Diplomarbeit zur Erlangung des akademischen Grades
Diplom-Informatiker

Weiterentwicklung einer Eclipse-Erweiterung für verteilte Paar-Programmierung im Hinblick auf Kollaboration und Kommunikation

Oliver Rieger

Institut für Informatik,
Freie Universität Berlin
Matrikelnummer: 3483220
rieger@inf.fu-berlin.de

Eingereicht bei:
Prof. Dr. Lutz Prechelt

Betreuer:
Christopher Özbek
Stephan Salinger

Berlin, 07.07.2008

Ich versichere, dass ich die vorliegende Arbeit mit dem Titel „Weiterentwicklung einer Eclipse-Erweiterung für verteilte Paar-Programmierung im Hinblick auf Kollaboration und Kommuniation“ selbstständig verfasst habe. Alle Stellen der Arbeit, die anderen Werken wörtlich oder sinngemäß entnommen sind, sind unter Angabe der Quelle als Entlehnung kenntlich gemacht.

Berlin, den 07.Juli 2008

Oliver Rieger

Gliederung

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

In dieser Diplomarbeit wird eine Eclipse-Erweiterung für verteilte Paar-Programmierung unter den Schwerpunkten der direkten Kommunikation in Weitverkehrsnetzen und der Kollaboration auf Schreibebene analysiert und weiterentwickelt.

Nach Analyse eines bestehenden Ansatzes der Werkzeugunterstützung für verteilte Paar-Programmierung werden Anforderungen an die Weiterentwicklung des bestehenden Ansatzes herausgearbeitet. Die Entwicklung der neuen Anforderungen ist durch eine Erweiterung des ursprünglichen Ansatzes von zwei auf beliebig viele Teilnehmer innerhalb einer Programmiersitzung notwendig, da die ursprünglichen Annahmen der Konzeption des Werkzeugansatzes nicht mehr aufrecht erhalten werden können.

Der für den Aufbau einer Projektsitzung initiale Abgleich aller Projektdateien basiert auf dem Replikations-Ansatz, der jedem Teilnehmer lokale Kopien aller Projektdateien zur Verfügung stellt. Die Umsetzung dieses Ansatzes wird vor allem in Hinblick auf der Notwendigkeit der Übertragung großer Datenmengen über das Weitverkehrsnetz bei steigender Teilnehmeranzahl analysiert. Der Austausch großer Datenmengen setzt jedoch eine Erweiterung der Dateiübertragung im Hinblick auf eine schnelle Verbindungsmöglichkeit voraus, die durch die Verwendung einer Punkt-zu-Punkt Verbindung ermöglicht werden kann. Aufbauend auf einer Analyse der bestehenden Konzepte für die Umsetzung einer direkten Verbindung in Weitverkehrsnetzen und der Darlegung der bestehenden NAT-Problematik, wird ein Konzept für eine direkte Dateiübertragung auf Basis des Jingle-Ansatzes erarbeitet. Die Umsetzung und Integration der direkten Dateiübertragung auf Basis des zuvor erarbeiteten Konzepts in die bestehende Eclipse-Erweiterung, ermöglicht die Synchronisation großer Projekte durch eine schnelle Übertragung aller benötigter Projektdateien und verbessert die bisherige Umsetzung der Projekt-Replikation entscheidend.

Für die Integration einer Kollaboration auf Schreibebene in die bestehende Eclipse-Erweiterung werden verschiedene Verfahren zur Realisierung der hierfür notwendigen Nebenläufigkeitskontrolle untersucht. Darauf aufbauend werden die Grundlagen für eine Konsistenzsicherung mit den Verfahren der Operational Transformation erläutert, da diese Verfahren die notwendigen Anforderungen für einen Einsatz im Umfeld der verteilten Paar-Programmierung erfüllen. Der Jupiter-Ansatz bildet hierfür die viel versprechende Basis zur Umsetzung einer Nebenläufigkeitskontrolle, die vor der Integration in die Eclipse-Erweiterung in einer autarken Implementierung umsetzen wird. Abschließend wird die

Jupiter-basierte Nebenläufigkeitskontrolle in die bestehende Eclipse-Erweiterung integriert und eine Kollaboration auf Schreibebene ermöglicht.

2. Eclipse-Erweiterung für verteile Paar-Programmierung

In diesem Kapitel stelle ich die Motivationsfaktoren und Hintergründe für die Entwicklung der Eclipse-Erweiterung Saros in der bisherigen Form vor. Hierzu werden die Begriffe Paarprogrammierung und verteilte Paarprogrammierung erläutert, um die Motivation der Entwicklung von Saros aufzuzeigen. Im Anschluss wird der bisherige technische Ansatz von Saros vorgestellt und die Motivation und Gründe der Weiterentwicklung dargelegt.

2.1 Paar-Programmierung

Paar-Programmierung (pair programming) bezeichnet eine Methodik aus dem Umfeld des Extreme Programming (XP) [1] und hat im Verlauf der vergangenen Jahre zunehmende Verbreitung gefunden. Innerhalb von XP stellt die Paar-Programmierung ein Konzept der gemeinsamen Arbeit zweier Entwickler an einem Problem dar. Beide Entwickler teilen sich einen gemeinsamen Arbeitsplatz und erfüllen innerhalb einer Paarprogrammierungs-Sitzung fest definierte Rollen, die in der Sitzung mehrfach gewechselt werden können.

Ein Entwickler übernimmt die Rolle des Drivers und führt die eigentliche Programmierarbeit aus, während der zweite Entwickler die Arbeit fortwährend in der Rolle des Observer (bzw. Navigator) kontrolliert. Der Driver erläutert seine Vorgehensweise dem Observer, der diese jederzeit kritisch hinterfragt und Einwände oder alternative Ansätze sofort kommuniziert. Gleichzeitig findet eine fortlaufende Überprüfung der Arbeitsergebnisse statt.

Außerhalb von XP ist Paar-Programmierung ein spontaner und bedarfsorientierter Vorgang ohne festgelegte Rollen, mit dem Ziel, Wissen auszutauschen oder bei Entwurfsentscheidungen und komplexen Problemstellungen zu unterstützen.

Vorteile

Das Konzept der Paar-Programmierung bietet erfolgversprechende Potentiale hinsichtlich der Codequalität und der Problemlösungsstrategien, welche bereits Gegenstand von empirischen Untersuchungen gewesen sind [2].

Die Kollaboration zweier Entwickler im Kontext der Paar-Programmierung verbessert zum einen die Qualität des erstellten Quell-Codes, da während der Implementierung eine fortlaufende Kontrolle durch den Observer erfolgt (pair reviews) und dadurch Fehler sofort aufgedeckt und behoben werden können. Des Weiteren findet eine Verbesserung im Entwurf und den Problemlösungsstrategien statt, da das gemeinsame Vorgehen durchgesprochen und gemeinsam durchdacht wird (pair relaying).

Beide Entwickler haben unter Umständen unterschiedliche Erfahrungen, ein voneinander abweichendes Verständnis der Problemstellung sowie verschiedene Lösungsstrategien. In der Phase der gemeinsamen Problemlösung werden die unterschiedlichen Strategien und Lösungsansätze ausführlich diskutiert und abgewogen, so dass die Wahrscheinlichkeit eines falschen Ansatzes im Vergleich zur Lösungsstrategie eines einzelnen Entwicklers stark gesenkt werden kann (pair think).

Die Anwesenheit des Partners führt zu einem positiven Gruppenzwang (pair pressure), der die Teilnehmer über einen längeren Zeitraum konzentrierter und motivierter arbeiten lässt. Durch die indirekte, gegenseitige Kontrolle werden Vorgehensweisen und Standards innerhalb des Entwicklungsprozesses besser eingehalten.

Durch das Wechseln der Rollen innerhalb der Paar-Programmierungs-Sitzung sind sowohl eine gesteigerte Motivation der Teilnehmer, als auch eine verbesserte Problemlösung zu beobachten. Der Driver muss zu jedem Zeitpunkt jegliche Schritte ausführlich erklären, was ein durchgehendes Durchdenken und Hinterfragen des aktuell verfolgten Ansatzes zur Folge hat (*debugging by explaining*).

Hinzu kommt der ständige Wissenstransfer (pair learning) zwischen den Entwicklern, der sowohl das Wissen des eigenen Projektcodes, als auch das allgemeine technische Wissen und Erfahrungen mit einbezieht. Dies führt zu einer Kultur der gemeinsamen Codeverantwortung und vermindert das Projektrisiko für den Ausfall von einzelnen Entwicklern.

Nachteile

Einer der häufigsten Kritikpunkte an der Methodik der Paar-Programmierung ist der Aspekt der erhöhten Kosten durch den gemeinsamen Einsatz von zwei Entwicklern an einem gemeinsamen Rechner, im Verhältnis zu zwei unabhängig arbeitenden Entwicklern. Jedoch lässt sich die Argumentation der erhöhten Kosten während der Entwicklung durch die verbesserte Qualität des Programmcodes und des Entwurfs, sowie der verbesserten Wartbarkeit der Software entkräften [3].

Paar-Programmierung benötigt die räumliche Nähe, welche bei verteilten Teams ein erhebliches Problem darstellen kann. Zudem benötigt Paar-Programmierung spezielle Arbeitsplätze, die das gemeinsame Arbeiten ermöglichen. Diese sind jedoch nicht immer gegeben. Zusätzlich zu den physischen Arbeitsplätzen müssen sich beide Entwickler auf eine gemeinsame Entwicklungsplattform einigen, was jedoch durch eine Standardisierung der Arbeitsplätze [1] kompensiert werden kann.

Ein weiterer wichtiger Kritikpunkt betrifft die etwaige persönliche Abneigung der jeweiligen Entwickler zum Thema Paar-Programmierung. Dieser Aspekt betrifft die persönliche Fähigkeit und Bereitschaft eines Entwicklers seinen persönlichen Arbeitsplatz zu öffnen und seine Arbeit einem kontinuierlichen impliziten Kontrollprozess zu unterwerfen.

2.2 Verteilte Paar-Programmierung

Die im vorhergehenden Kapitel aufgezeigten Vorteile der Paar-Programmierung basieren zum überwiegenden Anteil auf der direkten Kommunikation der beteiligten Entwickler. Wohingegen die Ursachen für die Nachteile, abgesehen vom Kostenfaktor, durch unpassende Arbeitsplätze und die nicht immer erfüllbare räumliche Nähe bedingt sind.

Vorteile

Ein großer Vorteil der verteilten Paar-Programmierung liegt in der Möglichkeit Paar-Programmierung auch in räumlich verteilten Teams einsetzen zu können. Entwicklerteams können sich über den gesamten Globus verteilen, so dass die „klassische“ Paar-Programmierung von vornherein ausgeschlossen ist. An dieser Stelle bietet verteilte Paar-Programmierung, unter Berücksichtigung gewisser Einschränkungen (siehe Nachteile), die Möglichkeit, die Vorteile der Paar-Programmierung auch in einem verteilten Umfeld einzusetzen.

Stotts et. al [4] beobachteten im Zusammenhang mit verteilter Paar-Programmierung einen höheren Grad an Parallelität, durch den die Arbeitsleistung des verteilten Paares im Verhältnis zur klassischen Paar-Programmierung gesteigert werden könnte. Die Arbeitsleistung könnte beispielsweise gesteigert werden, indem die Teilnehmer, nach vorheriger Koordination, für einen gewissen Zeitraum zur Erreichung einer gemeinsamen Aufgabe, parallele Arbeiten ausführen würden.

Nachteile

Ein Großteil der Vorteile der Paar-Programmierung basiert auf der direkten Kommunikation zwischen den Teilnehmern, die in einem verteilten Umfeld schwieriger zu realisieren ist. An dieser Stelle müssen geeignete Werkzeuge nicht nur die Veränderungen an den Dokumenten kommunizieren, sondern vor allem auch die durch die verteilten Arbeitsplätze entstehende Informationsverluste kompensieren. Auch ist eine Erläuterung der Verhaltensweisen bzw. der Beweggründe des jeweils entfernten Teilnehmer wünschenswert, um Missverständnisse zu vermeiden.

Man spricht im Bereich der Computer Supported Collaborative Work (CSCW) vom Begriff Awareness, der das Bewusstsein über den Zustand und die Aktivitäten der Teilnehmer in kollaborativen Szenarien beschreibt [5]. Es ist jedoch von einem Informationsverlust durch die Werkzeug-unterstützte Kompensation dieser Informationen auszugehen, da komplexe menschliche direkte Kommunikation weit über die eigentliche Sprache hinausgeht und Informationen wie beispielsweise Körpersprache oder Gestiken nur schwer abgebildet werden können. Die eigentliche Nähe und soziale Zusammengehörigkeit innerhalb eines Paares kann im verteilten Umfeld somit nicht im gleichen Umfang zum Tragen kommen.

2.3 Ansätze und technische Umsetzung der Eclipse-Erweiterung Saros

In Anschluss an die Einführung in die klassische und verteilte Paar-Programmierung werden im diesem Abschnitt die Ansätze und Entscheidungskriterien der Eclipse-Erweiterung Saros erläutert, deren Umsetzung auf der Diplomarbeit *Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierung verteilter Paar-Programmierung* von Riad Djemili [6] basiert.

2.3.1 Kriterien für die Entscheidung von Saros

Djemili stellt zur Realisierung eines technischen Werkzeugansatzes für verteilte Paar-Programmierung die Kriterien Übersetzbarkeit, Flexibilität, Technische Restriktionen, Sicherheit und Parallelität auf und vergleicht unter Zuhilfenahme dieser Kriterien unterschiedliche technische Ansätze von Desktop Sharing und Collarboration Awareness.

Der Begriff Desktop Sharing bezeichnet die Freigabe eines Desktops, der virtuellen Arbeitsfläche eines Rechners, über ein Netzwerk mit mehreren entfernten Rechnern und den gemeinsamen Zugriff auf alle Anwendungen des veröffentlichten Rechners. Collaboration Awareness bezeichnet die direkte Integration der Mehrbenutzer-Funktionalität in entsprechende Werkzeuge, wie beispielsweise integrierte Entwicklungsumgebungen (IDE) mit Mehrbenutzerunterstützung, für den verteilten Einsatz.

Desktop Sharing	**Collaboration Awareness**
+ Unterstützt alle Artefakte	- Unterstützt bestimmte Artefakte
+ Für Demonstrationen geeignet	- Für Demonstrationen nicht geeignet
+ guter Lerneffekt bzgl. Editor-Bedienung	- Schlechter Lerneffekt bzgl. Editor-Bedienung
+ Keine Probleme mit externen Abhängigkeiten	- Evtl. Probleme mit externen Abhängigkeiten
- Geringe Parallelität	+ Hohe Parallelität
- Keine spezielle PP-Unterstützung	+ Spezielle PP-Unterstützung
- Darstellungsprobleme	+ Immer optimale Darstellung
- Hohe Netzlast	+ Geringe Netzlast
- Gesamter Desktop wird frei gegeben	+ Mehr Datenschutz

Tabelle 2.1: Gegenüberstellung von Vor- und Nachteilen der beiden technischen Ansätze zur verteilten Paar-Programmierung [6]

Als Ergebnis der Gegenüberstellung dieser beiden Ansätze (siehe Tabelle 2.1) gelangt Djemili zu dem Ergebnis, dass der Collaboration Awareness Ansatz für die Realisierung einer Werkzeug-Unterstützung für verteilte Paar-Programmierung zu bevorzugen ist. Die notwendigen Voraussetzungen der speziellen Unterstützung der Paar-Programming, der Gewährleistung einer optimalen Darstellung und die geringe Netzlast bildeten die Hauptentscheidungskriterien für die Wahl des Collaboration Awareness Ansatzes.

Dieser Gegenüberstellung schließt sich eine öffentliche Umfrage über die Bedürfnisse und die als notwendig eingeschätzten Funktionen eines Werkzeugs für die Verteilte Paar-Programmierung an. Die Ergebnisse dieser Umfrage flossen in die Design-Entscheidungen und die umgesetzten Funktionalitäten ein und bestätigten den gewählten Ansatz der Collaboration Awareness.

2.3.2 Eclipse-Erweiterung Saros (DPP I)

In diesem Abschnitt werde ich die eingesetzten Technologien zur Umsetzung der Entwurfsentscheidungen und der Eclipse-Erweiterung erläutern, um im späteren Verlauf Probleme und Verbesserungspotentiale aufzeigen zu können.

Zur Realisierung einer Werkzeugunterstützung für verteilte Paar-Programmierung entschied sich Djemili [6] für die Entwicklung einer Java-basierten Eclipse-Erweiterung. Die Kommunikation erfolgt über das XML-basierte Protokoll *Extensible Messaging and Presence Protocol* (XMPP) [7] unter Verwendung der Java API *Smack* [8].

Die Entscheidungskriterien für Java liegen in der hohen Akzeptanz dieser Programmiersprache innerhalb des Instituts für Informatik der FU-Berlin, der persönlichen

Erfahrungen des Autors, sowie der vermuteten potentiellen Weiterentwicklung innerhalb des universitären Umfeldes. Ebenso wie Java hat die Entwicklungsumgebung Eclipse [9] eine große Verbreitung innerhalb des Instituts der Informatik und besitzt vor allem auch außerhalb des universitären Umfeld eine starke Verbreitung, v.a. unter nicht kommerziellen Entwicklungsumgebungen, so dass der Autor hofft einen möglichst großen Nutzerkreis ansprechen zu können. Eclipse besitzt ein quelloffenes, sehr flexibles Erweiterungssystem (plugin system), das eine Integration eigener Erweiterungen aufbauend auf vorhandener Funktionalität ermöglicht. Potentielle Nutzer, die bereits Eclipse als Entwicklungsumgebung einsetzen, können daher bekannte Funktionalitäten weiter nutzen. Die Kommunikation erfolgt über den Nachrichtenaustausch von XMPP-Nachrichten, die innerhalb eines dezentralen Serververbunds von Jabber-Servern vermittelt werden.

Eine elementare Anforderung stellt nach Ansicht von Djemili der vollständige, lokale Zugriff auf alle Dateien des gemeinsames Projekts dar, um alle IDE-Funktionalitäten im vollen Umfang nutzen zu können. Dies wird durch die Verwendung des Replikations-Ansatzes erreicht, der eine komplette lokale Kopie des gemeinsamen Projekts vorsieht. Ein weiterer Vorteil des Replikations-Ansatzes liegt in der verzögerungsfreien, netzwerkunabhängigen Ausführung der lokalen Funktionen, sowie in der anpassungsfreien Verwendung aller bestehenden IDE-Funktionalitäten.

Die Notwendigkeit einer vielschichtigen Awareness in der Werkzeugunterstützung verteilter Paar-Programmierung wurde unter *2.2 (Nachteile verteilter Paar-Programmierung)* bereits aufgeführt. Innerhalb von Saros wird Awareness einerseits über Kontaktlisten und anderseits über visuelle Erweiterungen der bestehenden Eclipse-Oberfläche realisiert. Über eine Roster-Kontaktliste kann der Benutzer die Anwesenheit potentieller Sitzungsteilnehmer sehen und entsprechend einen Teilnehmer aus dieser Liste zu einer gemeinsamen verteilten Paar-Programmierungssitzung einladen. Gleichzeitig kann er Teilnehmern aus dieser Liste unabhängig vom Status der aktuellen Sitzung über eine Instant-Messaging-Funktionalität Nachrichten schicken. Des Weiteren können über eine Sitzungsliste alle an der Sitzung teilnehmenden Benutzer mit ihrer aktiven Rolle angezeigt werden.

Innerhalb einer Projektsitzung wird Awareness in Saros [6] durch die Erweiterung der Eclipse Editor-Komponenten und der Eclipse-Dateiübersicht folgendermaßen realisiert:

1. **Dateimarkierungen**: Innerhalb der Eclipse-Dateiübersicht werden die vom Driver geöffneten Dateien und die derzeit bearbeiteten Dateien farbig hervorgehoben.

2. **Texthervorhebungen**: Innerhalb des Editors werden die Textänderungen des Drivers mit einem farbigen Hintergrund beim Observer dargestellt, um die kürzlich ausgeführten Operationen nachvollziehen zu können. Weiterhin werden Textselektionen sowohl vom Driver als auch vom Observer jeweils beim entfernten Partner farbig hervorgehoben. Dies ermöglicht eine bessere Fokussierung auf relevante Bereiche im Editor innerhalb einer Diskussion zwischen den verteilen Partnern.

3. **Cursorpositionen**: Nicht nur explizite Selektionen von Textstellen, sondern auch die Cursorpositionen der Partner werden im Editor dargestellt.

4. **Sichtbereich**: Der Sichtbereich gibt den im Editor des Drivers dargestellten Bereich innerhalb des aktuell geöffneten Dokuments an.

5. **Verfolger-Modus**: Der vom Observer wählbare Verfolger-Modus ermöglicht ihm, den Aktivitäten des Drivers zu folgen, um einer klassischen Paar-Programmierung möglichst nahe zu kommen. Öffnet der Driver eine Datei, so wird diese bei ausgewähltem Verfolger-Modus auch beim Observer geöffnet, zudem folgt der Observer automatisch dem Sichtbereich des Drivers.

Des Weiteren wurde für Saros die Entscheidung getroffen, sich stark an der Rollenverteilung der regulären Paar-Programmierung zu orientieren. Es existiert zu jedem Zeitpunkt nur ein Driver mit exklusiven Schreibrechten auf allen Projektdateien, sowie ein leseberechtigter Observer. Der Observer kann sowohl im Verfolger-Modus den Driver beobachten, als auch eigenständig mit lesendem Zugriff durch alle Projekt-Dateien navigieren. Dies ermöglicht einen höheren Grad an Parallelität als in der klassischen Paar-Programmierung, da z.B. vom Observer bestimmte Recherchearbeiten parallel zu der Implementierungsarbeit des Driver durchgeführt werden können.

Den beiden Partnern steht es offen, die Rollen innerhalb der Sitzung beliebig häufig zu wechseln, um die Anforderung des Rollenwechsels in der Paar-Programmierung zu realisieren. Der initiale Driver des Projekts kann einen Rollenwechsel initiieren, um seine Driver-Rolle auf den aktuellen Observer zu übertragen, während er gleichzeitig seine eigene Rolle in die Rolle des Observers wechselt. Zu jedem beliebigen Zeitpunkt kann dieser Rollenwechsel durch den initialen Driver wieder rückgängig gemacht werden.

2.3.3 Ausbau der Eclipse-Erweiterung Saros (DPP II)

Aufbauend auf der Diplomarbeit von Djemili und der existierenden Eclipse-Erweiterung Saros entstand im Zuge der Seminararbeit *Weiterentwicklung des Eclipse-Plug-Ins „Saros" zur Verteilten Paarprogrammierung (DPPII)* von Björn Gustavs [10] eine Erweiterung der bestehenden Umsetzung von Saros. Kernpunkt dieser Weiterentwicklung war die Motivation, die Akzeptanz der bestehenden Eclipse-Erweiterung im Umfeld der Open Source Programmierung zu erhöhen.

Um dies zu erreichen, erweiterte Gustavs den bisherigen strikten Paaransatz, der eine Begrenzung auf zwei Sitzungsteilnehmer vorsah. Damit schuf er die Möglichkeit eine beliebige Anzahl von Observern innerhalb einer Projektsitzung zuzulassen. Die bisherige Funktionalität wurde somit auf den Multi-Observer-Modus adaptiert. Dies ermöglicht den potentiellen Anwendern sowohl die Möglichkeit einer verteilten Paar-Programmierung, stark angelehnt an Becks Paradigmen, als auch die Verwendung außerhalb von XP in einem weniger stringenten Ansatz.

2.4 Anforderungen an die Weiterentwicklung

In diesem Kapitel werde ich die Einschränkungen des bisherigen Ansatzes erläutern und daraus die Ziele für die Weiterentwicklung der Eclipse-Erweiterung ableiten.

2.4.1 Replikation eines gemeinsamen Projekts

Durch die Verwendung des Replikations-Ansatzes wird eine vorausgehende Synchronisation der lokalen Projektkopien zu Beginn der Projektsitzung notwendig. Djemili weist bereits während der Konzeption der Eclipse-Erweiterung auf den erheblichen Aufwand dieses Ansatzes hin, da alle Teilnehmer eine identische Version des Projekts besitzen müssen. Vor allem in beliebig komplexen Netzwerktopologien stellt die Synchronisation vieler bzw. großer Dateien eine ernsthafte Hürde dar.

Er sieht jedoch diese Problematik durch die Annahme abgeschwächt, dass potentielle Paare vor Beginn einer Sitzung nur leicht voneinander divergierende lokale Projekt-Versionen besitzen. Dieser Umstand wurde durch die Ergebnisse seiner Umfrage weiter bestätigt. Daher bestand sein Lösungsansatz in dem Abgleich des entfernten Projekts mit einem lokal verfügbaren Projekt. Wird in Saros eine Projektsitzung eingeleitet, so werden beim Observer alle lokal verfügbaren Projekte mit dem des Drivers für die gemeinsame Projekt-Sitzung verglichen und nach Auswahl eines näherungsweise identischen Projekts nur die von

einander abweichenden oder fehlenden Dateien übertragen. Falls kein näherungsweise identisches Projekt existiert müssen alle Projekt-Dateien übertragen werden.

Die eigentliche Übertragung der Dateien erfolgt über die XMPP-basierte Nachrichten-Übermittlung. XMPP-Nachrichten eignen sich jedoch nur bedingt bis unzureichend für eine Übertragung von Dateien, da sie zum Einen nur begrenzte Übertragungsgrößen von wenigen kBytes zulassen und zum Anderen eine durch den Versand über den dezentralen Serververbund basierende relativ hohe Latenzzeit bei einem hohen Nachrichtenaufkommen besitzen, die bei zunehmender Größe der Nachrichten weiter ansteigt.

Durch die Erweiterung der ursprünglichen Teilnehmeranzahl in DPPII von zwei auf konzeptionell beliebig viele Teilnehmer, kann die ursprüngliche Annahme der Existenz fast identischer Projekte bei allen potentiellen Teilnehmern nicht mehr aufrecht erhalten werden. Die Synchronisations-Problematik des Replikations-Ansatzes muss daher durch alternative Lösungskonzepte abgeschwächt bzw. aufgehoben werden, da der Ansatz der derzeitigen Synchronisation eines vollständigen Projekts mit mehreren Teilnehmern zu einem erheblichen Zeitaufwand führt.

2.4.2 Parallelität auf Schreibebene

In der Weiterentwicklung der Eclipse-Erweiterung Saros innerhalb des Folgeprojekts DPP II wurde die zuvor auf zwei Personen limitierte Teilnehmerzahl auf eine beliebige Anzahl von Observern innerhalb der Sitzung erhöht. Ziel war es, die Akzeptanz der Eclipse-Erweiterung zu erhöhen, indem nicht mehr das klassische Programmierer-Paar zusammen an einen Projekt arbeitet, sondern es einer beliebigen Anzahl von Entwicklern ermöglicht wird, die Vorteile der vorhandenen Funktionalität auch in einem größeren Teilnehmerkreis nutzen zu können.

Der Lösungsansatz aus DPP II geht aber in seiner Umsetzung nicht weit genug. Wie im Kapitel 2.2 „Verteilte Paar-Programmierung“ bereits aufgeführt, ist einer der Vorteile in der verteilten Paar-Programmierung die mögliche Parallelität in der Ausführung von Arbeitsschritten. Djemili [6] identifiziert in diesem Zusammenhang drei Stufen der Parallelität:

Parallelität auf Programmebene – *Diese Parallelität ist gegeben, wenn der Observer während einer Sitzung die Möglichkeit hat, das Fenster des DPP-Werkzeugs in den Hintergrund zu schieben, um mit anderen Programmen zu interagieren.*

Parallelität auf Sichtebene *– Diese Parallelität ist gegeben, wenn eine Parallelität auf Programmebene gilt und der Observer die Möglichkeit hat von dem*

Aufmerksamkeitsbereich des Drivers abzuweichen und eigenständig Dateien eines Projekts zu durchstöbern.

Parallelität auf Schreibebene *– Diese Parallelität ist gegeben, wenn eine Parallelität auf Sichtebene gilt und es mehr als einen Driver geben darf, das heißt mehrere Personen bekommen gleichzeitig Schreibzugriff für die Dateien.*

Die Erweiterung um eine beliebige Anzahl von Observern ermöglicht jedoch weiterhin nur Parallelität auf Programm- und Sichtebene, da es auch in dieser Erweiterung nur einen schreibberechtigten Teilnehmer gibt. Dieses Schreibrecht muss nun unter einer beliebig großen Anzahl von Teilnehmern geteilt werden, was diesen Sachverhalt hinreichend komplex macht. Eine Akzeptanzsteigerung durch die Möglichkeit, dass mehr als zwei Personen an einer Projektsitzung teilnehmen können, wird die Notwendigkeit des Teilens eines exklusiven Schreibrechts und der daraus resultierenden problematischen Koordination der Vergabe negativ beeinflusst.

Um die ursprüngliche Akzeptanzsteigerung des Projekts DPP II zu verwirklichen, muss also eine Parallelität auf Schreibebene ermöglicht werden.

Die bisherige Konzeption eines Werkzeugs für verteilte Paar-Programmierung bildet einen Sonderfall für eine Werkzeug-Unterstützung aus dem Bereich der *Computer Supported Collaborative Work* (CSCW) für eine synchrone Zusammenarbeit von mehreren Teilnehmer in einer verteilten Umgebung ab. Durch die Erweiterung von Saros durch eine Parallelität auf Schreibebene könnten die bestehenden Funktionalitäten auch in einem erweitertem Umfeld der CSCW-Systeme erfolgreich eingesetzt werden.

2.4.3 Kommunikationsmittel

Konzeptionell war Saros für die Kommunikation zwischen zwei Teilnehmer ausgelegt und besitzt daher zusätzlich zur impliziten Kommunikation über die Awareness im Editor-Bereich einen Private-User-Chat, der den Austausch von Nachrichten zwischen zwei Teilnehmern ermöglicht.

Innerhalb einer Mehrbenutzer-Umgebung von mehr als zwei Teilnehmern ist der Funktionsumfang dieser Chat-Kommunikation nur begrenzt nutzbar. Beispielsweise müsste der Driver weiterführende Erklärungen immer explizit an jeden einzelnen Observer versenden, damit sich alle Observer auf demselben Kenntnisstand befinden.

Aus diesem Grund muss das bestehende Kommunikationsmedium des Nachrichtenaustauschs um die Funktion eines Multi-User-Chats erweitert werden, so dass Nachrichten an alle Teilnehmer der Sitzung verschickt werden können. Darüber hinaus sollte die Möglichkeit der Integration zusätzlicher Kommunikationsmedien wie Audio und Video in Betracht gezogen werden.

3. Analyse des bestehenden Replikations-Ansatzes

Wie im vorhergehende Abschnitt erläutert, bildet die Synchronisation und der damit verbundene Austausch großer Datenmengen eine sehr kritische Anforderung an ein Werkzeug für verteilte Paar-Programmierung mit mehr als zwei Teilnehmern. Basierend auf einer Analyse der Dateiübertragung und des gesamten Replikations-Prozesses der bestehenden Umsetzung nach Abschluss des Projekts DPP II, werden alternative Lösungsszenarien erarbeiten und ein Konzept für die Optimierung des Replikations-Prozesses vorstellt. Die Analyse betrachtet den konzeptionellen Ablauf einer Projektinitialisierung und den technischen Ansatz, sowie darüber hinaus die Umsetzung des Datenabgleichs.

3.1 Einführung in die Dateiübertragung

In diesem Abschnitt werde ich die technische Umsetzung der Projektsynchronisation im Hinblick auf die Dateiübertragung untersuchen, um im folgenden Abschnitt die verschiedenen Phasen unter Berücksichtigung der diagnostizierten Probleme betrachten und bewerten zu können.

3.1.1 Grundlagen

Der gesamten Informationsaustausch zwischen den verteilten Teilnehmern basiert auf Jabber, einer Sammlung XML-basierter Netzwerkprotokolle, deren Kern das XMPP Protokoll (*Extensible Messaging und Presence Protocol*) bildet. XMPP ist ursprünglich aus dem Jabber Projekt hervorgegangen, welches von Jeremie Miller 1998 gestartet wurde, um eine quelloffene Alternative zu den bis dato bestehenden proprietären Instant Messaging Lösungen zu schaffen und wird derzeit von der XMPP Standard Foundation spezifiziert und weiterentwickelt. Seit Anfang 2004 ist XMPP als Internetstandard für Instant Messaging durch IETF verabschiedet. Die XMPP-Nachrichten werden in einem dezentralisierten Serververbund, ähnlich der SMTP Netzarchitektur, durch einen sogenannten Jabber Identifier (JID) identifiziert und adressiert [11].

Zusätzlich zu den Nachrichten- und Anwesenheitsprotokollen existieren unter dem Namen *XMPP Extension Protocols (XEP)* [12] XMPP-Erweiterungen für erweiterte Funktionalitäten wie beispielsweise Dateiübertragung oder Konferenzen. Diese Protokolle werden in einem standardisierten Entwicklungsprozess von der XMPP Standards Foundation überwacht.

Für die technische Umsetzung des Jabber-Protokolls findet in Saros die quelloffene Softwarebibliothek für XMPP *Smack* [8] in der Version 2.2.1 Verwendung. Die Dateiübertragung in Saros kann sowohl über eine XMPP-Erweiterung, als auch über eine eigenständige Umsetzung mittels XMPP-Nachrichten geschehen.

3.1.2 Dateiübertragung nach XEP-0096

XEP-0096 File Transfer [13] ist ein von der XMPP Standards Foundation (XSF) spezifiziertes Protokoll für eine Dateiübertragung zwischen zwei XMPP Teilnehmern.

Diese Spezifikation bietet ein Konzept zur Lösung der Probleme der bisherigen „traditionellen“ Out-Of-Band Übertragung an. Die Probleme mit einer Out-Of-Band Übertragung bestanden in der fehlenden Zuverlässigkeit der Verbindung, sowie in der Problematik des Verbindungsaufbaus zwischen XMPP Teilnehmern, die sich hinter Firewalls oder NATs (Network Address Translation) befinden.

XEP-0096 bietet Mechanismen für eine zuverlässige Dateiübertragung mit entsprechenden Ausweichmechanismen (engl. fallback) und sichert den Verbindungsaufbau auch bei oben beschriebenen Netzwerktopologien. Das Protokoll ermöglicht vor Beginn der Dateiübertragung den Austausch von Informationen der zu übertragenen Dateien und der möglichen Übertragungskanäle. In Abhängigkeit dieser Informationen wird die eigentlichen Dateiübertragung entweder über einen Proxy-Server (Socks5) oder als XMPP-Nachrichten (In-Band-Bytestream) versendet. Falls eine Socks5 Übertragung nicht zustande kommt, erfolgt ein automatisches Ausweichen auf eine In-Band-Bytestream Übertragung.

Stream Initiation

XEP-0095 Stream Initiation [14] spezifiziert die Protokoll-Erweiterung, um den Verbindungsaufbau zwischen zwei XMPP Teilnehmern zu initiieren und durchzuführen. Das Protokoll erlaubt das Aushandeln einer gemeinsamen Verbindung und ermöglicht den Versand von Meta-Daten über zur Verfügung stehende Verbindungen.

Bevor eine Verbindung aufgebaut werden kann, werden über ein *XEP-0030 Service Discovery Protokoll* [15] beim verbundenen Jabber Server die zur Verfügung stehenden Protokolle (Socks5, IBB) für die Dateiübertragung ermittelt.

Socks5-Bytestream

Das *XEP-0065 Socks5-Bytestream* Protokoll [16] ermöglicht den Aufbau eines *Out-Of-Band Bytestreams* für eine Dateiübertragung zwischen zwei XMPP Teilnehmer mit Socks5. Der Bytestream kann sowohl direkt (Punkt-zu-Punkt Verbindung) oder vermittelt über spezielle Socks5-Proxy-Server erfolgen. Eine Direktverbindung kann jedoch nur erfolgen, wenn eine Verbindung zwischen den Teilnehmer nicht von Firewalls oder NATs verhindert wird. In

diesen Fällen wird versucht, eine Verbindung über einen Proxy Server aufzubauen. Die Verfügbarkeit eines Socks5-Proxy-Dienstes ist jedoch vom entsprechenden Jabber-Server abhängig, der diese Funktionalität zur Verfügung stellen muss.

In-Band-Bytestream

Das *XEP-0047 In-Band-Bytestream (IBB)* Protokoll [17] ist das Ausweichprotokoll der *XEP-0096 File Transfer* Spezifikation. Wenn eine Socks5 Verbindung nicht zustande kommen kann, wird auf IBB „umgeschaltet". Die Dateien werden in kleinere Pakete aufgeteilt und als XMPP Nachricht übertragen und beim empfangenen Teilnehmer wieder zusammengefügt. Diese Übertragungsform ist jedoch bei größeren Dateien sehr langsam, da jede Datei in sehr viele, nur wenige kByte große Blöcke aufgeteilt werden muss, die ihrerseits innerhalb des dezentralen Serververbunds als XMPP-Nachricht einzeln zugestellt werden müssen.

3.1.3 Nachrichten-basierte Dateiübertragung

Im Verlauf des Projekts DPP II stellte Gustavs erhebliche Einschränkungen in der Funktionsweise der Dateiübertragung mit dem XEP-0096 File Transfer Protokoll in Saros fest. Diese Einschränkungen werden in einem späteren Abschnitt detaillierter analysiert, zusammenfassen ist an dieser Stelle nur bemerken, dass eine Dateiübertragung nur in sehr wenigen Fällen zustande kommen kann.

Als alternative Lösungsstrategie wurde in DPP II die Dateiübertragung durch eine eigene auf XMPP-Nachrichten basierende (Chat-basierte) Implementierung nach dem Vorbild des In-Band-Bytestream Protokolls erweitert. Hierbei werden die Dateien in Blöcke aufgeteilt und als Base64-codierte Chat-Nachrichten innerhalb einer eigenen Erweiterung übertragen und auf der Empfänger-Seite zusammengesetzt.

Diese Art der Dateiübertragung wurde nicht als automatisches Ausweichprotokoll umgesetzt, sondern muss in den Saros Einstellungen manuell aktiviert werden. Durch die Aktivierung wird jedoch die API-basierte Dateiübertragung deaktiviert, so dass eine situationsbedingte Auswahl der Übertragungsart nicht möglich ist.

3.2 Analyse der Dateiübertragung

Die im vorherigen Abschnitt vorgestellten Ansätze der Dateiübertragung werden im Folgenden unter Berücksichtigung ihrer Funktionstüchtigkeit in der Saros Umsetzung nach Abschluss des Projekts DPP II und im Hinblick auf die veränderten Anforderungen an die Projekt-Synchronisation untersucht.

3.2.1 Dateiübertragung mit XMPP File Transfer Protokoll

Die unter XEP-0096 spezifizierten Protokolle sollten eine verbindungsorientierte Dateiübertragung einer einzelnen Datei ermöglichen. Zur Verbindungssicherung kann mit Hilfe des Stream Invitation Protokolls die Art der Verbindung ausgehandelt werden. Die eigentliche Übertragung wird durch Meta-Daten unterstützt, die Informationen über den Verlauf und das Ergebnis der Dateiübertragung bereitstellen. Somit können Fehler im Vorfeld oder während der Dateiübertragung erkannt werden. Nach erfolgreicher Übertragung oder nach einem Übertragungsabbruch wird die Verbindung wieder beendet und muss für eine weitere Datei neu aufgebaut werden.

Dieses Verhalten wirkt sich jedoch negativ bei einer Projekt-Synchronisation von vielen Dateien aus. Da die ursprüngliche Annahme der Synchronisation von vereinzelten Dateien für eine Projektsynchronisation in einem Umfeld beliebig vieler Teilnehmer nicht mehr aufrecht erhalten werden kann, muss die Dateiübertragung unter dem Aspekt der Synchronisation aller Projektdateien betrachtet werden. Java Projekte beispielsweise bestehen zum Großteil aus vielen Quellcode Dateien, die jeweils jedoch nur aus wenigen Kilobyte Daten bestehen. Zusätzlich existieren Bibliotheken, die mehrere MegaByte in Anspruch nehmen können. Bei einer Übertragung mit der XEP-0096 File Transfer XMPP Erweiterung kommt es vor Beginn der Übertragung bei jeder Datei zu einer Verbindungsaushandlung, die pro Datei mehrere Sekunden in Anspruch nehmen kann. Bei sehr vielen kleinen Dateien bedeutet dies einen erheblichen zeitlichen Aufwand, der die eigentlichen Übertragungszeit übersteigen kann und zu einer erhebliche Verzögerung des gesamten Synchronisationsprozesses führen kann.

Zudem ist die Überprüfung der Verbindungsart vor jeder Dateiübertragung innerhalb eines Projekt-Synchronisationsprozesses nur bedingt sinnvoll. Das Zustandekommen einer Socks5-Bytestream Verbindung ist sowohl von den Eigenschaften des Jabber-Servers als auch von Firewalls und der NAT-Topologie abhängig. Schlägt der Verbindungsversuch einer Socks5-Bytestream Verbindung während der Übertragung der ersten Projekt-Dateien fehl, ist die Wahrscheinlichkeit des Zustandekommens dieser Verbindungsart für die Übertragung der verbleibenden Dateien gering, da nicht zu erwarten ist, dass sich die abhängigen Komponenten in so kurzen Abständen ändern.

Die Übertragungsdauer hängt zudem stark vom eingesetzten Protokoll für die eigentliche Übertragung ab. Kann eine Socks5-Verbindung nicht zustandekommen, so wird auf das IBB-Ausweichprotokoll ausgewichen, was im Besonderen bei der Übertragung von größeren Bibliotheksdateien zu einer erheblichen zeitlichen Verzögerung führt. Um eine Socks5-

Proxy-Verbindung überhaupt ermöglichen zu können, muss der für diese Verbindung notwendig Proxy-Port zur Verfügung stehen. In privaten Netzwerken setzt dies die manuelle Konfiguration der entsprechenden NAT-Router und Firewalls voraus, in öffentlichen oder Firmen-internen Netzen ist eine solche Konfiguration nur bedingt möglich, so dass eine Socks5-Bytestream Verbindung in solchen Fällen nicht aufgebaut werden kann.

3.2.2 Dateiübertragung mit Chat-basierter Umsetzung

Die Chat-basierte Dateiübertragung ist in der Funktion mit dem standardisierten IBB Protokoll XEP-0047 [17] vergleichbar. Die Nachteile des IBB-Protokolls im Hinblick auf die Latenzzeit können daher auf die Chat-basierte Umsetzung übertragen werden.

Als Vorteil gegenüber dem XEP-0096 Protokoll kann die sofortige Übertragung aller Dateien ohne Verbindungsprüfung angesehen werden. Es wird nicht für jede Datei eine Verbindungsart ausgehandelt, sondern die Dateien werden sofort als Chat-Nachrichten übertragen. Im Gegensatz zum IBB Protokoll findet hier jedoch keine Verbindungssicherung während der Dateiübertragung statt, sondern alle Dateien werden aufgeteilt in einzelne nur wenige Kilobyte große Chat-Nachrichten ohne Prüfung oder zusätzliche Meta-Daten hintereinander versandt. Die Reihenfolge ist ausschließlich auf Grundlage eines internen Zählers gesichert. Die Problematik der Dateiübertragung von vielen Dateien innerhalb eines Projekt-Synchronisations-Prozesses sollte jedoch nicht auf der Übertragungsebene mit Hilfe eines eigenen Protokolls erfolgen, deren Vorteil einzig aus dem Fehlen notwendiger wichtiger Funktionen für die Dateiübertragung besteht. Es stellt vielmehr ein konzeptionelles Problem in Verbindung mit der Verwendung von XMPP Protokollen innerhalb der Projektsynchronisation dar.

Die Beweggründe für die Entwicklung dieses eigenen Protokolls lagen in der bis dato fehlenden Funktionsfähigkeit der Smack-Umsetzung des XEP-0096 Protokolls. Sollten diese Einschränkungen behoben sein, sollte auf die Jabber-spezifizierten Protokolle für eine Dateiübertragung zurückgegriffen werden, da eine IBB-Verbindung aufgrund ihrer hohen Latenzzeit für die Übertragung nur als letzte Ausweichlösung betrachtet werden darf. Dieses eigene Protokoll bietet für die Übertragung einer einzelnen Datei keinen Vorteil, sondern setzt nur Teile eines bereits bestehenden offiziellen Protokolls um.

Die XEP-0096-basierte Übertragung muss bisher explizit in der Konfiguration ausgeschaltet werden, damit die Chat-basierte Dateiübertragung verwendet werden kann. Somit ist eine situationsbedingte Auswahl der geeigneten Übertragungsart unter Verwendung dieses Protokolls nicht mehr möglich.

3.2.3 Einschränkungen der Smack Bibliothek

Für die Realisierung der Dateiübertragung in Saros wurde eine angepasste Smack Bibliothek in der Version 2.2.1 eingesetzt. Die Smack Bibliothek implementiert die XMPP-Protokolle auf Grundlage der vorgegebenen XMPP-Spezifikation.

Proxy-Port

Es besteht das Problem innerhalb der Smack API, dass der notwendige Server-Port für eine Socks5-Bytestream Verbindung nicht einstellbar ist, sondern über den fest eingestellten Port 7777 erfolgen muss. Somit kann keine Anpassung der Port-Konfiguration an bestehenden Firewall- oder Router-Konfigurationen erfolgen, was einen erfolgreichen Verbindungsaufbau verhindert, obwohl die Infrastruktur bei vorhandenem offenen Port einen Verbindungsaufbau zulassen würde. Diese Einschränkung ist der Smack Community bereits seit längerem bekannt und wurde als Verbesserung [18] aufgenommen, jedoch bis jetzt umgesetzt.

Um eine Port-Einstellung vornehmen zu können, wurde im Verlauf des Projekts DPP I die Smack Bibliothek manuell mittels eines inoffiziellen Patch erweitert. Wie eine Untersuchung des Patches ergab, wurde die API mit diesem Patch nicht nur um die Einstellmöglichkeit der Ports erweitert, sondern es wurden zusätzlich tiefgreifende Veränderungen innerhalb der Socks5 Implementierung der Smack API vorgenommen. Diese Veränderungen wurden nicht dokumentiert. Als Folge unterscheidet sich die Spezifikation der XEP-0096 Dateiübertragung von der eingesetzten Implementierung.

Blockgröße für IBB-Protokoll

In der aktuellen Version der Smack Bibliothek lässt sich die Blockgröße für die IBB Übertragung nicht einstellen, sondern ist mit 4096 Kilobyte festgesetzt. Jabber-Server lassen unter Umständen aber das zehnfache an Blockgrößen für eine IBB Übertragung zu. Da jede verschickte Nachricht eine Verzögerung bedeutet, könnte der Versand von weniger Nachrichten mit größeren Paketen die Geschwindigkeit der IBB Übertragung erhöhen.

Funktionseinschränkungen der API

Die XEP-basierte Dateiübertragung war nach Abschluss des Projekts DPP II weder in komplexen Netzwerktopologie noch in einem einfach privaten Netzwerk möglich. Nur durch explizites Unterbinden einer XEP-basierten Dateiübertragung war durch Verwendung des eigenen Chat-basierten Übertragungsprotokolls eine Dateiübertragung möglich. Bei dem Versuch einer XEP-basierten Dateiübertragung erfolgte nach dem Verbindungsaufbau kein Datenaustausch, es kam stattdessen innerhalb der Projekt-Synchronistation zu einer

Verklemmung (deadlock), die nur durch den expliziten Abbruch durch den Benutzer beendet werden konnte.

Die nicht dokumentierten Änderungen an der Smack Bibliothek wurden bereits im Abschnitt Proxy-Port aufgeführt und führten zu einem nicht mehr nachvollziehbaren Verhalten der Dateiübertragung, da dieses von der XEP-0096 Spezifikation abwich.

3.3 Analyse der Projekt-Synchronisation

Die bestehende Konzeption einer Projekt-Synchronisation zur Realisierung des Replikations-Ansatzes basiert, wie bereits einführend in Kapitel 2.4 dargelegt, auf dem Konzept eines Projekt-Abgleichs zwischen zwei Teilnehmern vor Beginn der Paarsitzung. Im Hinblick auf die Weiterentwicklung der Werkzeug-Unterstützung im Rahmen des Projekts DPP II, von zwei auf eine beliebige Teilnehmeranzahl, muss dieses Konzept neu bewertet werden, da Grundannahmen aus der ursprünglichen Konzeption nicht mehr gegeben sind.

3.3.1 Grundkonzept der Projekt-Synchronisation

Nach der Veröffentlichung eines Projekts (engl. Share Project) in Saros können Teilnehmer zu einer verteilten Paar-Programmierungs-Sitzung (DPP-Sitzung) eingeladen werden. Innerhalb dieses Einlade-Prozesses findet die Projektsynchronisation auf Basis des Replikations-Ansatzes statt.

Das Konzept der Projektsynchronisation wird von Djemili [6] im Abschnitt *5.3.3 Einladungen* beschrieben. Es werden hierbei die Rollenschreibungen „Einlader“ und „Eingeladener“ verwendet. Die einzelnen Phasen werden im folgenden sinngemäß vorgestellt.

1. **Einlader sendet Einladung:** Initialisierung durch Senden einer Einladungs-Nachricht als XMPP-Nachricht.

2. **Eingeladener sendet Bestätigung:** Senden einer Bestätigungs- oder Ablehnungs-Nachricht als XMPP-Nachricht zur Ausführung oder zum Abbruch der eigentlichen Projektsynchronisation.

3. **Einlader sendet Dateiliste:** Einlader erstellt eine Dateiliste aller Projektdateien mit Pfadangaben und sendet diese als Datei als den Eingeladenen.

4. **Eingeladener sendet Dateiliste:** Der Eingeladene empfängt die Dateiliste. Dem Ansatz des Kopieren von lokalen Ressourcen folgend wird für jedes Projekt im Eclipse-Workspace des Eingeladenen eine Dateiliste erstellt und mit der empfangenen Dateiliste verglichen. Alle Projektdateien des lokalen Projekts, dessen Dateien am ehesten mit dem entfernten Projekt übereinstimmen, werden in ein neues Projekt kopiert. Die Dateiliste des lokalen Projekts wird an den Einlader als Datei versandt.

   Dieses Vorgehen wurde in DPP II dahingehend angepasst, dass dem Eingeladenen die Auswahl zur Verfügung steht, auch das entsprechende Projekt fortzuführen, sollte es sich bereits in seinem Eclipse-Workspace befinden.

5. **Einlader sendet fehlende Ressourcen:** Nach Empfang der entfernten Dateiliste und dem Vergleich mit der lokalen Dateilist, sendet der Einlader nacheinander die fehlenden Dateien an den Eingeladenen.

6. **Einlader empfängt fehlende Ressourcen:** Der Eingeladene empfängt alle fehlenden Ressourcen. Nach dem Empfang der benötigten Dateien werden alle lokalen Dateien, die nicht zu dem entfernten Projekt gehören, gelöscht.

3.3.2 Einschränkungen

Das vorgestellte Konzept baut auf der Annahme auf, dass ein Großteil der vorhandenen Ressourcen zur Verfügung stehen. Aus diesem Grund wird immer ein Einlesen aller Projekte des Workspaces gestartet. Bei einer Vielzahl von Projekten führt dieser Ansatz zu einer großen Verzögerung, die anhängig von der Rechnerleistung und der Projektanzahl / -größe den Einladungsprozess unnötig verzögert.

Dem Eingeladenen steht nicht die Möglichkeit zur Verfügung, selbstständig ein bestehenden Basis-Projekt auszuwählen, er hat nur die Möglichkeit ein neues Projekt oder das vorgeschlagene Projekt als Grundlage für die Projekt-Synchronisation zu wählen. Die Bestimmung des wahrscheinlichsten Projekts basiert auf dem Vergleich der Dateilisten und stellt nur bedingt ein ausreichendes Mittel zur Wahl des geeigneten Projekts dar.

3.4 Fazit

Dem Anwender muss auf Seiten des Eingeladenen mehr manuelle Eingriff-Möglichkeit gegeben werden, da die bisherige Umsetzung dies nur bedingt ermöglicht. Der Fokus muss

an dieser Stelle vor allem auch auf der Art der Dateiübertragung liegen. Die Entscheidung für die Replikation lokaler Ressourcen muss die Verbindungsmöglichkeiten mit einbeziehen, da die hohe Wahrscheinlichkeit eines zur Verfügung stehenden ähnlichen lokalen Projekts nicht mehr gewährleistet werden kann.

Der aktuelle Lösungsansatz der Dateiübertragung während der Projekt-Synchronisation besitzt konzeptionelle sowie technische Einschränkungen. Das Aushandeln einer Verbindung vor jeder Übertragung mit XEP-0096 erhöht den zeitlichen Aufwand in einem unvertretbaren Maße. Zudem bietet die technischen Umsetzung in der aktuell verwendeten Smack Bibliothek keine stabile Grundlage. Die im Projekt DPP II realisierte eigene chat-basierte Dateiübertragung stellt im Hinblick auf die Notwendigkeit einer schnellen Übertragung vieler Dateien keine Alternative dar.

Eine Dateiübertragung allein basierend auf dem XEP-0096 Protokoll ist für die Dateiübertragung ganzer Projekte nur als Ausweichlösung geeignet. Direktverbindungen (Punkt-zu-Punkt Verbindung, kurz P2P Verbindung) würden für die Projekt-Synchronisation einen erheblichen Mehrwert bringen, da die Übertragungsgeschwindigkeit nur noch abhängig von der zur Verfügung stehen Bandbreite wäre. Im folgenden Kapitel werden daher die Grundlagen für den Aufbau einer Direkt-Verbindung zwischen den Teilnehmern und Lösungsansätze für die Integration in Saros erarbeitet.

4. P2P Kommunikation in Weitverkehrsnetzen

Der Aufbau einer Punkt-zu-Punkt Verbindung (kurz P2P) bzw. direkten Verbindung zwischen zwei Rechnern außerhalb eines lokalen Netzwerks stellt eine große Herausforderung dar. In Weitverkehrsnetzen (globales Internet) können alle Geräte nicht mit einer global eindeutigen IP-Adresse adressiert werden, da die Vielzahl von Geräten den IP-Adressbereich des aktuell eingesetzten IPv4 Protokolls übersteigt. Da öffentliche IP-Adressen immer knapper werden, müssen in lokalen (privaten) Netzwerken private IP-Adressen zum Einsatz kommen, die auf den öffentlichen IP-Adressen abgebildet werden müssen. Um eine übergreifende Kommunikation zu ermöglichen, müssen die internen privaten IP-Adressen an der Schnittstelle zum Internet (oder einem anderen Netzwerk) in öffentliche IP-Adressen übersetzt werden.

NAT (Network Address Translation) stellt ein Verfahren zur Abbildung der privaten IP-Adressen auf öffentliche IP-Adressen dar, welches 1994 von *IETF (Internet Engineering Task Force)* [19] als *RFC (Requests for comments)*[1] [20] veröffentlicht wurde. Hierbei bezeichnet NAT sowohl die eigentliche Translation, als auch das jeweilige Gerät oder die jeweilige Software, die diese Abbildung vornimmt.

4.1 Network Address Translation (NAT)

Zur Darstellung der Funktionsweise von NAT nehmen wir einen NAT-Router an, der den Translations-Vorgang zwischen einem lokalen Netzwerk und dem öffentlichen Netz vornimmt. Hierbei wird während dieses Vorgangs zwischen der *Source-NAT,* bei dem die Quell-Netzwerk-Adresse ersetzt wird, und der *Destination-NAT*, bei dem die Ziel-Netzwerk-Adresse ersetzt wird, unterschieden. Die Netzwerk-Adresse besteht hierbei aus der IP-Adresse und dem zugehörigen Ports für UDP / TCP.

Der NAT Router empfängt ein IP-Paket aus dem lokalen Netz und speichert die Adresse in einer NAT-Tabelle ab. Im Anschluss ersetzt er die lokale IP-Adresse im Paket durch die öffentliche IP-Adresse und sendet das Paket an die Ziel-Adresse. Nun akzeptiert der NAT-Router eingehende Pakete von dieser öffentlichen Adresse *(NAT-Endpunkt)* und leitet sie an den privaten Host weiter. In diesem Zusammenhang sind folgende Punkte zu beachten:

– Die Abbildung hängt von der Port-Nummer des Senders ab. Zwei verschiedene Ports des privaten Hosts ergeben zwei unterschiedliche NAT-Endpunkte.

[1]RFCs ist eine vom IETF veröffentlichte Dokumenten-Sammlung zur Festlegung eines Internet Standards, welche technische Festlegungen und Übereinkommen beinhaltet.

– Der private Host muss als erster ein Paket senden, da ansonsten kein eingehendes Paket an ihn weitergeleitet werden kann.

– NAT sollte völlig transparent im Hintergrund ablaufen, so dass der eigentliche Anwender bzw. Client von diesen Vorgängen nichts mitbekommt.

4.2 NAT-Traversal Problematik

Ein großes Problem im Zusammenhang mit NAT Routern stellt die fehlende Standardisierung dar. Das eigentliche Verhalten der Translation kann sich daher unterscheiden. Abhängig von ihrem Verhalten können NATs in vier Klassen ([21],[22]) eingeteilt werden: *Full Cone NAT*, *Restricted Cone NAT*, *Port Restricted Cone NAT* und *Symmetric NAT*

Full Cone NAT

Bei einem *Full Cone NAT* werden alle Anfragen einer internen Netzwerkadresse (bestehend aus IP-Adresse und Port) auf eine statische externe Netzwerkadresse abgebildet. Es erlaubt insbesondere, dass externe Hosts über die externe Adresse des NAT-Gateways Verbindungen zu internen Hosts aufbauen.

Restricted Cone NAT

Die internen Netzwerkadressen werden wie beim Full Cone NAT statisch auf externe Netzwerkadressen abgebildet. Im *Restricted Cone NAT* erlaubt das Gateway die Kontaktaufnahme eines externen mit einem internen Host jedoch nur, wenn diesem Verbindungsversuch eine Kontaktaufnahme des internen Hosts zur IP-Adresse des externen Hosts vorausging.

Port Restricted Cone NAT

Das *Port Restricted Cone NAT* schränkt die Voraussetzung aus dem Restricted Cone NAT-Szenario für der Kontaktaufnahme des externen Hosts zusätzliche auf den externen Port ein, über den die vorausgangene umgekehrte Kontaktaufnahme stattfand.

Symmetric NAT

Im *Symmetric-NAT* werden alle Anfragen einer internen Netzwerkadresse zu einer spezifischen Empfänger-Adresse auf eine externe abgebildet. Ein Paket vom selben Host mit der gleichen Netzwerkadresse aber einer anderen Empfänger-Netzwerkadresse wird auf

eine andere externe Netzwerkadresse abgebildet. Nur der externe Host, der zuvor das Paket erhalten hat, kann ein Paket über die externe Netzwerkadresse an den internen Host senden.

Diese Klassifizierung bildet jedoch nur eine Grundklassifizierung für verfügbare NAT-Systeme, da sich diese oft aus Mischformen dieser klassischen Ansätze zusammensetzen und zwischen zwei oder mehr Verhaltensmustern dynamisch wechseln.

Probleme

NAT stellt jedoch nur einen vorübergehenden Ansatz zur Lösung der Adressenknappheit des IPv4 Protokolls dar. Das größte Problem besteht aus der fehlenden eindeutigen Zuordnung eines Hosts zu einer eindeutigen Adresse. Es können sich durch das Umschreiben der Adressen in den Paketen eine Vielzahl von Protokollkomplikationen ergeben [23]. Sowohl Verschlüsselungsverfahren auf Netzwerk- und Transportebene mit NAT, als auch Out-of-Band Netzwerkdienste mit Rückkanälen, wie z.B. VoIP-Protokolle [22], haben große Probleme mit diesem Vorgehen. Das NAT Verfahren setzt sich zudem über die strenge Trennung des OSI-Schichtenmodells hinweg.

4.3 Konzepte zur Lösung der NAT Problematik

Es existieren unterschiedliche Ansätze der NAT Problematik zu begegnen und einen Verbindungsaufbau zu ermöglichen. Diese Verfahren können unter dem Oberbegriff *NAT-Traversal* zusammengefasst werden und beinhalten sowohl manuelle als auch automatische Verfahren.

4.3.1 Router Konfiguration

NAT-Traversal kann eingeschränkt auf lokale Netzwerke am lokalen NAT-Router durchgeführt werden, um mittels manueller oder spezieller Routerfunktionalität Einstellungen im Router vornehmen zu können.

Port Forwarding

Die Technik des Port Forwarding ist eine manuelle Konfiguration des NAT-Routers mit dem Ziel, dass bestimmte Datenpakete an einen bestimmten lokalen Rechner weitergeleitet werden können. Es wird die Port-Nummer (oder der Port-Bereich) und die IP-Adresse des lokalen Rechners benötigt, so dass der lokale Rechner über die feste Weiterleitung anhand der Port-Nummer außerhalb des lokalen Netzwerks erreichbar ist.

Diese Technik hat den großen Nachteil, dass ein Zugang zum Router und das entsprechende Verständnis Voraussetzung ist. Die festgelegten Ports (oder Port-Bereiche) können somit nur von einem lokalen Rechner verwendet werden. Andere Rechner innerhalb desselben Netzwerks können diese nicht mehr nutzen.

Ein weiteres Problem dieses Verfahrens stellt die dynamische Port-Vergabe von Anwendungen zur Laufzeit dar. Da die verwendeten Ports vorher nicht bestimmt werden können, ist eine manuelle Voreinstellung nicht möglich.

Universal Plug and Play (UPnP)

UPnP ist ein Ansatz, um Port Forwarding innerhalb des Routers zu automatisieren und somit die Nachteile der manuellen Konfiguration des Port Forwardings zu kompensieren. UPnP wurde nicht für NAT-Traversal entwickelt, sondern für eine herstellerunabhängige Steuerung von elektronischen Geräten über ein IP-Netzwerk. Es kann jedoch für NAT-Traversal verwendet werden [24]. Der NAT-Router muss eine UPnP-Unterstützung beinhalten, um Anwendungen des lokalen Client-Rechners eine Router-Konfiguration über die UPnP-Schnittstelle zu ermöglichen.

Dieser Ansatz ist jedoch nur in privaten lokalen Netzwerken sinnvoll einsetzbar, da die programmgesteuerte, automatische Konfiguration des Routers ansonsten ein erhebliches Sicherheitsrisiko darstellt.

4.3.2 STUN

Simple Traversal of UDP through NATs (STUN) ist ein Client-Server-Protokoll zur Realisierung eines Datenaustauschs zwischen Client und Server durch einfache Request und Response-Nachrichten auf Basis von UDP. Es wurde 2003 als Standard definiert [21] und zählt zu einem der ersten Lösungsansätzen der NAT-Problematik.

Der Empfang von Paketen wird zum einen von der Unkenntnis der Ziel-IP-Adresse des sendenden Hosts, im Zusammenhang mit der Tatsache beeinträchtigt, dass auch der entfernte Host seine IP-Adresse nicht kennt. Zum anderen kann die Weiterleitung der Pakete durch NATs verhindert werden. Mit Hilfe des STUN Protokolls ist es dem Client möglich, die Konstellation eventuell verfügbarer NATs zwischen sich und dem öffentlichen Internet zu bestimmen. Desweiteren kann er den NAT-Typ und seine öffentliche IP-Adresse inklusive eines verfügbaren offenen Ports bestimmen. Auf Grund dieser Informationen ist es nun möglich eine P2P Kommunikationen aufzubauen.

Es handelt sich beim STUN Protokoll um ein Request-Response-Protokoll zwischen einem STUN Client und einem im öffentlichen Internet befindlichen STUN Server. Durch eine Strategie von aufeinander folgenden Anfragen des Clients an den Server kann der Client aus den Antworten bzw. dem Ausbleiben von Antworten Rückschlüsse auf die Umgebung, die eingesetzten NATs, sowie seine IP-Adresse und die verfügbaren Ports ziehen.

Es sind durch den Einsatz von STUN, im Gegensatz zu Port Forwarding und UPnP, keine Modifikationen an der NAT-Router-Konfiguration notwendig. Die Unterstützung von vielen NAT-Typen, mit Ausnahme des symmetrischen NAT, sorgt für eine große Verbreitung dieses Protokolls, welches jedoch nur auf UDP Verbindungen beschränkt ist. Zusätzlich zum eigentlichen NAT-Typ ist der Erfolg des NAT-Traversal Verfahrens jedoch auch abhängig von der verwendeten Router-Hardware und den entsprechenden Firewalls. Aktuelle Studien [25] belegen hierbei jedoch einen erfolgreichen Verbindungsaufbau in ca. 90% aller Fälle.

4.3.3 Traversal Using Relay NAT (TURN)

Traversal User Relay NAT (TURN) befindet sich in der Entwurfsphase als Internet Standard Protokoll [26] und wurde als Erweiterung des bestehenden STUN Protokolls entwickelt. Die beim STUN Protokoll bestehende Problematik mit symmetrischen NATs wird in TURN durch den Ansatz einer Relay-basierten Verbindung mit dem Ziel der Simulation restriktiver NATs begegnet. Im Gegensatz zu STUN ermöglicht TURN die Verwendung von UDP und TCP als Kommunikationsbasis.

Der Client kommuniziert mit einem öffentlichen TURN-Server über einfache Request- und Response-Nachrichten. Der TURN-Server dient hierbei als Datenrelay und leitet Daten zwischen dem Client und den externen Peers weiter. Er stellt dem Client hierfür eine öffentliche Netzwerkadresse zur Verfügung, die der Client als emulierte Absenderadresse verwendet, um von beliebigen externen Peers erreichbar zu sein.

Funktionsweise

Der Client befindet sich in einem privaten Netzwerk hinter einem NAT, der TURN-Server hingegen befindet sich im öffentlichen Weitverkehrsnetz und ist von externen Hosts erreichbar. Der TURN-Server generiert nach eingehendem TURN-Request des Clients einen internen 5-Tupel (TURN-clientseitig) mit den Verbindungsinformationen des Clients und des gewünschten Kommunikationspartners. Für die Kommunikation mit externen Hosts verwendet der TURN-Server einen externen 5-Tupel, der die emulierte Netzwerkadresse des Clients und die Verbindungsinformationen des externen Hosts enthält. Im Anschluss

leitet der TURN-Server eingehende Pakete mit der emulierten Netzwerkadresse des Clients, die im externen 5-Tupel enthalten ist, unter Verwendung des internen 5-Tupels an den Client weiter und emuliert somit ein Restricted Cone NAT.

4.3.4 Hole Punching

Der Begriff des *Hole Punching* bezeichnet ein Verfahren zum Aufbau einer direkten Punkt zu Punkt Verbindung zwischen zwei Rechnern, wenn sich beide innerhalb verschiedener privater Netzwerke und somit in NAT-Umgebungen befinden.

Wie in den vorhergehenden Abschnitten beschrieben, existieren NAT-Traversal Verfahren, um hinter einem NAT befindlichen Clients eine direkte Kommunikation mit externen Hosts zu ermöglichen. Basierend auf dem zuvor vorgestellten STUN Ansatz, wird im folgenden ein Überblick das Hole Punching gegeben [27].

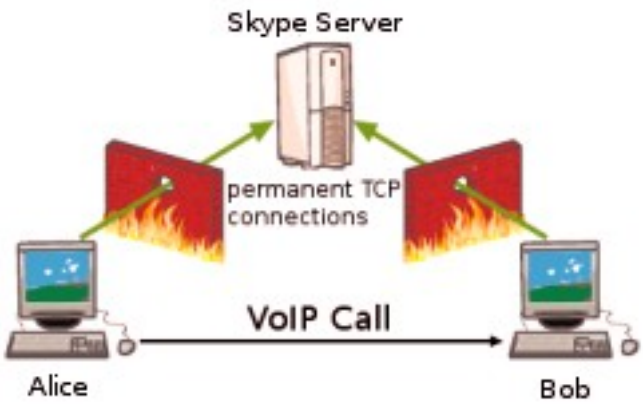

Abbildung 4.1: Anfrage von Allice an Bob über den Skype Server [27]

Beide Clients (Alice und Bob) schicken eine Anfrage an einen STUN Server (z.B. Skype Server in Abbildung 4.1) der die öffentlichen Netzwerkadressen der NATs und privaten Netzwerkadressen beider Clients ermittelt. Daraufhin schickt der Bob eine STUN Nachricht an Alice (siehe Abbildung 4.2) und öffnet somit einen Verbindungsendpunkt in seinem NAT für Alice. Die Nachricht von Bob wird vom NAT des Netzwerks von Alice blockiert.

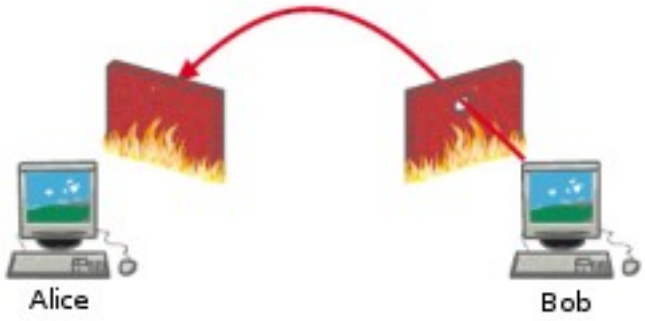

Abbildung 4.2: Bob schickt STUN Nachricht an Alice [27]

Alice schickt daraufhin eine STUN-Nachricht an Bob und öffnet somit einen Verbindungsendpunkt im NAT für Bob. Die Nachricht wird vom NAT des Netzwerks von Bob weitergeleitet (siehe Abbildung 4.3), da zuvor bereits ein Verbindungsendpunkt für Alice geöffnet wurde. Beide NATs besitzen nun offene Verbindungsendpunkte um eine Direktverbindung aufzubauen.

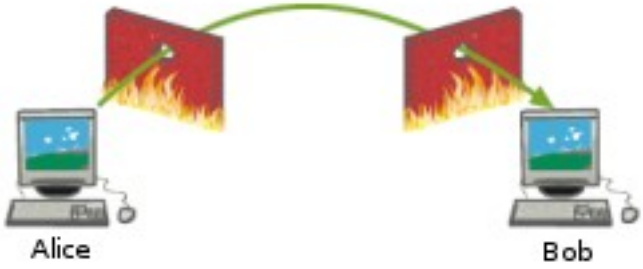

Abbildung 4.3: Aufbau der Direktverbindung zwischen Alice und Bob [27]

Hole-Punching funktioniert jedoch nicht mit beliebigen NAT-Typen, da besonders die symmetrischen NATs nicht traversiert werden können. Es muss in diesem Fällen auf Ausweichlösungen wie die Verbindung über einen Relay-Server mit z.B. TURN zurückgegriffen werden.

4.3.5 ICE

Das Interactive Connectivity Establishment (ICE) Protokoll ermöglicht den Aufbau einer Verbindung zwischen zwei Clients in beliebigen Szenarien. In diesem Zusammenhang kombiniert ICE die bekannten Verfahren STUN und TURN, um ein verbessertes NAT-Traversal zu ermöglichen und symmetrische NATs zu überwinden. Es wird im ICE angenommen, dass Rechner mehrere Schnittstellen und Verbindungsmöglichkeiten

besitzen, um eine Direktverbindung aufbauen zu können. Aus diesem Grund wird eine Kombination der Verfahren STUN und TURN verwendet, um Informationen über alle Verbindungsmöglichkeiten und Netzwerkadressen gewinnen zu können. Auf Grundlage dieser Informationen wird die beste Verbindungsvariante ausgewählt, eine Verbindung zwischen zwei Clients herzustellen. ICE versteht sich als universeller Ansatz, um der NAT Problematik zu begegnen.

ICE befindet sich momentan im Entwurfsprozess um als Internet Standard [28] aufgenommen zu werden. Es basiert auf der Grundmotivation der Realisierung von Multimedia-Verbindungen mit Hilfe eines universell einsetzbaren NAT-Traversal Verfahrens und wird bereits in kommerziellen Voice-Over-IP Anwendungen wie z.B Skype oder Google Talk eingesetzt [29].

4.4 Jingle

Ein Großteil der Protokoll-Erweiterungen von XMPP betreffen textuelle oder XML-basierte Kommunikation und sind für Media Kommunikation wie Sprache, Video oder Dateiübertragung nicht geeignet. 2005 entstand daher eine Sammlung von XMPP Erweiterungen unter dem Namen *Jingle* zur Konzeption eines *Multimedia Session Managements*, deren Verbindungsaufbau über XML-Nachrichten gesteuert und deren Datenübertragung über standardisierte Internet Protokolle wie dem *Real-Time Transport Protokoll (RTP)* [30] verlaufen sollte. Zur selben Zeit entwickelte Google mit Google Talk einen XMPP-basierten Dienst für Instant Messaging (IM) und voice over IP (VoIP). In einer Kooperation mit Google entwickelte das XMPP Entwickler Team eine XMPP Protokoll-Erweiterung für Multimedia Dienste, die sich momentan im XMPP Standardisierungsprozess für XMPP-Erweiterungen befinden. [29]

4.4.1 Jingle Einführung

Das Jingle Protokoll [31] ermöglicht den Auf- und Abbau, sowie die Verwaltung von Multimedia-Sessions (Jingle-Session) zwischen zwei XMPP Teilnehmern. Das Aushandeln und die Verwaltung selbst erfolgt über XMPP, die eigentliche Übertragung findet jedoch außerhalb von XMPP statt.

Die Spezifikation [29] definiert einen modularen Ansatz für den Aufbau und die Verwaltung einer Sitzung (engl. Session):

- Eine Sitzung hat ein oder mehrere Paare von ausgehandelten Inhalts-Typen des Übertragungsinhalts.

- Eine Inhaltsbeschreibung definiert den zu übertragenen Inhalt (z.B. Sprache [32] oder Video [33]).
- Ein Transport-Manager, welcher spezifiziert wie der Inhalt transportiert wird. (UDP [34] oder ICE [28])
- Ein Inhalts-Typ ist eine Kombination aus einer Inhaltsbeschreibung und einem oder mehreren Transportprotokollen.

Der modulare Aufbau von Jingle erlaubt die Verwendung von vielfältigen Multimedia-Session Typen für Audio-Video Verbindungen, Dateiübertragung und mehr. Die eigentliche Übertragung innerhalb einer Session ist weiter modular aufgebaut, so dass für jeden Multimedia-Session-Type eine entsprechende Übertragung verwendet werden kann. Für eine Jingle-basierte Dateiübertragung existieren bereits Entwürfe für einen Migrationsprozess des bestehenden XEP-0096 Protokolls in eine neue XMPP-Erweiterung XEP-0234 [35].

4.4.2 Transport Manager

Die für eine Dateiübertragung relevanten Jingle Transport Manager sind Jingle Raw UDP Transport [34] und Jingle ICE-UDP Transport [36].

XEP-0177 sendet die Media Daten über eine UDP Socket Verbindung, deren Verbindungsaufbau unter Verwendung eines STUN-Servers erfolgt. Der Einsatz eines STUN Servers ist jedoch nicht für alle Netztopologien zur Lösung des NAT-traversal Problems geeignet. Für den Aufbau einer Verbindung mit XEP-0176 wird der Interactive Connectivity Establishment (ICE) Ansatz verwendet, der eine Verbindung in allen Netztopologien sicherstellen soll. Es kann zudem mit XEP-0177 sowohl eine Übertragung über UDP als auch über TCP erfolgen.

Verbindungsaufbau mit Jingle

Der Verbindungsaufbau von Jingle erfolgt abhängig von der vorherrschenden Netzwerk- und NAT-Topologie mit dem Ziel ein universelles Verfahren einer Multimedia-Verbindung bereitzustellen.

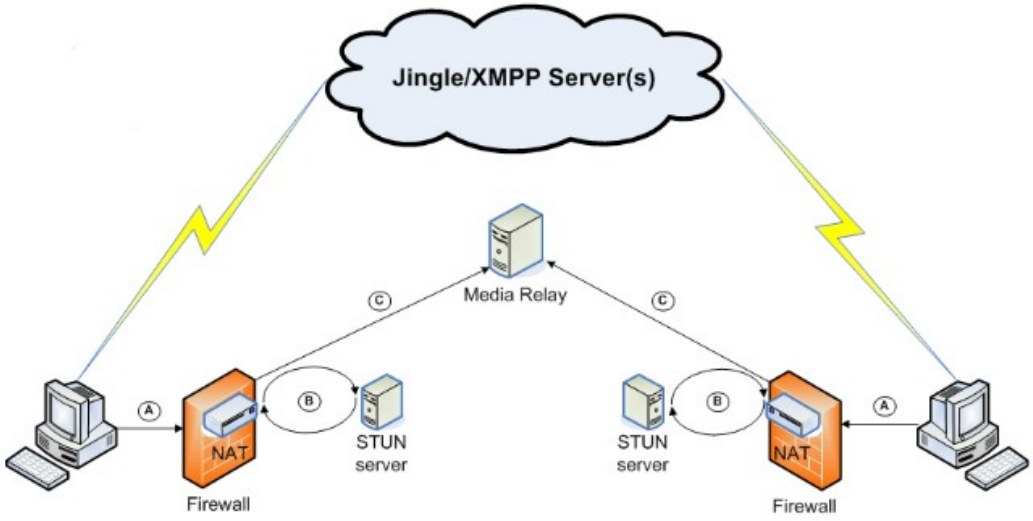

Abbildung 4.4: Jingle Verbindungs-Architektur [37]

Die Jingle-Architektur für den Verbindungsaufbau wird wie in Abbildung 4.4 dargestellt, in drei Stufen unterschieden:

a. **Lokale IP-Adressen (P2P):** In der ersten Stufe erfolgt der Verbindungsaufbau als Direktverbindung zwischen den beteiligten Rechner. Dieses Verfahren hat jedoch nur Erfolg, wenn sich beide Teilnehmer innerhalb desselben lokalen Netzwerks befinden. Es stellt den klassischen Ansatz einer Direktverbindung dar.

b. **Globale IP-Adressen (P2P mediated by NAT):** Sollte der Verbindungsaufbau in Stufe Eins fehlschlagen, wird das NAT-Traversal Verfahren STUN für einen Verbindungsaufbau verwendet, um eine Direktverbindung in NAT-Umgebungen zu ermöglichen.

c. **Media Relay (Verbindung über Proxy):** Sollte der Verbindungsaufbau in Stufe Zwei fehlschlagen, wird nach dem Vorbild des TURN Ansatzes eine Verbindung über einen Media-Proxy aufgebaut.

4.5 Jingle-Erweiterung der Smack API

Die bisherige Eclipse-Erweiterung Saros basiert auf der Smack API in der Version 2.2.1 und enthält noch keine Umsetzungen der Jingle-Erweiterungen, da diese Umsetzungen zum damaligen Zeitpunkt noch nicht zur Verfügung standen. In der aktuell verfügbaren Version 3.0.4 wurden die Jingle-Erweiterungen erstmalig in die Smack API integriert.

Die Umsetzung der Jingle-Erweiterungen in Smack implementieren die veröffentlichten XMPP-Jingle-Spezifikationen und stellen Schnittstellen zur Nutzung von Jingle-basierten Multimedia-Sessions und Übertragungskanälen zur Verfügung. Für die Implementierung der Transport-Manager-Protokolle XEP-00176 und XEP-00177 und die hierfür notwendigen NAT-Traversal Verfahren findet in Smack die Java-basierte Implementierung JSTUN [38] Verwendung.

4.5.1 Vorteile

Die XMPP-Jingle-Erweiterungen werden über einen standardisierten Prozess für die Protokoll-Definitionen durch die XMPP Standards Foundation gesteuert, so dass ein großes Potential für die Stabilität und die Weiterentwicklung angenommen werden kann. Zusätzlich zu einer Verwendung von Jingle zur Realisierung einer verbesserten Dateiübertragung bietet Jingle für Saros den Ausgangspunkt zur Integration einer modularen Multimedia-Schnittstelle, die für die Integration erweiterter Funktionalitäten wie Audio- und Video-Übertragung in Saros verwendet werden kann.

4.5.2 Nachteile

Die Jingle-Erweiterungen wurde erstmalig in der aktuellen Version 3.0.4 in der Smack API aufgenommen. Leider existiert noch keine Dokumentation über die Jingle-Implementierung in Smack. Die aktuelle Version 3.0.4 der Smack API beinhaltet bereits Umsetzungen vereinzelter Szenarien, die den aktuellen Funktionsumfang aufzeigen. Es sind noch nicht alle Erweiterungen in dieser Version enthalten. Zum jetzigen Zeitpunkt sind jedoch nur die Grundstruktur und die notwendigsten Protokolle für eine Multimedia-Kommunikation implementiert.

Eine direkte Umsetzung einer Jingle-basierten Dateiübertragung ist zum jetzigen Zeitpunkt noch nicht realisiert, jedoch kann eine Dateiübertragung basierend auf einer Multimedia-Session mit Hilfe der Jingle-Erweiterungen umgesetzt werden.

4.6 Fazit

DPP I und DPP II verwenden die Smack API in der Version 2.2.1. Für den Ausbau der Kommunikation zum Lösen der Netzwerkproblematik in Weitverkehrsnetzen ist die Verwendung der API in der Version 3.0.4 sinnvoll, da erst ab dieser Version die Jingle Erweiterung im vollen Umfang in die API integriert sind und die notwendigen Erweiterungen für Multi-Media Kommunikation und NAT-traversal Protokoll-Umsetzungen beinhalten.

Aufbauend auf der bisherigen Entwicklung von Saros können nach Umstellung auf die Version 3.0.4 der Smack API die bisherigen Funktionen voll erhalten bleiben und weiterentwickelt werden.

Besonders die Jingle Erweiterungen in der weiterentwickelten Smack API bieten für den Ausbau von Saros im Hinblick auf das NAT-Traveral den geeigneten Ansatz für die Realisierung der notwendigen Dateiübertragung und bieten darüber hinaus die Möglichkeit für einen späteren Ausbau mit erweiterter Multimedia Funktionalität. Der Jingle-Ansatz und die Java-basierte Implementierung mit Smack bieten einen universellen Lösungsansatz zur Realisierung von Multimedia Kommunikation in Weitverkehrsnetzen. Durch die ständige Weiterentwicklung und Standardisierung von Jingle durch die XSF ist von einer langfristigen Umsetzung und Unterstützung dieser Protokolle auszugehen.

5. Implementierung eines erweiterten Replikations-Ansatzes

Im vorhergehenden Kapitel wurden Ansätze zur Realisierung von Multimedia-Kommunikation durch den Einsatz von NAT-Traversal Verfahren betrachtet und die Implementierungen dieser Verfahren vorgestellt. Auf Basis dieser Untersuchung wird im Folgenden der bestehende Replikations-Ansatz erweitert, um die erarbeiteten Anforderungen aus Kapitel 3 erfüllen zu können.

5.1 Entwurfsentscheidungen

Die in Kapitel 3 erarbeiteten Einschränkungen des bisherigen Replikations-Ansatzes offenbarten Verbesserungspotential sowohl in Hinblick auf technische als auch konzeptionelle Ansätze, um die in Kapitel 2 aufgestellten Anforderungen erfüllen zu können. Im folgenden werden die Konzeptionen für einen erweiterten technischen Ansatz der Dateiübertragung vorgestellt und in den überarbeiteten Projekt-Synchronisations-Prozess eingebunden.

5.1.1 Dateiübertragung mit Jingle

Die Probleme in Bezug auf eine Direktverbindung in Weitverkehrsnetzen wurde ausführlich im vorhergehen Kapitel dargelegt. Als Lösungsansatz bietet die Smack API ab Version 3.0.4 erweiterte Jingle-Bibliotheken, die NAT-Traversal-Konzepte beinhalten und somit eine Direktverbindung zwischen zwei Teilnehmern ermöglichen. Vor Beginn einer Übertragung der Daten mit Jingle wird eine initiale Jingle-Session aufgebaut, um eine Direktverbindung zu etablieren. Im Anschluss können über die bestehende Verbindung Daten zwischen den beiden Teilnehmern der Session ausgetauscht werden.

Innerhalb eines Projekt-Synchronistation-Prozesses ist ein Austausch von sehr vielen und ggf. großen Datenmengen von mehreren Megabytes zu erwarten. Innerhalb einer etablierten Jingle-Session können verschiedene Übertragungskanäle einen Datenaustausch zwischen den Teilnehmern ermöglichen. Ein Verbindungsaufbau für jede zu übertragende Datei ist daher nicht notwendig, sondern alle Projekt-Dateien können über die bestehende Verbindung übertragen werden. Nach Abschluss der Projekt-Synchronisation wird die Jingle-Session durch den Projekt-Host beendet.

Zum jetzigen Zeitpunkt existieren keine Erweiterungen zur Dateiübertragung innerhalb der Jingle-Erweiterungen in Smack, zudem ist der Standardisierungsprozess für eine Jingle-Dateiübertragung durch die XSF noch in einem sehr frühen Stadium der Entwicklung. Aus

diesen Gründen ist eine eigene Dateiübertragung aufbauend auf dem derzeitigen Jingle-Erweiterungen für Multimedia-Kommunikation in Jingle notwendig. Nach dem Zustandekommen einer Jingle Session mit ICE kann, basierend auf den standardisierten Internetprotokollen UDP und TCP, eine Dateiübertragung erfolgen. Der modulare Aufbau von Jingle ermöglicht die spätere Adaption der Dateiübertragung an eine mögliche Weiterentwicklung der Jingle-Implementierung in Smack. Zur Realisierung der eigenen Dateiübertragungs-Erweiterung sind keine Eingriffe in die Smack-API notwendig, da das API alle notwendigen Schnittstellen zur Realisierung eigener Erweiterungen bereitstellt.

5.1.2 Ausweich-Protokoll XEP-0096

Um eine Dateiübertragung immer zu ermöglichen, muss zusätzlich zur Jingle-basierten Dateiübertragung ein Ausweich-Protokoll zur Verfügung stehen, welches im Fehlerfall zu Anwendung kommen kann. Sollte eine Jingle-basierte Übertragung nicht zustande kommen, wird eine Dateiübertragung mit dem XEP-0096 Protokoll angestrebt. Unter Berücksichtigung der in Kapitel 3.2.1 analysierten Nachteile dieses Protokolls im Kontext eines Projekt-Synchronisations-Prozesses, muss die Verwendung dieses Protokolls innerhalb von Saros angepasst werden, da die Verwendung des Protokolls zum Versand von vielen Dateien, aufgrund des Verbindungsauf- und -abbaus für jede zu sendende Datei, sehr ineffizient ist.

Als alternatives Lösungsszenario wird nach einem fehlerhaften Verbindungsaufbau einer Jingle-Dateiübertragung ein Archiv aller benötigten Projekt-Dateien für den Abschluss der Projekt-Synchronisation erstellt. Diese einzelne Archiv-Datei wird im Anschluss über die XEP-0096-basierte Dateiübertragung versandt.

5.1.3 Anpassung des Projekt-Synchronisations-Prozesses

Der bisherige Ansatz des Projekt-Synchronisation sah nur ein geringes manuelles Eingreifen von Seiten des Anwenders auf Empfänger-Seite vor. Der Anwender hatte entweder die Möglichkeit, ein neues Projekt zu erstellen oder aufbauend auf einer automatischen Workspace-Analyse ein vorgegebenes Projekt zu verwenden.

Da in der veränderten Situation von beliebig vielen Anwendern innerhalb einer Projekt-Sitzung nicht mehr von den ursprünglichen Annahmen ausgegangen werden kann, muss der Prozess der Projekt-Synchronisation dahingehend anpasst werden, dass der Anwender situationsbedingt die bestmögliche Projekt-Synchronisation durchführen kann. Die Projekt-Synchronisation ist sowohl von den lokal verfügbaren Projekten, als auch von der zur Verfügung stehenden Übertragungsart abhängig.

5.2 Implementierung

Basierend auf den Entwurfsentscheidung und der Analyse der Projekt-Synchronisation aus Abschnitt 3.2.3 wird im folgenden die Umsetzung der Projekt-Synchronisation unter Verwendung einer Jingle-basierten Dateiübertragung beschrieben.

5.2.1 Projekt-Synchronisation

Die Einschränkungen der momentanen Projekt-Synchronisation wurden ausführlich im *Kapitel 3.3* dargelegt. Aufbauend auf den Analyse-Ergebnissen werden die Funktionalitäten der bestehenden Dateiübertragung adaptiert und an die bestehenden Anforderungen angepasst.

Verbindungsinformationen

Nach Bestätigung der Projekt-Einladung durch den Eingeladenden[2] im zweiten Schritt, überträgt der Einladende eine Projekt-Liste des veröffentlichten Projekts an den Eingeladenen. Diese Übertragung erfolgt als Dateiübertragung der zuvor erstellten Projekt-Liste. Anhand der Informationen aus der empfangenen Projekt-Liste wird der Projekt-Auswahl-Dialog beim Eingeladenen wie in Abbildung 5.1 dargestellt, initiiert.

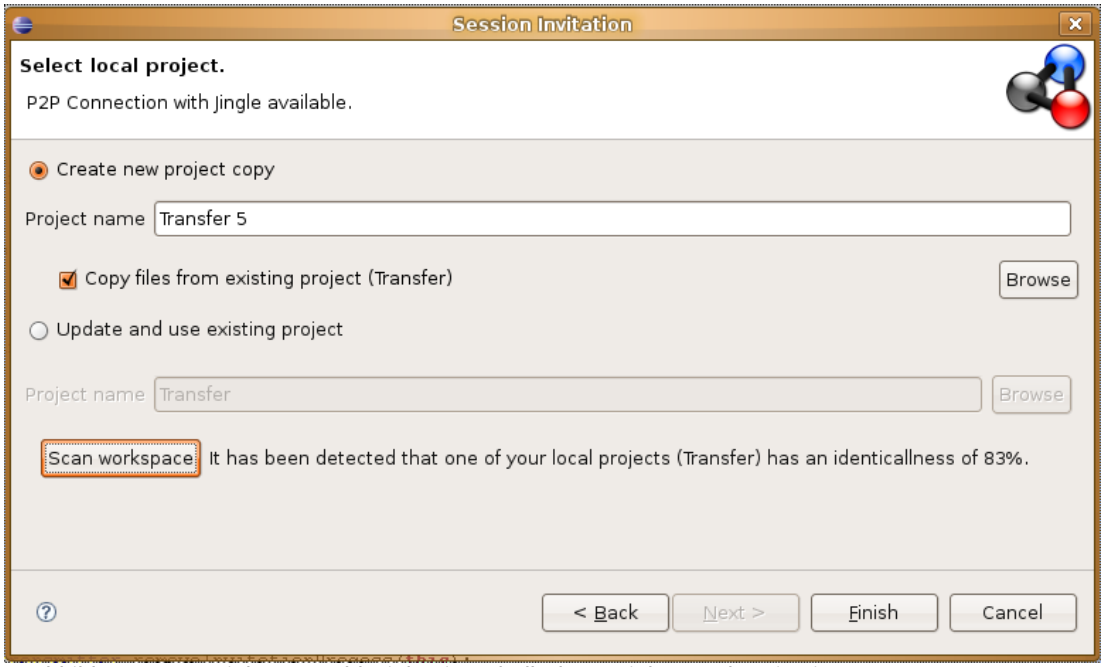

Abbildung 5.1: Projekt-Auswahl Dialog innerhalb der Projekt-Synchronisation

[2]Die folgenden Beschreibungen der Schritte betreffen das im *Kapitel 3.3.1 Grundkonzept der Projekt-Synchronisation* beschriebenen schrittweise Vorgehen.

Konnte erfolgreich eine Jingle-Sitzung etabliert werden, wurde die Dateiliste über die nun bestehende Direktverbindung zwischen den Teilnehmern übertragen. Die Sitzung kann nun für die Übertragung der Dateiliste des eingeladenen Teilnehmers, sowie der Projekt-Dateien verwendet werden. Der Anwender wird über das verwendete Verfahren der Dateiübertragung informiert.

Projekt-Auswahl

Der Eingeladene kann nun aufgrund der zur Verfügung stehenden Informationen über die bestehenden Verbindungsart seine Strategie für die Projekt-Synchronisation der Situation entsprechend auswählen. Sollte keine Direktverbindung zur Verfügung stehen, erfolgt nach einem entsprechenden Hinweis die Aufforderung ein bestehendes Projekt anzugeben, um die Synchronisation zu verkürzen.

Ihm obliegt, wie bereits in DPPII, die Wahl der Fortsetzung eines bestehenden Projekts, jedoch kann er dieses explizit auswählen und ist nicht auf eine einzige Vorgabe festgelegt. Unterstützend kann explizit ein Suchprozess des Eclipse-Workspace erfolgen, in dessen Verlauf jedes Projekt aus dem Workspace mit dem veröffentlichten Projekt verglichen wird. Im Anschluss wird das Projekt, welches nach Analyse der Projektstruktur mit dem Projekt aus der Dateiliste am ehesten übereinstimmt, als Basisprojekt vorgeschlagen. Dieser Suchprozess kann je nach Größe und Anzahl der in Workspace befindlichen Projekte einen längeren Zeitraum in Anspruch nehmen und muss daher explizit vom Anwender gestartet werden.

5.2.2 Jingle-basierte Dateiübertragung

Die Umsetzung einer Jingle-basierten Dateiübertragung bedingt den notwendigen Wechsel der verwendeten Version der Smack Bibliotheken von Version 2.2.1 auf die Version 3.0.4, um die integrierten Jingle-Bibliotheken nutzen zu können. Ein Versionswechsel der Hauptbibliotheken der Netzwerkschicht von Saros birgt die Notwendigkeit der Anpassung der bestehenden Funktionalität an veränderte Schnittstellen oder tiefgreifende Änderungen im Vergleich zur Vorversion. Diese Migration stellt an dieser Stelle jedoch einen vertretbaren Aufwand dar, da nach Abschluss der Migration die bestehende Funktionalität im vollem Umfang zur Verfügung steht.

Smack API 3.0.4

Nach erfolgreicher Migration auf die Smack API in der Version 3.0.4 konnte auf Grundlage der zur Verfügung stehenden Jingle-Bibliotheken eine Jingle-basierte Dateiübertragung

umgesetzt werden. Eine Dateiübertragung ist bisher nicht in der Jingle-Erweiterung implementiert, jedoch bietet die API in der nun vorliegenden Version Schnittstellen für die Realisierung eigener Multimedia-Erweiterung an. Eine direkte Manipulation an der Jingle API ist nicht notwendig, da das modulare Framework von Jingle alle Schnittstellen für den Aufbau einer Jingle-Sitzung bereitstellt.

Hierzu wurde eine eigene Media-Manager Implementierung mit einer entsprechenden Jingle-Session Erweiterung realisiert. Innerhalb der Sitzung kommen eigene Transmitter und Receiver-Implementierung mit auf TCP basierten Java Sockets zum Einsatz, die ein eigenes Protokoll für die Dateiübertragung realisieren.

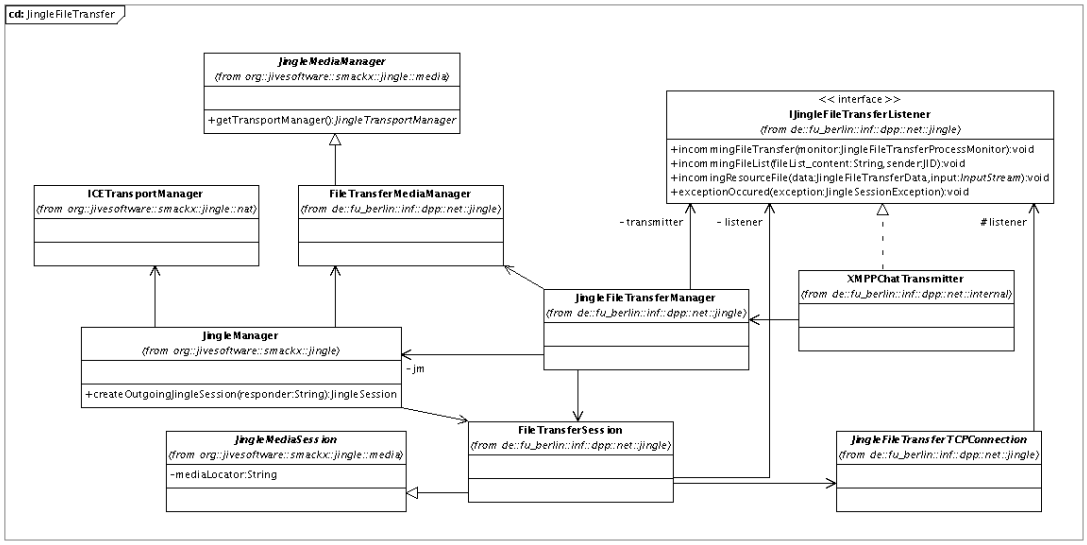

Abbildung 5.2: Klassendiagramm des JingleManagers und der beteiligten Klassen

Jingle-Manager

Jingle ist ein Session basiertes Protokoll, dessen Umsetzung in Smack eine out-of-band Verbindung ermöglicht, um Multimedia-Kommunikation (z.B. Voice-over-IP, Video-Übertragung oder Dateiübertragung) zu realisieren. Eine Jingle-Session beinhaltet zum Einen die Übertragungsmethode (bzw. den entsprechenden *Transport-Manager*), welche die notwendigen IP-Adressen mit den zugehörigen Ports (*transport candidate*) ermitteln. Mit Hilfe dieser Netzwerkadressen können die Pakete an die entsprechenden Teilnehmer adressiert werden. Weiterhin wird für eine Jingle-Session die Information über den innerhalb der Sitzung kommunizierten Inhalt bzw. Codec (*payload type*) benötigt. Bei einer etablierten Jingle-Session können Multimedia-Daten des definierten Codec-Typs zwischen zwei Teilnehmer aufgrund ihrer ermittelten Netzwerkadressen außerhalb von XMPP übertragen werden.

Die Klasse *JingleManager* der Smack Jingle API (siehe Abbildung 5.2) verwaltet den Aufbau und Abbau einer Jingle-Session, die durch die Klasse *JingleSession* umgesetzt sind und stellt die benötigten Schnittstellen für die Verwaltung der Session bereit.

`JingleFileTransferManager`: Innerhalb von Saros erfolgt der Auf- und Abbau einer Jingle-Session, sowie die Steuerung der Dateiübertragung über die Schnittstellen der Klasse `JingleFileTransferManager`, die von Fassaden-Klasse[3] der Netzwerkkommunikation angesprochen wird. Der `JingleFileTransferManager` integriert den `JingleManager` und steuert die Lebenszyklen der Jingle-Sessions.

`IJingleFileTransferListener`: Diese Schnittstelle ermöglicht den Zugriff auf den eintreffenden Media-Daten der Dateiübertragung. Die Fassaden-Klasse `XMPPChatTransmitter` wurde zum Empfang der eintreffenden Media-Daten um diese Schnittstelle erweitert.

Transport-Manager

Als Transport-Manager für den Aufbau einer Jingle-Session und die Bestimmung der Netzwerkdadressen wird die Implementierung des XEP-0176: Jingle ICE-UDP Protokolls [36] verwendet. Der ICE-Ansatz bietet wie in Kapitel 4 vorgestellt, den derzeitig erfolgversprechenden Ansatz für den Aufbau einer Direktverbindung unter Verwendung von NAT-Traversal Verfahren.

Jingle-Media-Manager

Mit Hilfe der Klasse `JingleMediaManager` bietet die Smack API eine abstrakte Klasse zur Realisierung eigener Multimedia-Erweiterungen an. Die Kontrolle der entsprechenden Codec-Typen bzw. unterschiedlichen Multimedia-Typen wird innerhalb des `JingleMediaManager` gesteuert und somit vor der Klasse `JingleManager` verborgen. Durch diesen modularen Ansatz ist es möglich, neue Multimedia-Erweiterungen in eigenen Jingle-Media-Manager zu implementieren und in den Jingle-Manager einzubinden. Innerhalb des Media-Managers wird der Codec-Typ festgelegt und die Verwaltung der entsprechenden Jingle-Media-Sessions gesteuert.

`FileTransferMediaManager`: Diese Klasse implementiert die `JingleMediaManager`-Schnittstelle zur Realisierung eines Multimedia-Codec für eine Dateiübertragung mit

[3] Die Klasse `XMPPChatTransmitter` implementiert die Fassaden-Schnittstelle `ITransmitter` für die gesamte Netzwerkkommunikation von Saros.

Jingle. Für die Dateiübertragung wurde innerhalb der Klasse der Codec-Typ *filetransfer* definiert.

Jingle-Media-Session

Nach erfolgreich aufgebauter Jingle-Session stehen für eine Mulitmedia-Verbindung die Netzwerkadressen der beteiligten Partner und der entsprechende Codec-Typ zur Verfügung. Die Smack Jingle API beinhaltet keine Methoden zum Senden und Empfangen von Media-Daten, sondern stellt durch die Klasse `JingleMediaSession` die Schnittstelle zur Verwendung der Jingle API bereit. Durch eine Implementierung der `JingleMediaSession`-Schnittstelle können eigene Umsetzungen zum Senden und Empfangen der Media-Daten erfolgen.

`FileTranferSession`: Diese Klasse implementiert die `JingleMediaSession` für den im `FileTranferMediaManager` definierte Codec-Typ *filetransfer*. Aufbauend auf den ermittelten Netzwerkadressen erfolgt innerhalb dieser Klasse der Aufbau des eigentlichen Übertragungskanals für die Dateiübertragung.

TCP-basierte Dateiübertragung

Die eigentliche Dateiübertragung innerhalb einer `JingleMediaSession` erfolgt über die Umsetzung einer TCP Übertragung via Java-Socket. Zur Realisierung einer Dateiübertragung wird ein eigenes Protokoll definiert, um zusätzlich zum Dateiinhalt Meta-Informationen zu übertragen.

Diese Meta-Informationen sind notwendig, um unterschiedliche Dateitypen (z.B. Dateiliste für Projekt-Initiierung oder Projekt-Ressource-Datei) zu unterscheiden und durch Pfad- und Projekt-Informationen eindeutig zuordnen zu können. Des weiteren ist es hierdurch möglich nicht nur einzelne Dateien zu senden, sondern die gesamte Übertragung aller Projektdateien innerhalb einer Jingle-Session zu realisieren.

`IJingleFileTransferConnection`: Dieses Schnittstelle repräsentiert den Übertragungskanal der innerhalb der *Jingle-Media-Session* aufgebaut wird.

`JingleFileTransferTCPConnection`: Diese Klasse implementiert die IjingleFileTransferConnection-Schnittstelle und baut eine TCP-Verbindung zum Senden der Daten innhalb der Jingle-Sitzung auf.

Senden und Empfangen von Dateien

Im Anschluss an die Vorstellung der wichtigsten Klassen für die Dateiübertragung wird der technische Ablauf des im Abschnitt 5.2 vorgestellten Projekt-Synchronisations-Prozesses erläutert. Der Einladungsprozess durchläuft hierbei mehrere Stufen:

1. Innerhalb des Einladungsprozesses sendet der Einladende im dritten Schritt des im Abschnitt 3.3.1 beschriebenen Einladungsprozesses die Dateiliste an den Eingeladenen. Die Fassaden-Klasse `ITransmitter` übergibt die zu sendende Dateiliste an den `JingleFileTransferManager`.

2. Der `JingleFileTransferManager` erzeugt eine `JingleSession` und startet den Jingle-basierten Verbindungsaufbau zum Eingeladenen. Parallel hierzu übergibt der `JingleFileTransferManager` die zu sendenden Transfer-Daten an den `FileTransferMediaManager`, der das Senden und Empfangen von Daten innerhalb der `JingleSession` verwaltet.

3. Auf Seiten des Eingeladenen empfängt der `JingleFileTransferManager` die Anfrage einer eingehenden Jingle-Sitzungsanfrage und startet daraufhin eine eingehende `JingleSession`.

4. Nach erfolgreichem Aufbau der `JingleSession` wird der Transport-Kanal auf beiden Seiten initiiert. Dies erfolgt durch den Aufbau einer TCP-Verbindung zwischen den Teilnehmern durch die Initialisierung der TCP-Verbindung über die Klasse `JingleFileTransferTCPConnection`.

5. Nach erfolgreichem Aufbau des Transport-Kanals werden die dem `FileTransferMediaManager` übergebenen Daten über die bestehende TCP-Verbindung übertragen.

6. Nach Empfang der Daten beim Eingeladenen werden die diese an den Fassaden-Klasse `ITransmitter` weitergeleitet. Die `JingleSession` bleibt nach dem Empfang der Daten bestehen.

7. Die Dateiliste des Eingeladenen wird - wie unter Schritt 4 im Abschnitt 3.3.1 beschrieben - an den Einladenden gesendet. Hierzu übergibt der `ITransmitter` die Daten an den `JingleFileTransferManager`, der diese an den

`FileTransferMediaManager` der offenen `JingleSession` übergibt. Dieser leitet die Daten zum Senden an `JingleFileTransferTCPConnection` weiter.

8. Wie im Schritt 5 aus Abschnitt 3.3.1 beschrieben, werden nun alle noch benötigten Dateien einzeln vom Einladenden über die bestehende `JingleSession` an den Eingeladenen gesendet.

9. Nach Beendigung der Übertragung aller Dateien, beendet der Einladende die `JingleSession`.

Ausweichprotokoll bei fehlerhaften Jingle-Verbindungsaufbau

1. Kann die `JingleSession` unter Punkt 3 des vorhergehenden Abschnitt nicht aufgebaut werden, wird eine `JingleSessionException` an den `ITransmitter` weitergeleitet. Dieser benachrichtigt den `IOutgoingInvitationProcess` des Einladenden.

2. Die Dateiliste des Einladenden wird nun über das XEP-0096-basierte Ausweichprotokoll an den Eingeladenen übertragen.

3. Der Eingeladene sendet seine lokale Dateiliste in Folge ebenfalls über die XMPP-basierte Übertragung an den Einladenden zurück.

4. Der Versand aller Projekt-Dateien durch den Einladenden muss nun ebenfalls über das XMPP-basierte Ausweichprotokoll erfolgen. Alle zu sendenden Projekt-Dateien werden im `IOutgoingInvitationProcess` in einem Transfer-Archiv zusammengefasst und an den `ITransmitter` weitergeleitet. Der `ITransmitter` sendet das Archiv dann an den Eingeladenen.

5.2.3 XMPP-basierte Dateiübertragung

Die bestehende XMPP-basierte Dateiübertragung der Smack API beinhaltet die im Abschnitt 3.2.3 aufgeführten Einschränkungen, welche die Funktionen der Dateiübertragung im Projekt-Synchronistationsprozess maßgeblich beeinträchtigen. Die daher notwendigen konzeptionellen sowie technischen Anpassungen werden im folgenden Abschnitt beschrieben.

Anpassungen innerhalb des Einladungsprozesses

Um die XMPP-basierte Dateiübertragung mit Hilfe des XEP-0096 Protokolls als Ausweichprotokoll für die Dateiübertragung mit Jingle zu verbessern, muss das Senden der Dateidateien konzeptionell überarbeitet werden. Bisher wurden die Dateien zum Senden dem `ITransmitter` einzeln übergeben. Dieses Vorgehen ist bei einer Jingle-basierten Dateiübertragung weiterhin sinnvoll, da die Dateien innerhalb einer offenen Jingle-Sitzung übertragen werden können.

Dem Senden einer Datei mit dem XMPP-Protokoll geht jedoch immer ein Verbindungsaufbau voraus, der den gesamten Übertragungsprozess aller Projekt-Dateien stark verlangsamt. Wie bereits im vorhergehenden Abschnitt aufgeführt, wurde innerhalb des Einladungsprozesses das Senden der Dateien dahingehend angepasst, dass bei einer XMPP-basierten Übertragung alle zu übertragenden Projektdateien in ein Archiv verpackt werden. Im Anschluss daran wird das Projekt-Archiv übertragen, wodurch das Aushandeln der Verbindung nur ein einziges Mal durchgeführt werden muss.

Anpassungen der Smack API

Desweiteren waren technische Anpassungen der Smack API notwendig, um eine effizientere Übertragung von größeren Dateien zu ermöglichen. In der vorhergehenden wie auch aktuellen Version sind wichtige Parameter des XEP-0096 Protokolls innerhalb der Smack API Implementierung nicht einstellbar (siehe hierzu Abschnitt 3.2.3).

Um zum Einen die Geschwindigkeit einer IBB-Übertragung durch wählbare Blockgrößen zu erhöhen und zum Anderen eine Socks5-Verbindung durch Konfiguration des Proxy-Ports zu ermöglichen, wurde die Klasse `FileTransferManager` der Smack API Erweiterung für die Dateiübertragung um Einstellmöglichkeiten für diese Parameter erweitert. Zusätzlich kann der Aufbau einer Socks5-Verbindung durch einen weiteren Parameter unterbunden werden, so dass nur eine IBB-Verbindung ermöglicht wird. Da die Socks5-Unterstützung vom jeweiligen Jabber-Server abhängig ist, können etwaige Probleme beim Verbindungsaufbau einer Socks5-Verbindung durch diesen Parameter unterbunden werden.

5.3 Test-Szenarien

Um die Umsetzung des Replikations-Ansatzes zu testen, wurden für folgenden Test-Szenarien Verbindungstests zwischen zwei Rechnern in beide Richtungen durchgeführt und jeweils 10-mal wiederholt. Vereinzelt konnte im Vorfeld ein Verbindungsaufbau auf Grund zu

restriktiver Einstellungen der Betriebssystem-Firewall nicht erfolgen, obwohl eine Direktverbindung hätte aufgebaut werden können. Die verwendeten Testrechner wurden dahingehend konfiguriert, das ein möglicher Verbindungsaufbau nicht von der Betriebssystem-Firewall blockiert werden kann. Die Bezeichnungen *WLAN* und *LAN* beziehen sich auf die Netzwerktopologien des Informatik-Instituts der Freien Universität Berlin.

Test-Szenario 1: WLAN – zu – LAN

Alle Projektdateien konnten über eine aufgebaute Direktverbindung im Zuge eines simulierten Einladungsprozesses kopiert werden.

Test-Szenario 2: LAN – zu – LAN

Alle Projektdateien konnten über eine aufgebaute Direktverbindung im Zuge eines simulierten Einladungsprozesses kopiert werden.

Test-Szenario 3: DSL – zu – DSL

Eine Direktverbindung zwischen den Teilnehmer konnte nicht aufgebaut werden. Während des Einladungsprozesses wurden die Projektdateien mittels des Ausweichprotokolls als Archiv über die IBB-Verbindung übertragen.

Test-Szenario 4: DSL – zu – LAN

Eine Direktverbindung zwischen den Teilnehmer konnte nicht aufgebaut werden. Während des Einladungsprozesses wurden die Projektdateien mittels des Ausweichprotokolls als Archiv über die IBB-Verbindung übertragen.

Test-Szenario 5: DSL (VPN-Verbindung) – zu – LAN

Alle Projektdateien konnten über eine aufgebaute Direktverbindung im Zuge eines simulierten Einladungsprozesses kopiert werden. Der über eine private DSL-Verbindung mit dem Weitverkehrsnetz verbundenen Rechner war über eine zusätzliche VPN-Verbindung mit dem Uni-Netzwerk verbunden.

5.4 Fazit

Mit der Umsetzung des Jingle-basierten Verfahres konnte der Aufbau einer Direktverbindung in verschiedenen Test-Szenarien sowohl zwischen verschieden Netzwerkbereichen des Informatik-Instituts der FU-Berlin, als auch zwischen institutinternen und externen Rechner

(mit bestehender VPN-Verbindung), erfolgreich aufgebaut werden. In den Test-Szenarien, in denen keine Direktverbindung aufgebaut werden konnte, erfolgte der Datentransfer über das Ausweichprotokoll. In allen Fällen konnte der Einladungsprozess erfolgreich abgeschlossen werden.

Eine Weiterentwicklung sowohl der Jingle-Erweiterungen der Smack-API, als auch der Jingle-Integration durch das XMPP Entwickler Team ist sehr wahrscheinlich, da dieser universelle Ansatz, sowohl im wissenschaftlichen, als auch in wirtschaftlichen Umfeld eine starke Akzeptanz gefunden hat.

Der Replikations-Ansatz konnte zum einen durch die Integration der Direktverbindung, als auch durch die Anpassung der XMPP-basierten Übertragung entscheidend verbessert werden.

6. Mehr-Schreiber-Funktionalität

Rechnergestützte Gruppenarbeit (Computer Supported Cooperative Work) oder Echtzeit-Gruppeneditoren ermöglichen einer Gruppe von Teilnehmern die Bearbeitung eines gemeinsamen Dokuments zur selben Zeit unter besonderer Berücksichtigung der räumlichen Trennung der Teilnehmer bei gleichzeitiger Verbindung über eine Breitbandkommunikation. Für die Umsetzung der Anforderung „Parallelität auf Schreibebene“ aus Abschnitt 2.4.2 werden Anforderungen an Echtzeit-Gruppeneditoren aufgestellt und im folgenden Verfahren zur Nebenläufigskontrolle vorgestellt. Abschließend werden aktuelle Umsetzungen von Echtzeit-Gruppeneditoren betrachtet.

6.1 Anforderungen an Echtzeit-Gruppeneditoren

Eine große Herausforderung in diesem Zusammenhang ist die Sicherung der Konsistenz der gemeinsamen Dokumente unter den Voraussetzungen kurzer Antwortzeiten und der Unterstützung blockierungsfreier und gleichzeitiger Bearbeitung der Dokumente in einem verteilten Umfeld.

Für die Festlegung der Anforderung für die Eclipse-Erweiterung Saros in Hinblick auf die Mehr-Schreiber-Funktionalität können die Anforderungen von Echtzeit-Gruppeneditoren herangezogen werden, da es sich bei verteilter Paar-Programmierung um eine Spezialisierung des Echtzeit-Gruppeneditoren-Ansatzes handelt.

Folgende Kriterien ([39],[40]) müssen von einem Echtzeit-Gruppeneditor erfüllt werden:

1. **Echtzeit:** Lokale Operationen müssen beinahe verzögerungsfrei, fast so schnell wie bei einem Ein-Benutzer-Editor, ausgeführt werden können. Die Latenzzeit für die Ausführung entfernter Operationen sollte klein sein (idealer Weise so klein wie die Latenzzeit der externen Übertragung).

2. **Verteilt:** Die zusammenarbeitenden Benutzer arbeiten auf verschiedenen Systemen und sind über unterschiedliche Netzwerke mit einer nichtdeterministischen Latenzzeit verbunden.

3. **Uneingeschränkt:** Mehrere Benutzen können konkurrierend und blockierungsfrei auf alle Dokumentenabschnitte zu jeder Zeit zugreifen und diese bearbeiten.

Um Konflikte während der gleichzeitigen Bearbeitung von Dokumenten durch mehrere Teilnehmer aufzulösen, muss eine Nebenläufigkeitskontrolle oder Konsistenzsicherung verwendet werden, die den Teilnehmer die Bearbeitung ermöglicht.

6.2 Nebenläufigkeitskontrollverfahren

Nach P. Bernstein [41] wird unter einer Nebenläufigkeitskontrolle die Koordination der Aktionen verschiedener Prozesse verstanden, die gleichzeitig auf gemeinsame Daten zugreifen und sich damit potentiell behindern, sowie die Wahrung der Konsistenz der gemeinsam genutzten Dokumente.

Nebenläufigkeitskontrollverfahren wurden ursprünglich für Datenbank-Anwendungen entwickelt und auf das Umfeld der Echtzeit-Gruppenanwendungen adaptiert. In früheren Ansätzen wurden Konflikte durch den Einsatz von Lock- oder Transaktionsverfahren aufgelöst.

Für die Umsetzung einer Nebenläufigkeitskontrolle gibt es prinzipiell zwei Verfahrensarten: Verfahren zur Konfliktvorbeugung und Verfahren zur Konfliktauflösung.

6.2.1 Sperrverfahren

Turn-Taking

Der „Turn-Taking“ Ansatz [40] ist ein sehr einfacher Ansatz, um die Konsistenz der Dokumente zu gewährleisten. Nur eine Person kann zur selben Zeit an dem Dokument arbeiten. Der Zugriff auf das Dokument erfolgt entweder über eine software-gesteuerte Zuweisung des Schreibrechts oder über externe soziale Protokolle. (z.B. Ablaufbeschreibungen innerhalb einer Organisation). Es gibt somit immer nur eine gültige Dokumentenversion, so dass keine Versionierung oder Synchronisation von Dokumenten stattfinden muss.

Dieser Ansatz schränkt die gleichzeitige Bearbeitung auf eine einzelne Person ein und kann somit nicht die Minimalanforderung der gleichzeitigen Bearbeitung eines Dokuments erfüllen. Es handelt sich hierbei um eine spezielle Form eines pessimistischen Sperrverfahrens.

Pessimistische Sperrverfahren

Pessimistische Sperrverfahren sind Verfahren zur Konfliktvorbeugung und zählen im Umfeld von Datenbankanwendungen zu den verbreitetsten Sperrverfahren.

Bei Sperrverfahren wird ein Bereich (z.B. Wort, Zeile, Sektion, Dokument) gesperrt, bevor er bearbeitet werden kann, so dass nur ein Benutzer zur selben Zeit auf diesen Bereich zugreifen kann. Das Sperren dieser Bereiche verhindert das Auftreten von Konflikten während der Bearbeitung des Dokuments durch mehrere Benutzer.

Die einfachste Umsetzung eines pessimistischen Sperrverfahrens ist das exklusive Sperren (exclusive locking). Ohne eine verfeinerte Granularität des Sperrbereichs wird das gesamte Dokument gesperrt, was dem oben beschriebenen Turn-Taking Ansatz entspricht. Bei einer kleineren Unterteilung des Sperrbereichs in einzelne Dokument-Kapitel, Absätze und oder Teilbereiche können mehrere Benutzer gleichzeitig auf dem selben Dokument arbeiten, jedoch nur in unterschiedlichen Bereichen des Dokuments.

Pessimistische Verfahren können weiterhin in zentrale (z.B. Token-Verfahren) und dezentrale Kontrolle (z.B. Votierungsverfahren, Floor-Passing-Verfahren) für die Vergabe der Schreibrechte und das Setzen der Sperren unterschieden werden.

Pessimistische Verfahren eignen sich nach Borghoff und Schlichter [42] besonders dann, wenn große Datenblöcke (mehrere Kilobytes) bzw. ganze Dateien als Zugriffseinheit verwendet werden. Für diese Einsatzbereiche ist eine gewisse Latenzzeit für die Vergabe der Sperrberechtigung unter den Leistungsgesichtspunkten vertretbar. In Echtzeit-Editoren kommen Sperrverfahren jedoch sehr selten zum Einsatz.

Optimistische Sperrverfahren

Ein Verfahren zur Konfliktauflösung ist das optimistische Sperrverfahren. Es basiert auf der Annahme, dass ein präventives Sperren durch relativ selten auftretende Konflikte einen unnötig hohen Aufwand und Leistungseinbußen nach sich ziehen würde.

Es verfolgt daher den Ansatz, dass alle Dokumentenbereiche bearbeitet werden können und im Anschluss validiert und kommuniziert werden. Erst zum Zeitpunkt der Veröffentlichung der Änderungen werden die Dokumentenbereiche für die Validierung gesperrt und es wird überprüft, ob Konflikte mit anderen Benutzern aufgetreten sind. Diese Konflikte müssen entweder manuell behoben werden (z.B. nachdem der Benutzer über die aufgetretenen Konflikte informiert wurde) oder die Änderungen müssen, vergleichbar mit einer fehlgeschlagenen Datenbanktransaktion, rückgängig gemacht werden.

Der Vorteil von optimistischen Sperrverfahren liegt in der gleichzeitigen Bearbeitung des Dokuments durch alle Benutzer. Die Konflikterkennung ist jedoch mit einem hohen Aufwand verbunden und birgt nur bedingt die Möglichkeit für eine automatische Konfliktauflösung. Vor

allem bei einer groben Granularität und einer steigenden Benutzerzahl ist die Häufigkeit von Validierungsfehlern und der Aufwand für eine Konfliktauflösung sehr hoch.

Der Aufwand beim Einsatz von Sperrverfahren liegt nicht nur in der Konflikterkennung, sondern auch in der Verwaltung der gesperrten Bereiche. Je kleiner die Granularität des Sperrbereichs ist, um so höher ist der Aufwand für die Verwaltung. Je größer der Sperrbereich ausfällt, um so mehr wird die gleichzeitige Bearbeitung eingeschränkt.

Sperrverfahren sind für den Einsatz in Saros ungeeignet, da der große Zeitaufwand, den die Sperrung ansich beansprucht, und die damit entstehende Verzögerung bzw. Blockierung des Arbeitsprozesses, die an Saros gestellten Anforderung nicht erfüllen kann. Bei einer kleineren Granularität würde der Aufwand für den Einsatz von Sperrverfahren zu hoch sein. [43]

6.2.2 Operational Transformation (OT)

Im Gegensatz zu Sperrverfahren wird bei der Operational Transformation auf kleine Einheiten zugegriffen, wie etwa einzelne Wörter oder einzelne Zeichen. Sie wird deshalb häufig für eine eng gekoppelte, synchrone Gruppenarbeit verwendet. Änderungen werden bei diesem Verfahren sofort mitgeteilt, so dass die Nutzer die Anzeige aller Modifikationen nach dem WYSIWIS-Prinzip (What You See Is What I See) übermittelt bekommen [42].

Operational Transformation ist nach Gerlicher [43] ein Verfahren zur Konsistenzerhaltung durch Konfliktauflösung. Jeder Benutzer hat eine lokale Kopie des gemeinsamen Dokuments und führt die Operationen sofort auf seiner lokalen Kopie des Dokuments aus. Im Anschluss werden lokal durchgeführte Operationen an die anderen Benutzer verteilt, die diese auf ihren Kopien ausführen. Eine empfangende Operation kann vor ihrer Ausführung transformiert werden, um die Konvergenz der lokalen Dokumentkopien zu erreichen.

Die Ansätze von Operational Transformation (OT) erfüllen alle für den Einsatz in Saros notwendigen Bedingungen. Da alle Änderungen sofort auf der lokalen Kopie ausgeführt und im Anschluss an die anderen Benutzer übertragen werden, erfüllt OT die Bedingung der Echtzeit-Anforderung im vollen Umfang. Entfernte Operationen werden nur bei Konflikten zur Sicherung der Konvergenz transformiert, anderenfalls sofort beim entfernten Benutzer ausgeführt. Jeder Benutzer hat eine eigene Kopie auf seinem Rechner und kommuniziert nur die Änderungen, was eine hohe Performance sichert und die Anforderung an verteilte Anwendungen im vollen Umfang erfüllt. Jeder Benutzer kann somit blockierungsfrei auf alle Dokumentabschnitte zugreifen, da die eigentliche Konsistenzprüfung erst beim Empfang der entfernten Operationen anderer Benutzer erfolgt. Daher bieten diese blockierungsfreien

Verfahren dem Benutzer die größtmögliche Flexibilität und erfüllen die Zwanglosigkeits-Anforderung im vollem Umfang.

Die wissenschaftlichen Arbeiten der jüngsten Vergangenheit zum Thema Echtzeit-Gruppeneditoren beschäftigen sich fast ausschließlich mit Methoden und Verfahren zur Verbesserung der Konsistenzerhaltung mit Verfahren der Operational Transformation.

6.3 Vergleich von Echtzeit-Gruppeneditoren

Im Folgenden werden aktuelle Entwicklungen im Bereich der Echtzeit-Gruppeneditoren im Hinblick auf die eingesetzten Nebenläufigkeitsverfahren betrachtet.

6.3.1 IRIS

IRIS [44] basierte in den frühen Phasen der Entwicklung auf einer pessimistischen Nebenläufigkeitskontrolle mit Vortierungsverfahren. Nach einer Neustrukturierung und Neuimplementierung 1996 mit Java wurde ein optimistisches Sperrverfahren verwendet, da schlechte Performance-Erfahrungen mit pessimistischen Sperrverfahren in Weitverkehrsnetzen einen Wechsel notwendig machten.

Die Dokumente werden in einzelne Strukturelemente unterteilt, welche dann einzeln bearbeitet werden können. IRIS beinhaltet keine Algorithmen zur automatischen Konfliktauflösung, sondern enthält Historie-Informationen zur Rekonstruktion des Änderungsverlaufs, so dass eine manuelle Konfliktauflösung durch den Anwender erfolgen kann. Dieses Vorgehen ermöglicht IRIS sowohl eine synchrone als auch eine asynchrone Kollaboration. Die Kommunikation erfolgt dezentral über Multicast-Nachrichten.

6.3.2 SubEthaEdit

SubEthaEdit [45] ist ein von Apple angebotener kommerzieller kollaborativer Editor, der gleichzeitigen Schreibzugriff gewährt, ohne Teilbereiche des Textes zu sperren. Die Konsistenzsicherung auf der Basis der Operational Transformation wird ohne Eingreifen der Anwender automatisch durchgeführt. Es besitzt zur Unterstützung der gemeinsamen Arbeit an Dokumenten zusätzliche Awareness-Informationen und visuelle Informationen über die Urheber einzelner Textpassagen. Durch den Einsatz der von Apple entwickelten Bonjour Technologie, einer Umsetzung der zero-conf-Technik zur konfigurationsfreien Vernetzung von Geräten in lokalen Rechennetzen, sind veröffentlichte Dokumente automatisch in einem lokalen Netzwerk verfügbar.

Leider ist SubEthaEdit ein für Apple-Systeme vorbehaltenes kommerzielles Produkt ohne aktuelle Bestrebungen der Portierung auf andere Betriebssysteme.

6.3.3 Eclipse Communication Framework

Das Eclipse Communication Framework (ECF) realisiert einen Kommunikationsserver, der die Servlet-Bridge aus dem Eqnuinox-Incubator-Projekt nutzt, um OSGi auf dem Server innerhalb eines Webservers ablaufen zu lassen.

Bei **OSGi** (Open Services Gateway Initiative) handelt es sich um die Bezeichnung einer hardwareunabhängigen dynamischen Softwareplattform, auf der Anwendungen und Dienste entsprechend dem SOA-Paradigma ausgeführt werden können.

Es abstrahiert die Kommunikations-Protokolle als ECF-Protokoll-Adapter, so dass die eigentliche Semantik der Protokolle innerhalb der Adapter implementiert werden. Es können mit ECF sowohl Punkt-zu-Punkt, als auch Client/Server Anwendungen realisiert werden. Dieses Framework bildet die Basis für Projekte aus dem Umfeld der Echtzeit-Gruppeneditoren.

Eclipse Shared Text Editor

Die Eclipse-Shared-Text-Editor-Erweiterung [46] ist eine Beispielimplementierung eines Gruppeneditors auf Basis des Eclipse Communication Frameworks. Zur Kommunikation zwischen den Teilnehmern muss ein ECF-Server verfügbar sein, über den die eigentliche Kommunikation verläuft.

Es ermöglicht das Veröffentlichen einzelner Dateien und das gemeinsame Arbeiten an diesen Dateien innerhalb einer Sitzung. Diese Umsetzung besitzt weder Mechanismen zur Nebenläufigkeitskontrolle noch zusätzliche Awareness-Informationen. Diese Umsetzung eines Echtzeit-Gruppeneditors ist vielmehr eine Machbarkeits-Studie, die die Flexibilität und die Einsatzmöglichkeiten des ECF von Seiten der Entwickler aufzeigen soll.

Real-Time Shared Editor

Aufbauend auf der bereits vorgestellten Eclipse-Shared-Text-Editor-Erweiterung hat das ECF Entwickler-Team eine kollaborative Eclipse-Editor-Erweiterung mit Nebenläufigkeitskontrolle unter dem Namen Real-Time Shared Editor [47] veröffentlicht. Es ermöglicht zwei Anwendern gleichzeitig auf ein zuvor veröffentlichtes Dokument schreibend zuzugreifen.

Die Nebenläufigkeitskontrolle basiert auf dem Konzept der Operational Transformation. Als Algorithmus wird der Jupiter Ansatz verwendet, um ein blockierungsfreies Arbeiten beider Anwender und eine automatische Konfliktauflösung zu ermöglichen.

Auch dieses Projekt ist eher eine Machbarkeits-Studie durch das ECF-Entwickler-Team. Es limitiert die gleichzeitige Bearbeitung eines Dokuments auf zwei Anwender und stellt keine Umsetzung von zusätzlichen Awareness-Informationen (z.B. welcher Anwender welche Textstelle eingefügt hat) zur Verfügung.

Die Dokumente müssen zuvor, wie bereits im Eclipse-Shared-Text-Editor, einzeln veröffentlicht werden.

6.3.4 ACE

Das Projekt ACE [48] ist ein an der Berner Hochschule für Technik und Informatik entwickelter kollaborativer Editor. Es handelt sich hierbei um die Eigenentwicklung eines Editors, der das gemeinsame blockierungsfreie Arbeiten von mehreren Anwendern an einem Dokument ermöglicht. Die Entwickler legten großen Wert auf detaillierte Awareness-Informationen mit farbigen Syntaxhervorhebungen und Schreibzugriffsinformationen der einzelnen Anwender in den kollaborativen Dokumenten.

Die Verwendung der zero-conf Technologie ermöglicht das automatische Auffinden veröffentlichter Dokumente innerhalb eines lokalen Subnetzes (vgl. Bonjour Technologie), die kollaborative Arbeit an den gemeinsamen Dateien erfolgt über eine auf BEEP basierende Netzwerkkommunikation.

Auch in diesem Projekt basiert die Nebenläufigkeitskontrolle auf dem Konzept der Operational Transformation auf Basis des Jupiter Algorithmus, um einen blockierungsfreien, uneingeschränkten Echtzeit-Gruppeneditor mit automatischer Konfliktauflösung zu realisieren.

6.4 Fazit

Die Betrachtung der vorgestellten aktuellen Ansätze von Echtzeit-Gruppeneditoren offenbart eindeutige Gemeinsamkeiten.

Alle Echtzeit-Gruppeneditoren mit der primären Anforderung der synchronen kollaborativen Echtzeitbearbeitung von Dokumenten basieren auf den Verfahren der Operational Transformation. Das blockierungsfreie, ungezwungene Bearbeiten von Dokumenten steht bei diesen Ansätzen im Vordergrund, so dass Sperrverfahren für die Umsetzung ausgeschlossen werden müssen. IRIS verwendet als einziges Projekt Sperrverfahren, da es sowohl synchrone als auch asynchrone Kommunikation erfüllen muss und daher

Einschränkungen in der synchronen Bearbeitung in Kauf nimmt. Auffällig ist die häufige Umsetzung der Nebenläufigkeitskontrolle auf Basis des Jupiter Ansatzes.

Im folgenden Kapitel werden die Verfahren zur Operational Transformation einer detaillierteren Betrachtung unterzogen, da sie nach erster Untersuchung für eine Umsetzung der Nebenläufigkeitskontrolle den aufgestellten Anforderungen am ehesten entsprechen.

7. Operational Transformation (OT)

Operational Transformation (OT) ist ein Verfahren zu Realisierung einer Nebenläufigkeitskontrolle bei eng gekoppelter, synchroner Gruppenarbeit. Zum Verständnis werden in diesem Kapitel Konsistenz-Probleme und ein Modell zur Konsistenz-Sicherung definiert, um darauf aufbauend den Ansatz von Operational Transformation vorzustellen.

7.1 Vorgeschichte

Erstmalig wurde ein OT Algorithmus von Ellis und Gibbs 1989 [39] vorgestellt. Es handelt sich dabei um den dOPT (distributed Operational Transformation) Algorithmus, der erstmals im GROVE System implementiert wurde. Das gemeinsame Dokument liegt jedem Teilnehmer als lokale Kopie vor. Die Änderungsoperationen werden sofort auf dem lokalen Dokument ausgeführt und an die anderen Teilnehmer kommuniziert. Der dOPT Algorithmus kann die Konvergenz nicht in allen Szenarien sicherstellen. Das signifikante Konfliktszenario ist unter dem Namen dOPT-Puzzle veröffentlicht worden [49]. Der dOPT-Algorithmus bildet jedoch die Grundlage für spätere Entwicklungen von OT Algorithmen.

Der REDUCE Ansatz erweitert die in GROVE identifizierten Probleme von Divergenz und Kausalitätsverletzung um das Problem der Intentionsverletzung und ermöglicht die Aufstellung des als Grundlage für zukünftige OT Algorithmen notwendige Konsistenz-Modells. Zur Lösung des dOPT-Puzzles wurde in REDUCE [50] der Generic Operational Transformation (GOT) Algorithmus entwickelt, der später optimiert im GOTO (optimized GOT) Algorithmus weiterentwickelt wurde.

7.2 Gemeinsames Dokumentenmodell

Jeder Teilnehmer besitzt eine lokale Kopie der Datei und kann diese lokal manipulieren. Im Anschluss werden die Änderungen an alle Teilnehmer kommuniziert. Das gemeinsame Dokument ist ein aus einer Folge von Zeichen bestehendes Textdokument, deren Zeichen von Null bis zur Gesamtanzahl der Zeichen im Dokument indexiert sind. Die Manipulation eines solchen Dokuments erfolgt durch folgende primitive Operationen:

- Einfügen: mit dem Befehl insert(p,s) werden an der Stelle p die Zeichen s eingefügt
- Löschen: mit dem Befehl delete(p,l) werden ab der Position p alle Zeichen bis zur Position p+l gelöscht.

7.3 Definitionen

Wir definieren nach Lamport [51] eine kausale (partielle) Ordnungsrelation der Operationen in Hinblick auf ihre Erzeugungs- und Ausführungsreihenfolge.

Definition 1: Kausale Ordnungsrelation „→“

Gegeben sind zwei Operationen O_1 und O_2 die von den Teilnehmern i und j erzeugt wurden. Dann gilt $O_1 \rightarrow O_2$, wenn

1. i = j und die Erzeugung von O_1 vor der Erzeugung von O_2 erfolgt ist, oder
2. i ≠ j und die Ausführung von O_1 vom Teilnehmer i vor der Erzeugung von O_2 erfolgt ist, oder
3. es existiert eine Operation O_x, so dass $O_1 \rightarrow O_x$ und $O_x \rightarrow O_2$ gilt

Definition 2: Abhängige und unabhängige Operationen

Gegeben sind zwei Operationen O_1 und O_2.

1. O_2 ist abhängig von O_1, wenn gilt $O_1 \rightarrow O_2$
2. O_1 und O_2 sind unabhängig (oder nebenläufig), ausgedrückt durch $O_1 \parallel O_2$, wenn weder $O_1 \rightarrow O_2$ noch $O_2 \rightarrow O_1$ gilt

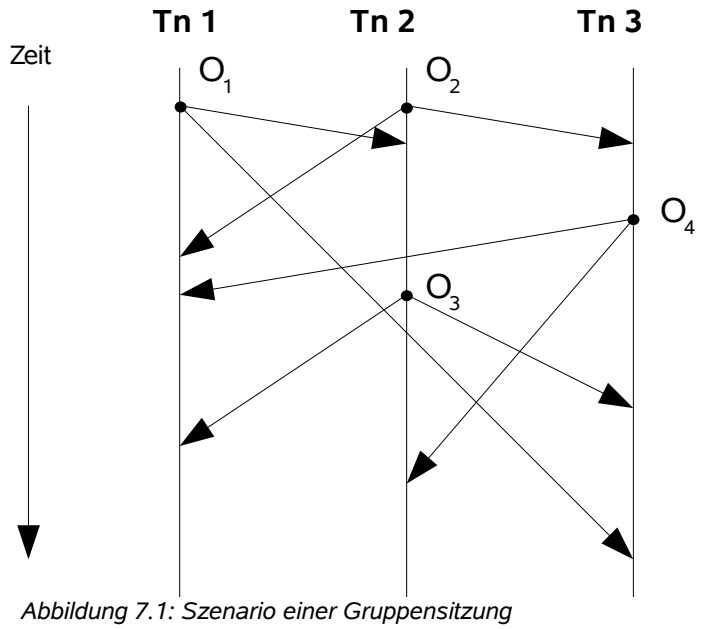

Abbildung 7.1: Szenario einer Gruppensitzung

Abbildung 7.1 verdeutlicht eine Gruppensitzung mit drei Teilnehmern (Tn) auf einem Zeitgraphen. Es gibt vier Änderungsoperationen in diesem Szenario: O_1 wird durch

Teilnehmer 1 (Tn 1) erzeugt, die Operationen O_2 und O_3 durch Teilnehmer 2 (Tn 2) und die Operation O_4 durch den Teilnehmer 3 (Tn 3). In diesem Szenario werden die Operationen sofort lokal ausgeführt und anschließend an die anderen Teilnehmer übermittelt und dort lokal ausgeführt.

Die Pfeile im Graphen repräsentieren die Übertragung der Operationen zwischen den Teilnehmern. Die vertikalen Linien repräsentieren die durch den entsprechenden Teilnehmer ausgeführten Aktivitäten.

Bezug nehmend auf die Definitionen 1 und 2 existieren drei Paare abhängiger Operationen in diesem Szenario: $O_1 \rightarrow O_3$, $O_2 \rightarrow O_3$ und $O_2 \rightarrow O_4$
Die Ausführung von O_1 erfolgt vor der Erzeugung von O_3, die Erzeugung von O_2 erfolgt vor der Erzeugung von O_3, und die Ausführung von O_2 erfolgt vor der Erzeugung von O_4.

Des Weiteren gibt es drei Paare unabhängiger Operationen im obigen Szenario: $O_1 \parallel O_2$, $O_1 \parallel O_4$, sowie $O_3 \parallel O_4$, da in keinem dieser Paare eine Operationsausführung vor der Erzeugung erfolgt.

7.4 Konsistenz-Probleme

Das Szenario aus Abbildung 7.1 beinhaltet verschiedene Konsistenz-Probleme, die im Folgenden untersucht werden. Beim dem GROVE-Ansatz [39] liegt das gemeinsame Dokument bei jedem Teilnehmer als Kopie vor. Im Falle einer Änderungsoperation wird diese sofort auf der lokalen Kopie des Teilnehmers ausgeführt und im Anschluss an alle Gruppenteilnehmer übertragen.

Entfernte Operationen anderer Teilnehmer werden nach ihrer Ankunft und nicht nach ihrer eigentlichen Ordnung ausgeführt.

Divergenz

Das in Abbildung 7.1 dargestellte Szenario einer Gruppensitzung beinhaltet vier Operationen, die nach ihrer Ankunft in der folgenden Reihenfolge ausgeführt werden.

- Teilnehmer 1: O_1,O_2,O_4,O_3
- Teilnehmer 2: O_2,O_1,O_3,O_4
- Teilnehmer 3: O_2,O_4,O_3,O_1

Wenn nicht alle Operationen kommutativ sind, wird das Ergebnis der lokalen Ausführung nicht bei allen Teilnehmer identisch sein. Dieses Problem wird Divergenz genannt.

Kausalitätsverletzung (Causality Violation)

Wenn Teilnehmer Operationen ohne eine Synchronisation erzeugen und kommunizieren, können diese bei den anderen Teilnehmern nach ihrem Eintreffen abweichend von der ursprünglichen Reihenfolge ausgeführt werden.

In Abbildung 7.1 wird die Operation O_3 nach dem Empfang der Operation O_1 auf Seiten des Teilnehmers 2 erzeugt. Somit gilt: $O_1 \rightarrow O_3$

Teilnehmer 3 empfängt jedoch O_3 vor O_1, so dass die Ausführung von O_3 vor O_1 zu einem inkonsistenten Zustand führt. Dieses Problem wird Causality Violation genannt, wenn die Operationsausführung außerhalb der ursprünglichen Reihenfolge erfolgt, so dass es zu einer Verletzung der Abhängigkeiten von Operationen kommt.

Intentionsverletzung (Intention Violation)

Das Intention-Problem ist ein Konsistenz Problem, welches im REDUCE-Ansatz [50] erstmalig im Zusammenhang mit unabhängigen Operationen identifiziert wurde und sich in einem wichtigen Punkt vom Problem der Divergenz unterscheidet. Divergenz kann immer durch die Verwendung eines Protokolls zur Serialisierung der Operationen aufgelöst werden. Wohingegen eine Intentionsverletzung nicht durch Serialisierung aufgelöst werden kann, wenn die Operationen bereits in ihrer ursprünglichen Form ausgeführt wurden.

Um das Problem zu verdeutlichen, werden die zwei unabhängigen Operationen O_1 und O_2 aus Abbildung 7.1 betrachtet. Die Ausführung von O_2 auf dem Dokument, welches durch die vorhergehende Operation O_1 verändert wurde, führt zu einer Veränderung des Dokumentenzustands, der vom Teilnehmer 2 nicht beabsichtigt gewesen ist.

Zur Verdeutlichung der obigen Situation wird angenommen:

1. Der initiale Dokumentenzustand besteht aus der Zeichenfolge „ABCDEF“
2. Die Operation O_1 = insert (1, “12“) fügt zwischen „A“ und „B“ die Zeichenfolge „12“ ein.
3. Die Operation O_2 = delete(2, 2) löscht die Zeichenfolge „CD“ aus dem Dokument.

Nach Ausführung beider Operationen ohne eine Verletzung der Intention, sollte der finale Dokumentenzustand aus der Zeichenfolge „A12BE“ bestehen.

Bei Teilnehmer 1 wird nach lokaler Ausführung von O_1 die ankommende Operation O_2 auf dem neuen Dokumentenzustand „A12BCDE“ ausgeführt. Der resultierende Dokumentenzustand nach Ausführung beider Operationen ist „A1CDE“. Die Ausführung von O_2 löscht nicht, wie von Teilnehmer 2 beabsichtigt die Zeichenfolge „CD“, sondern die

Zeichenfolge „2B“. Gleichzeitig wird die beabsichtigte Wirkung von O_1 verletzt, da durch die Ausführung von O_2 das zuvor einfügte Zeichen „2“ gelöscht wurde.

Der Einsatz von Serialisierung würde die Ausführungsreihenfolge von O_1 vor O_2 sicherstellen, der resultierende Dokumentenzustand „A1CDE“ würde jedoch ebenfalls die Absicht der Operationen O_1 und O_2 verletzen.

7.5 Konsistenz-Modell

Auf Grundlage der zuvor identifizierten Probleme kann ein Konsistenz-Modell wie folgt definiert werden:

1. **Konvergenz** (Convergence): wenn sich alle Kopien eines Dokuments nach Ausführung eine Menge von Operationen in einem identischen Endzustand befinden
2. **Kausalitäts-Erhaltung** (Causality-preservation): für jedes Paar von Operationen O_a und O_b gilt: wenn $O_a \rightarrow O_b$ dann muss die Ausführung von O_a bei allen Teilnehmern vor der Ausführung von O_b erfolgen
3. **Intentions-Erhaltung** (Intention-preservation): für jede Operation O muss das Ergebnis nach Ausführung von O bei allen Teilnehmern identisch mit der ursprünglichen Absicht bei der Erzeugung von O sein. Das Ergebnis der Ausführung von O darf sich nicht auf die Effekte von entfernten unabhängigen Operationen auswirken.

Die Algorithmen zur Operational Transformation wurden entwickelt um die Probleme der Divergenz, Intentions- und Kausalitätsverletzung zu beheben. Ein Echtzeit-Gruppeneditor wird dabei als konsistent bezeichnet, wenn der eingesetzte Operation Transformation Algorithmus immer den Erhalt von Konvergenz, Intention und Kausalität gewährleistet [43].

7.6 Operational Transformation

In der Arbeit Ellis und Gibbs [39] wurde erstmalig ein neuer Ansatz für einen Algorithmus der Nebenläufigkeitskontrolle unter dem Namen Operational Transformation (OT) vorgestellt.

Der OT Ansatz kann vereinfacht dargestellt in zwei Hauptkomponenten unterteilt werden:

1. Der Integrations-Algorithmus ist verantwortlich für den Empfang, die Verbreitung und die Ausführung von Operationen. Diese Komponente gewährleistet die Konvergenz und die Kausalität-Erhaltung.
2. Die Transformationsfunktion ist verantwortlich für das Zusammenführen zweier konkurrierender Operationen und gewährleistet die Intention-Erhaltung.

Transformationsfunktionen in OT Algorithmen transformieren Operationen mit dem Ziel die Auswirkungen anderer Operationen durch Transformationen einzubeziehen oder zu unterbinden. Vereinfacht dargestellt, verändert eine Transformation zweier unabhängiger, konkurrierender Operationen die Positionsangabe einer Operation derart, dass das Ergebnis nach Ausführung die Intention-Erhaltung erfüllt.

Formal kann zwischen zwei Arten von Transformationen unterschieden werden: Inclusion Transformation und Exclusion Transformation.

Eine Transformationsfunktion T muss für jede Kombination von zwei Operationen definiert sein. Auf Basis der zuvor definierten primitiven Operationen Einfügen und Löschen auf einem linearen Textdokument, existieren je vier Kombinationen der Operationen für IT und ET, (T (einfügen,einfügen), T(einfügen, löschen), T(löschen, einfügen), T(löschen, löschen)).

7.6.1 Definitionen

Um die notwendigen Beziehungen zwischen Operationen für eine korrekte Transformation zu definieren, wird nach Sun et. al [40] der Begriff des Operation-Kontexts CT_O eingeführt. Dieser Kontext besteht aus der Menge von Operationen, deren Ausführung notwendig ist, um vom initialen Dokumentenzustand in den Zustand vor Erzeugung der Operation O zu gelangen (Definition-Kontext).

Die Ausführung einer Operation kann nur innerhalb dieses Kontextes korrekt erfolgen. Wenn der aktuelle Kontext (Ausführung-Kontext) sich vom Definition-Kontext unterscheidet, muss die Operation transformiert werden, um sie innerhalb des aktuellen Kontextes korrekt ausführen zu können.

Für die Spezifizierung der Vor- und Nachbedingungen von Transformationsfunktionen werden im Folgenden zwei Kontext-basierende Operations-Relationen definiert:

Definition 3 : *Kontext-Äquivalenz-Relation* ⊔

Gegeben sind zwei Operationen O_1 und O_2 mit den entsprechenden Kontexten CT_{O1} und CT_{O2}. O_1 und O_2 sind Kontext-äquivalent, wenn gilt: $CT_{O1} = CT_{O2}$

Die Kontext-Äquivalenz-Relation ist somit offensichtlich auch transitiv.

Definition 4: *Kontext-Vorgänger-Relation* ↦

Gegeben sind zwei Operationen O_1 und O_2 mit den entsprechenden Kontexten CT_{O1} und CT_{O2}. O_1 ist Kontext-Vorgänger von O_2, wenn gilt: $CT_{O2} = CT_{O1} + [O_1]$

Es ist zu beachten, dass die Kontext-Vorgänger-Relation ↦ per Definition nicht transitiv ist.

7.6.2 Inclusion Transformation (IT)

Inclusion Transformation bezeichnet die Transformation, die den Effekt einer Operation in eine andere Operation "einbindet". Wenn also die Operation O_a gegen die Operation O_b so transformiert wird, dass die Transformation die Auswirkung von O_b beinhaltet.

Spezifikation 1: $(IT(O_a,O_b) : O'_a)$

1. Vorbedingung der Eingabe-Parameter: $O_a \sqcup O_b$

2. Nachbedingung der Ausgabe: $O_b \mapsto O'_a$, wobei die Auswirkung von O'_a auf den Kontext $CT_{O'a}$ mit der Auswirkung von O_a auf den Kontext CT_{Oa} übereinstimmt.

7.6.3 Exclusion Transformation (ET)

ET ist notwendig, um die Auswirkungen der unterschiedlichen abhängigen Operationen aufzuheben. Dadurch kann ein einheitlicher Dokumentenstand erzeugt werden, so dass im Anschluss mit IT die eigentliche Transformation durchgeführt werden kann. Es bezeichnet ein Transformationsverfahren, in welchem die Operation O_a gegen eine andere Operation O_b derart transformiert wird, dass die Auswirkung von O_b in O_a ausgeschlossen wird.

Beispiel für ET:

O_1 und O_4 in Abbildung 7.1 sind unabhängige Operationen, die auf unterschiedlichen Dokumentenstatus erzeugt werden. Sobald O_4 bei Teilnehmer 1 eintrifft, kommt es zu einem inkonsistenten Dokumentenzustand, wenn O_4 gegen O_1 transformiert wird. Um den inkonsistenten Zustand zu vermeiden, muss eine ET auf O_4 angewandt werden, um O_4 gegen die kausale Vorgänger-Operation O_2 zu transformieren. O'_4 beinhaltet somit nicht die Ausführung von O_2 auf O_4 und operiert nun auf demselben Dokumentenstand wie O_1. Im Anschluss kann O'_4 mit IT gegen O_1 transformiert werden.

Spezifikation 2: $(ET(O_a,O_b) : O'_a)$

1. Vorbedingung der Eingabe-Parameter: $O_b \mapsto O_a$

2. Nachbedingung der Ausgabe: $O_b \sqcup O'_a$ und die Auswirkung O'_a auf den Kontext $CT_{O'a}$ stimmt mit der Auswirkung von O_a auf den Kontext CT_{Oa} überein.

7.6.4 Umkehrbarkeit

Zusätzlich zu den Vor- und Nachbedingungen müssen Transformationsfunktionen die nachfolgende Umkehrbarkeits-Anforderung erfüllen.

Definition 5: Umkehrbarkeits-Anforderung

Gegeben sind zwei Operationen O_a und O_b :

1. Für $O_b \sqcup O_a$ und $O'_a = IT(O_a,O_b)$ muss $O_a = ET(O'_a,O_b)$ gelten

2. Für $O_b \mapsto O_a$ und $O'_a = ET(O_a,O_b)$ muss $O_a = IT(O'_a,O_b)$ gelten

7.6.5 Transformationseigenschaften

Die folgenden Transformationseigenschaften sind für OT Algorithmen hinreichend und notwendig, um die Konvergenz in beliebig komplexen Szenarien zu gewährleisten. [52]

Transformationseigenschaft 1 (TP1):

- Die Bedingung **TP1** definiert eine Zustands-Äquivalenz. Die Ausführung von O_1, gefolgt von $T(O_2, O_1)$, muss denselben Dokumentenzustand erzeugen, wie die Ausführung von O_2 gefolgt von $T(O_1, O_2)$:

  TP1 : $O_1\ O'_2 \equiv O_2\ O'_1$

Transformationseigenschaft 2 (TP2):

- Die Bedingung TP2 gewährleistet, das die Transformation einer Operation nicht abhängig ist von der Reihenfolge, in der die Operationen ausgeführt werden.

  TP2 : $T(T(O, O_1) , O'_2) \equiv T(T(O, O_2) , O'_1)$, für ein beliebiges O

7.7 Auswahl des geeigneten OT Algorithmus

Aufbauend auf den Arbeiten von Ellis und Sun [49] wurde eine Vorauswahl an OT Algorithmen getroffen, welche die im vorhergehenden Kapitel definierten Eigenschaften und Anforderungen erfüllen, so dass Konvergenz, Kausalität und Intention in beliebigen Szenarien sichergestellt werden können.

7.7.1 adOPTed

Der adOPTed Algorithmus [53] basiert auf dem dOPT Algorithmus und verwendet einen n-dimensionalen Interaktion-Modell-Graphen, um die Konvergenz und die Kausalität sicherzustellen. Zur Gewährleistung der Intention muss die Transformationsfunktion sowohl TP1 als auch TP2 sicherstellen, welches die Komplexität des Algorithmus erheblich erhöht.

7.7.2 GOTO

Der GOTO Algorithmus [40] ermöglicht, den GOT Algorithmus durch Reduzierung der Anzahl der auszuführenden Transformationen zu optimieren. Er arbeitet auf einem 1-dimensionalen Historie-Puffer mit einer Vektor-Zahl und benötigt sowohl IT als auch ET um das Konvergenz-Modell zu erfüllen.

7.7.3 Jupiter

Der Jupiter Algorithmus [54] verwendet einen zweidimensionalen Vektor-Raum und sichert in Verbindung mit einer Nachrichtenwarteschlange die Konvergenz und Kausalitäts-Erhaltung. Zur Gewährleistung der Intention muss die verwendete Transformationsfunktion in Form einer Transformations-Matrix nur IT unter Einhaltung von TP1 sicherstellen. Die Kommunikation der Änderungsoperationen erfolgt über einen zentralen Jupiter-Server, der die Änderungen an die Clients kommuniziert.

7.8 Fazit

Bereits in der Voruntersuchung von Echtzeit-Gruppeneditoren im Abschnitt 6.3 war die häufige Verwendung des Jupiter-Algorithmus zur Realisierung einer Nebenläufigkeitskontrolle in synchronen Echtzeit-Gruppeneditoren zu beobachten. Jupiter ist ein stabiler Algorithmus zur Sicherung der Konvergenz [40] und bietet sich sowohl von der geringere Komplexität der Umsetzung her, als auch vom geringeren Kommunikationsaufwand gegenüber den anderen Ansätzen an. Der Jupiter-Ansatz ist daher

ein vielversprechender Ansatz zur Realisierung und Integration einer Nebenläufigkeitskontrolle in Saros.

8. Jupiter Algorithmus

In diesem Kapitel wird das Konzept des Jupiter-Ansatzes zur Realisierung einer Konsistenz-Sicherung vorgestellt, die Implementierungsdetails erläutert und die durchgeführten Test-Szenarien beschrieben.

8.1 Einführung Jupiter

Der Konsistenz-Sicherungs-Algorithmus in Jupiter wurde vom dOPT Algorithmus aus GROVE [42] abgeleitet, um diesen Algorithmus in einer Umgebung mit mehreren Clients und einem zentralen Server umzusetzen.

Jeder Client besitzt eine Kopie des gemeinsamen Dokuments wie bereits in GROVE. Der Unterschied besteht darin, dass die Änderungen des gemeinsamen Dokuments von einem zentralen Server verwaltetet werden und die Kommunikation zwischen jeweils einem Client und dem Server stattfindet.

Die Hauptbestandteile von Jupiter sind die Transformationsfunktion, die Nachrichtenwarteschlange und ein zweidimensionaler Vektor-Raum.

8.1.1 Transformationsfunktion

- Transformationsfunktion mit folgender Eigenschaft
  - xform(a,b) = (a',b'), so dass a → b' = b' → a
  - a auf b' angewandt liefert das selbe Ergebnis, wie b auf a' angewandt.

8.1.2 Zweidimensionaler Vektor Raum

Der zweidimensionale Vektor Raum (x,y) bildet den aktuellen Dokumenten-Zustand auf Grundlage der empfangen und gesendeten Operationen ab. Jede Seite besitzt eine zwei-elementige Vektor Nummer bestehend aus:

- x: Anzahl der lokal erzeugten Operationen
- y: Anzahl der empfangenen Operationen

Wird eine lokale Operation ausgeführt und versendet, wird die Operation mit dem Dokumenten-Zustand zum Zeitpunkt ihrer Ausführung [x,y] versehen und im Anschluss daran der lokale Zähler erhöht. [x+1, y]

Bei einer eintreffenden Nachricht wird nach der Transformation der Zähler der entfernten Operationen der Empfänger-Seite erhöht [x,y+1].

8.1.3 Nachrichtenwarteschlange

In der Nachrichtenwarteschlange werden alle lokal ausgeführten Operationen mit ihrer aktuellen Vektor Nummer als Status gespeichert und verbleiben dort, bis eingehende Nachrichten den Empfang auf Grund ihrer Vektor Nummer bestätigen.

Eine Operation O_1 wird beim Client im Dokumenten-Zustand [0,0] lokal ausgeführt, versendet und in der Warteschlange abgelegt. Empfängt der Client zu einem späteren Zeitpunkt eine Server-Nachricht mit dem Dokumentenzustand [0,1] löscht er die lokale Nachricht aus der Warteschlange, da diese Operation durch den Zähler der entfernten Operationen des Server bestätigt wurde.

Empfängt eine Seite eine Nachricht mit der Vektor Nummer [c,d], löscht sie alle Operationen mit dem Status [a,b] aus der Warteschlange, für die a < d gilt, da die lokale Operation des Clients in der Vektor-Nummer des Servers als entfernte Operation vermerkt ist. Im obigen Beispiel hat der Server die Client Operation empfangen und seinen Zähler der entfernten Operationen erhöht.

8.2 Zwei-Wege Protokoll

Jede Nachricht wird mit dem Status des Senders zum Zeitpunkt vor der Erzeugung der Nachricht versehen. Diese Labels werden zur Erkennung von Konflikten verwendet, die die xform-Funktion dann auflöst. Der Algorithmus garantiert, dass egal wie weit die Client-Server-Zustände auf dem Graphen divergieren, ein konsistenter Zustand hergestellt werden kann.

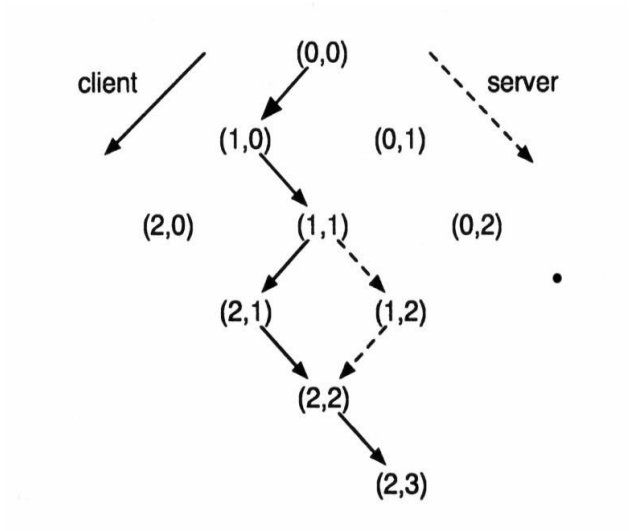

Abbildung 8.1: Zweidimensionaler Status-Raum [55].

In der Abbildung 8.1 erzeugt der Client eine lokale Nachricht und sendet sie an den Server. Nach dem Empfang der Nachricht befinden sich beide Seiten im Zustand (1,0) des Status-Raums. Lokal habt der Client den Status [1,0] und der Server den Status [0,1].

Sendet der Server im Anschluss eine erzeugte Nachricht an den Client, befinden sich beide nach Empfang der Nachricht durch den Client im Zustand (1,1), da beide zu diesem Zeitpunkt jeweils eine Operation ausgeführt haben.

In diesem Stand erzeugen sowohl Server als auch Client eine konkurrierende Operation, woraufhin ihre Dokumentenzustände divergieren. Im Status Raum wird dies durch divergierende Zustände (2,1) und (1,2) abgebildet. Empfangen Client und Server die jeweiligen entfernten Operationen, müssen sie diese gegen ihre lokalen Operationen transformieren und ausführen. Durch die Transformation wird für das Nachrichtenpaar mit demselben Ausgangszustand ein gemeinsamer Ergebniszustand erreicht.

Solange Client und Server nur einen Schritt auseinander liegen, kann xform direkt angewendet werden. Wenn diese jedoch weiter divergieren, ist die Situation schwieriger. Abbildung 8.2 veranschaulicht das Konvergieren der Zustände auch bei größer werdenden Abständen.

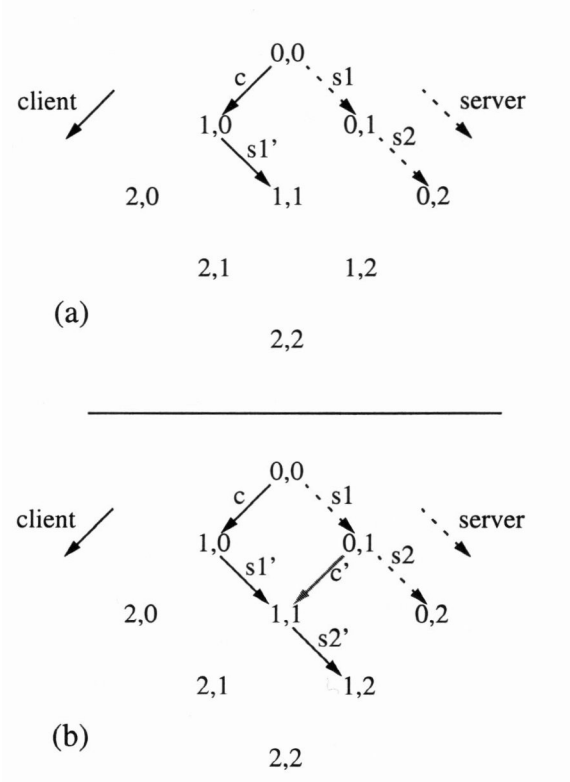

Abbildung 8.2: Berechnung der hypothetischen Nachrichten [55].

Wenn der Client c ausführt und die Konflikt-Nachricht s1 erhält, berechnet xform s1'. Erzeugt der Server eine Nachricht s2 ohne zuvor c auszuführen, kann der Client nicht mehr xform(c,s2) ausführen, da c und s2 nicht im gleichen Startzustand erzeugt wurden.

Die Lösung: Wenn der Client s1' berechnet, muss er sich ebenfalls c' als Ergebnis von xform(c,s1') merken. Diese repräsentiert die **hypothetische Operation**, um den gemeinsamen Zustand zu erreichen. Wenn s2 eintrifft, kann der Client c' zur weiteren Berechnung von xform verwendet. xform (c',s2) = c'', s2'.

Die Operation s2 wird ausgeführt, um den Status (1,2) zu erreichen. Wenn der Server nun c lokal ausführt, erreicht er ebenso den Status (1,2). Wenn die Nachricht c noch nicht beim Server eingetroffen ist, muss der Client auch c'' speichern.

Die Speicherung dieser Operationen erfolgt in der Nachrichtenwarteschlange. Alle eingehenden Nachrichten werden somit gegen die noch nicht bestätigten eigenen Operationen in der Nachrichtenwarteschlange transformiert, um die Zwischenzustände zu berechnen.

8.3 n-Wege Jupiter Protokoll

Das Zwei-Wege-Protokoll von Jupiter wird zu einem n-Wege Protokoll ([54],[55]) erweitert, indem der Server für jeden Client den Zwei-Wege-Algorithmus umsetzt. Hierzu besitzt der Server für jeden Client einen Proxy, der das Gegenstück im Jupiter Protokoll repräsentiert.

Empfängt der Server eine Client-Nachricht, wird diese Operation in seinem entsprechenden Proxy als entfernte Operation empfangen und gegen die Operationen aus seiner Nachrichtenwarteschlange transformiert. Im Anschluss wird die transformierte Operation an die anderen Proxies übermittelt, dort als lokal generierte Operation gespeichert und an die entsprechende Client-Seite gesendet. Dieser Servervorgang muss atomar sein, d.h. er darf nicht durch das Eintreffen weiterer Client-Nachrichten unterbrochen werden.

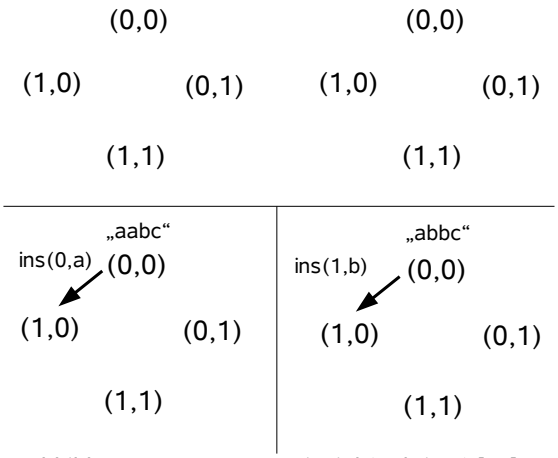

Abbildung 8.3: n-Wege Beispiel (Schritt 1) [55]

Beispiel: Die Funktionsweise der n-Wege-Adaption von Jupiter wird im folgenden Beispiel mit zwei Client-Seiten veranschaulicht. Im ersten Schritt in Abbildung 8.3 erzeugen beide Clients eine konkurrierende Operation auf demselben Dokumentenzustand und senden ihre Änderungsoperation als Nachricht an den Server. Die Übertragung der Nachricht erfolgt zeitversetzt über eine Breitbandkommunikation, so dass die Nachricht von Client 1 zuerst beim Server eintrifft.

Im zweiten Schritt in Abbildung 8.4 wird die eintreffende Nachricht beim Proxy des entsprechenden Clients transformiert und als Remote-Operation abgespeichert. Im Anschluss wird die transformierte Operation an die übrigen Proxies übertragen und dort als lokal ausgeführte Operation eingefügt und an Client 2 übertragen.

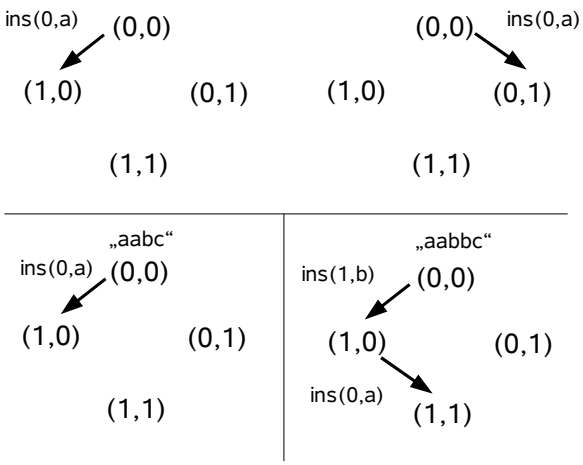

Abbildung 8.4: n-Wege Beispiel (Schritt 2) [55]

Empfängt Client 2 die Operation ins(0,a), transformiert er diese gegen seine lokalen Operationen und führt sie aus.

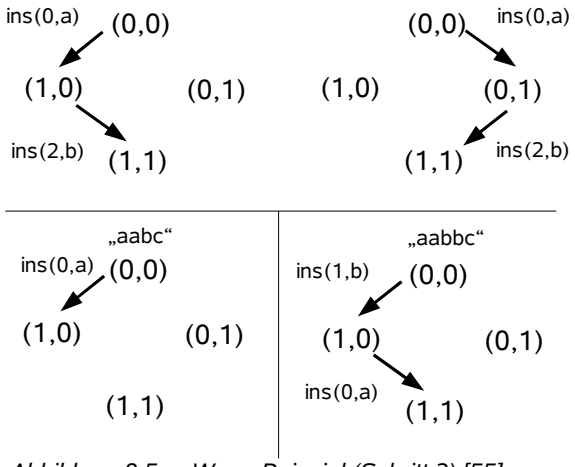

Abbildung 8.5: n-Wege Beispiel (Schritt 3) [55]

In Abbildung 8.5 empfängt der Server die zeitlich versetzte Nachricht von Client 2 und transformiert diese gegen die lokal ausgeführte Operation ins(0,a). Die transformierte Operation ins(2,b) überträgt er an die übrigen Proxies, die diese Operation als lokale Operation einfügen und an ihre entsprechenden Client-Seiten übertragen.

In Abbildung 8.6 empfängt Client 1 die Nachricht des Servers und transformiert sie gegen seine lokal ausgeführten Operationen. In diesem Fall kann er gleich die Operation ins(2,b) ausführen, da diese bereits vom Server transformiert wurde.

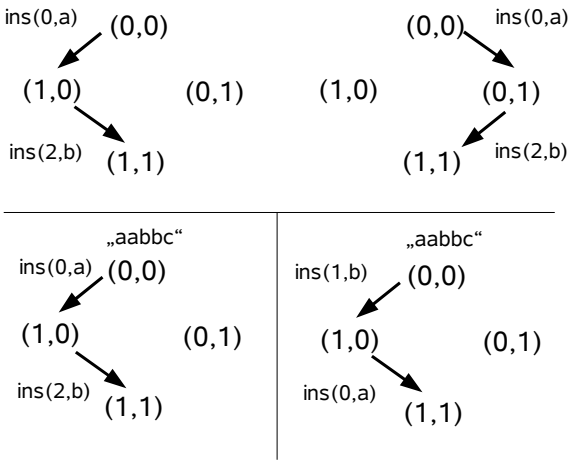

Abbildung 8.6: n-Wege Beispiel (Schritt 4) [55]

Beide Clients haben nun denselben Dokumentenzustand und befinden sich beide im Status (1,1), da je einer von beiden eine lokale Nachricht erzeugt und eine entfernte Nachricht erhalten hat.

8.4 Transformationsfunktion

Der ursprünglich dem Jupiter Algorithmus zugrunde liegende dOPT-Algorithmus verwendet eine Transformations-Matrix zur Berechnung der transformierten Operationen. Diese Transformationsfunktion ist auf die Berechnung der Transformation von einelementigen Operationen eingeschränkt. In Saros erfolgt die Übermittlung der Änderungen nicht nur einelementig, sondern aus Performance-Gründen werden Zeichenfolgen übertragen, so dass die einelementige Transformationsfunktion des dOPT Algorithmus nicht die notwendigen Anforderungen erfüllt.

Die Implementierung des GOTO Algorithmus basiert auf dem Pseudo-Code von Sun et. al [40] und wurde unter anderem bereits im ACE Projekt [56] als Transformationsfunktion innerhalb des Jupiter-Algorithmus eingesetzt. Dieser Algorithmus ermöglicht die Transformation von Zeichenfolgen und bietet sowohl eine Umsetzung für Inclusion Transformation als auch für Exclusion Transformation an. Da der Jupiter Algorithmus konzeptionell nur **Inclusion Transformation (IT)** benötigt, kann auf die Umsetzung der ET Funktion von GOTO verzichtet werden. Basierend auf dieser Umsetzung wurde die Transformationsfunktion des GOTO Algorithmus für Saros adaptiert.

8.4.1 Operationen

Operationen beschreiben die am Dokument durchgeführten Veränderungen und beinhalten sowohl den Inhalt der Änderungen, als auch die Position der durchgeführten Änderungen innerhalb des Dokuments. Die primitiven einelementigen Operationen für das Einfügen und Löschen von Zeichen aus Abschnitt 7.2 wurden zur Verarbeitung von Zeichenfolgen erweitert, um die Zeichenfolgen-basierten Änderungsinformationen in Saros verarbeiten zu können.

Änderungsoperationen

Für textuelle Änderungen an der lokalen Dokumentenkopie werden Einfüge- und Löschoperationen mit Zeichenfolgen als Eingabe-Parameter verwendet. Operationen des Ersetzens einer bestehenden durch eine neue Zeichenfolge werden durch die Kombination einer Lösch- und einer anschließenden Einfüge-Operation simuliert. Ergänzend zu diesen Änderungsoperationen existieren zusätzliche Hilfsoperationen, die in bestimmten Szenarien notwendig sind.

Zweiteilige Operation

Eine zweiteilige Operation (split operation) ist notwendig, wenn sich eine Einfüge- und eine Löschoperation dahingehend überlagern, dass die Einfügeoperation innerhalb des Löschbereichs durchgeführt. Soll diese Löschoperation im Anschluss an die Einfügeoperation ausgeführt werden, muss der Bereich vor, sowie der Bereich hinter der eingefügten Zeichenfolge gelöscht werden, um die Intention zu sichern.

Leere Operation

Bei der Transformation zweier Operationen kann eine Umwandlung in eine leere Operation erfolgen, wenn eine Änderung bereits von einem anderen Teilnehmer durchgeführt wurde. Die leere Operation ist notwendig, um den Status der Nachrichtenwarteschlange und den Vektor-Zustand zu aktualisieren.

8.4.2 Spezialfälle

Es existieren Spezialfällen [55], in denen die Transformationsfunktion nur auf der Grundlage der Positionsparameter keine eindeutige Transformation durchführen kann. Für den Spezialfall des Einfügens von Zeichen an dieselbe Position muss eine Erweiterung der Transformationsfunktion durchgeführt werden. Zwei Einfüge-Operationen (z.B. insert(0,a)

und insert(0,b)) können aufgrund ihrer identischen Positionen von der Transformationsfunktion nicht eindeutig auf der Client- und der Server-Seite ohne eine Priorisierung ausgeführt werden. Transformationsfunktionen müssen daher, unterstützt durch erweiterte Prioritätsregeln, Transformationen von Operationen auf gleichen Positionen durchführen. Angenommen die Transformationsfunktion besitzt eine Prioritätsregel, nachdem Zeichen absteigend alphabetischer Ordnung bevorzugt werden, kann die Intention nicht sichergestellt werden.

Das Szenario aus Abbildung 8.7 beschreibt ein Konfliktszenario, in dem die Transformationsfunktion keine eindeutige Transformation durchführen kann.

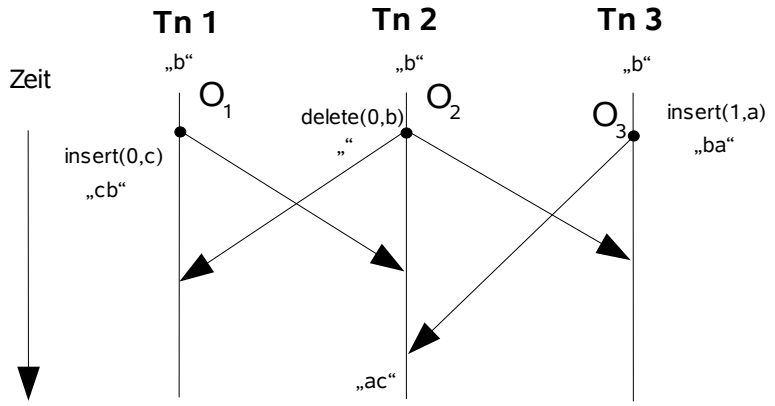

Abbildung 8.7: Spezialfall einer Transformation [55]

Alle Teilnehmer (Tn) haben den Dokumentzustand „b“ und führen lokale Operationen aus, die sie im Anschluss an die anderen Teilnehmer übertragen. Teilnehmer 2 führt nach dem Löschen des Zeichens „b“ die entfernte Operation O_1 und O_3 aus. O_3 wird aufgrund der Abhängigkeit zur Löschoperation transformiert (nach insert(0,a)), so dass im Anschluss die Operationen insert(0,c) und insert(0,a) transformiert werden müssen. Als Ergebnis der Transformation mit der Priorisierung nach absteigend alphabetischer Ordung wird O_1 transformiert und Teilnehmer 2 erhält den Dokumentzustand „ac“. Dieser Dokumentzustand verletzt die Intention der Urheber.

Zur Sicherung der Intention werden Einfüge- und Löschoperationen um einen zusätzlichen dritten Parameter erweitert, der die ursprüngliche Position bei der Erzeugung der Operation beinhaltet und durch Transformationen nicht verändert. Die zuvor falsche Transformation erfolgt nun unter Zuhilfenahme der ursprünglichen Positionen, so dass die Operationen

insert(0,c,0) und insert(0,a,1) nun korrekt unter Einbeziehung der ursprünglichen Position korrekt transformiert werden.

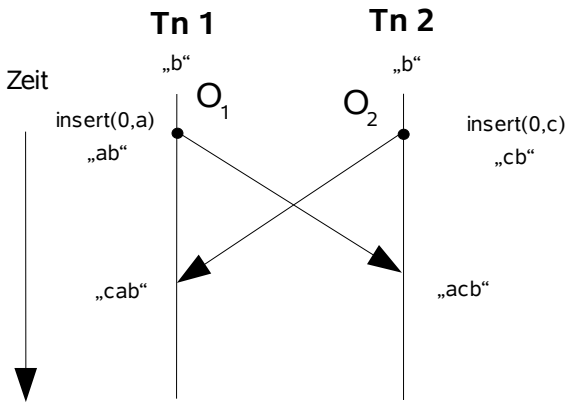

Abbildung 8.8: Spezialfall 2 einer Transformation

Ein weiterer Spezialfall zweier konkurrierender Einfügeoperationen an derselben Position ist in Abbildung 8.8 dargestellt. Ohne eine Prioritätsregel der Transformationsfunktion würde es zu einem inkonsistenten Dokumentzustand kommen, da die Operation O_1 und O_2 nicht eindeutig transformiert werden können. Um eine Eindeutigkeit in diesem Szenario zu gewährleisten, bevorzugt die Transformationsfunktion die Server-Seite. Die vom Server erzeugten Operationen werden bevorzugt behandelt, so dass die Eindeutigkeit der Transformationen gewährleistet werden kann.

8.5 Umsetzung des Jupiter Algorithmus

Im vorherigen Abschnitt wurde die Funktionsweise des Jupiter-Algorithmus vorgestellt. Im folgenden Abschnitt wird die Umsetzung des Jupiter-Algorithmus erläutert.

8.5.1 Jupiter Request

Ein Jupiter-Request bildet die Grundlage für den Datenaustausch der Änderungsoperationen zwischen dem Jupiter-Client und dem Jupiter-Server. Die verschiedenen Änderungsoperationen (siehe Abschnitt 8.4.1) werden durch die Operation-Objekte der Schnittstelle `Operation` realisiert [56].

`Operation`: Zusätzlich zu den Änderungs-Operationen `InsertOperation` und `DeleteOperation` existieren die Hilfsoperationen `NoOperation` und `SplitOperationen`.

Request: Um die Änderungen an Dokumenten übertragen und transformieren zu können, werden alle benötigten Informationen durch ein `Request`-Objekt [56] repräsentiert. Es beinhaltet wie in Abbildung 8.9 dargestellt, die Operationen, die Vektor-Nummer und die Seiten-Information (Server- oder Client-Seite) zur Priorisierung der Transformation.

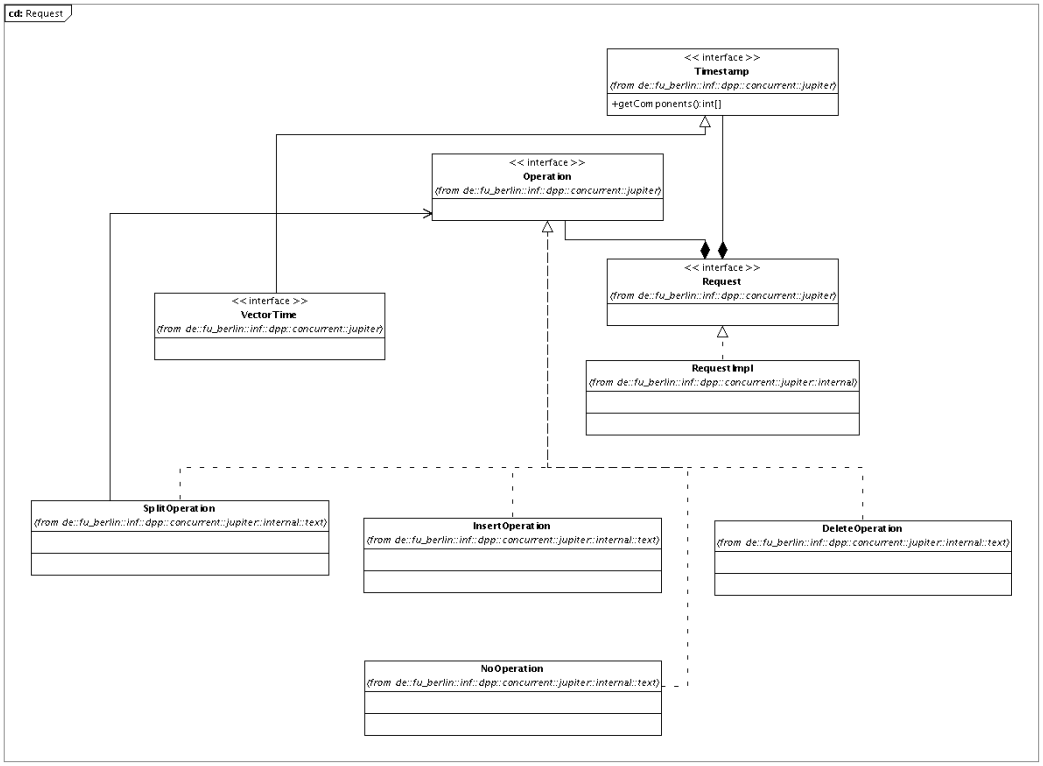

Abbildung 8.9: Klassendiagramm der Jupiter-Requests

8.5.2 Grundaufbau des Algorithmus

Die Umsetzung des Jupiter-Algorithmus konzentriert sich um die Klasse Jupiter [56] herum, die die Schnittstelle `Algorithm` zur Synchronisation der lokalen und entfernten Operationen implementiert. Das Klassendiagramm in Abbildung 8.10 gibt einen Überblick über die wichtigsten Klassen des Algorithmus und der Transformationsfunktion.

Jupiter: Die Jupiter Klasse implementiert die Schnittstelle `Algorithm` und bildet die zentrale Klasse des Jupiter Algorithmus. Alle `Request`-Instanzen werden über die `InclusionTransformation` Schnittstelle berechnet.

GOTOInclusionTransformation: Die Transformationsfunktion des Jupiter Algorithmus wird durch eine Implementierung des Pseudo-Codes des GOTO Algorithmus [40] für

Inclusion Transformation in der Klasse `GOTOInclusionTransformation` realisiert und implementiert die Schnittstelle `InclusionTransformation` [56].

Erzeugen von Jupiter-Requests

Jupiter berechnet für eine lokal erzeugte Operation über den Aufruf der Methode `generateRequest(Operation op)` einen Jupiter-Request, der im folgenden an die entsprechende Gegenseite kommuniziert wird. Nach dem Erzeugen des Jupiter-Request wird der lokale Operations-Zähler der Vektor-Nummer inkrementiert und die Operation der lokalen Nachrichten-Warteschlange hinzugefügt.

Empfang entfernter Jupiter-Requests

Beim Empfang einer entfernten Operation durch den Empfang eines Jupiter Request wird die lokal auszuführende Operation durch den Aufruf der Methode `receiveRequest(Request req)` wie folgt ermittelt.

1. Auf der Basis der Vektor-Nummer des empfangenen Request werden die von der Gegenseite bestätigten Operationen aus der Nachrichtenwarteschlagen entfernt.

2. Die Transformationsfunktion berechnet die lokal auszuführende Operation durch Transformation der empfangenen Operation mit jeder noch verbliebenen Operation der Nachrichten-Warteschlange. Nach Abschluss der Transformationen wird der entfernte Operations-Zähler der Vector-Nummer inkrementiert.

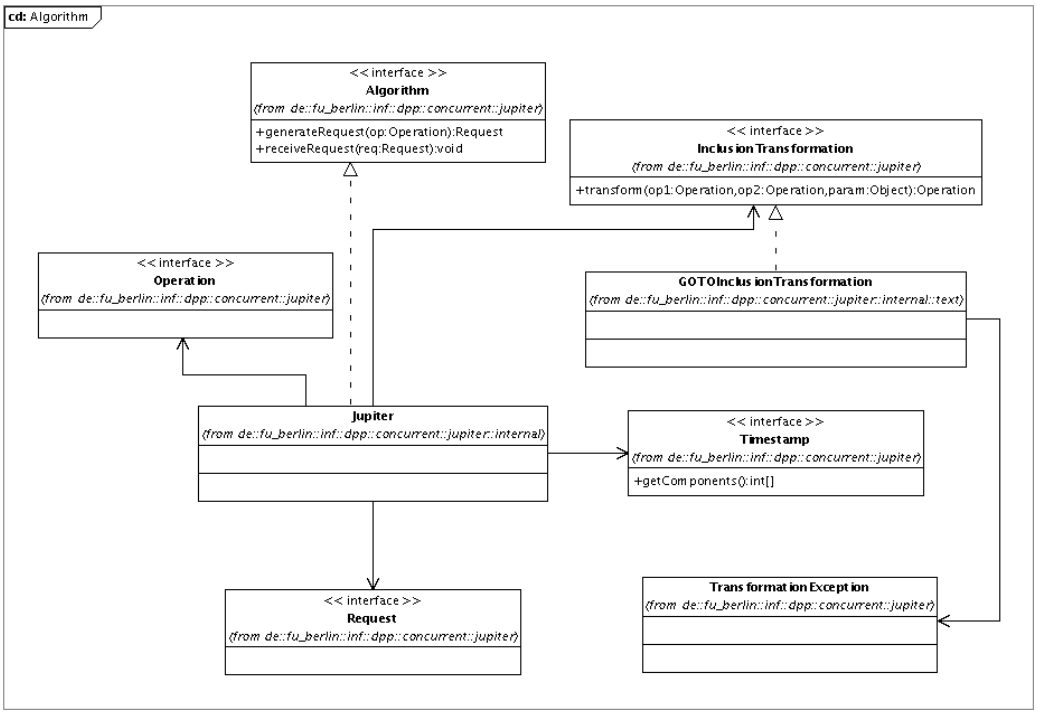

Abbildung 8.10: Klassendiagramm des Jupiter Algorithmus

8.5.3 Jupiter Client

Jedes lokale Dokument, welches sich in einem nebenläufigen Bearbeitungsprozess befindet, wird durch eine Instanz der Klasse `JupiterDocumentClient` repräsentiert und realisiert innerhalb des Jupiter-Ansatzes die Client-Seite des Jupiter-Protokolls.

Erzeugen lokaler Änderungen

Nach der lokalen Ausführung einer Änderungsoperation in der grafischen Komponente des Systems wird ein entsprechender Jupiter-Request durch den Aufruf der Methode `generateRequest(Operation op)` der `JupiterClient`-Instanz des Dokuments erzeugt. Innerhalb dieser Klasse wird der erzeugte `Request` zur Weiterleitung an die Netzwerk-Komponente, eine Instanz der Klasse `RequestForwarder`, übergeben.

Ausführen entfernter Änderungen

Entfernte Änderungen werden als empfangene Request-Instanzen an den `JupiterDocumentClient` übergeben. Innerhalb dieser Klasse wird die auszuführende

transformierte Operation erzeugt, die im Anschluss auf der lokalen Kopie des entsprechenden Dokuments ausgeführt werden kann.

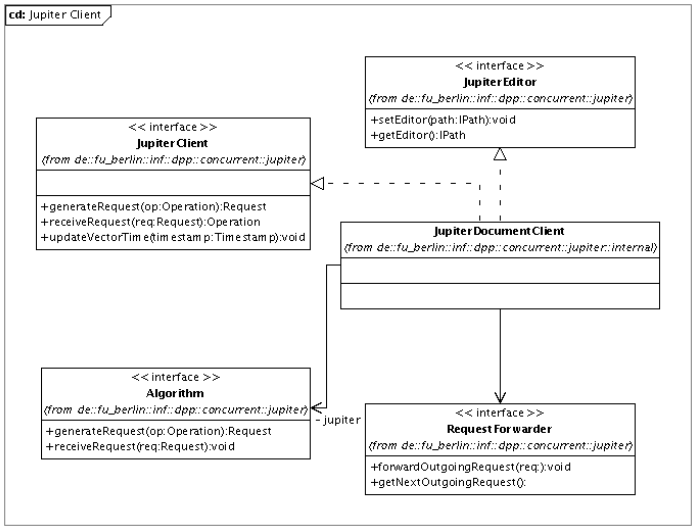

Abbildung 8.11: Klassendiagramm der Jupiter-Client-Komponente

8.5.4 Jupiter Server

Die Jupiter-Server-Komponente `JupiterDocumentServer` eines Dokuments realisiert eine n-Wege Kommunikation auf Basis von n-vielen Zwei-Wege Synchronisations-Protokollen. Jeder Teilnehmer (bzw. Client) des Bearbeitungsprozesses wird durch eine `ProxyJupiterDocument` Instanz repräsentiert, die die Server-Seite des Jupiter-Protokolls implementiert. Wird ein neuer schreibender Teilnehmer für das entsprechende Dokument im System angemeldet, wird ein neuer Jupiter-Proxy für diesen Teilnehmer erzeugt und entsprechend nach Abmelden des Teilnehmers aus der Server-Komponente entfernt.

Wie im Abschnitt 8.3 dargelegt, muss die Bearbeitung eines Requests innerhalb des Servers atomar erfolgen und darf nicht durch weitere eintreffende Requests unterbrochen werden. Dies wird zum einen durch Warteschlangen für eintreffende und zu versendende Requests und zum anderen durch die Klasse `Serializer` erreicht, die einen Request störungsfrei bearbeitet.

1. Eintreffende Requests werden in die eingehende Warteschlange des `JupiterDocumentServers` eingefügt, welche durch die Schnittstelle `SynchronizedQueue` repräsentiert wird.

2. Wurde ein neuer Request der `SynchronizedQueue` hinzugefügt, startet der Bearbeitungsprozess im Serializer. Der Request wird mit der Proxy-Komponente des entfernten Clients synchronisiert und im Anschluss als serverseitig erzeugte Operation zur Synchronisation an alle verbliebenen Proxies übergeben.

3. Jedes `ProxyJupiterDocument` erzeugt einen Jupiter-Request zur Ausführung der Änderungsoperation auf Client-Seite.

4. Die erzeugten Requests der `ProxyJupiterDocument` Instanzen für die entsprechenden Clients werden über die Schnittstelle `RequestForwarder` der Warteschlange für zu versendende Request-Nachrichten hinzugefügt.

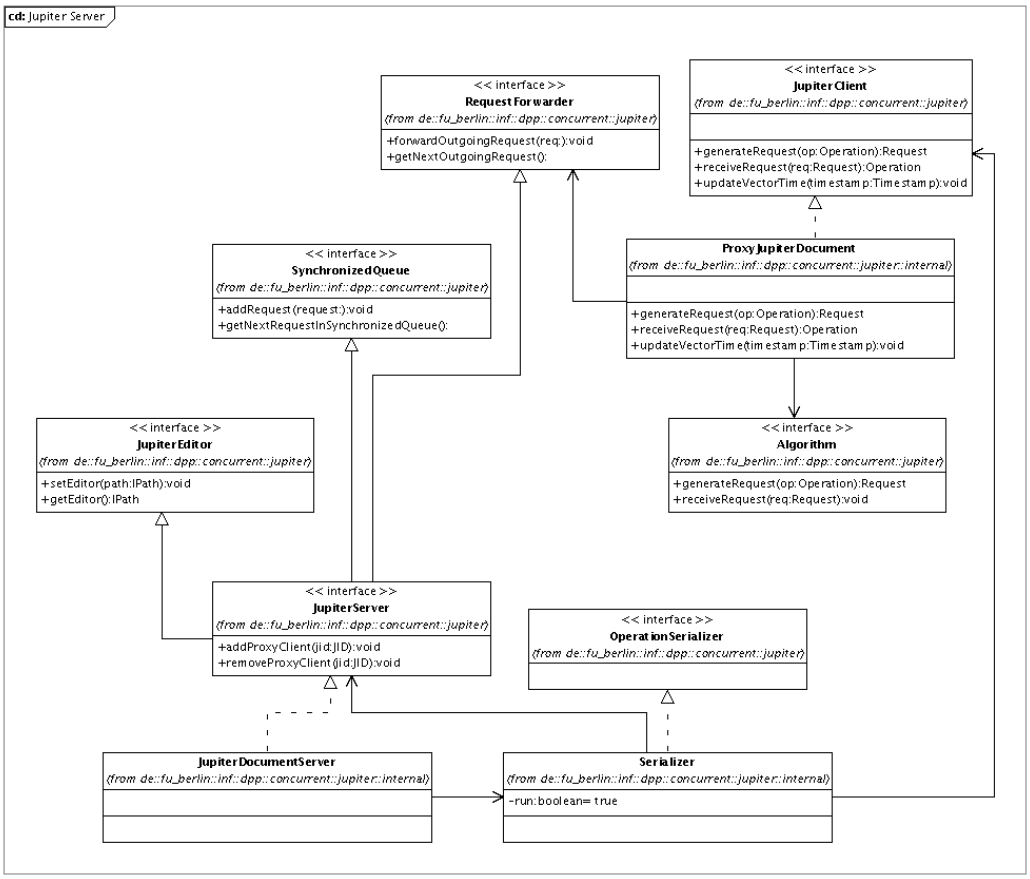

Abbildung 8.12: Klassendiagramm der Jupiter-Server-Komponente

8.6 Test Szenarien

Um die im vorherigen Abschnitt beschriebene Umsetzung des Jupiter Algorithmus zu testen, wurden die beschriebenen Komponenten in ein auf JUnit[4] basierendes Test-Framework integriert, welches ein Netzwerk für die Übertragung der Nachrichten zwischen Client und Server simuliert. Die Clients melden sich am Jupiter-Server an und erhalten die Nachrichten der anderen Clients als vom Server transformierte Nachrichten über das simulierte Netzwerk. Nach dem Empfang führen sie die Nachrichten lokal aus.

Während des Sende-Vorgangs einer Client-Operation kann eine Verzögerung der Übertragung über die zusätzliche Angabe eines Verzögerungswertes simuliert werden. Hierdurch kann die Reihenfolge der Ausführung der Operationen gesteuert werden und der zeitlich versetze Empfang von Nachrichten simuliert werden. Dies ermöglicht die Umsetzung unterschiedlicher Test-Szenarien.

8.6.1 Test Szenarien für das 2-Wege Jupiter Protokolls

Die folgenden Testfälle beschreiben alle Kombinationen von Einfüge- und Löschoperationen zweier gleichzeitig ausgeführter Operationen O_1 und O_2. Die Operation O_1 wird in diesen Testfällen vor der Operation O_2 ausgeführt. In allen aufgeführten Testfällen konnte die Umsetzung des 2-Wege Jupiter Protokolls das Konsistenz-Modell im vollen Umfang erfüllen. Die Testfälle für die Inclusion Transformation [55] wurden im JUnit-Test `InclusionTransformationTest` umgesetzt.

Einfüge Operationen

Testfall 1: O_1 fügt ein Zeichen (oder eine Zeichenfolge) vor oder an derselben Stelle, wie die Operation O_2 ein. Die Transformation ist in diesem Fall nicht nur von der Position abhängig, sondern die Transformation ist sowohl von den ursprünglichen Positionen, als auch der jeweiligen Erzeuger-Seite abhängig. (Siehe hierzu Abschnitt 8.4.2).

Testfall 2: O_1 fügt ein Zeichen (oder eine Zeichenfolge) innerhalb oder hinter dem Bereich der Operation O_2 ein. Operation O_1' muss um Länge des eingefügten Bereichs von O_2 inkrementiert werden.

[4] ***JUnit*** *ist ein Framework zum Testen von Java-Programmen, das besonders für automatisierte Unit-Tests einzelner Units (meist Klassen oder Methoden) geeignet ist.[57]*

Einfüge und Lösch-Operationen

Testfall 3: Einfüge-Operation O_1 fügt ein Zeichen an die Position vor der Lösch-Operation O_2 ein.

Testfall 4: Einfüge-Operation O_1 fügt ein Zeichen (oder eine Zeichenfolge) hinter der Lösch-Operation O_2 ein. Die Position von O_1' muss um den Bereich von O_2 dekrementiert werden.

Testfall 5: Einfüge-Operation O_1 fügt ein Zeichen (oder Zeichenfolge) innerhalb des Bereichs der Lösch-Operation O_2 ein. Die Operation O_1' muss dekrementiert werden.

Lösch- und Einfüge-Operationen

Testfall 6: O_1 löscht einen Bereich vor O_2.

Testfall 7: Lösch-Operation O_1 löscht an der selben Position wie Operation O_2. Die Position der Löschoperation O_1' muss um die Länge des eingefügten Bereichs von O_2 inkrementiert werden.

Testfall 8: Einfüge-Operation O_2 befindet sich innerhalb des Bereichs der Lösch-Operation O_1. Die Löschoperation O_1' muss daher aufgeteilt werden, um die zuvor ausgeführte Einfüge-Operation nicht aufzuheben und die Intention sicherzustellen. (Siehe Zweiteilige Operation im Abschnitt 8.4.1)

Lösch- / Lösch-Operationen

Testfall 9: O_1 löscht einen Bereich hinter O_2. Die Position von O_1' muss um den gelöschten Bereich von O_2 dekrementiert werden.

Testfall 10: Die Löschoperationen O_1 und O_2 überlagern sich. O_2 beginnt an einer Position vor oder an derselben Position wie O_1 und endet hinter dem Bereich von O_1. O_1 wird nicht mehr ausgeführt, da der Bereich bereits von O_2 gelöscht wurde, stattdessen wird eine leere Operation erzeugt.

Testfall 11: Die Löschoperationen O_1 und O_2 überlagern sich. O_2 startet vor oder an der selben Position wie O_1 und endet vor O_1. Der überlagernde Teil wird von O_2 gelöscht, während O_1' nur den restlichen Teil hinter dem Bereich von O_2 löscht.

Testfall 12: Die Löschoperationen O_1 und O_2 überlagern sich. O_2 beginnt innerhalb des Bereichs von O_1 und endet hinter oder an der gleichen Position wie O_1. Der überlagernde Teil wird von O_2 gelöscht, so dass O_1' nur noch den verbleibenden Teil löscht.

Testfall 13: Die Löschoperationen O_1 und O_2 überlagern sich. Der Bereich von O_2 befindet sich innerhalb des Bereichs von O_1. Der überlagernde Teil beider Operationen wird von O_2 gelöscht. O_1' löscht daher nur noch den Bereich vor und hinter O_2.

8.6.2 Test-Szenarien für das n-Wege Jupiter Protokoll

Die in vielen Arbeiten behandelte Problematik der Korrektheit von Transformationsfunktionen und der Konsistenzsicherung in verteilten Echtzeit-Systemen bildet die Grundlage für die folgenden Test-Szenarien. Diese Test-Szenarien beinhalten komplexe Probleme, die für eine Vielzahl von Algorithmen als Gegenbeispiel für die Korrektheit verwendet werden [55]. In sämtlichen aufgeführten Test Szenarien konnte die Einhaltung des Konsistenz-Modells sichergestellt werden.

Counter Szenarien

Aus der Veröffentlichung „Proving Correctness of Transformation Functions in Real-Time Groupware“ [59] wurden die Szenarien aus den Abbildung 3, 4 und 5 als Testfälle (`CounterExampleTest`) umgesetzt.

Convergence Problems

Aus der Veröffentlichung „Achieving Convergence with Operational Transformation in Distributed Groupware Systems“ [60] wurden die Szenarien „C2 puzzle P1“ und „C2 puzzle P2“ als Testfälle (`ConvergenceProblemTest`) umgesetzt.

dOPT Puzzle

Aus der Veröffentlichung „Operational Transformation in Real-Time Group Editors: Issues, Algorithms, Achievements“ [49] wurde das „dOPT puzzle“ Szenario als Testfall (`DoptPuzzleTest`) umgesetzt.

9. Umsetzung einer Mehr-Schreiber-Funktionalität

Für einen konkurrierenden Schreibzugriff in einer Mehr-Schreiber-Umgebung sind Konflikterkennung und –behebung notwendig, um die Konsistenz der Projektdateien bei allen Projektteilnehmern sicherstellen zu können. Die Auswahl und Umsetzung eines geeigneten Ansatzes der Konsistenzsicherung für Saros wurden in den vorherigen Kapiteln begründet und dargelegt. Im folgenden Kapitel wird auf die Umsetzung einer Mehr-Schreiber-Funktionalität eingegangen und die konzeptionellen Ansätze sowie die Integration in die Eclipse-Erweiterung Saros beschrieben.

9.1 Kommunikationsstruktur

Einer der zentralen Punkte für die Integration einer Mehr-Schreiber-Funktionalität ist die Integration einer Konsistenzsicherung in das bestehende System. Hierfür ist eine Anpassung des Benachrichtigungskonzepts an die neuen Anforderungen notwendig.

An dieser Stelle wird der Begriff der ***konfliktfreien Operation*** eingeführt, um zwischen den Operationen, die einer Konsistenzsicherung unterliegen, und denen, die ohne weitere Überprüfung kommuniziert werden können, zu unterscheiden. Sie haben keinen Einfluss auf die Konsistenz der Dokumentenstände oder der Projektstruktur.

Nicht konfliktfreie Operationen betreffen alle Veränderung am Dokumentzustand durch Einfügen und Entfernen von Zeichen, als auch Veränderungen an der Projektstruktur durch Einfügen, Löschen, Umbenennen oder Verschieben von Dateien oder Ordner bzw. Paketen.

In der bisherigen Umsetzung bestand keine Notwendigkeit, Konflikte zwischen gleichzeitig ausgeführten Operationen mehrerer Teilnehmer erkennen zu müssen, da auf Grund des exklusiven Schreibzugriffs des Drivers alle nicht konfliktfreien Operationen nur vom einem Driver ausgeführt werden konnten. Im Anschluss informierte der Driver alle Observer nacheinander über die ausgeführten Operationen. Die Observer informieren ihrerseits alle anderen Teilnehmer nacheinander über ihre Aktivitäten. Da es sich hierbei jedoch um konfliktfreie Operationen handelt, die lediglich Awareness-Informationen des jeweiligen Observers beinhalten und keinen Einfluss auf die Bearbeitung durch andere Projekt-Teilnehmer haben, können sich diese Nachrichten gefahrlos zeitlich überlagern. Der Verlust von Nachrichten der konfliktfreien Operationen hat keinen Einfluss auf das Konsistenz-Modell und bedarf daher keiner gesonderten Absicherung.

9.1.1 Zentrale Architektur

Die im vorherigen Abschnitt identifizierten nicht konfliktfreien Operationen müssen über eine Konsistenzsicherung gegen eine Verletzung des Konsistenz-Modells abgesichert werden und dürfen in einer Mehr-Schreiber-Umgebung nicht ohne Synchronisation durch die Konsistenzsicherung an alle Teilnehmer kommuniziert werden. Das Vorgehen des Kommunizierens dieser Änderungsoperationen ist von dem zum Einsatz kommenden Verfahren der Konsistenzsicherung abhängig. Die Integration einer Jupiter-basierten Konsistenzsicherung bedingt eine zentrale Server-Instanz für die Synchronisation der Änderungsoperationen. Um die zentrale Server-Instanz der Jupiter-Server-Komponente in die bestehende Eclipse-Erweiterung zu integrieren, wird die bisherige Rolle des Projekt-Hosts erweitert.

In der bisherigen Umsetzung sah die Rolle des Projekt-Host folgende zusätzliche Eigenschaften vor:

Einladen weiterer Teilnehmer: Nach dem Projekt-Start der gemeinsamen Projekt-Sitzung obliegt es dem Projekt-Host, weitere Teilnehmer zu der bestehenden Sitzung einzuladen.

Rollen-Vergabe: Er besitzt zudem das Recht die Driver-Rolle an Teilnehmer der Sitzung zu vergeben und wieder zu entfernen. Während der Vergabe die Rolle des Drivers an einen anderen Teilnehmer der Sitzung fungiert der Projekt-Host als Observer, kann sich die exklusive Driver Rolle jedoch zu jedem Zeitpunkt wieder zuweisen.

Die bisherigen Aufgaben der Koordination der Projekt-Teilnehmer und deren Rollen durch den Projekt-Host wird um die Rolle der Jupiter-Server-Instanz erweitert. Nicht konfliktfreie Nachrichten werden von den anderen Teilnehmern mit der Rolle Driver nicht an alle Teilnehmer kommuniziert, sondern zur Synchronisation der nebenläufigen Bearbeitung der Dokumente an den Projekt-Host gesendet. Der Projekt-Host kommuniziert nach Synchronisation der Änderungsoperationen eine konsistente Operationsausführung mit allen Observern der Sitzung.

Erweiterte Rollen-Vergabe: Das bisherige Recht des Projekt-Hosts der Rollen-Vergabe wird erweitert. Der Projekt-Host hat nun die Möglichkeit jeden Teilnehmer der Sitzung beliebig die Rolle Driver oder Observer, unabhängig von den vorkommenden Rollen der anderen Teilnehmer, zuweisen. Dies ermöglicht eine beliebige Anzahl von

Drivern innerhalb der Projektsitzung. Als Erweiterung besitzt der Projekt-Host die Möglichkeit alle aktuellen Driver-Rollen zu entfernen und den entsprechenden Teilnehmern die Rolle Observer zuzuweisen.

9.1.2 Multi-User-Chat Protokoll

In der bisherigen Umsetzung basiert die Kommunikation auf den sogenannten „Chatthread“ [6] für die Kommunikation zwischen zwei Teilnehmern. Um die Nachrichten-Kommunikation an die steigende Anzahl der Sitzungsteilnehmer anzupassen, in der nicht jeder Projekt-Teilnehmer seine konfliktfreien Operationen oder Aktivitäten einzeln an jeden anderen Teilnehmer versendet, wird Saros um das Multi-User-Chat Protokoll XEP-0045 [58] erweitert.

Ein Jabber-Server stellt für dieses Protokoll einen virtuellen **Jabber-Conference-Room (JCR)** bereit. Dieses Protokoll ermöglicht es Nachrichten gleichzeitig an alle Teilnehmer innerhalb des JCR zu versenden. Eine Historie-Funktionalität des Protokolls ermöglicht es zeitlich zurückliegende Nachrichten auszulesen und bietet darüber hinaus ein Rechte- und Rollenkonzept der Teilnehmer.

Der Projekt-Host erzeugt zu Beginn einer Projekt-Sitzung einen Jabber-Confercence-Room zur Kommunikation der Projekt-Aktivitäten und registriert sich automatisch als Teilnehmer. Während des Beitritts zur aktuellen Projektsitzung meldet sich jeder Teilnehmer in diesem JCR an. Sendet ein Teilnehmer nun eine konfliktfreie Operation bzw. Aktivität an den JCR, wird diese automatisch an alle angemeldeten Teilnehmer versandt. Jede Nachricht des JCR beinhaltet als zusätzliche Information den Urheber zur eindeutigen Identifizierung.

Jeder Teilnehmer sendet seine konfliktfreien Operationen an den JCR und nicht mehr einzeln an alle Teilnehmer. Es kann bei konfliktfreien Operationen ebenfalls zu geringfügigen Konflikten kommen, die jedoch nur den Zustand der Anzeige der Awareness-Informationen betreffen können, so dass z.B. Markierungen für einen kurzen Zeitpunkt falsch angezeigt werden, wenn sie in der falschen Reihenfolge eintreffen. Diese Awareness-Informationen dienen der Kommunikation zwischen den Projekt-Teilnehmern und beeinflussen nicht die Konsistenz der Projekt-Dateien. Aus diesem Grund wird von einer Konflikterkennung und -behebung dieser Operationen abgesehen.

Nach Überprüfung und ggf. Korrektur der kritischen Operationen der Driver durch die Konfliktsicherung und anschließender lokalen Ausführung beim Projekt-Host, werden diese nun konfliktfreien Operationen ebenfalls an alle Projekt-Teilnehmer übermittelt.

9.2 Konsistenzsicherung

Die Konflikterkennung und -behebung werden durch eine optimistische Nebenläufigkeitskontrolle realisiert, deren Kommunikationsstruktur auf einer Client-Server-Architektur aufbaut. Die Jupiter-basierte Konsistenzsicherung ermöglicht einen vertretbaren Kommunikationsaufwand für die Synchronisation der nebenläufigen Änderungsoperation, da jeder Driver seine Operationen nur direkt an den Jupiter-Server[5] sendet. Der Jupiter-Server kommuniziert die Änderungsoperationen an alle weiteren Clients[6].

9.2.1 Jupiter-basierte Konsistenzsicherung

Das Grundkonzept der Jupiter-basierten Konsistenzsicherung besteht aus der sofortigen lokalen Ausführung der erzeugten Operationen. Erst nach der Ausführung werden die Operationen an den Jupiter-Servers übermittelt. Wie im vorherigen Abschnitt bereits beschrieben, erfolgt die serversseitige Konflikterkennung beim Projekt-Host. Des Weiteren besitzt jeder Driver eine eigene Jupiter-Client-Instanz, die der Synchronisation der erzeugten Operationen mit dem Jupiter-Server und zur Ausführung der entfernten Operationen anderer Driver dient.

Im Anschluss an eine Operationsausführung wird eine konfliktfreie Abfolge der Operationen vom Projekt-Host an alle Teilnehmer versendet. Jede vom Projekt-Host kommunizierte Operation eines Drivers beinhaltet als zusätzliche Information den ursprünglichen Urheber zur eindeutigen Identifizierung.

Somit wird eine doppelte Ausführung einer Operation bei einem beteiligten Driver vermieden, da jeder Driver anhand der Urheber-Identifizierung erkennen kann, ob diese Operation bereits lokal ausgeführt wurde.

Integration des Jupiter Algorithmus

Die Umsetzung des im Kapitel 8 beschriebenen Jupiter-Algorithmus wird als Verfahren zur Konsistenzsicherung in die bestehende Eclipse-Erweiterung integriert.

`IActivityProvider` Die Klasse `IActivityProvider` regelt das Erzeugen und Verarbeiten von `IActivity`-Exemplaren, das Verhalten bzw. die ausgeführte Operation beinhaltet. [6]

`AcitivitySequencer` Der `ActivitySequencer` [6] implementiert die Schnittstelle `IActivityManager` und empfängt daher alle erzeugten Aktivitätsexemplare und

[5] 1:1-Beziehung für die Kommunikation auf Client-Seite

[6] 1:N-Beziehung für die Kommunikation auf Server-Seite

sammelt diese zum Versenden an alle Teilnehmer in einer Liste. Das `ISharedProject` repräsentiert die laufende Projektsitzung und leert durch Aufruf der `flush`()-Methode die Liste der aufgelaufenen Aktivitäten, um sie im Anschluss an die Netzwerkkomponente weiterzuleiten. Gleichzeitig implementiert diese Klasse die Schnittstelle `IActivitySequencer` und erfüllt die Rolle der Aktivitätsablaufsteuerung. Eintreffende Aktivitäten werden von der Netzwerkkomponente an die Klasse `ActivitySequencer` weitergeleitet.

Die Klasse `ActivitySequencer` bietet den Ansatzpunkt für die Integration einer Konsistenzsicherung innerhalb des bestehenden Systems, da sie die Schnittstelle zwischen Netzwerkkomponente und Oberfläche darstellt. Nach Ausführung einer lokalen Aktivität wird das erzeugte Aktivitätsexemplar an diese Klasse weitergeleitet. Konfliktfreie Operationen werden an die Netzwerkkomponente weitergeleitet, wohingegen Änderungsoperationen am Dokumentenzustand gesondert über die Konsistenzsicherung behandelt werden müssen. Gleichzeitig werden alle eintreffenden Operationen entfernter Teilnehmer von der Netzwerkkomponente an diese Klasse weitergeleitet, um sie an die entsprechenden `IActivityProvider` zu verteilen.

`ConcurrentManager` Die Klasse `ConcurrentManager` verwaltet sowohl die Jupiter-Client-[7], als auch die Jupiter-Server-Komponente[8] und bildet die Schnittstelle für die Konsistenzsicherung. Weiterhin implementiert diese Klasse die Schnittstelle `ISharedProjectListener`, um auf Ereignisse der Projektsitzung, wie Rollenwechsel und Verlassen von Teilnehmer, reagieren zu können.

`IRequestManager` Diese Schnittstelle empfängt Jupiter-Request-Nachrichten für die Synchronisation in den Jupiter-Client- und Jupiter-Server-Instanzen.

Die Klasse `IActivitySequencer` wird um die Schnittstelle `RequestForwarder` erweitert, um Jupiter-Request-Nachrichten an die Netzwerkkomponente weiterleiten zu können. Die Basis für die Kommunikation zwischen Jupiter-Client und Jupiter-Server bilden Jupiter-Request-Nachrichten, die Informationen über die durchgeführten Operationen, die Vektor-Nummer zur Berechnung der Transformationen und die Seiten-Identifikation beinhalten. Hierzu wurde das Konzept der Übertragung von Aktivitäten erweitert, um getrennt von den konfliktfreien Operationen bzw. Aktivitäten, Jupiter-Request-Nachrichten zu versenden.

[7] Nähere Informationen zur Jupiter-Client-Komponente unter Abschnitt 8.5.3

[8] Nähere Informationen zur Jupiter-Server-Komponente unter Abschnitt 8.5.4

Die Schnittstelle `ITransmitter` wurde um die Methoden `processRequest` und `sendJupiterRequest` erweitert, die ein Jupiter-Request-Objekt[9] über die Umsetzung einer XMPP-Paketerweiterung für Jupiter-Requests als XMPP-Nachricht versenden. Die Paketerweiterung wurde entsprechend in den Klassen `RequestPacketExtension` und `RequestExtensionProvider` umgesetzt.

Starten und Stoppen der Jupiter-Komponenten

Die Projekte der Programmsitzung können hunderte oder tausende Dateien beinhalten, die zu jedem Zeitpunkt von mehreren Drivern bearbeitet werden könnten. Jedes Dokument muss in der Konsistenzsicherung durch einen Jupiter-Server und einen entsprechende Jupiter-Client-Instanzen bei den Drivern vertreten sein. Um nicht im Vorfeld alle Dokumente beim Start der Projektsitzung mit den entsprechenden Jupiter-Komponenten zu initialisieren, werden Dokumente erst bei der Bearbeitung durch mehrere Driver der Konsistenzsicherung hinzugefügt.

Beim Öffnen eines neuen Dokuments im Eclipse-Editor durch einen Driver wird ein lokaler Jupiter-Client beim Driver initialisiert, der für eine Änderungsoperation einen Jupiter-Request erzeugt, um die Änderungsoperationen mit dem Jupiter-Server des Projekt-Hosts zu synchronisieren. Gleichzeitig wird der Jupiter-Proxy des Drivers dem Jupiter-Server des Dokuments auf Seiten des Projekt-Hosts hinzugefügt. Sollte noch kein Jupiter-Server für dieses Dokument existieren, wird es entsprechend im ConcurrentManager des Projekt-Hosts angelegt und zusätzlich zum Driver wird für den Projekt-Host ein Jupiter-Proxy am Jupiter-Server angemeldet.

Beim Schließen eines Dokuments im Eclipse-Editor des Drivers, wird dieser aus der Konsistenzsicherung des Dokuments entfernt. Hierzu beendet der Driver seinen Jupiter-Client des Dokuments. Gleichzeitig entfernt der Projekt-Host den Jupiter-Proxy des Drivers aus dem Jupiter-Server. Der Driver empfängt Änderungsoperationen für dieses Dokument nicht mehr als Jupiter-Request-Nachrichten, sondern führt die Aktivitätsexemplare aus, die vom Projekt-Host an den JCR des Projekts gesendet werden.

Dieses Vorgehen reduziert den Kommunikationsaufwand zwischen dem Projekt-Host und den Drivern der Projektsitzung. Zusätzliche Jupiter-Request-Nachrichten werden nur an die Driver versendet, die die betreffende Datei aktiv bearbeiten.

[9] Nähere Informationen unter Abschnitt 8.5

Erzeugen und Senden von Jupiter-Request-Nachrichten

Änderungsoperationen an Dokumentenzuständen werden, wie in den folgenden Schritten beschrieben, durch die Konsistenzsicherung als Jupiter-Request-Nachrichten zu den entsprechenden Gegenstellen gesendet:

1. Alle `IActivityProvider` melden erzeugte Aktivitätsexemplare der durchgeführten Operationen dem `ActivtiySequencer` durch Aufruf der `activityCreated`-Methode. Operationen, die Änderungen an Dokumentenzuständen betreffen, werden nicht wie bisher an alle Teilnehmer kommuniziert, sondern innerhalb dieser Methode an die Konsistenzsicherung weitergeleitet. Dies erfolgt durch Aufruf der `activityCreated`-Methode der Klasse `ConcurrentManager`.

2. Der `ConcurrentManager` wandelt die `IActivity` in eine Instanz der Klasse `Operation` um und übergibt die Operation an den Jupiter-Client des Dokuments durch Aufruf der `generateRequest`-Methode.

3. Der Jupiter-Client erzeugt ein Jupiter-Request der Operation und fügt ihn in die ausgehende Warteschlange des `RequestForwarder` zum Versenden ein.

4. Das ISharedProject, welches die laufende Projektsitzung repräsentiert, übergibt die `Request`-Exemplare an den `ITransmitter`, der Fassaden-Klasse der Netzwerkkomponente. Sollte es sich bei dem ausführenden Client um den Projekt-Host handeln, wird die Nachricht an den `IRequestManager` durch Aufruf der `receiveRequest`-Methode zur Synchronisation mit dem Jupiter weitergegeben.

5. Der ITransmitter wandelt alle Jupiter-Requests in ihre XML-Repräsentation um und sendet sie als XMPP-Nachrichten direkt an den Jupiter-Server über das Jabber-Netzwerk (in diesem Fall dem Projekt-Host).

Synchronisieren der Jupiter-Request-Nachrichten

Die Jupiter-Request-Nachrichten der Driver werden vom Projekt-Host empfangen und im Jupiter-Server des entsprechenden Dokuments falls notwendig transformiert und an die entsprechenden Driver übermittelt.

1. Der `ITransmitter` des Projekt-Host empfängts die Jupiter-Request-Nachricht und erstellt aus dem XML-Inhalt der Nachricht ein Jupiter-`Request`-Exemplar. Diesen `Request` leitet er an den `IRequestManager` weiter.

2. Der die Schnittstelle `IRequestManager` implementierende `ConcurrentManager` empfängt den `Request` und leitet ihn an den betreffenden Jupiter-Server des Dokuments weiter.

3. Innerhalb des Jupiter-Servers wird der Request mit den Proxy-Clients synchronisiert. Die Proxy-Clients erzeugen als Ergebnis des Synchronisationsprozesses Jupiter-Requests zur Ausführung an ihre betreffenden Clients. Hierfür leiten sie die erzeugten Jupiter-Requests an den `RequestForwarder` weiter.

4. Alle Nachrichten in der ausgehenden Warteschlange des `RequestForwarder` werden nach ihrem Hinzufügen in der Warteschlange nacheinander von `ISharedProject` zum Versenden an den `ITransmitter` weitergeleitet.

5. Der `ITransmitter` sendet nach Umwandlung des `Request` in einer entsprechenden XML-Repräsentation die Nachricht direkt an den zugehörigen Client über das Jabber-Netzwerk.

Empfangen und Ausführen von Jupiter-Request-Nachrichten

Jupiter-Request-Nachrichten des Jupiter-Servers werden mit dem lokalen Jupiter-Client des Drivers synchronisiert und im Anschluss als Operation auf der lokalen Dokumentenkopie ausgeführt.

1. Der `ITransmitter` eines Drivers empfängt die Jupiter-Request-Nachricht und erstellt aus dem XML-Inhalt der Nachricht ein Jupiter-`Request`-Exemplar. Diesen `Request` leitet er an den `IRequestManager` weiter.

2. Der die Schnittstelle `IRequestManager` implementierende `ConcurrentManager` empfängt den `Request` und leitet ihn an den betreffenden Jupiter-Client des Dokuments weiter.

3. Der Jupiter-Client synchronisiert den `Request` und liefert eine `Operation` als Ergebnis zurück. Diese wird im `ConcurrentManager` als `IActivity` umgewandet

und an den `IActivityManager` durch Aufruf der `execTransformedActivity`-Methode weitergeleitet.

4. Der `IActivityManager` leitet die `IActivity` an die `IActivityProvider` weiter, die für die Ausführung der Aktivitäten zuständig sind.

Beinhaltet der ausführende Driver zusätzlich die Rolle des Projekt-Hosts, fügt er die soeben ausgeführte Aktivität in die ausgehende Nachrichtenwarteschlange des `IActivitySequencer` ein. Alle Teilnehmer der Sitzung erhalten daraufhin die `IActivity` der durchgeführten Aktivität und führen diese, sofern sie nicht Driver des Dokuments sind, lokal aus.

9.2.2 Erweiterte Konsistenzsicherung

Der Jupiter-Algorithmus ermöglicht eine Konsistenzsicherung ohne manuelles Eingreifen durch den Anwender. Dies setzt jedoch voraus, dass alle Änderungsoperationen kommuniziert werden können. Die in Weitverkehrsnetzen zwangsläufig auftretende Latenzzeit während der Übertragung von Änderungsoperationen und der hieraus resultierenden Überlagerung und dem zeitlich versetztem Eintreffen der Änderungsnachrichten wird in Jupiter durch die Berechnung von hypothetischen Operationen (siehe 8.2) aufgelöst.

Die bisherige Nachrichtenübermittlung basiert auf der Übertragung von XMPP-Nachrichten über einen Jabber-Serververbund. Diese Übertragungsart birgt das Risiko des Verlusts einzelner Nachrichten, so dass die Konsistenzsicherung mit Jupiter in diesem Fall nicht mehr erfolgen kann, da der Status innerhalb des Vektor-Raums nicht mehr fehlerfrei rekonstruiert werden kann. Der Verlust einer Nachricht wird durch die zu erwartende Vektor-Nummer identifiziert, die Konsistenz kann infolge nicht mehr sichergestellt werden.

Die Konvergenz des gemeinsamen Dokuments muss in diesem Fall durch eine zusätzliche Konsistenzsicherung sichergestellt werden. Wird eine Ausnahme bzw. ein Konflikt durch die Jupiter-basierte Konsistenzsicherung gemeldet, müssen alle Proxy- und Client-Endpunkte in einen Start-Zustand gesetzt und der Dokumentenzustand abgeglichen werden.

Der den Konflikt auslösende Teilnehmer muss seinen Bearbeitungsprozess unterbrechen und wird über einen Informationsdialog über den Konflikt informiert. Im Anschluss muss die Konfliktdatei mit dem Dokumentenzustand des Projekt-Hosts synchronisiert werden. Hierfür wird die Datei global gespeichert, um den aktuellen Bearbeitungsvorgang bei allen

Teilnehmern sicher zu beenden. Anschließend empfängt der konfliktauslösende Teilnehmer eine neue Version der Datei vom Projekt-Host.

Der Verlust der Informationen der bereits beim Teilnehmer durchgeführten Änderungen ist in diesem Fall vertretbar, da es sich um kleinere Textfragmente handelt und die bisherigen Änderungen des Teilnehmer vor Auftreten des Konflikts in das Dokument des Projekt-Hosts integriert wurden.

Erweiterte Überprüfung

Der Abschluss eines Bearbeitungsprozesses durch einen Driver wird durch die Speicherung der bearbeiteten Datei signalisiert. Eine Speicher-Operation ist eine konfliktfreie Operation des Drivers, die an alle Teilnehmer kommuniziert wird. Die anderen Teilnehmer speichern das Dokument nach Erhalt der Speicher-Operation und bestätigen die Speicherung ihrerseits durch Senden einer eigenen Speicher-Operation.

An dieser Stelle erfolgt eine erweiterte Überprüfung der Konsistenz aller Dokumentenkopien der Teilnehmer des bearbeiteten Dokuments. Der Vergleich der Dokumentenkopien erfolgt auf Grundlage der Dateiprüfsumme, die nach Ausführung der Speicherung von der Datei erstellt wird. Die Dateiprüfsumme wird als zusätzlicher Parameter der Speicher-Operation hinzugefügt, so dass der Projekt-Host alle Dokumentenversionen vergleichen kann. Als Referenz dient an dieser Stelle immer der Dokumentenstand des Projekt-Hosts. Wird eine Ungleichheit der Prüfsummen zwischen Projekt-Host und einem Teilnehmer festgestellt, wird eine neue Dokumentenkopie der Datei vom Projekt-Host an den Teilnehmer übertragen, der über den identifizierten Konflikt und deren Behebung informiert wird.

9.3 Exklusives Schreibrecht

Die Konsistenzsicherung ermöglicht den Drivern die blockierungsfreie Arbeit innerhalb der Dokumente des Projekts und synchronisiert die Änderungsoperationen. Driver-Operationen an der Projektstruktur können jedoch nicht durch die Konsistenzsicherung synchronisiert werden, da diese Operationen Bereiche außerhalb der Dokumentenstruktur betreffen.

Folgende Operationen an der Projekt-Struktur können nicht nebenläufig ausgeführt werden, sondern benötigen ein exklusives Schreibrecht:

Verschieben oder Umbenennen von Ressourcen: Das Verschieben oder Umbenennen von Paketen bzw. Dateien kann in Konflikt mit einer Verschiebe-Operation

(bzw. Umbenennen-Operation) eines anderen Drivers stehen. Des Weiteren können Konflikte entstehen, wenn andere Driver Dateien innerhalb der Projektstruktur geöffnet und nach Ausführung der Operation verschoben oder umbenannt haben.

Löschen von Ressourcen: Das Löschen von Ressourcen besitzt dieselbe Konflikt-Problematik wie sie bereits unter Verschieben oder Umbenennen von Ressourcen beschrieben wurde.

Diese Konflikt-Operationen werden durch ein exklusives Schreibrecht für das gesamte Projekt unterbunden. Ein exklusiver Schreibzugriff eines Drivers nur für Teilbereiche des Projekts würde das bestehende Rechtekonzept verändern, da es in diesem Fall erweiterte Driver-Rollen geben würde, die nicht mehr auf alle Dateien bzw. Ordner schreibenden Zugriff haben.

Im Kontext der Paar-Programmierung und der damit verbundenen hohen Kommunikation zwischen den Teilnehmern, ist die Verwendung eines globalen exklusiven Schreibzugriffs und die hieraus resultierende Koordination zwischen den Teilnehmern eine akzeptable Einschränkung zur Sicherung der Konsistenz.

Ausnahmen für ein exklusives Schreibrecht von Projektstruktur-Operationen bilden an dieser Stelle Operationen des Einfügens neuer Ressourcen, da diese Vorgänge die Zugriffsbereiche der anderen Driver nicht beeinflussen. Driver können daher ohne ein exklusives Schreibrecht neue Pakete (oder Ordner) und Dateien innerhalb der Projektstruktur erstellen und diese Operationen kommunizieren.

Ausführen einer Projektstruktur-Operation

Projektstruktur-Operationen, die ein exklusives Schreibrecht benötigen, können nur vom Projekt-Host unter der Bedingung ausgeführt werden, dass alle Projekt-Teilnehmer die Rolle Observer zum Zeitpunkt der Ausführung der Projektstruktur-Operation innehaben. Da der Projekt-Host für die Zuweisung der Rollen innerhalb der Projektsitzung verantwortlich ist, kann er Operationen an der Projektstruktur durchführen, die aktiven Driver-Rollen entziehen und den entsprechenden Teilnehmern die Rolle des Observers zuweisen. Der Projekt-Host ist somit der exklusive Driver und besitzt daher automatisch ein exklusives Schreibrecht.

Erfolgt eine Projektstruktur-Operation ohne ein exklusives Schreibrecht durch den Projekt-Host, wird der entsprechende Anwender darüber informiert.

10. Fazit

10.1 Zusammenfassung

Ziel meiner Diplomarbeit war es, zum einen den bestehenden Replikations-Ansatz der Eclipse-Erweiterung für verteilte Paar-Programmierung „Saros“ um die Umsetzung einer Punkt-zu-Punkt Verbindung für den Prozess des Datenabgleichs zu erweitern. Eine direkte Punkt-zu-Punkt Verbindung ermöglicht nicht nur eine, nur durch die Bandbreite limitierte schnelle Datenübertragung, sondern schafft darüber hinaus die Voraussetzung für die Umsetzung erweiterter Multimedia-Kommunikation. Zum anderen sollte der bestehende Ansatz des exklusiven Schreibrechts eines Drivers, mit dem Ziel der Kollaboration auf Schreibebene, um die Möglichkeit mehrerer Driver innerhalb einer Projektsitzung erweitert werden.

Nach Untersuchung der bestehenden Problematik und aktueller Lösungskonzepte zur Realisierung direkter Verbindungen im Weitverkehrsnetzen habe ich eine Umsetzung auf Basis des Jingle-Ansatzes unter Verwendung der Smack-API in die bestehende Eclipse-Erweiterung integriert. Den bestehenden Replikations-Ansatz habe ich zur Verwendung einer Punkt-zu-Punkt Verbindung entsprechend adaptiert. Für die Realisierung der Punkt-zu-Punkt Verbindung unter Verwendung der Jingle-Erweiterungen der Smack-API war eine Umstellung der Version notwendig.

Zur Realisierung einer Kollaboration auf Schreibebene habe ich die verschiedenen Ansätze von Verfahren zur Nebenläufigkeitskontrolle untersucht und für eine Umsetzung innerhalb der bestehenden Eclipse-Erweiterung bewertet. Die Verfahren der Operational Transformation ermöglichen eine eng gekoppelte, synchrone Gruppenarbeit und erfüllen durch die Eigenschaften des blockierungsfreien Zugriffs auf alle Dokumente mit automatischer Konfliktauflösung die notwendigen Voraussetzungen für einen Einsatz im Umfeld der verteilten Paar-Programmierung. Hierbei bot sich der Jupiter-Ansatz, sowohl von der geringeren Komplexität der Umsetzung her als auch vom geringen Kommunikationsaufwand her, gegenüber anderen Ansätzen an.

Eine von mir autarke Umsetzung einer Jupiter-basierten Nebenläufigkeitskontrolle wurde vor der Integration in die Eclipse-Erweiterung in einer zu diesem Zweck von mir entwickelte Testumgebung verifiziert.

Nach erfolgreichem Abschluss dieser Testphase wurde die Implementierung des Jupiter-Algorithmus von mir in die bestehende Eclipse-Erweiterung als Nebenläufigkeitskontrolle

integriert und eine Mehr-Schreiber-Funktionalität durch Adaption der bestehenden Eclipse-Erweiterung umgesetzt. Hierdurch ist es nun möglich, dass mehrere Teilnehmer der Sitzung gleichzeitig an Projektdateien arbeiten können.

10.2 Bewertung

Durch die Integration eines universellen Verfahrens zum Aufbau einer Punkt-zu-Punkt Verbindung konnte der bestehende Ansatz des Datenaustauschs entscheidend verbessert werden. Die von mir integrierte Umsetzung der Jingle-basierten Lösung für das NAT-Problem schafft zudem die Voraussetzung für die Verwendung einer Direktverbindung zur Optimierung der bestehenden Kommunikation und zur Umsetzung erweiterter Multimedia-Kommunikation. Der für die Integration notwendige Versionswechsel der Smack-API gestaltete sich auf Grund eingeschränkter Dokumentation der Schnittstellen und der Veränderung bestehender Schnittstellen nach dem Versionswechsel, schwieriger als im Vorfeld angenommen. Stellenweise reagierten Schnittstellen trotz unveränderter Spezifikation nach der Umstellung der Version mit einem veränderten Verhalten, wodurch bestimmte Funktionen der bestehenden Eclipse-Erweiterung nicht mehr funktionierten und zusätzlich angepasst werden mussten. Im Ergebnis kann mit dieser Umsetzung eine Direktverbindung in verschiedenen Netztopologien aufgebaut werden.

Weiterhin stellte die Umsetzung der Mehr-Schreiber-Funktionalität in die bestehende Eclipse-Erweiterung mit der damit verbundenen Adaption der bestehenden Funktionalität an eine Mehr-Schreiber-Umgebung eine große Herausforderung dar. Durch die zu Beginn autarke Umsetzung der Nebenläufigkeitskontrolle konnte ich erfolgreich den Jupiter-Ansatz in einen Algorithmus zur Konsistenzsicherung umsetzen. Die Integration in die bestehende Eclipse-Erweiterung und Adaption derselbigen durch eine Mehr-Schreiber-Funktionalität konnte initial erfolgen, so dass mehrere Driver konfliktfrei an einem gemeinsamen Dokument arbeiten können. Es existieren sowohl durch den gestiegenen Kommunikationsaufwand zur Konsistenzsicherung und der relativ hohen Latenzzeit der bestehenden XMPP-basierten Kommunikationsstruktur, als auch durch die tiefgreifenden Anpassungen der bestehenden Eclipse-Erweiterung noch Einschränkungen, die im Zuge der Diplomarbeit nicht abgedeckt werden konnten.

10.3 Ausblick

Auch wenn die Schwerpunkte der Nebenläufigkeitskontrolle und der Umsetzung eines universellen Verfahrens zum Aufbau einer Direktverbindung innerhalb dieser Diplomarbeit umgesetzt werden konnten, birgt die bestehende Eclipse-Erweiterung viel Potential um zukünftig weiter ausgebaut werden zu können.

Vorrangig wäre an dieser Stelle die Adaption der bestehenden XMPP-basierten Kommunikationsstruktur zu nennen. Der langsame Versand sowohl der Änderungs- als auch der Awareness-Information über den Jabber-Serververbund kann durch die Verwendung einer Direktverbindung entscheidend verbessert werden. Die für die Umsetzung der direkten Dateiübertragung integrierten Mechanismen zur Lösung des NAT-Problems bieten die Voraussetzung zur Umsetzung weiterer Kommunikationsmittel. Auf Basis dieses Ansatzes können Multimedia-Kommunikationen (z.B. Audio- und Video) zur Verbesserung der Kommunikation der Sitzungsteilnehmer umgesetzt werden.

Die bisherige Umsetzung des zeitweise exklusiven Schreibrechts des Projekt-Hosts bei Veränderungen an der Projektstruktur könnte durch einen Mechanismus des exklusiven Schreibrechts für beliebige Driver erweitert werden. Alle anderen Driver würden für den Änderungsvorgang ihre Driver-Rollen abgeben und nach Durchführung der Änderungen automatisch wieder zugewiesen bekommen.

Auch die Nachverfolgung der Änderungsoperation verschiedener Driver an einem Dokument könnte dahingehen verbessert, dass die Änderungsoperationen eindeutiger den einzelnen Driver zuzuordnen wären. Bisher werden die Änderungsoperationen der anderen Driver durch einen grauen Hintergrund dargestellt, um die entfernten von den lokal ausgeführten Änderungen zu unterscheiden.

Literaturverzeichnis

[1] Kent Beck: " Extreme Programming Explained: Embrace Change", 1999

[2] Williams, Laurie: The Collaborative Software Process, 2000

[3] A. Cockburn, L. Williams: " The Costs and Benefits of Pair Programming", 2001

[4] D. Stotts, L Williams, N. Nagappan, P. Baheti, D. Jen and A. Jackson: "Extreme Programming and Agile Methods - XP/Agile Universe 2003", 2003

[5] P. Dourish, V. Bellotti: "Awareness and coordination in shared workspaces", 1992, CSCW'92

[6] R. Djemili: Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierung verteilter Paarprogrammierung, 2006

[7] P. Saint-Andre: " Streaming XML with Jabber / XMPP", 2005

[8] Jive Software: "Smack API", http.://www.jivesoftware.org/smack (02.06.2008)

[9] Eclipse Foundation: "Eclipse", http://www.eclipse.org (02.06.2008)

[10] B. Gustavs: "Weiterentwicklung des Eclipse-Plug-Ins Saros zur Verteilten Paarprogrammierung (DPPII)", https://www.inf.fu-berlin.de/w/SE/ThesisDPPII (28.05.2008)

[11] Wikipedia: "Jabber", http://de.wikipedia.org/wiki/Jabber (22.03.2008)

[12] XMPP Standards Foundation: "XMPP Extensions", http://www.xmpp.org/extensions/ (23.02.2008)

[13] T. Muldowney, M. Miller, R. Eatmon: "File Transfer (XEP-0096)", http://www.xmpp.org/extensions/xep-0096.html (15.02.2008)

[14] T. Muldowney, M. Miller, R. Eatmon: "XEP-0095: Stream Initiation", http://www.xmpp.org/extensions/xep-0095.html (15.02.2008)

[15] J. Hildebrand, P. Saint-Andre, R. Eatmon, P. Millard: "XEP-0030: Service Discovery", http://www.xmpp.org/extensions/xep-0030.html (15.02.2008)

[16] D. Smith, M. Miller, P. Saint-Andre: "XEP-0065: SOCKS5 Bytestreams", http://www.xmpp.org/extensions/xep-0065.html (13.12.2007)

[17] J. Karneges: "XEP-0047: In-Band Bytestreams (IBB)", http://www.xmpp.org/extensions/xep-0047.html (13.12.2007)

[18] A. Wenckus: "Smack - 137 - File Transfer Settings", http://www.igniterealtime.org/issues/browse/SMACK-137 (05.06.2008)

[19] IETF: "The Internet Engineering Task Force", http://www.ietf.org/ (07.06.2008)

[20] K. Egevang, P. Francis: "The IP Network Address Translator (NAT)", http://tools.ietf.org/html/rfc1631 (02.06.2008)

[21] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy: "RFC 3489 - STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", http://tools.ietf.org/html/rfc3489 (02.06.2008)

[22] Wikipedia: "Network Address Translation", http://de.wikipedia.org/wiki/Network_Address_Translation (02.06.2008)

[23] M. Holdrege, P. Srisuresh: "RFC 3027 - Protocol Complications with the IP Network Address Translator", http://tools.ietf.org/html/rfc3027 (02.06.2008)

[24] SarWiki. Humboldt Universitäst - Informatik: "NAT traversal", http://sarwiki.informatik.hu-berlin.de/NAT_Traversal (02.06.2008)

[25] A. Klenk, A. Müller: "Network Address Translator - Tester", http://gex.cs.uni-tuebingen.de/?mod=start (15.02.2008)

[26] J. Rosenberg, R. Mahy, P. Matthews: "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", http://tools.ietf.org/html/draft-ietf-behave-turn-07 (02.06.2008)

[27] J. Schmidt: "The hole trick", http://www.heise-online.co.uk/security/How-Skype-Co-get-round-firewalls--/features/82481/0 (22.06.2008)

[28] J. Rosenberg: "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", http://tools.ietf.org/html/draft-ietf-mmusic-ice-19 (02.06.2008)

[29] S. R. Smith, Jingle: Jabber Does Multimedia, 2007, http://doi.ieeecomputersociety.org/10.1109/MMUL.2007.14

[30] H. Shulzrinne et al.: "RTP: A Transport Protocol for Real-Time Applications", RFC 3550 July 2003

[31] J. Beda, P. Saint-Andre, S. Ludwig, J. Hildebrand, S. Egan: "XEP-0166: Jingle", http://www.xmpp.org/extensions/xep-0166.html (15.02.2008)

[32] S. Ludwig, P. Saint-Andre, S. Egan, R. McQueen: "XEP-0167: Jingle RTP Sessions (Scenario voicechat)", http://www.xmpp.org/extensions/xep-0167.html#scenarios-voicechat (08.06.2008)

[33] P.Saint-Andre, M. Chen: "XEP-0180: Jingle Video via RTP", http://www.xmpp.org/extensions/xep-0180.html (08.06.2008)

[34] J. Beda, P. Saint-Andre, S. Ludwig, J. Hildebrand, S. Egan: "XEP-0177: Jingle Raw UDP Transport Method", http://www.xmpp.org/extensions/xep-0177.html (08.06.2008)

[35] P. Saint-Andre: "XEP-0234: Jingle File Transfer", http://www.xmpp.org/extensions/xep-0234.html (14.06.2008)

[36] J. Beda, P. Saint-Andre, S. Ludwig, J. Hildebrand, S. Egan: „XEP-0176: Jingle ICE-UDP Transport Method“, http://www.xmpp.org/extensions/xep-0176.html (14.06.2008)

[37] Matt Tucker: "Jingle: Cutting Edge VoIP", http://www.slideshare.net/mattjive/

[38] T. King: ""JSTUN" - Java Simple Traversal of User Datagram Protocol (UDP)", http://jstun.javawi.de/ (15.12.2007)

[39] C.A. Ellis, S.J. Gibbs: "Concurrency control in groupware systems", 1989, Proceedings of the ACM SIGMOND Conference on Management of Data (May)

[40] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen: „Achieving Convergence, Causality-preservation, and Intention-preservation“, 1998

[41] P. Bernstein, A. V. Hadzilacos, and et al: „Concurrency control and recovery in database systems“, 1987,

[42] Uwe M. Borghoff, Johann H. Schlichter, Rechnergestützte Gruppenarbeit: Eine Einführung in verteilte Anwendungen, 1998,

[43] A. Gerlicher: "Erweiterung bestehender Anwendungen um kollaborative Funktionen mit Hilfe des Collaborative Editing Framework for XML (CEFX)", 2005

[44] M. Koch, H. Schlichter, D. Köhler: "Iris - Collaborative Editing Environment", http://sunschlichter0.informatik.tu-muenchen.de/proj/iris/

[45] Apple: "SubEthaEdit - collaboration text editing", http://www.codingmonkeys.de/subethaedit/edit.html (12.02.2008)

[46] Ken Gilmer: "Example Shared Text Editor", http://wiki.eclipse.org/index.php/Example_Shared_Text_Editor (20.05.2008)

[47] Mustafa K. Isik: "Real-Time Shared Editing", http://wiki.eclipse.org/RT_Shared_Editing (20.05.2008)

[48] M. Bigler, S. Räss, L. Zbinden: "ACE - a collaborative editor", http://de.wikipedia.org/wiki/ACE_%28Editor%29 (20.05.2008)

[49] C. Ellis, C. Sun: "Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements", 1998, Proc. of 1998 ACM Conference on Computer Supported Cooperative Work

[50] C. Sun, Y. Yang, Y. Zhang, and D. Chen: "A consistencey model and supporting schemes for real-time cooperative editing systems", Jan. 1996, Proc. of The 19th Australasian Computer Science Conference

[51] L. Lamport: "Time, clocks, and the ordering of events in a distributed system", 1978, CACM 21(7)

[52] M. Ressel, D. Nitsche-Ruhland, R. Gunzenhäuser: "An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors", 1996, Proceedings of the 1996 ACM conference on Computer supported cooperative work

[53] C. Ellis, C. Sun: "Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements", 1998, Proc. of 1998 ACM Conference on Computer Supported Cooperative Work

[54] D. Nicholas, P. Curtis, M. Dixon, and J. Lamping: ""High-latency, low-bandwidth windoing in the Jupiter collaboration system"", Nov. 1995, ACM Symposium on User Interface Software

[55] M. Bigler, S. Raess, L. Zbinden: ACE Report Implementation Algorithm, 2005

[56] M. Bigler, S. Rass, L. Zbinden: "ACE - a collaborative editor", http://sourceforge.net/projects/ace/ (23.06.2008)

[57] Wikipedia: "JUnit", http://de.wikipedia.org/wiki/JUnit (22.04.2008)

[58] P. Saint-Andre: "XEP-0045: Multi-User Chat", http://www.xmpp.org/extensions/xep-0045.html (22.02.2008)

[59] A. Imine, P. Molli, G. Oster, M. Rusinowitch: „Proving Correntness of Transformation Functions in Real-Time Groupware“, 2003, Proceedings of the eighth conference on European Conference on Computer Supported Cooperative Work

[60] A. Imine, P. Molli, G. Oster, M. Rusinowitch: „Achieving Convergence with Operational Transformation in Distributed Groupware Systems“, 2004