Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Software Engineering

Entwicklung eines Server-Prototypen für Saros

Nils Hauke Bussas
Matrikelnummer: 4462430
nilsbu@zedat.fu-berlin.de

Betreuer: Franz Zieris
Gutachter: Prof. Dr. Lutz Prechelt
Zweitgutachter: Prof. Dr. Elfriede Fehr

Berlin, 20.02.2014

Zusammenfassung

In dieser Arbeit geht es um die Entwicklung eines Server-Prototypen für das Eclipse-Plug-In Saros. Dieses Programm unterstützt Entwickler bei der verteilten, kollaborativen Softwareentwicklung. Dazu wird eine Sitzung aufgebaut, über deren Dauer alle Beteiligten Zugriff auf alle geteilten Projekte haben. Im bisherigen Sitzungsmodus, werden die Sitzungen von einem der Beteiligten geleitet. Dies ist nicht immer erwünscht, darum soll es die Möglichkeit geben, die Sitzungen von einem Server leiten zu lassen.

Ziel dieser Arbeit war es, den Prototypen eines solchen Servers zu entwickeln. Dieser ist von mir als zusätzliche Funktion im Eclipse-Plug-In realisiert worden. Wichtige implementierte Features sind das Starten von Sitzungen auf dem Server, der Beitritt eines Benutzers in eine solche Sitzung, das Einladen von Benutzern in die Sitzung und die Auskunft darüber, ob auf einem Server aktuell eine Sitzung läuft.

Zudem habe ich eine Liste von Erweiterungen zusammengetragen, die den Server verbessern könnten. Anhand dieser Liste soll der Server weiterentwickelt und schließlich veröffentlicht werden.

Eidesstattliche Erklärung

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

20.02.2014

_____

Nils Hauke Bussas

Inhaltsverzeichnis

1 Einleitung

Softwareentwicklung ist oft risikoreich. In der Planungsphase kann es passieren, dass an die Software gestellte Anforderungen unmöglich umzusetzen oder nicht sinnvoll sind. Ebenso kann es geschehen, dass sich die Anforderungen im Laufe des Entwicklungsprozesses verändern. Auch ist es möglich, dass der Aufwand, den es benötigt, die Software zu entwickeln, stark unterschätzt wird, um nur einige Beispiele für Risiken zu nennen.[11]

Im Laufe der Jahrzehnte sind eine Menge von Mechanismen entwickelt worden, um die Risiken zu minimieren. Einer dieser Mechanismen ist das Entwickeln von Prototypen, dem so genannten *Prototyping*. Bei diesem Verfahren wird ein bereits lauffähiges Modell der gewünschten Software oder einigen Teilen davon entwickelt. Anhand der Erfahrungen, die während der Entwicklung des Prototypen gemacht wurden, kann man unter anderem die Umsetzbarkeit der Software testen, man kann den Aufwand besser abschätzen und je nach Design des Prototypen ist es sogar möglich, die Reaktionen der Benutzer zu testen. [4, Kapitel 1]

Diese Arbeit beschäftigt sich mit der Entwicklung eines Server-Prototypen im Open-Source-Projekt Saros. Die Entwicklung des Servers ist zu aufwendig, um in einem Schritt abgeschlossen zu werden, insbesondere da mit dem Server Neuland für das Entwicklerteam betreten wird. Erschwerend kommt hinzu, dass Saros zu großen Teilen von Studenten im Rahmen von Abschlussarbeiten entwickelt wird und daher eine hohe Fluktuation an neuen Entwicklern hat.

Daher war gewünscht, hierfür zunächst einen Prototypen zu entwickeln. Dieser Prototyp sollte Klarheit darüber verschaffen, ob die Idee des Servers, sowie die einzelnen Features sinnvoll sind. Darüber hinaus sollte erforscht werden, wie die technische Umsetzung am besten zu bewerkstelligen ist. Wenn möglich sollte der entwickelte Code für die Entwicklung des finalen Servers, der veröffentlicht wird, verwendet werden können.

1.1 Was ist Saros?

Saros ist ein Plug-In für die Entwicklungsumgebung Eclipse für verteilte, kollaborative Softwareentwicklung. Es unterstützt Entwickler dabei, gemeinsam und gleichzeitig an einem Softwarerojekt zu arbeiten, selbst wenn sie sich nicht am selben Ort befinden. Um gemeinsam an einem Projekt aus dem Eclipse-Workspace arbeiten zu können, baut man eine Sitzung auf, über deren Dauer alle Sitzungsteilnehmer Zugriff auf die geteilten Dateien erhalten. Sämtliche Änderungen, die ein Teilnehmer an den Dateien macht, werden bei allen anderen nachvollzogen, sodass bei allen Mitarbeitern der Inhalt aller Editoren (Bearbeitungsfenster in Eclipse) und Dateien konsistent ge-

halten wird. Das bedeutet, dass die Inhalte der Editoren und Dateien zu jedem Zeitpunkt entweder gleich sind oder es zumindest sein werden, wenn alle versendeten Benachrichtigungen über Textänderungen empfangen und die beschriebenen Änderungen lokal ausgeführt sind.

Neben dieser Kernfunktionalität gibt es weitere Features, die die Nutzer bei der verteilten Entwicklung unterstützen sollen. Im *Follow Mode* kann man den Bewegungen, die ein anderer Sitzungsteilnehmer innerhalb des Projektes macht, automatisch folgen. Solche Bewegungen können der Wechsel in eine andere Datei oder das Scrollen innerhalb einer Datei sein. Es gibt ein so genanntes *Whiteboard*, eine Zeichenfläche, auf der man gemeinsam Skizzen erstellen kann. Es gibt einen eingebauten Chat, sowohl für alle Teilnehmer, als auch für Einzelgespräche, und eine Einbindung von Skype.

Jeder Benutzer hat eine Kontaktliste, die mit seinem Account verknüpft ist. Sitzungen können mit den Kontakten aus dieser Liste aufgebaut werden.

Saros ist ein Open-Source-Projekt, dessen Entwickler hauptsächlich Informatikstudenten der Freien Universität Berlin, sowie Mitarbeiter der Arbeitsgruppe Software Engineering des Instituts für Informatik sind.

1.2 Ziel der Arbeit

Die Arbeit umfasst die drei folgenden Ziele:

1. Entwicklung eines Server-Prototypen für Saros, der über alle grundlegenden Features verfügt. Konkret gemeint ist damit:
   - die Möglichkeit, einen Server einzurichten
   - das Starten einer Sitzung auf dem Server
   - das ”Hochladen”von Dateien/Projekten auf den Server
   - den Beitritt eines Benutzers in eine Sitzung
   - das Einladen von Personen in eine Sitzung
   - das Abfragen von Sitzungsinformationen vom Server
2. Bewertung von Nutzen und Umsetzung der oben genannten Features
3. Erstellung einer Liste der den Server betreffenden Aufgaben für nachfolgende Entwickler

2 Analyse

2.1 Sitzungen in Saros

Wenn mehrere Personen über Saros zusammen an einem Projekt arbeiten möchten, müssen sie dafür eine gemeinsame Sitzung aufbauen.

Zu Beginn der Sitzung wird vom Initiator der Sitzung ausgewählt, welche Mitarbeiter die Sitzung haben soll und an welchen Projekten in der Sitzung gearbeitet werden soll. Projekte können ganz oder nur teilweise geteilt werden. Das Teilen von lediglich einer Auswahl der Dateien eines Projekts wird *Partial Sharing* genannt. Wenn die Mitarbeiter noch nicht alle geteilten Dateien lokal verfügbar haben oder diese veraltet sind, werden sie vom Initiator der Sitzung, dem die Rolle des „Hosts“ zufällt, automatisch an die Mitarbeiter gesendet. Dadurch ist sichergestellt, dass alle Sitzungsteilnehmer nach dem Sitzungsaufbau identische Dateien haben.

Der Prozess, in dem die Dateien zwischen dem Host und einem anderen Teilnehmer abgeglichen werden, wird als *Project Negotiation* bezeichnet. In ihr wählt der Benutzer, der die Dateien erhält, zunächst aus, an welchem Ort die Projekte der Sitzung gespeichert werden sollen. Dies kann ein schon bestehendes Projekt (in der Regel eine lokal vorhandene Version des geteilten Projekts) sein oder ein anderer Speicherort. Danach wird überprüft, ob dem Benutzer am ausgewählten Ort Dateien fehlen. Falls nötig werden sie an ihn gesendet.

In einer laufenden Sitzung werden alle Änderungen, die ein Mitarbeiter an einer der geteilten Dateien macht, an alle anderen Teilnehmer gesendet und lokal nachvollzogen. Technisch ist dies so geregelt, dass die Informationen zunächst an den Host gesendet werden, sofern er die Änderung nicht selbst gemacht hat. Von dort aus werden sie dann den weiteren Teilnehmern mitgeteilt.[13]

Sitzungen enden dann, wenn der Host sie beendet. Einzelne Mitarbeiter können sie jedoch auch vor dem Sitzungsende verlassen.

Bezüglich des Hosts sind einige Besonderheiten zu beachten. Ihm ist das Sonderrecht gewährt, zu einer laufenden Sitzung zusätzliche Projekte hinzuzufügen. Grund dafür ist, dass es zu Inkonsistenzen, also unauflösbaren Unterschieden zwischen Dateiinhalten, kommen kann, wenn Nicht-Hosts weitere Projekte in die Sitzung einfügen.[13] Gleiches gilt für das Hinzufügen von zusätzlichen Teilnehmern. Auch hier hat sich gezeigt, dass es beim Hinzufügen durch Nicht-Hosts zu Inkonsistenzen kommen kann.[14] Darum ist das Einladen zusätzlicher Sitzungsteilnehmer ebenfalls ausschließlich dem Host gestattet.

Diese Sonderrechte und -pflichten können zurzeit nicht an andere Teilnehmer

übertragen werden, daher wird eine Sitzung beendet, wenn der Host sie verlässt.

2.2 Problematik des Hosts

Die bisherige Regelung ist aus mehreren Gründen unpraktisch.

Wenn man sich zu dritt in einer Sitzung befindet und der Host die Sitzung verlässt, wird auch die Verbindung zwischen den beiden anderen Personen abgebrochen, da Saros nicht über die Möglichkeit verfügt, die Rolle des Hosts an eine andere Person zu übergeben. Wenn jemand anderes als der Host ein weiteres Projekt zur Sitzung hinzufügen möchte, muss die Sitzung abgebrochen und eine neue aufgebaut werden, in der die andere Person der neue Host ist.

Außerdem gibt es Probleme beim zeitversetzten Arbeiten. Wenn zwei Personen (im Folgenden Alice und Bob genannt) zusammen mit Hilfe von Saros an einem Projekt arbeiten und Bob die Sitzung verlässt (z. B. weil er eine Mittagspause einlegt), ist der aktuelle Stand des Projekts nur noch bei Alice lokal verfügbar, da die Änderungen, die sie vornimmt, nicht mehr an Bob übertragen werden. Wenn Bob später weiterarbeiten möchte, muss Alice entweder erreichbar sein, damit sie eine Sitzung mit Bob aufbauen kann, oder Bob die aktuellen Daten auf eine andere Weise verfügbar machen. Saros ist in diesem Fall als Kommunikationsmittel nicht ausreichend, da für die Synchronisation von Daten zwingend beide Personen an ihrem Computer aktiv sein müssen. Wenn sie das Problem lösen möchten, müssen sie auf andere Mittel, wie z. B. ein Versionsverwaltungssystem oder für beide zugreifbare Ordner, in die Alice die Dateien kopiert, bevor sie ihren Arbeitsplatz verlässt, zurückgreifen.

2.3 Vision eines Saros-Servers

Aus den oben genannten Gründen soll Saros einen neuen Sitzungsmodus erhalten, in dem die Rolle des Hosts von einem Server übernommen wird. Auf diesem Server soll eine beliebige Anzahl an Sitzungen verwaltet werden können. Sie sollen dort ohne Unterbrechung laufen können, sodass sich Mitarbeiter einer Arbeitsgruppe jederzeit einklinken können. Die Benutzer bekommen dann vom Server den aktuellen Stand der Arbeit zugesendet, arbeiten so lange sie wollen daran und verlassen schließlich die Sitzung, die aber ohne sie weiterläuft. Gestartet werden sollen die Sitzungen dadurch, dass ein Benutzer einen Server auswählt, auf dem die Sitzung laufen soll, dazu die Projekte, die geteilt werden, und evtl. weitere Mitarbeiter. Der Initiator der Sitzung, der im alten System der Host ist, kann hier jederzeit die Sitzung verlassen, ohne dass diese dadurch beendet wird. Des Weiteren müs-

sen die einzelnen Mitarbeiter eines Projekts nicht gleichzeitig am Rechner sein, um einander den aktuellen Stand des Projekts via Saros zu übermitteln, da dieser über den Server jederzeit verfügbar ist. Wie in bisherigen Sitzungen sollen Benutzer weitere Teilnehmer zur Sitzung einladen können. Ungeklärt ist noch, wie eine Sitzung beendet wird. Denkbar ist, dass nur der Initiator sie beenden kann oder dass jeder Teilnehmer das Recht dazu hat.

Da das Projekt zu groß ist, um in einem Schritt entwickelt zu werden, hat das Entwicklerteam sich für eine evolutionäre Vorgehensweise entschieden. Im ersten Schritt, den ich in meiner Arbeit realisiert habe, wird ein Prototyp des Servers entwickelt. Anhand der Erfahrungen, die ich gemacht habe, kann die Spezifikation verfeinert und der Server weiterentwickelt werden. Ich bin im Rahmen meiner Arbeit nicht in der Lage gewesen, eine solche verfeinerte Spezifikation aufzustellen, jedoch habe ich Vorschläge für weitere Features, sowie Probleme im von mir entwickelten Prototypen zusammengetragen und in den Abschnitten 5.2 und 5.3 aufgeschrieben. Im Folgenden beschreibe ich, welche Entscheidungen ich beim Entwurf der Spezifikation meines Prototypen treffen musste.

3 Planung

Ich habe in Abschnitt 2.3 beschrieben, warum die Entwicklung eines Serverfeatures gewünscht ist und warum es sinnvoll war, hierfür zunächst einen Prototypen zu entwickeln. Nicht jedoch habe ich erläutert, welche Form dieser Prototyp haben sollte und welche Funktionalitäten er enthalten würde. Meine Überlegungen dazu sind Thema dieses Kapitels.

3.1 Vorüberlegungen

3.1.1 Stand-Alone oder Elipse-Plug-In?

Bei der Implementierung des Servers gab es einige Schwierigkeiten. Zur Zeit meiner Arbeit war Saros ausschließlich ein Eclipse-Plug-In, also nicht eigenständig lauffähig. Es war aber gewünscht, den Server aus der bisherigen Codebasis zu entwickeln, da dies den Arbeitsaufwand erheblich verringern würde. Daher standen grundsätzlich zwei Wege offen. Ich konnte den Code von Eclipse loslösen, sodass eine Codebasis zur Verfügung stünde, aus der ich einen eigenständigen Server hätte entwickeln können, oder ich konnte den Server als zusätzliches Feature im Eclipse-Plug-In implementieren.

Es ist geplant, Saros neben Eclipse in Zukunft auch für andere Entwicklungsumgebungen verfügbar zu machen. Daher ist die Trennung des bisherigen Saros-Codes in einen allgemeinen Teil, den sogenannten „Core“, und einen Eclipse-spezifischen vorgesehen. Zu Beginn meiner Arbeit war diese Trennung jedoch noch in der Planungsphase. Da es für die Entwicklung eines Stand-Alone-Servers erforderlich ist, dass der Core bereits von Eclipse losgelöst ist, war es für mich nicht möglich, diesen Weg einzuschlagen.

Daher entschied ich mich dazu, die zweite Möglichkeit umzusetzen und den Server als ein aktivierbares Feature im Saros-Plug-In zu entwickeln. Wenn der Code des Saros-Kerns vom eclipse-spezifischen Teil getrennt ist, kann der Schritt zu einem eigenständigen Server nachgeholt werden (siehe 5.2.2).

3.1.2 Anzahl der Sitzungen pro Server

Saros war bisher nicht darauf ausgelegt, mehrere Sitzungen zur selben Zeit zu unterstützen, da es für die bisherige Benutzung nicht nötig war. Für einen Server ist diese Einschränkung jedoch relevant, da es durchaus erwünscht sein kann, mehrere Sitzungen über einen Server zu koordinieren.

Im Code wird an vielen Stellen die Annahme gemacht, dass nie mehr als eine Sitzung zur gleichen Zeit läuft. Da die Sitzungen ein zentraler Aspekt von Saros sind, besitzt der sie betreffende Code, unter anderem die Klassen *SarosSession* und *SarosSessionManager*, viele Schnittstellen zu ande-

ren Codeteilen, wie den graphischen Oberflächen oder der Netzwerkschicht. Daher ist davon auszugehen, dass die Erweiterung auf mehrere, parallele Sitzungen zu aufwendig wäre, um im Rahmen dieser Arbeit absolviert zu werden. Aus diesem Grund wird der Prototyp des Servers zu jedem Zeitpunkt nur eine Sitzung verwalten können. Die Umstellung ist daher eine Aufgabe für zukünftige Entwickler (siehe 5.2.4).

3.1.3 Proof of Concept oder Alpha-Version?

Es stellte sich außerdem die Frage, welchen Wert der von mir entwickelte Code für nachfolgende Entwickler haben soll. Der von mir entwickelte Server könnte ein reiner Prototyp („Proof of Concept“) sein, der nie für die Benutzer zugänglich gemacht werden soll, sondern lediglich zum Sammeln von Erfahrungen entwickelt wird.

Entwickler, die später an dem Server arbeiten, würden dann zwar von den Erfahrungen, die im Rahmen dieser Arbeit gemacht wurden, profitieren, müssten den Code aber nochmals von Grund auf neu entwickeln. Dies hätte den Vorteil, dass strukturelle Fehler des Codes bei einer neuen Planung mit bedacht werden könnten, jedoch nicht am bestehenden System behoben werden müssten. Die Kosten eines Scheiterns wären also niedriger. Darüber hinaus würde diese Vorgehensweise ermöglichen, den Fokus darauf zu legen, ein lauffähiges Produkt zu entwickeln und auf Codedokumentation und -qualität weniger Wert zu legen. Dies hätte wahrscheinlich zur Folge, dass eine größere Anzahl von Features erforscht werden könnte.

Alternativ dazu könnte die Arbeit auch als eine frühe Alpha-Version begriffen werden, die zwar auch noch nicht für die Benutzer zugänglich sein soll, aber von Nachfolgern lediglich weiterentwickelt werden muss, um veröffentlicht werden zu können. Diese Vorgehensweise wird als „evolutionäres Prototyping“ bezeichnet.[9, S. 434] Dieser Ansatz hat den Vorteil, dass weniger Arbeit doppelt erledigt werden muss, jedoch den Nachteil, dass vermutlich weniger Funktionalität realisiert werden könnte, da die einzelnen Teilfeatures mit größerer Sorgfalt entwickelt werden müssten.

Zu treffen war diese Entscheidung nur in Verbindung mit der Klärung der folgenden Frage über kontinuierliche Integration.

3.1.4 Kontinuierliche Integration

Bei kontinuierlicher Integration[7] werden die Teilfeatures des Servers schon vor Abschluss der Arbeit in das Hauptprojekt integriert. Dabei wird ein Reviewprozess durchlaufen, in dem sowohl die Idee des Features an sich, als auch die konkrete Umsetzung im Code von anderen Saros-Entwicklern beurteilt und deren Verbesserungsvorschläge von mir berücksichtigt werden

können.[1]

Die Entscheidung über die kontinuierliche Integration konnte nicht unabhängig von der vorhergehenden bezüglich Proof of Concept bzw. einer Alpha-Version getroffen werden, weil ein Proof of Concept nicht in das Projekt integriert werden muss, dies jedoch Sinn ergibt, wenn geplant ist, spätere Entwickler an meinem Code weiterarbeiten zu lassen. Aus diesem Grund ähneln die Argumente für und gegen die kontinuierliche Integration denen aus dem vorherigen Abschnitt.

Wenn die Ergebnisse meiner Arbeit nicht laufend in das Hauptprojekt integriert würden, die Entwicklung des Codes also parallel stattgefunden hätte, käme ich voraussichtlich schneller voran, jedoch entginge mir das Feedback der anderen Saros-Entwickler. Wenn ich mich dafür entscheiden würde, müsste ich einen Teil meiner Arbeitszeit auf die Kommunikation mit den anderen Entwicklern verwenden. Jedoch würde deren Feedback die Codequalität erhöhen und mir dabei helfen, die Features, die ich entwickeln würde, sinnvoller zu gestalten.

Es war mir bei meiner Arbeit wichtiger, die Features, die ich entwickle mit hoher Sorgfalt zu entwickeln, als möglichst viele Features des Servers zu implementieren. Des Weiteren hat sich beim Aufstellen der Spezifikation (siehe 3.4) gezeigt, dass vermutlich viele Teile des von mir entwickelten Codes von meinen Nachfolgern ohne große Änderungen übernommen werden können. Aus diesen Gründen habe ich den Code laufend in das Hauptprojekt integriert, sodass nachfolgende Entwickler ihn lediglich weiterentwickeln müssen. Der Prototyp stellt somit keinen reinen „Proof of Concept“ dar.

Die entwickelten Features durften trotz der kontinuierlichen Integration zunächst nicht für die Benutzer zugänglich sein, obwohl sie bereits Teil der Codebasis sind. Um dieses Problem zu lösen, entschied ich mich dazu , einen so genannten „Feature-Toggle“ einführen (siehe 4.2). Ein Feature-Toggle oder Feature-Flag ist eine Variable, in der Regel ein Wahrheitswert (*Boolean*), die dazu dient, ein Feature im Code an- oder auszuschalten. Dies kann nötig sein, wenn das Entwickeln eines Features zu lange dauert, um zwischen zwei Releases implementiert und zu werden. In diesem Fall dürfen Teile des Features, die bereits implementiert sind, im Release nicht enthalten bzw. abrufbar sein. Über den Feature-Toggle kann man das Feature fürs Release deaktivieren, es zum Testen jedoch aktivieren.[8]

3.2 Anwendungsfälle

Um auswählen zu können, welche Features den Prototypen bilden sollten, war es notwendig, zunächst die Nutzerseite zu betrachten. Dazu sind im

[1] Das Entwicklungsteam von Saros benutzt das Review-System *Gerrit*[1].

Folgenden einige zentrale Anwendungsfälle beschrieben, die der Prototyp abdecken soll.

3.2.1 Server-Feature aktivieren

Da der Server Teil des Plug-Is ist, muss es möglich sein, ihn zu aktivieren und zu deaktivieren. Dazu soll der Administrator, die Person, die den Server verwaltet, im gestarteten Eclipse in den Einstellungen von Saros ein Häkchen setzen können, das den Server aktiviert. Wenn er sich dann mit seinem Account verbindet, sofern dies noch nicht der Fall war, soll der Server für die Personen in seiner Kontaktliste erreichbar sein.

3.2.2 Server-Sitzung starten

Wenn ein Benutzer auf einem Server eine Sitzung mit seinen gewünschten Projekten starten möchte, müssen dafür sowohl der Benutzer als auch der Server mit ihrem Account verbunden sein. Außerdem müssen sie einander als Kontakte hinzugefügt haben. Weder Server noch Benutzer dürfen aktuell Teilnehmer einer Sitzung sein.

Zum Starten der Sitzung öffnet der Benutzer in seiner Saros-Instanz einen Wizard (ein Fenster, in dem auf mehreren Seiten Einstellungen vorgenommen werden können), in dem er zunächst nach dem Server gefragt wird, der die Sitzung verwalten soll. Ihm wird dazu eine Liste der Server in seiner Kontaktliste angezeigt, jeweils mit der Zusatzinformation, ob dort gerade eine Sitzung läuft. Er wählt den gewünschten Server aus. Danach wird er nach den zu teilenden Projekten gefragt und wählt auch diese aus. Zum Abschluss sendet er die Daten durch einen Klick auf den Finish-Button an den Server. Dieser empfängt die Daten und startet eine Sitzung, in die er den Benutzer einfügt. Der Benutzer akzeptiert die Einladung.

Ist dies geschehen, soll eine Sitzung gestartet sein, deren Host der Server ist und in der neben ihm der Benutzer der einzige Teilnehmer ist. Die geteilten Projekte sind die vom Benutzer ausgewählten.

3.2.3 Server-Sitzung mit weiteren Teilnehmern starten

Als Abwandlung des vorhergehenden Anwendungsfalles soll es für den Benutzer ebenfalls möglich sein, beim Sitzungsstart weitere gewünschte Sitzungsteilnehmer anzugeben. Diese müssen ebenfalls mit ihren Accounts verbunden und in der Kontaktliste von Server und Benutzer gespeichert sein. Auch sie dürfen aktuell nicht Teilnehmer einer Sitzung sein.

Im Folgenden wird der Benutzer aus dem letzten Anwendungsfall Alice genannt. Der zusätzliche gewünschte Sitzungsteilnehmer ist Bob, das Angeben

und Einladen von mehr als einem weiteren Teilnehmer funktioniert analog zur folgenden Beschreibung.

Im Anschluss an die Auswahl der Projekte wird Alice gefragt, ob sie weitere Personen in die Sitzung einladen möchte. Sie wählt Bob aus.

Nachdem die Sitzung aufgebaut ist, also nach Abschluss sämtlicher Schritte aus 3.2.2, verschickt der Server eine Einladung an Bob, die dieser annimmt.

Nun ist die gleiche Sitzung aufgebaut wie in 3.2.2 mit dem Unterschied, dass zusätzlich Bob Teil der Sitzung ist.

3.2.4 Beitritt zu einer Server-Sitzung

Wenn ein Benutzer in eine Server-Sitzung eintreten will, müssen Server und Benutzer einander in der Kontaktliste gespeichert haben und der Server muss zurzeit Host einer Sitzung sein. Der Benutzer muss mit seinem Account verbunden sein und darf nicht Teilnehmer einer Sitzung sein.

Der Benutzer bewegt den Mauszeiger in seiner Kontaktliste in der *Saros-View*[2] über den Server, dessen Sitzung er betreten möchte. Es erscheint ein Tooltip (ein kleines, texthaltiges Pop-Up-Fenster, das unterhalb des Mauszeigers erscheint), der ihm anzeigt, dass auf dem Server aktuell eine Sitzung läuft, welche Projekte geteilt sind und welche Teilnehmer die Sitzung hat. Er führt einen Rechtsklick aus. Es erscheint ein Kontextmenü, welches ein Feld enthält, das zum Sitzungsbeitritt verwendet werden kann. Der Benutzer klickt auf dieses Feld. Saros sendet dem Server eine Nachricht, in der um einen Sitzungsbeitritt gebeten wird. Der Server empfängt die Bitte und schickt eine Sitzungseinladung an den Benutzer. Dieser nimmt die Einladung an.

Nach Ausführung dieser Schritte ist der Benutzer ein Teilnehmer der Sitzung des Servers.

3.2.5 Einladung zu einer Server-Sitzung

Es soll möglich sein, einen Kontakt zu einer laufenden Server-Sitzung einzuladen. Hierfür muss die einladende Person (Alice) Teilnehmer der Sitzung sein. Die eingeladene Person (Bob) muss mit ihrem Account angemeldet sein und in der Kontaktliste von Alice und dem Server gespeichert sein. Außerdem darf sie kein Sitzungsteilnehmer einer laufenden Sitzung sein.

Alice wählt Bob in der Kontaktliste in ihrer *Saros-View* mit einem Rechtsklick aus. Im erscheinenden Kontextmenü klickt sie das Einladungsfeld an. Saros

[2] Die *Saros-View* ist ein Fenster in Eclipse, üblicherweise im unteren Bereich, über das Saros sich steuern lässt.

sendet eine Bitte, Bob einzuladen, an den Server. Dieser empfängt die Nachricht und sendet eine Einladung an Bob.

Nach diesem Ablauf hat Bob die gewünschte Einladung in die Sitzung erhalten.

3.3 Auswahl der zu implementierende Features

Nun müssen auf Basis der in Abschnitt 3.1 getroffenen Entscheidungen und der Anwendungsfälle die zu implementierenden Features extrahiert werden. Ist dies geschehen, kann eine technische Spezifikation aufgestellt werden.

3.3.1 Server aktivieren

Um den Server zu aktivieren, muss, wie in 3.2.1 beschrieben, eine Checkbox in den Einstellungen von Saros hinzugefügt werden, über welche die Server-Funktion an- und ausgeschaltet werden kann. Die Information, ob der Server aktiviert ist oder nicht, muss dauerhaft, also über das Beenden von Eclipse hinaus, gespeichert werden, damit der Server nicht bei jedem Programmstart neu aktiviert werden muss.

3.3.2 Server bekannt machen

Um die Anforderungen zu erfüllen, die in den Abschnitten 3.2.2, 3.2.3 und 3.2.4 beschrieben sind, ist es notwendig, dass ein Benutzer von den Servern in seiner Kontaktliste erfährt, dass bei ihnen das Server-Feature aktiviert ist.

3.3.3 Sitzungsstart

Aus den Anwendungsfällen in 3.2.2 und 3.2.3 geht hervor, dass der Server um den Aufbau einer Sitzung gebeten wird, damit eine Sitzung gestartet werden kann. Zusätzlich müssen die Projekte, die geteilt werden sollen, übertragen werden. Außerdem muss die Möglichkeit bestehen, dem Server mitzuteilen, welche Benutzer zusätzlich zum Initiator Sitzungsteilnehmer sein sollen.

Der Prozess muss, wie leicht ersichtlich ist, mit der Bitte um den Sitzungsstart beginnen, da alles andere davon abhängt, dass der Server dieser Aufforderung nachgeht. Unklar ist jedoch, in welcher Form das Übertragen der Projekte geschieht und wann dem Server mitgeteilt wird, welche Personen einzuladen sind.

Beim Beitritt einer Person in eine Sitzung werden eine *Session Negotiation* und eine *Project Negotiation* vom Host gestartet. In der *Session Negotia-*

tion sind einige Aktionen vereint, die beim Beitritt zu einer Sitzung ausgeführt werden müssen. Dazu gehört eine Kompatibilitätsüberprüfung der Saros-Versionen, die Einladung zur Sitzung selbst und die Zuweisung einer Benutzerfarbe.[2] Nach ihrem Abschluss ist der Benutzer ein Teilnehmer der Sitzung. In der *Project Negotiation* werden ihm die geteilten Projekte übermittelt. Beides sind komplexe Prozeduren. Ich werde diese daher, so weit es mir möglich ist, unverändert lassen und serverspezifische Kommunikation vor oder nach diesen Prozessen stattfinden lassen, selbst wenn das bedeutet, dass während des Sitzungsstarts oder -beitritts inhaltliche Dopplungen in den versendeten Nachrichten auftreten.

Eine sinnvolle Strategie scheint es daher zu sein, den Server zunächst um den Start einer leeren Sitzung zu bitten, d. h. es sind keine Projekte geteilt und keine Sitzungsteilnehmer neben dem Server selbst enthalten. Wenn der Server die Anfrage erhalten hat, schickt er eine Absage, wenn schon eine Sitzung läuft, oder startet eine Sitzung, wen dies nicht der Fall ist. Danach sendet er dem Absender der Nachricht eine Einladung in die Sitzung. Nachdem dieser akzeptiert hat, laufen *Session Negotiation* und *Project Negotiation* wie in einer klassischen Saros-Sitzung ab. Bei der *Project Negotiation* werden keine Dateien übertragen, da die Sitzung noch frei von Projekten ist.

Im Anschluss daran muss der Client die gewünschten Projekte zur Sitzung hinzufügen. Im klassischen Saros ist die einzige Person, die in einer laufenden Sitzung zusätzliche Projekte hinzufügen kann, der Host. Diese Beschränkung wurde von Patrick Schlott im Rahmen seiner Masterarbeit[13] eingeführt, um inkonsistente Dateizustände, die entstehen konnten, wenn andere Teilnehmer als der Host Projekte hinzufügen, zu verhindern. Diese Inkonsistenzen entstehen jedoch nur, wenn die Sitzung mindestens drei Beteiligte hat. Aus diesem Grund ist es möglich, das Feature unter der Einschränkung, dass die Sitzung zur Zeit nur zwei Teilnehmer hat, wieder einzuführen. Direkt nach dem Beitritt des Benutzers hat die Sitzung nur zwei Teilnehmer, daher kann sicher eine *Project Negotiation*, die vom Benutzer gestartet wird, ausgeführt werden. In dieser fügt er die gewünschten Projekte zur Sitzung hinzu. Siehe dazu Abbildung 1.

3.3.4 Einladung von Personen

Nachdem dies geschehen ist, können weitere Personen zur Sitzung eingeladen werden. Technisch lässt sich hier die gleiche Methode verwenden, die auch zur Umsetzung des in Abschnitt 3.2.5 beschrieben Anwendungsfalles verwendet wird. In beiden Fällen wird der Server darum gebeten, Benutzer einzuladen. Hierzu muss lediglich eine Nachricht, die diese Bitte enthält, an den Server gesendet werden. Wenn dieser die Person in seiner Kontaktliste gespeichert hat, versendet er daraufhin eine Einladung.

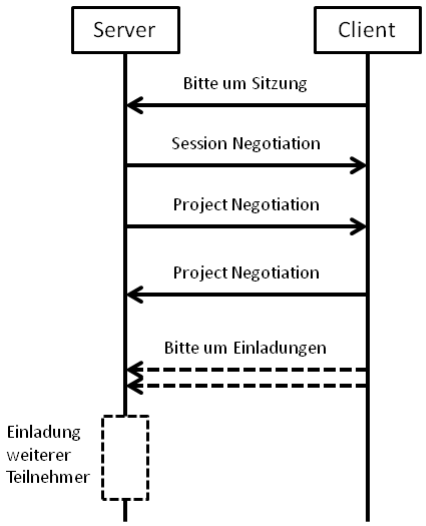

Abbildung 1: Sitzungsstart mit weiteren Teilnehmern

3.3.5 Abfrage von Sitzungsinformationen

Sowohl in Abschnitt 3.2.2, als auch in Abschnitt 3.2.4 wird gefordert, dass Benutzer Auskunft über die Aktivität des Servers erhalten. Im ersten Fall wird die Information benötigt, ob ein Server aktuell eine Sitzung leitet. Im zweiten Fall werden die geteilten Projekte und Sitzungsteilnehmer abgefragt. Beides lässt sich zu einer einzelnen Abfrage zusammenfassen.

3.3.6 Beitritt in eine Sitzung

In Anwendungsfall aus Abschnitt 3.2.4 wird gefordert, dass Benutzer sich selbst in laufende Sitzungen einladen können. Hierfür muss der Benutzer dem Server eine Anfrage senden, auf die dieser mit einer Einladung reagiert.

3.4 Technische Spezifikation der zu entwickelnden Software

In diesem Abschnitt beschreibe ich die ursprünglich geplante Spezifikation, die sich aus den im vorherigen Abschnitt ausgesuchten Features ergibt. Aus verschiedenen Gründen, die in Abschnitt 4 beschrieben sind, hat sich die Spezifikation im Laufe der Entwicklung an mehreren Stellen geändert.

3.4.1 Server aktivieren

Um den Server aktivieren und deaktivieren zu können, wird in der bestehenden Einstellungsseite „`Saros>Advanced`“ eine Checkbox mit dem Text „`Activate Server`“ (oder einem vergleichbaren) hinzugefügt.

Der Zustand dieses Feldes wird im sogenannten *PreferenceStore* gespeichert, ein von Eclipse bereitgestellter Speicher, der es einem Plug-In ermöglicht, Daten dauerhaft zu speichern[6, S. 230]. Sämtliche anderen Features des Servers können diesen Wert dort auslesen, um festzustellen, ob der Server aktiviert ist oder nicht.

3.4.2 Server bekannt machen

Für die Accountverwaltung benutzt Saros das Nachrichtenprotokoll XMPP. Das Protokoll erlaubt es, eigene Erweiterungen zu definieren und diese über eine so genannte „Service Discovery“[10] abfragen zu lassen. Dazu werden Namespaces definiert, die eine Erweiterung repräsentieren. Neu eingeführte Nachrichtentypen, die zu dieser Erweiterung gehören, werden dem Namspace untergeordnet.

Saros verwendet den Namespace „`de.fu_berlin.inf.dpp`“. Kontakte in der Liste eines Benutzers, die diese Erweiterung unterstützen, sind dementsprechend andere Saros-Instanzen. Um unter diesen Kontakten die Saros-Server zu finden, werde ich den Sub-Namespace „`de.fu_berlin.inf.dpp.server`“ einführen. Um herauszufinden, ob ein Kontakt ein Server ist, muss also über eine *Service Discovery* abgefragt werden, ob er diesen Service unterstützt. Eigene Nachrichtentypen wird der Namespace nicht haben, da alle den Server betreffenden Nachrichten auch dem regulären Saros-Namespace untergeordnet werden können. Technisch ist dies leichter umzusetzen.

3.4.3 Abfrage der Sitzungsinformationen

Um die Sitzungsinformationen des Servers abzufragen, sendet der Benutzer eine Anfrage an den Server. Dieser antwortet mit einer Nachricht, die folgende Informationen enthält: Der Angabe, ob eine Sitzung läuft und wenn ja, einer Liste der geteilten Projekte und einer Liste der Sitzungsteilnehmer. Wenn ein Projekt nur partiell geteilt ist, wird das zusätzlich angegeben.

Angezeigt wird der Status des Servers im neu eingeführten Wizard, der in Abschnitt 3.4.8 beschrieben ist.

Hinweis: Aufgrund von Bedenken bzgl. des Datenschutzes, die in Abschnitt 4.6 thematisiert sind, habe ich diesen Teil der Spezifikation nachträglich abgeändert.

3.4.4 Sitzungsstart

Zum Start einer leeren Sitzung (ohne zusätzliche Teilnehmer und ohne geteilte Projekte) wird eine neue Nachricht eingeführt, die vom Client bei Bedarf an den Server gesendet wird. Die Nachricht enthält die Information, ob es sich um einen Sitzungsstart oder einen -beitritt handelt (siehe dazu 3.4.5). Erhält der Server eine solche Nachricht, die um einen Sitzungsstart bittet, prüft er, ob bereits eine Sitzung läuft. Sollte dies nicht der Fall sein, leitet er entsprechende Schritte, basierend auf der bisherigen Funktionalität von Saros, ein. Er verschickt eine Sitzungseinladung, auf die eine *Session Negotiation* folgt. Wenn der Sitzungsstart nicht möglich ist, verschickt der Server eine Absage, ein ebenfalls neugeschaffener Nachrichtentyp.

Hinweis: Der Prozess des Sitzungsstarts wurde anders umgesetzt, als hier beschrieben (siehe Abschnitt 4.5).

3.4.5 Sitzungsbeitritt

Aus Gründen der Redundanzvermeidung habe ich mich dazu entschieden, für den Beitritt in eine bestehende Sitzung den gleichen Nachrichtentyp, der zum Sitzungsstart benutzt wird, wiederzuverwenden. Beide dienen dazu, den Server darum zu bitten, den Absender in eine Sitzung einzuladen. In einem Fall muss die Sitzung zunächst noch gestartet werden, im anderen nicht.

Wenn der Server eine Nachricht, die um einen Sitzungsbeitritt bittet, erhält und eine Sitzung läuft, verschickt er eine Einladung.

3.4.6 Projekte zur Sitzung hinzufügen

Wenn der Client den Server zum Start einer Sitzung aufgefordert hat, leitet dieser gemäß des in Saros üblichen Prozederes zunächst eine *Session Negotiation* und dann eine *Project Negotiation* ein. Letzteres ist prinzipiell unnötig, da noch keine Projekte geteilt sind, jedoch schadet es, von einem kleinen Zeitaufwand abgesehen, auch nicht. Da der von mir entwickelte Server lediglich ein Prototyp ist, halte ich es noch nicht für nötig, diesen Vorgang zu unterbinden.

Sind beide Vorgänge abgeschlossen, wird eine zweite *Project Negotiation* vom Client angestoßen, in der er die gewünschten Projekte zur Sitzung hinzufügt. Hierzu sind clientseitig kleine Änderungen am Code vorzunehmen, die ihm einen solchen Prozess erlauben. Serverseitig, also auf der Empfängerseite, sind die vorzunehmenden Änderungen größer, da der Empfänger bisher über eine grafische Oberfläche entscheidet, an welchem Ort die gesendeten Dateien gespeichert werden. Hier muss eine zweite Annahmemethode bereitgestellt werden, durch die der Server automatisiert einen Speicherort

im Workspace auswählt.

Hinweis: Statt einer Änderung der Project Negotiation *habe ich mich dazu entschieden eine neue Negotiation einzuführen (siehe Abschnitt 4.8).*

3.4.7 Benutzer zur Sitzung einladen

Nur der Host, dessen Rolle vom Server übernommen wird, kann neue Teilnehmer in eine Sitzung einladen. Weil der Server passiv ist, müssen die Einladungen von den Clients angestoßen werden. Geplant ist daher ein Nachrichtentyp, dem eine Jabber-ID[3] beigelegt ist. Wenn der Server eine solche Nachricht von einem Sitzungsteilnehmer erhält, prüft er, ob ihm der angegebene Kontakt bekannt ist. Wenn ja, lädt er ihn in die laufende Sitzung ein.

Es gibt zwei Anwendungsfälle für diese Einladungen. Nach der zweiten *Project Negotiation* (siehe 3.4.6) sendet der Client automatisch Einladungsanfragen für alle vor Sitzungsstart angegebenen Sitzungsteilnehmer an den Server. Während einer laufenden Sitzung kann jeder Sitzungsteilnehmer andere Kontakte einladen. Verwendet werden dafür dieselben GUI-Elemente, über die ein Host aktuell Kontakte einladen kann.

Hinweis: Die Umsetzung dieses Features wurde von mir nicht fertiggestellt (siehe Abschnitt 4.7).

3.4.8 Benutzeroberfläche für den Sitzungsstart

Für den normalen Sitzungsstart kennt Saros drei Vorgehensweisen. Zwei beinhalten Kontextmenüs. Über einen Rechtsklick auf ein Projekt im Workspace findet man einen Eintrag, der einen die Kontakte auswählen lässt, mit denen man das angeklickte Projekt teilen möchte. Umgekehrt gelangt man durch einen Rechtsklick auf einen Saros-Kontakt zu einem Menüeintrag, der die Auswahl eines Projekts erlaubt, welches zusammen mit diesem Kontakt bearbeiten werden soll. Darüber hinaus gibt es einen Wizard, der einen sowohl die Kontakte, als auch die Projekte auswählen lässt. Für den Start einer Sitzung mit einem Server ist geplant, einen neuen Wizard, der an diesem bestehenden angelehnt ist, einzuführen. Es wird eine neue Seite hinzugefügt, auf welcher der Server ausgewählt wird.

Diese Vorgehensweise ist nicht die komfortabelste für die Benutzer, allerdings ist sie einfach umzusetzen. Da das Ziel dieser Arbeit in erster Linie ist, die Möglichkeiten eines Saros-Servers zu erkunden, und nicht, ein Serverfeature für die Endanwender bereitzustellen, kann dieser Nachteil in Kauf genommen werden.

[3] Adresse eines Kontakts in XMPP

4 Implementierung

4.1 Generelles

Wie in Abschnitt 3.1.4 beschrieben, sind die Teilerfolge meiner Programmierung laufend in den Master-Branch des Saros-Projekts eingepflegt worden. Da ich aus diesem Grund laufend im Austausch mit dem Rest des Entwicklerteams stand, werde ich hier kurz einen Überblick über die wichtigsten Kommunikationstools im Umgang mit der Entwicklung an Saros geben.

Bevor ein Patch in den Master-Branch integriert wird, durchläuft er einen Reviewprozess über das Review-System Gerrit[1]. Das System wird sowohl zur Bewertung des Codes als auch zur Diskussion von Implementationsansätzen verwendet. Darüber hinaus gibt es eine Mailingliste, über die Diskussionen geführt werden können.

4.2 Feature-Toggle

Ein Feature-Toggle ermöglicht es mir, Teile der Funktionalität in das Hauptprojekt zu integrieren ohne die Entwicklung der anderen Mitarbeiter zu stören. Daher sind alle im folgenden beschriebenen Funktionen, die nur aktiv sein sollen, wenn der Server eingeführt ist, über einen solchen Boolean abgekapselt. Dazu habe ich in der Datei, die die Einstellungen für das Saros-Plug-In enthält („`saros.properties`“) die Variable „`de.fu_berlin.inf.dpp.server.SUPPORTED`“ angelegt.

4.3 Aktivierung des Servers in den Einstellungen

Ich habe im *PreferenceStore* einen neuen Boolean unter dem Namen „`server_activated`“ angelegt. Der Wert dieser Variable wird über die Ausführungsdauer des Plug-Ins hinaus gespeichert. Beim Empfang einer Nachricht, die den Server zu einer Aktion auffordert, wird der Zustand dieser Variable überprüft und bei positivem Ergebnis mit dem Bearbeiten der Anfrage begonnen.

Aktiviert wird der Server über die Saros-Einstellungen im Eclipse-Menü. Dort habe ich unter „`Saros > Advanced`“ eine Checkbox hinzugefügt, über die sich der Inhalt der oben genannten Variable ändern lässt. Die Beschriftung der Checkbox lautet „`Activate server (requires reconnect) [experimental]`“ (siehe Abbildung 2).

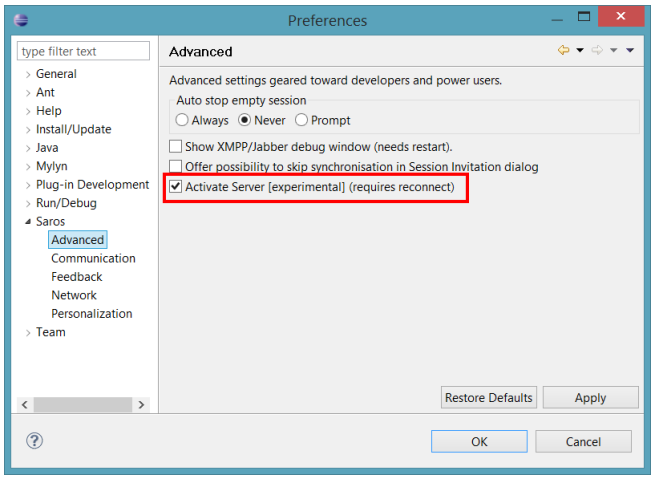

Abbildung 2: Checkbox zur Aktivierung des Servers

4.4 Bekanntmachen des Servers

Um anzuzeigen, dass die Serverfunktion in einer bestimmten Saros-Instanz aktiviert ist, habe ich eine XMPP-Erweiterung, die durch den Namespace „`de.fu_berlin.inf.dpp.server`“ charakterisiert ist, eingeführt.

Wenn ein Benutzer sich in Saros mit einem Konto anmeldet und das Server-Feature aktiviert ist, wird nun, zusätzlich zum bisherigen Dienst, auch der Server-Dienst über XMPP bekannt gegeben. Dieser kann nun über eine sogenannte *Service Discovery*[10] abgefragt werden.

Durch Experimente hat sich gezeigt, dass es nicht zuverlässig funktioniert, den Dienst nach der Anmeldung bekanntzumachen. Daher kann es nötig sein, sich nach der Aktivierung des Servers neu zu verbinden. Entsprechend habe ich der Checkbox, die den Server aktiviert, den Hinweis „`requires restart`“ hinzugefügt (siehe 4.3).

4.5 Starten und Beitritt zu einer Sitzung

Der Typ einer XMPP-Nachricht wird über eine Erweiterung (*Extension*[12, S. 23]), eine Art Anhang, bestimmt. Ich habe in einer ersten Version eine neue Erweiterung für die Bitte um einen Sitzungsstart oder -beitritt eingeführt. Sie trug den Namen *JoinSessionRequestExtension* und enthielt einen Boolean, der zwischen der Bitte um einen Beitritt und einer Bitte um einen Sitzungsstart unterschied. Ich habe beide Szenarien in einem Nachrichtentyp zusammengefasst, da sie sich sehr ähneln und durch die Zusammenlegung Code eingespart werden kann. Nachteile schienen nicht zu entstehen.

Beim Empfang einer solchen Nachricht reagiert der Server auf die Bitte, sofern es möglich ist. Wenn ein Beitritt gefragt ist, muss aktuell eine Sitzung laufen. Ist ein Sitzungsstart gefragt, darf es keine bestehende Sitzung geben. Sollten die nötigen Voraussetzungen nicht gegeben sein, reagiert der Server mit einer Absage, einer Nachricht, die eine neu geschaffene *JoinSessionRejectedExtension* enthält. Erhält der anfragende Nutzer eine solche Nachricht, wird ein Dialogfenster angezeigt, das die Absage mitteilt (siehe Abbildung 3).

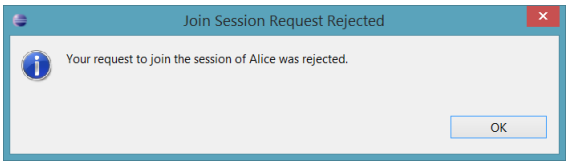

Abbildung 3: Ablehnung einer Einladung zum Sitzungsbeitritt

Der Benutzer löst die Nachricht zum Sitzungsstart über die in Abschnitt 4.9 beschriebene Oberfläche aus.

Ich habe versucht, den Sitzungsbeitritt über einen Eintrag in einem Kontextmenü zu steuern. Bei einem Rechtsklick auf einen Kontakt sollte die Option „`Join server session`“ erscheinen. Sie sollte nur bei Servern aktiviert sein und sonst deaktiviert. Wie sich jedoch gezeigt hat, war dieser Ansatz ungeeignet, da eine *Service Discovery*[10] nötig sein kann, um zu bestimmen, ob der ausgewählte Kontakt ein Server ist. Diese wird asynchron ausgeführt. Da ein Feld eines Kontextmenüs in Eclipse nicht aktiviert oder deaktiviert werden kann, solange das Menü geöffnet ist, kann nicht sichergestellt werden, dass das Feld bei Servern aktiviert und sonst deaktiviert ist. Aus diesem Grund habe ich beschlossen, auch diese Aktion in einen Wizard zu verlagern, es zeitlich jedoch nicht geschafft, diese Oberfläche zu erstellen.

Im Rahmen der *Server Session Negotiation* (siehe 4.8) war es nötig, einen eigenen Nachrichtentyp für die Bitte um einen Sitzungsstart einzuführen, da die Übertragung zusätzlicher Daten notwendig wurde. Daher ist die *JoinSessionRequestExtension* jetzt nur noch für den Beitritt in eine laufende Sitzung zuständig. Der enthaltene Boolean wurde entfernt.

4.6 Serverinformationen abfragen

Um den Benutzern eine Übersicht über die auf einem Server laufende Sitzung zu geben, habe ich eine Statusabfrage-Funktion eingeführt. Über diese sollte ein Client die Information erhalten, ob auf dem Server eine Sitzung läuft, und wenn ja, welche Personen und Projekte daran beteiligt sind. Dazu sollte eine Nachricht mit einer neu geschaffenen *SessionStatusRequestExtension* an

den Server geschickt werden. Dieser wiederum sollte mit einer *SessionStatus-ResponseExtension* antworten, die jeweils eine Liste mit Projektnamen und Jabber-IDs der Teilnehmer enthält.

Von anderen Entwicklern wurde ich darauf hingewiesen, dass das Mitteilen der beteiligten Personen aus Gründen des Datenschutzes bedenklich ist. Da ich den Einwand für gerechtfertigt hielt – schließlich haben die Nutzer keine Kontrolle darüber, wem der Server, auf dem sie an einer Sitzung teilnehmen, Auskunft erteilt – habe ich die Liste der Teilnehmer aus der Antwort des Servers entfernt. Außerdem wurde mir vorgeschlagen, statt der Liste der Projektnamen eine Beschreibung der Sitzung zu senden. Inhalt dieser Beschreibung solle zunächst nach wie vor die Liste der Projekte sein. Durch ein solches Vorgehen lassen sich möglicherweise später Inkompatibilitäten zwischen verschiedenen Saros-Versionen vermeiden. Diese treten immer dann auf, wenn die Art der Daten in den versendeten Nachrichten, also Menge oder Datentyp der enthaltenen Felder, geändert werden. Wenn man z. B. in der Zukunft die Anzahl der Teilnehmer in die Statusinformation einschließen möchte, erfordert dies nur noch eine Änderung des Beschreibungstextes und nicht das Hinzufügen eines zusätzlichen Feldes, welches diese Information enthält.

Angezeigt werden soll die Sitzungsinformation in Wizards zum Sitzungsstart und -beitritt. Zum Zeitpunkt der Einreichung der Arbeit war dies noch implementiert.

4.7 Einladung von Benutzern in eine Sitzung

Zum Einladen von Kontakten in die Sitzung war vorgesehen, dass der Client eine Nachricht an den Server sendet. Dieser führt die Einladung aus, sofern er den Kontakt in seiner Kontaktliste gespeichert hat.

Dazu habe ich eine *InvitationRequestExtension* erstellt, die als Nachricht an den Server geschickt werden kann, um diesen zur Einladung des Kontaktes, dessen Jabber-ID in der Nachricht enthalten ist, auszuführen.

Ausgelöst werden kann diese Nachricht über dieselben Wege, die man in einer klassischen Sitzung nutzen kann, um Kontakte einzuladen. Diese sind zum einen ein Eintrag im Kontextmenü, den man erreicht, indem man mit der rechten Maustaste auf einen Kontakt in der *Saros-View* klickt, der kein Teilnehmer der Sitzung ist, und zum anderen ein Wizard, der über das Eclipse-Menü zu erreichen ist.

Bisher hatte ich noch keine Maßnahmen unternommen, um klassische Saros-Sitzungen von Serversitzungen zu unterscheiden. Dies war nicht nötig, da keines der eingeführten Features das Verhalten der Teilnehmer in der Sitzung beeinflusst hat. Da das nun eingeführte Feature nur zugänglich sein

soll, wenn die Sitzung von einem Server geleitet wird, habe ich der Klasse *SarosSession* eine neuen Variable, die genau diese Information speichert, hinzugefügt. In der *Session Negotiation*, die einen Teilnehmer in die Sitzung einfügt, teilt der Host dem Client mit, ob es sich um eine Serversitzung handelt oder nicht.

Hinweis: Zum Zeitpunkt des Abschlusses dieser Arbeit ist dieses Feature nicht im Master-Branch von Saros integriert. Grund dafür ist, dass es Uneinigkeit innerhalb des Entwicklerteams darüber gab, ob das Feature überhaupt nötig ist, schließlich ist die Selbsteinladung möglich. Aus Zeitgründen habe ich mich daher auf die folgenden Features konzentriert.

4.8 Server Transfer Negotiation

Meinen ursprünglichen Plan, die *Project Negotiation* für den Initiator der Sitzung zu öffnen, um die Projekte hinzuzufügen, habe ich nach einer Diskussion mit dem Entwicklerteam verworfen. Zum einen wurde bemängelt, dass ein solcher Einsatz den Ablauf, den Teilnehmer beim Eintritt in eine Sitzung durchlaufen, verletzen würde (vgl. *User States*[3][13]). Zudem wurde argumentiert, dass es aus architektonischer Sicht unklug sei, die *Project Negotiation* vom Client aus zu starten. Es durchbricht die gewünschte Client-Server-Architektur.

Alternativ zur *Project Negotiation* bestand die Möglichkeit, die Projekte vor dem Sitzungsstart an den Server zu übermitteln. Ich habe mich für dieses Verfahren, im Folgenden *Server Transfer Negotiation* genannt, entschieden, weil es zum einen die beschriebenen Probleme umgeht und weil zum anderen das Verfahren auch auf ein anderes offenes Probleme anwendbar ist. Es kann verwendet werden, um Clients in einer laufenden Sitzung die Möglichkeit zu geben, Projekte hinzuzufügen (siehe 5.3.1).

Ein Client bittet um den Beginn einer *Server Transfer Negotiation*, indem er eine Nachricht mit einer *StartSessionRequestExtension* an den Server sendet. Diese enthält Angaben über die geteilten Projekte, deren Namen, die Liste der enthaltenen Dateien und der Information, ob bzw. welche der Projekte partiell geteilt werden sollen (*Partial Sharing*). Wenn auf dem Server aktuell keine Sitzung läuft, tritt auch er in die *Server Transfer Negotiation* ein und teilt dies dem Client mit, wenn nicht, erteilt er eine Absage, was beim Client zum Abbruch des Prozesses führt. Die Antwort erfolgt in Form einer *StartSessionAnswerExtension*, die einen Boolean mit der Antwort enthält.

Hat der Client eine Zusage erhalten, verpackt er analog zur *Project Negotiation*, in der dieser Prozess ähnlich abläuft, die zu teilenden Dateien projektweise in ZIP-Archive und packt anschließend die entstandenen Archive in ein übergeordnetes Archiv. Dies hat den Vorteil, dass nur eine Datei gesendet werden muss und dass die Projekte beim Empfang sequenziell gespeichert

werden können, da jeweils der Inhalt eines Archivs den Inhalt eines Projekts ergibt.

Nachdem das Archiv per *FileTransferJob* an den Server gesendet wurde, entpackt dieser die Dateien und speichert sie im lokalen Workspace. Die Namen der Projekte sind mit den lokalen Versionen des Clients identisch. Wenn beim Server schon gleichnamige Projekte existieren, wird deren Inhalt zunächst gelöscht.

Nachdem dies geschehen ist, erstellt der Server eine Sitzung mit den soeben erhaltenen Projekten und dem Client als weiteren Teilnehmer. Damit ist die *Session Transfer Negotiation* abgeschlossen (siehe Abbildung 4).

Hinweis: Zum Zeitpunkt des Abschlusses dieser Arbeit ist dieses Feature nicht im Master-Branch von Saros integriert.

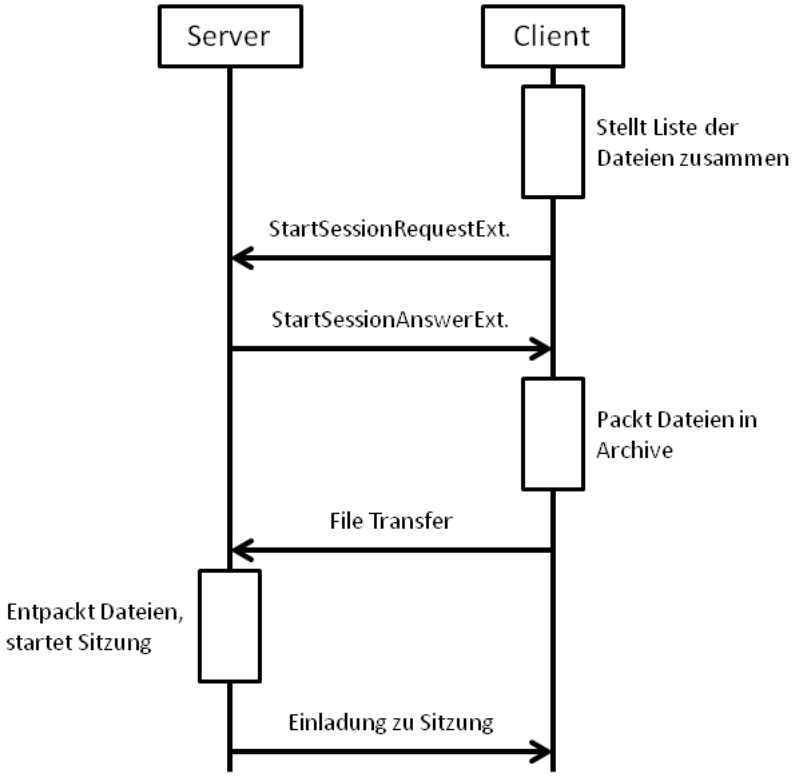

Abbildung 4: Session Transfer Negotiation

4.9 Benutzeroberfläche zum Sitzungsstart

Zum Starten einer Serversitzung habe ich einen Wizard eingeführt, der drei Seiten hat. Auf der ersten wählt man den Server aus, auf dem die Sitzung laufen soll. Die zweite dient dazu, die Projekte auszuwählen, die in der Sitzung geteilt werden sollen. Auf der dritten Seite kann man weitere Sitzungsteilnehmer auswählen, die nach Sitzungsstart eingeladen werden sollen.

Zur Zeit des Abschlusses dieser Arbeit war die dritte Seite noch nicht erstellt, da das Einladen von Sitzungsteilnehmern ebenfalls nicht möglich war (siehe 4.7).

Auf der ersten Seite ist eine Liste der Server zu sehen (siehe Abbildung 5). Um zu erkunden, welche der Kontakte in der Kontaktliste das Serverfeature aktiviert haben, wird jeweils eine *Service Discovery* gestartet, um dies herauszufinden. Zum Zeitpunkt des Abschlusses dieser Arbeit wurde dort noch nicht angezeigt, ob auf dem jeweiligen Server eine Sitzung läuft.

Hat der Benutzer einen der Server ausgewählt, kann er fortfahren und gelangt zur nächsten Seite. Diese entspricht exakt der entsprechenden Seite des Wizards zum Start einer klassischen Saros-Sitzung (siehe Abbildung 6).

Nachdem der Benutzer mindestens eine Datei ausgewählt hat, kann er über den Button „Finish“ den Wizard schließen und leitet damit die *Server Transfer Negotiation* ein. Ist diese erfolgreich abgeschlossen, erhält er eine Einladung in die erstellte Sitzung.

Zu erreichen ist der Wizard im Eclipse-Menü unter „`Saros > Share via server...`“

Hinweis: Zum Zeitpunkt des Abschlusses dieser Arbeit ist dieses Feature noch nicht vollendet worden.

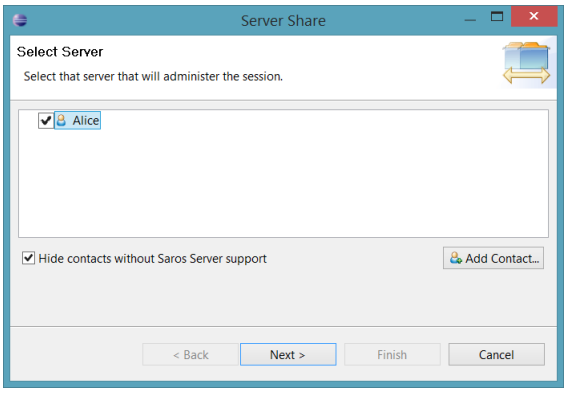

Abbildung 5: Auswahl des Servers

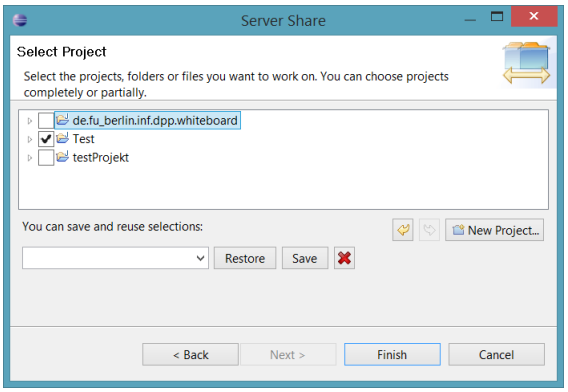

Abbildung 6: Auswahl der Projekte

5 Auswertung

5.1 Probleme mit den implementierten Features

Bei der Entwicklung eines Prototypen ist es normal, dass unvorhergesehene Probleme auftauchen. So auch bei der Entwicklung des Server-Prototypen. In diesem Abschnitt beschreibe ich die Probleme, die ich bei den von mir entwickelten Features sehe. Diese beziehen sich auf den Stand zum Zeitpunkt des Abschlusses der Arbeit.

5.1.1 Bekanntmachen des Servers

In Experimenten haben sich zwei Probleme gezeigt, die bei meiner Implementierung des Features zum Bekanntmachen des Servers auftauchen (siehe 4.4). Zum einen ist es nötig, sich nach dem Aktivieren des Servers in den Einstellungen neu mit seinem Account zu verbinden. Zum anderen hat die *Service Discovery* zum Teil relativ lange gedauert (>10 Sekunden). Das Senden einer einfachen Nachricht (z. B. eine Einladung in eine Sitzung) schien in der Regel schneller zu gehen. Wenn sich dieser Eindruck in weiteren Tests bestätigt, dann muss man daraus den Schluss ziehen, dass der von mir gewählte Ansatz nicht optimal ist. Siehe dazu 5.2.12.

5.1.2 Server Transfer Negotiation

Mit der Arbeit an der *Server Transfer Negotiation* bin ich nicht fertig geworden, daher fehlen hier eine Reihe von Sicherheitsmechanismen, die sicherstellen, dass der Prozess beim Auftreten von Fehlern korrekt abgebrochen wird.

Darüber hinaus werden bei jedem Sitzungsstart alle Dateien eines Projektes an den Server gesendet. Analog zur *Project Negotiation* wäre es jedoch möglich, lediglich die Dateien zu übertragen, die auf dem Server nicht vorhanden sind. Gerade bei großen Projekten kann dabei viel Zeit gespart werden. Siehe dazu 5.2.8.

5.2 Mögliche Verbesserungen am Saros-Server

Es war nie geplant, meinen Entwurf des Servers in der von mir entwickelten Form zu veröffentlichen. Stattdessen war es meine Aufgabe, die Grundlage für nachfolgende Entwickler zu legen. Dazu gehörte auch, Probleme aufzudecken und Vorschläge für weitere Features zu machen, die ich in meiner Arbeit noch nicht umsetzen kann.

Die folgenden Listen von Features in diesem und dem folgenden Abschnitt sind daher als Vorschlag und Ansatzpunkt für die Weiterentwicklung zu verstehen. Die einzelnen Punkte sind nicht eingehend auf Nutzen oder Umsetzbarkeit untersucht worden.

Wenn nicht anders angegeben, können die einzelnen Punkte unabhängig von den anderen umgesetzt werden.

Ich habe versucht, die Priorität der beschriebenen Erweiterungen zu beurteilen. Diese Einschätzungen sind lediglich als Vorschläge zu verstehen. Priorität *sehr hoch* bedeutet, dass das Feature vor einer Veröffentlichung unbedingt umgesetzt werden muss, *hoch*, dass das Feature aus meiner Sicht zur Grundausstattung des Server gehört. Erweiterungen mit Priorität *mittel* und *niedrig* sind zunächst optional.

5.2.1 Server Transfer Negotiation beenden

Priorität: *sehr hoch*

Ich habe es nicht geschafft, die Entwicklung der *Server Transfer Negotiation* zu beenden. Daher gehört es zu den ersten Aufgaben bei der Weiterentwicklung des Servers, sie fertigzustellen.

Ein großer Teil der folgenden Features sind von diesem abhängig.

5.2.2 Server aus dem Eclipse-Plug-In herauslösen

Priorität: *mittel*

Eclipse ist relativ schwergewichtig, beansprucht somit viel Arbeitsspeicher und CPU-Zeit. Daher ist es sinnvoll, den Server als eigenständiges Programm zu realisieren, das ressourcensparender ist. Da parallel zu und nach Abschluss dieser Arbeit daran gearbeitet wird, Saros von Eclipse zu trennen, um es für weitere Entwicklungsumgebungen wie IntelliJ bereitstellen zu können, wird es nach Abschluss dieser Trennung möglich sein, einen Stand-Alone-Server zu entwickeln. Dieser kann beispielsweise ein kleines Kommandozeilenprogramm sein, der auf den Saros-Core zugreift.

Dieses Feature macht 5.2.10 überflüssig.

5.2.3 Sitzungen beenden

Priorität: *hoch*

Zurzeit können vom Server geleitete Sitzungen nicht von den Clients gestoppt werden. Dies kann ausschließlich lokal in der Eclipse-Instanz des Servers getan werden. Es ist denkbar, dass dieses Recht allen Kontakten des

Servers oder nur dem Initiator der Sitzung gegeben wird.

5.2.4 Mehrere Sitzungen pro Server erlauben

Priorität: *hoch*

Es ist erwünscht, dass ein Server mehrere Sitzungen gleichzeitig verwalten kann. Ich musste die Einschränkung, bloß eine Sitzung zu unterstützen, in Kauf nehmen, da die Festlegung auf eine Sitzung zu tief in Saros verankert ist. Prinzipiell sollte es jedoch keine unlösbaren Probleme bei der Einführung dieses Features geben. Nachdem dieses Feature für den Server eingeführt wurde, ist es denkbar, es auf das reguläre Saros-Plug-In zu erweitern, sodass die Benutzer ebenfalls an mehreren Sitzungen parallel teilnehmen können.

5.2.5 Weitere Benutzer beim Start angeben

Priorität: *mittel*

Beim bisherigen Sitzungsstart ohne Server ist es möglich, Mitarbeiter auszuwählen. Es ist für den Serverbetrieb nicht nötig, dies zu ermöglichen, allerdings erhöht es die Benutzbarkeit.

Dieses Feature ist von 5.3.3 abhängig.

5.2.6 Server nicht als Sitzungsteilnehmer behandeln

Priorität: *mittel*

Der Server wird aktuell in der Liste der Sitzungsteilnehmer dargestellt, er belegt eine der fünf Sitzungsfarben, man kann ihm im *Follow Mode* folgen. Solange der Server Teil des Saros-Plug-Ins ist, kann man über seinen Editor die geteilten Dateien manipulieren und wenn eine Datei in seinem Eclipse-Fenster geöffnet ist, können andere Sitzungsteilnehmer dies sehen. All das widerspricht der Funktion des Servers und es sollte in Erwägung gezogen werden, dies entsprechend zu ändern.

5.2.7 Lokalen Stand in laufende Sitzung einbinden

Priorität: *niedrig*

Einer der Vorteile, die ein Server bei der Koordinierung von Sitzungen hat, ist, dass Sitzungen nicht unterbrochen werden müssen, wenn einige oder alle Teilnehmer sie verlassen. Dies bringt allerdings das Problem mit sich, dass ein Mitarbeiter immer Teil der Sitzung sein möchte, wenn er an dem Projekt arbeitet. Wenn er außerhalb der Sitzung Änderungen vornimmt und dann in die Sitzung eintritt, wird der lokale Stand der Arbeit vom Server

überschrieben. Daher ist es denkbar, die Option, beim Sitzungseintritt den lokalen Stand auf den Server zu übertragen, einzuführen.

Den Nutzen in klassischen Sitzungen schätze ich gering ein, da die Sitzungen hier nicht ununterbrochen laufen. Zudem besteht das Problem, dass der lokale Stand des Host überschrieben werden würde. Anders als in einer Server-Sitzung ist dieser nicht bloß eine Kopie der Arbeit eines anderen Mitarbeiters, sondern kann einzigartige Fortschritte enthalten.

Dieses Feature ist von 5.2.1 abhängig.

5.2.8 Server Transfer Negotiation effizienter gestalten

Priorität: *hoch*

In der *Server Transfer Negotiation* werden alle Dateien des Projekts, das in der Sitzung geteilt werden soll, übertragen. Wenn der Server in der Vergangenheit bereits eine das Projekt enthaltende Sitzung geleitet hat, ist ein Speichen der Dateien auf dem Server denkbar, sodass sie später wiederverwendet werden können. Dann ist es nur noch nötig, veränderte Dateien auf den neuesten Stand zu bringen. Dies kann insbesondere bei größeren Projekten nützlich sein.

Der beschriebene Mechanismus wird in der *Project Negotiation* bereits verwendet.

Dieses Feature ist von 5.2.1 abhängig.

5.2.9 Dialoge bei Sitzungsbeginn/-beitritt überarbeiten

Priorität: *niedrig*

Ich habe den Prozess des Sitzungsstarts, also die *Session Negotiation* und *Project Negotiation* nicht abgeändert, sondern nur ergänzt. Daraus resultiert, dass einige der Dialoge nicht passend sind.

Wenn man den Server auffordert, eine Sitzung zu starten, schickt dieser eine Einladung. Dem Client wird daraufhin ein Dialog angezeigt, der ihm anbietet, diese Einladung anzunehmen oder abzulehnen. Dieses Dialogfenster ist überflüssig. Ein Ablehnen der Einladung ergibt nur in den allerseltensten Fällen Sinn, z. B. wenn der Benutzer die Sitzung lediglich für die spätere Benutzung zur Verfügung stellen möchte, jedoch selbst nicht vorhat teilzunehmen. Doch selbst in diesem Fall bietet dieses Dialogfenster dem Benutzer keinen Vorteil, da ein Abbrechen der *Project Negotiation* denselben Effekt hat. Der gleiche Dialog erscheint, wenn ein Benutzer sich selbst in eine Sitzung einlädt. Auch hier ist er überflüssig.

In der *Project Negotiation* werden die Benutzer aufgefordert, den Ort aus-

zusuchen, an dem die in der Sitzung geteilten Projekte gespeichert werden sollen. Standardmäßig wird eine Kopie erstellt, wenn bereits ein gleichnamiges Projekt existiert. In Sitzungen, die nicht serverbasiert sind, ist dies eine sinnvolle Wahl, da sich dadurch ein versehentliches Überschreiben der Projekte verhindern lässt. Beim serverbasierten Sitzungsstart hat der Client das betreffende Projekt soeben erst an den Server gesendet. Daher wird es nur in den allerseltensten Fällen Unterschiede zwischen dem lokalen Projekt und dem des Servers geben, nämlich nur dann, wenn der Benutzer es in der kurzen Zwischenzeit geändert hat. Darüber hinaus werden die Benutzer in der Regel ihr lokales Projekt in der Sitzung verwenden wollen. Daher ist es aus meiner Sicht sinnvoll, in der *Project Negotiation* beim Starten des Servers die Standardauswahl auf das lokale Projekt zu ändern.

5.2.10 Umgang mit leeren Sitzungen

Priorität: *niedrig*

In den Saros-Einstellungen können Nutzer auswählen, wie ihre Saros-Instanz sich verhalten soll, wenn sie in einer Sitzung Host sind und alle sonstigen Teilnehmer die Sitzung verlassen haben (siehe Abbildung 7). Sie können auswählen zwischen automatischem Beenden, dem Erscheinen eines Dialogfensters, das nachfragt, was getan werden soll, und keiner Aktion. Wenn das Server-Feature aktiviert ist, ergibt nur letzteres Sinn. Daher könnte die Einstellung automatisch auf „Keine Aktion“ gesetzt werden oder es könnte, wenn nötig, ein Dialog erscheinen, der darauf hinweist, dass die aktuellen Einstellungen zu unsinnigem Verhalten führen.

Dieses Feature ist überflüssig, wenn 5.2.2 umgesetzt ist.

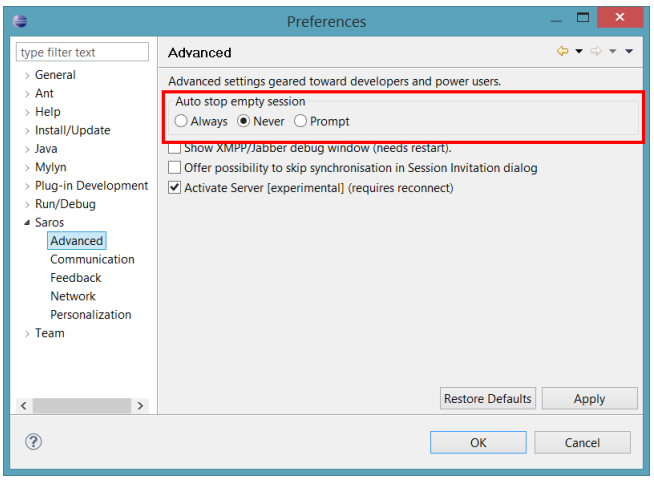

Abbildung 7: Umgang mit leeren Sitzungen

5.2.11 Sichtbarkeit des Servers in der Saros-View

Priorität: *niedrig*

Aktuell werden Server in der *Saros-View* genau wie andere Kontakte dargestellt. Besser wäre es, sie als solche erkenntlich zu machen. Denkbar wäre, sie farblich anders darzustellen oder mit einem eigenen Symbol zu versehen.

5.2.12 Alternative Bekanntmachung des Servers

Priorität: *niedrig*

Wenn sich bei den in Abschnitt 5.1.1 beschriebenen Tests zeigt, dass der von mir gewählte Ansatz nicht optimal ist, sollte ein anderer Mechanismus gewählt werden. Zum Herausfinden, ob ein Kontakt ein Server ist, könnten Clients ihren Kontakten bei Bedarf eine Nachricht schicken, auf die diese dann entsprechend antworten.

5.3 Mögliche Verbesserungen an Saros

Im Rahmen meiner Arbeit bin ich auf mögliche zusätzliche Features für den Server gestoßen, von denen auch das klassische Saros profitieren kann. Sie sind im Folgenden beschrieben.

5.3.1 Als Client Projekte zu einer Sitzung hinzufügen

Priorität: *mittel*

In älteren Saros-Versionen konnte jeder Sitzungsteilnehmer während einer Sitzung weitere Dateien oder Projekte zur Sitzung hinzufügen. Da das jedoch zu Inkonsistenzen, also unterschiedlichen Dateiinhalten bei den Sitzungsteilnehmern, führen konnte, wurde das Feature von Patrick Schlott entfernt.[13] Jetzt ist ausschließlich dem Host dieses Recht gewährt.

Das Feature kann jedoch wieder eingeführt werden, indem das zu teilende Projekt während der Sitzung mit einer *Server Transfer Negotiation* (siehe 4.8) zum Server bzw. Host gesendet wird und dieser es dann zur Sitzung hinzufügt. Bei diesem Vorgang können die bemängelten Inkonsistenzen nicht auftreten.

Dieses Feature ist von 5.2.1 abhängig.

5.3.2 Projekte aus Sitzung entfernen

Priorität: *niedrig*

Dieses Feature ist sowohl für reguläre als auch Server-Sitzungen nützlich, aber ist bei serverbasierten Sitzungen weitaus angebrachter, da hier die Sitzungen darauf angelegt sind, besonders lang ohne Unterbrechung zu laufen. Bisher ist es jedoch nötig, die Sitzung neu zu starten, wenn man die Menge der in der Sitzung geteilten Dateien verkleinern möchte.

Es ist denkbar, entweder dem Initiator der Sitzung oder jedem Teilnehmer das Recht zu geben, den Server aufzufordern, ein Projekt oder einige Dateien aus der Sitzung zu entfernen. Dieser leitet diese Information dann an alle Beteiligten weiter. In Sitzungen, die nicht serverbasiert sind, fiele dieses Recht dem Host zu.

5.3.3 Als Client Einladungen verschicken

Priorität: *mittel*

In den bisherigen Sitzungen hat der Host die Möglichkeit, in einer laufenden Sitzung weitere Personen einzuladen, anderen Sitzungsteilnehmern war dies nicht möglich. In der ursprünglichen Spezifikation für den Server-Prototypen war angedacht, das Recht andere einzuladen auf alle Teilnehmer auszuweiten. (siehe 3.4.7). Jedoch ist das Feature aus Gründen, die in Abschnitt 4.7 beschrieben sind, aktuell nicht enthalten.

Sollte das Entwicklerteam sich dazu entschließen, auch in Serversitzungen Einladungen zu erlauben, kann es dazu meine Vorarbeit, die in Gerrit zu finden ist[5], verwenden.

Der von mir verwendete Mechanismus kann auch dazu verwendet werden, um Nicht-Hosts in regulären Sitzungen die Möglichkeit, Einladungen zu verschicken, geben. Dort wurde es ihnen entzogen, da in der folgenden *Project Negotiation* Inkonsistenzen entstehen konnten.[14]

Dieses Feature ist von 5.2.1 abhängig.

5.4 Ausblick

Die Entwicklung des Servers wird, vermutlich von anderen Entwicklern, fortgeführt werden. Da ich es nicht geschafft habe, alle von mir geplanten Features zu implementieren, muss zuerst die Entscheidung getroffen werden, ob diese fertiggestellt oder durch andere Implementierungen ersetzt werden. Danach kann der Server durch die in Abschnitt **??** beschriebenen Features erweitert werden. Mittelfristiges Ziel sollte aus meiner Sicht entweder sein, den Server als experimentelles Feature im Saros-Plug-In zu veröffentlichen oder, wenn die Trennung von Saros und Eclipse schnell genug geschieht, die Veröffentlichung eines eigenständigen Saros-Servers, der nicht Teil des Plug-Ins ist.

Literatur

[1] Gerrit Code Review. `https://code.google.com/p/gerrit/`. Stand: 16.02.2014.

[2] Saros - Session Negotiation. `http://www.saros-project.org/sessionNegotiation`. Stand: 16.01.2014.

[3] Saros - User States. `http://www.saros-project.org/userStates`. Stand: 17.02.2014.

[4] J. Arnowitz, M. Arent, and N. Berger. *Effective Prototyping for Software Makers*. Interactive Technologies. Elsevier Science, 2010.

[5] Nils Bussas. [FEATURE] Let clients add users to server based session. `http://saros-build.imp.fu-berlin.de/gerrit/#/c/1325/`, December 2013. Stand: 15.01.2014.

[6] B. Daum. *Professional Eclipse 3 for Java Developers*. Wrox professional guides. Wiley, 2006.

[7] Martin Fowler. Continuous Integration. `http://martinfowler.com/articles/continuousIntegration.html`, May 2006.

[8] Martin Fowler. FeatureToggle. `http://martinfowler.com/bliki/FeatureToggle.html`, October 2010. Stand: 14.01.2014.

[9] Pete Goodliffe. *Code Craft: The Practice of Writing Excellent Code*. No Starch Press, 2006.

[10] Joe Hildebrand, Peter Millard, Ryan Eatmon, and Peter Saint-Andre. Xep-0030: service discovery. *XMPP Standards Foundation, Tech. Rep*, 2008.

[11] Hooman Hoodat and Hassan Rashidi. Classification and analysis of risks in software engineering. *World Academy of Science, Engineering and Technology*, 56:446–452, 2009.

[12] P. Saint-Andre, K. Smith, and R. Tronçon. *XMPP: The Definitive Guide*. Definitive Guide Series. O'Reilly Media, 2009.

[13] Patrick Schlott. Analyse und Verbesserung der Architektur eines nebenläufigen und verteilten Softwaresystems. Master's thesis, Freie Universität Berlin, Inst. für Informatik, 2013.

[14] Sebastian Starroske. Improving the reliability of Saros using Root Cause Analysis. Master's thesis, Freie Universität Berlin, Inst. für Informatik, 2013.