentation
 */
class Example {
    int[] myArray = { 1, 2, 3, 4, 5, 6 };
    int theInt = 1;
    String someString = "Hello";
    double aDouble = 3.0;

    void foo(int a, int b, int c, int d, int e, int f) {
        switch (a) {
        case 0:
            Other.doFoo();
            break;
        default:
            Other.doBaz();
        }
    }

    void bar(List v) {
        for (int i = 0; i < 10; i++) {
            v.add(new Integer(i));
        }
    }
}

enum MyEnum {
    UNDEFINED(0) {
        void foo() {
        }
    }
}

@interface MyAnnotation {
    int count() default 1;
}
```

Braces

All opening braces "{" should be on the same line as the declaration statement. The closing brace "}" starts a line by itself and is indented to match its corresponding opening statement.

```
void bar(List v) {
    for (int i = 0; i < 10; i++) {
        v.add(new Integer(i));
    }
}
```

Every code block (in “if” or “while” statements) must be enclosed by braces even if the block only contains one statement. By not doing this, one relies on the code layout for correctness. Bugs are easily introduced this way.

Blank lines

Blank lines should appear after each package, import, class, member and method declaration. There should be no successive empty lines.

New lines

There should be new lines before “do-while”, “try-catch”, “try-finally” and “else if” statements. “If” statements containing only a single statement should have new line.

```
/**
 * If... else
 */
class Example {
    void bar() {
        do {
            // do something
        }
        while (true);

        try {
            // do something
        }
        catch (Exception e) {
            // exception handling
        }
        finally {
            // cleanup
        }
    }

    void foo2() {
        if (true) {
            return;
        }

        if (true) {
            return;
        }
        else if (false) {
            return;
        }
        else {
            return;
        }
    }
```

```
    void foo(int state) {
        if (true) {
            return;
        }
        if (true) {
            return;
        }
        else if (false) {
            return;
        }
        else {
            return;
        }
    }
}
```

Line Wrapping

The maximum line width should be 100 characters. Lines should be wrapped according to the following rules:

- Break after a comma.
- Break before an operator.
- Prefer higher-level breaks to lower-level breaks.
- Align the new line with the beginning of the expression at the same level on the previous line.
- If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

Comments

There are three different ways to use comments in Java. The documentation comments (/** ... */) and two ways for implementation comments ("/* ... */" and "// ... "). Documentation comments should be used to comment classes, methods and public fields. Non-public fields and multi-line comments within a method should use the "/* ... */" implementation comments. Single-line comments and trailing comments should use the "// ... " implementation comments.

```
/**
 * Class Example
 */
public class Example {

  /**
   * Foo constant
   */
  public static final String THE_FOO = "FOO";

  /* foo */
  private String foo;

  /**
   * Constructor
   */
  public Example() {
    // some initialization code
  }

  /**
   * This method does something
   */
  public int doThings() {

    /*
     * heavy documentation about lots of
     * stuff done in this method
     */
    ...

    if (condition) {
      return 1;  // trailing comment 1
    }
    else {
      return -1; // trailing comment -1
    }
  }

}
```

Attached is a code style configuration for eclipse which is based on the built-in Java code style with slight modifications.

A.1.3 Coding rules

License header

Each file should have a license header stating that the code is under the Apache License 2.0

```
/**
 * All rights reserved. Licensed under the Apache License, Version 2.0 (the "
     License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
```

Packages

org.jivesoftware.smack Contains the core classes of Smack to create connections to XMPP servers, to send and receive Packets.

org.jivesoftware.smackx Contains packages for implementations of the XEP specifications. Each extension should have its own subpackage named according to the feature it implements (e.g. `org.jivesoftware.smackx.nicknames` for XEP-0172: User nicknames).

org.jivesoftware.smackx.(feature).packet In order to implement a feature it is often required to create subclasses of `org.jivesoftware.smack.packet.IQ` which should be located in a subpackage of the feature package named "packet".

org.jivesoftware.smackx.(feature).provider The classes that parse the XML representation into a Java object are called providers and are subclasses of `org.jivesoftware.smack.provider.IQProvider` or `org.jivesoftware.smack.provider.PacketExtensionProvider`. There providers should be located in a subpackage of the feature package named "provider".

Visibility

Only classes and and methods that are useful to developers using Smack should be made public. All fields should be private and only be accessible via getter and setter methods. Other methods that are only useful for the internal logic of the feature should be made private or protected.

Field initialization

Instance variables should be initialized in their declaration if possible. This improves readability of the code and reduces code in the constructor.

Variable names

Variable names should be a good compromise between being self-explaining and being short. Common local variables such as primitive variables for loop iteration or condition testing may be named with a short name that matches their class name.

```
int i;
String s;
File f;
float f;
double d;
Iterator<...> it
```

All other variables should have full names that help understanding their use. In many cases the class name with an initial lower-case letter is a good choice.

```
ServiceDiscoveryManager serviceDiscoveryManager;
```

Control flow

Test whether you can return from a method instead of testing whether you should execute a block of code.
Instead of:

```
public void ... {
    ...
    if (condition) {
        // long block of code
    }
}
```

write:

```
public void ... {
    ...
    if (!condition) {
        return;
    }

    // long block of code
}
```

Documentation

At least all public methods should be fully documented including the JavaDoc Tags for the parameters, the return value and possible exceptions. Potentially long-running and blocking methods should always state in the documentation that they are blocking.

Exception handling

If your code raises exceptions related to an XMPP error (e.g. errornous responses, packet reply timout, IO exception) you may want to rethrow this exception as an XMPPException. To do this use the constructor of XMPPException that allows you to wrap the causing exception and add a message to it. Do not add the cause by concatenating the exceptions message to the message string.

```
throw new XMPPException("something went wrong: " + e.getMessage());
```

Instead wrap the Exception:

```
try {
    ...
}
catch (TimeoutException e) {
    throw new XMPPException("something went wrong", e);
}
```

Also avoid catching exceptions by:

```
catch (Exception e) {
    ...
}
```

This will not only catch all checked but also all non-checked and runtime exceptions (e.g. `IllegalArgumentException`, `NullPointerException`, etc.) which typically indicate problems in the application code and should not be hidden.

Thread Safety

Document concurrency related behavior. It should be clear if a particular class is intended to be thread safe, or not. Use the JCIP-based annotations to do this. Apart from documenting the code, static analyzers (such as FindBugs) can make use these annotations, allowing for better analysis of the code.

```
@ThreadSafe
public class Example {
    // thread safe methods
}
```

Prefer immutable objects. Immutable objects are by definition thread-safe. Objects are immuntable if they fulfill the following conditions:

- the class can not be overridden (make class final or use static factories)
- the object can be constructed completely in one step (no setter methods)
- the final keyword is used for every field this is not changed other than by the constructor
- the object has no methods that change the state

Best Practices

- To convert a primitive to a `String` use `String.valueOf(...)`. Don't use "" + i.
- To convert from a primitive use `Integer.parseInt(...)` and similiar.
- Don't use `StringBuffer` unless you need threadsafety, use `StringBuilder` instead.
- Similarly don't use `Vector`, use `ArrayList`.
- Never subclass `Thread`, always pass a `Runnable` into the constructor instead.
- If equals is implemented in a class it is MANDATORY to implement `hashCode` as well, otherwise `HashSets` and other classes which rely on the contract of `hashCode` will not work. IDEs like Eclipse provide generators to implement `equals` and `hashCode`.
- Use enum instead of a bunch of static final int fields. Then you have type safety for related constants and more readable debugging output is possible because you get meaningful names instead of "magic" numbers.
- Avoid the use of static methods and attributes if possible. This is no good object oriented programming style and it also makes testing your code more difficult.

A.1.4 Testing

You should always write unit tests for your code to verify the behavior of your classes and methods and to make it easier for reviewers to understand what your code does and that you thought about some border cases.

For example if you have found a bug in Smack you should write a test that verifies that bug then fix the bug and run the test again to see that it doesn't fail anymore.

Writing test for all features and bugs verifies the correctness of the code and prevents regressions that might be caused by other changes and fixes.

There are two locations for tests in the Smack repository. The "test-unit" folder contains normal JUnit4 tests that test all classes and methods that don't depend on any running infrastructure like an XMPP server.

The folder "test" contains test that are running against a local or remote XMPP server. All test cases subclass `SmackTestCase` which provides a pre-configured execution context. By overwriting `getMaxConnections` you can specify how many connection you want to use in your test. Invoking `getConnection(int index)` returns the preconfigured connection.

```
public class ExampleTest extends SmackTestCase {

    public ExampleTest(String arg0) {
        super(arg0);
    }

    public void testSomething() {
      Connection initiator = getConnection(0);
      Connection target = getConnection(1);

      // send some packets between connections
      ...

      // verify
      ...
    }

    protected int getMaxConnections() {
        return 2;
    }

}
```

The tests can be configured by editing the file "test/config/test-case.xml". Here you can set the host and port of the XMPP server and a prefix for the usernames of the users that will be automatically created for the tests.

Note: If you want to run the tests from Eclipse make sure that the "smack-

config.xml" file is in the class path. Otherwise test may fail because Smack is not initialized correctly. You can do this by configuring the JUnit Run Configuration and add the folder "build/resources" to the classpath or by linking the "build/resources/META-INF" folder in the output folder of the project where the compiled binaries are located.

```
ln -s ../build/resources/META-INF bin/
```

Test coverage and tools

There are tools for Java that can calculate the code accessed by unit tests like Emma, Cobertura or the Eclipse plugin EclEmma. A good test suite should cover about 80% of the code that you have written.

Another good tool to prevent coding errors is FindBugs which comes as a commandline tool or as Eclipse plugin. It analyzes the code and warns you about potential programming errors.

A.1.5 Creating and Publishing Patches

Before starting to write your patch for Smack you should checkout the latest version of Smack from the SVN repository.

```
svn co http://svn.igniterealtime.org/svn/repos/smack/trunk Smack
```

After that you can start coding in that working copy of Smack. This has the advantage that you can create your patch using the SVN tools and don't have to compare two directories with the "diff" command. Additionally you can always update the working copy to the latest version and check that your code doesn't conflict with any of the changes committed to the repository in the meantime.
If you are done with your work you should create a patch file.
If your patch contains new files or directories you first have to put them under version control by executing the following line.

```
svn add --force *
```

Done that you can create the patch via svn diff command.

```
svn diff > patch.patch
```

You can also create patches with Eclipse by right clicking on the project in the Package Explorer -> Team -> Create Patch...

Note that the patch will only contain changes of files that are under version control.

If your patch tends to be very long (more than 1000 lines changed or added) you should probably split your patch in multiple small patches to make reviewing this patch easier. A good way to do this is to split it in functional units like “Adding method x to core class Y to fix ...”, “Implementation of feature X senders side”, “Implementation of feature X receivers side”, “Test cases for feature X” and so on. Also keep in mind that the patches should not contain any unnecessary formatting changes.

Now the only thing left to do is posting your patch as a discussion in the Smack Developers Board and contact one of the Smack developers about your patch.
You should also check out the “How to contribute code” document.
Happy coding!

A.2 E-Mail zu XEP-0065

```
Von:      Henning Staib <henning.staib@fu-berlin.de>
Datum:    15. Februar 2010 11:55:19 MEZ
An:       editor@xmpp.org
Betreff: Errata on XEP-0065 SOCKS5 Bytestreams

Hi,
I have been working on a project that implements the XEP-0065 SOCKS5
Bytestreams features.  In making  use of  the specification,  I have
found what I  believe to be some errors and  inconsistencies. I have
gone to the trouble of compiling a list of the suspected errors, and
have appended  them to this  email, in the  hope that you  find them
useful:

1.  In  Section  4.5  Example  12  there  is  an  error  message
<not-acceptable /> with code "406" and type "auth". XEP-0086 defines
this error message  with the type "modify". So it  should be changed
to "modify" in the example too.

2. In Section 4.4 Example 7 the network address request to the proxy
```

contains a session ID:

```
<query xmlns='http://jabber.org/protocol/bytestreams'
       sid='vxf9n471bn46'/>
```

Although Section 6.1 says that the session ID MUST be present this information is not useful in this step. The SOCKS5 Bytestream session has not started at that time because the Initiator is still retrieving all needed information to start a request to the Target. There is no need for a seesion ID at this time bacause it is still unclear if the proxy will be uses for the SOCKS5 Bytestream session. Popular Implementation of SOCKS5 proxies like ejabberd and Openfire ignore the session ID in the network address request and also don't include it in the response to that request.

3. Section 4.4 Example 8 also contains the session ID in the response to the network address request.

```
<query xmlns='http://jabber.org/protocol/bytestreams'>
       sid='vxf9n471bn46'>
```

Additionally this is invalid XML (there are two '>').

4. Section 6.1 states that the <query/> element "... has a single attribute for the stream session identifier ..." and three sentences later it says it has a second attribute "mode".

5. Section 6.1 says "The sid specifies the bytestream session identifier. This attribute MUST be present.". According to point 2.) in this email maybe this should be changed to something like "This attribute MUST be present in the IQ-set from the Initiator to the Target, in the IQ-result from the Target to the Initiator and in the IQ-set from the Initiator to the Streamhost to request activation."

6. Section 4.6 states "If the Target tries but is unable to connect to any of the StreamHosts and it does not wish to attempt a connection from its side, it MUST return a <item-not-found/> error to the Initiator.". And in Section 5.2 the alternate flow for the

same case states "Target returns <remote-server-not-found/>" (A3). Probably the second error mentioned is the right one.

7. Section 5 contains some wrong references between the primary flow and the alternate flows. Because it would be very difficult and not very comprehensible to write down every change, I copied the section in this email and marked all necessary changes red. One alternate flow is missing and one refers to the Target although the Initiator is meant.

5.1 Primary Flow

1. Initiator wishes to establish a bytestream with Target
2. Initiator sends an IQ-set to Target specifying a StreamID and the network addresses of one or more StreamHosts [A1]
3. Target wishes to establish a bytestream with Initiator [A2]
4. Target requests TCP connection with a StreamHost [A3]
5. Target receives TCP acknowledgement from StreamHost [A4]
6. Target provides authentication credentials to StreamHost via SOCKS5
7. Target receives acknowledgement of authentication with StreamHost via SOCKS5 [A5]
8. Target requests connection with StreamHost via SOCKS5
9. Target receives acknowledgement of successful connection with StreamHost via SOCKS5 [A6]
10. Target sends IQ-result to Initiator announcing successful connection to StreamHost [A9]
11. Use Case Ends (bytestream is established and ready for use)

5.2 Alternate Flows

A1. Initiator does not know the full network address of a StreamHost (i.e., Proxy)

1. Initiator sends IQ-get to Proxy
2. Initiator receives IQ-result from Proxy containing network address [A7][A8]
3. Return to P2

A2. Target does not wish to establish a bytestream with Initiator
  1. Initiator receives <not-acceptable/> error from Target
  2. UCE unsuccessfully

A3. No more StreamHosts in list (Target is unable to reach any of the provided StreamHosts)
  1. Target returns <remote-server-not-found/> error to Initiator
  2. UCE unsuccessfully

A4. Target cannot reach StreamHost
  1. Return to P4

A5. Target authentication with StreamHost fails
  1. Return to P4

A6. Target is unable to connect to StreamHost
  1. Return to P4

A7. Proxy is unwilling to act as a StreamHost for Initiator
  1. Initiator receives <forbidden/> error from Proxy
  2. Return to P2

A8. Proxy is unable to act as a StreamHost for Initiator
  1. Initiator receives <not-allowed/> error from Proxy
  2. Return to P2

A9. Initiator connects to a Proxy
  1. Initiator requests TCP connection with Proxy ~~[A9]~~
  2. Initiator receives TCP acknowledgement from Proxy [A10]
  3. Initiator authenticates with Proxy via SOCKS5
  4. Initiator receives acknowledgement of authentication with Proxy via SOCKS5 [A11]
  5. Initiator requests connection with Proxy via SOCKS5
  6. Initiator receives acknowledgement of successful connection with Proxy via SOCKS5 [A12]
  7. Initiator sends IQ-set to Proxy requesting activation of bytestream
  8. Initiator receives IQ-result from Proxy acknowledging activation of bytestream [A13]
  9. Return to P11

A10. Initiator is unable to reach Proxy
  1. UCE unsuccessfully

A11. Initiator is unable to authenticate with Proxy
  1. UCE unsuccessfully
A12. Initiator is unable to connect to Proxy
  1. UCE unsuccessfully
A13. Proxy is unable to activate bytestream
  1. Initiator receives <internal-server-error/> error from Proxy
  2. UCE unsuccessfully

I hope you find this information useful.

Best regards,
  Henning Staib