Freie Universität Berlin

Arbeitsgruppe

Software Engineering

Verbesserte Präsenz durch Screensharing für ein Werkzeug zur verteilten Paarprogrammierung

Stephan Lau

Matrikelnummer: 4134110

lau@inf.fu-berlin.de

Betreuer:

Stephan Salinger und Karl Beecher

Eingereicht bei:

Prof. Dr. Lutz Prechelt

30. April 2010

Eidesstattliche Erklärung

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

Berlin, den 30. April 2010

Stephan Lau

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API *Application Programming Interface*, dt. Schnittstelle zur Anwendungsprogrammierung, bezeichnet die Schnittstelle die von einem Softwaresystem anderen Programmen zur Anbindung verfügbar gemacht wird

ARM *Assistant Releasemanager*

CB Community Bar

CSCW *Computer Supported Cooperative Work* oder *Computer Supported Collaborative Work* (CSCW), dt. computerunterstützte oder rechnergestützte Gruppenarbeit ist ein interdisplinäres Forschungsgebiet welches untersucht wie Individuen in Gruppen zusammen arbeiten und dabei durch Informations- und Kommunikationstechnologie unterstützt werden können

DPP verteilte Paarprogramierung, engl. distributed pair programming

FMS *Flash Media Server* ist ein Streaming-Server von Adobe Systems Inc.

FPS *Frames per Second*, Aktualisierungsrate der Bilder eines Videos

IBB *In-Band Bytestreams*, eine Möglichkeit Binärdaten über XMPP auszutauschen[KSA09]

MMX *Multi Media Extension*

MVC *Model View Controller* (dt. Model, Präsention, Steuerung) ist ein Architekturmuster zur Strukturierung von Software[GHJV09]

P2P *Peer-to-Peer*, dt. Rechner-Rechner-Verbindung

PP Paarprogrammierung, engl. pair programming

RM *Releasemanager*

RTP *Real-Time Transport Protocol*

SSE *Streaming SIMD Extensions*

SSF *Stream Service Framework*, siehe Abschnitt 7.1

TCP *Transmission Control Protocol*

UDP *User Datagram Protocol*

VBV *Video Buffer Verifier*

VoIP *Voice over IP*, dt. IP-Telefonie

VSF *Videosharing Framework*, siehe Kapitel 6

XP Extreme Programming, dt. Extremprogrammierung, ist ein bekanntes Vorgehensmodell in der Softwaretechnik

XMPP *Extensible Messaging and Presence Protocol*

1 Einleitung

Die hier vorliegende Bachelorarbeit beschäftigt sich mit der Erweiterung von Saros um *Screensharing*. Saros ist ein Plugin für Eclipse und ist an der Freien Universität Berlin – Arbeitsgruppe Softwareengineering des Fachbereichs Informatik entstanden und wird dort auch kontinuierlich weiterentwickelt. Es ist ein Werkzeug für verteilte Paarprogramierung (DPP) mit dem es mehreren Entwicklern ermöglicht wird an einem Projekt von Eclipse gemeinsam zu arbeiten.

Das Hauptproblem bei einem Werkzeug für DPP ist die Gestaltung eines kollaborativen Arbeitsumfeldes, denn im Gegensatz zur klassischen Paarprogrammierung (PP) sind die Entwickler nicht am gleichen Ort. Daher muss es ihnen klar ersichtlich sein *was* die Anderen *wo* tun, die sogenannten Präsenzinformationen. Diese sind schon an vielen Stellen vorhanden, doch es gibt noch Lücken die geschlossen werden müssen. Es existieren einige Erweiterungen bzw. Funktionen von Eclipse, deren Zustand noch nicht vermittelt werden kann. Zudem entziehen sich Werkzeuge und Artefakte, mit denen ausserhalb von Eclipse gearbeitet wird, ganz der Aufmerksamkeit der anderen Entwickler.

An diesen Stellen setzt *Screensharing* an, um dort die vermittelte Präsenz zu verbessern und dadurch einen transparenteren Arbeitsprozess und bessere Kollaboration zu ermöglichen.

1.1 Ziele

Diese Arbeit hat das konkrete Ziel Saros dahingehend zu erweitern, dass *Screensharing* in einer Saros-Sitzung möglich ist. Dabei steht es frei, wie dies umgesetzt wird. Es existieren schon diverse fertige Lösungen für *Screensharing*, deswegen geht es hier vorrangig darum die Möglichkeit zu schaffen diese auch einfach integrieren zu können, sowie auch eine mögliche Lösung zu implementieren.

Weiterhin muss noch geklärt werden, ob und inwiefern *Screensharing* den Arbeitsprozess mit Saros verbessert.

1.2 Aufbau dieser Arbeit

Bevor es um die Umsetzung der gestellten Aufgabe geht, ist es wichtig zu verstehen was Saros ist und auf welchen Konzepten es basiert. Dazu klärt Kapitel 2 über die konzeptuellen Grundlagen von Saros auf, also die Programmiertechniken PP und DPP und wie, rein technisch gesehen, Kollaboration mit Hilfe von Computern umgesetzt werden kann.

Kapitel 3 stellt dann Saros im Detail vor, wie damit gearbeitet wird und welche Probleme dabei entstehen können, um damit die Motivation für diese Arbeit zu begründen.

Danach werden ähnliche Arbeiten in Kapitel 4 vorgestellt, bevor die konkreten Anforderungen an die *Screensharing*-Implementierung in Kapitel 5 beschrieben werden.

Ein Framework zur Übertragung beliebiger Bildsequenzen bildet die Basis für die Umsetzung der gestellten Aufgabe. Nachdem das Design und die Spezifikation dieses Frameworks in Kapitel 6 erläutert wurde, beschreibt Kapitel 7 wie damit *Screensharing* in Saros integriert wurde.

Abschließend gebe ich in Kapitel 8 einen Überblick über meine Arbeit im Saros Projekt und werde in Kapitel 9 noch die ausstehenden Tätigkeiten und mögliche Erweiterungen meiner Implementierung vorstellen.

1.3 Notationen

Englische Begriffe, soweit nicht in die deutsche Sprache übernommen[1], werden *hervorgehoben* und groß geschrieben.
Klassennamen, Methodensignaturen etc. sind in `Schreibtmaschinenschrift` gesetzt. Wenn kein Namensraum bei einem Klassennamen angegeben wird, ist davon ausgehen, dass sie sich in dem Namensraum `de.fu_berlin.inf.dpp` oder darunter befindet und dort eindeutig ist.

1 deutsche Wörter laut Duden sind z. B. Host, Client, Desktop, Encoder, Bug [Kra07]

2 Grundlagen

In diesem Kapitel werden die für diese Arbeit nötigen Grundlagen erklärt. Dazu werde ich eine kurze Einführung in die Programmierpraktiken PP und DPP geben.

Weiterhin gehe ich auf die Möglichkeiten der technische Umsetzung verteilter Kollaboration und deren Vergleich ein.

2.1 Paarprogrammierung

Entwickler können alleine oder aber auch zu zweit programmieren. Eine der bekanntesten Konzepte hierzu ist PP. Es stammt aus Extreme Programming (XP) und ist dort eine der traditionellen Praktiken [BA04].

Es gibt eine feste Rollenverteilung bei PP. Ein Programmierer nimmt die Rolle des *Drivers* ein, d.h. er übt die eigentliche Arbeit am Rechner aus, während der andere Entwickler, der *Observer*, ihn beobachtet. Dabei kann er das Vorgehen nochmals bedenken und ggf. kritisch hinterfragen. Dies gewährleistet eine ständige Überprüfung der Arbeitsergebnisse.

Die Rollen müssen nicht festgelegt sein, es empfiehlt sich sogar sie regelmäßig zu wechseln [WK02].

2.1.1 Vorteile

Es erscheint zunächst reine Zeitverschwendung wenn ein Entwickler dem anderen nur beim Programmieren zuschaut, doch laut [CW01] hat PP positiven Einfluß auf die Qualität von Code und Entwurf sowie deren Wartbarkeit.

Dies wird bedingt durch folgende Aspekte [Wil00]:

Pair Reviews Durch die ständige Kontrolle des *Observers* wird der Code auf Fehler überprüft, die der *Driver* übersehen kann [Wil00].

Pair Thinking Dadurch, dass zwei Entwickler sich einer gemeinsamen Sache widmen, können unterschiedliche Gedanken entwickelt und gegeneinander abgewogen werden [Wil00].

Pair Relaying Im Paar lassen sich komplexere Probleme schneller Lösen. Durch das *Pair Thinking* unterstützen sich beide Programmierer gegenseitig der Findung eines Lösungsweges [Wil00].

Pair Pressure Die Programmierer arbeiten motivierter als wenn sie alleine wären. Sie wollen sich gegenseitig nicht „hängen lassen“ und lassen sich nicht so einfach ablenken. Dies erzeugt einen positiven Druck [Wil00].

Debugging by Explaining Durch das gegenseitige Erklären wird das Vorgehen bei Beiden bewusster durchdacht und hinterfragt [Wil00].

Team Building PP verbessert den Zusammenhalt in der Gruppe der Entwickler. Dadurch lernen die Teilnehmer einfacher und öfter miteinander zu kommunizieren [Wil00].

Project Risk Das Wissen über die gestellten Probleme und deren erarbeiteten Lösungen verteilt sich automatisch im Team. Dadurch verringert sich das Projektrisiko falls mehrere Entwickler ausfallen [Wil00]. Dies ist auch bekannt als *Truck Number* [WK02], eine informelle Metrik, die Aussagen über das individuelle Projektrisiko machen kann. Es muss einfach die Frage „How many or few would have to be hit by a truck (or quit) before the project is incapacitated?“[WK02, S.41] beantwortet werden. Die schlimmste Antwort wäre *Einer!*, also höchstes Projektrisiko, die beste *Alle!* [WK02].

2.1.2 Nachteile

PP hat auch einige potentielle Nachteile.

Erhöhte Kosten Ein offensichtlicher Nachteil sind die erhöhten Kosten, da zwei Entwickler an der Lösung eines Problems arbeiten. Durch die bessere Qualität vom Code und Entwurf wird dies aber wieder relativiert [CW01].

Räumliche Nähe Bedingt durch das Konzept der PP müssen beide Teilnehmer an einem Rechner sitzen, sich also an einem Standort aufhalten. Dies ist bei verteilten Teams jedoch nicht möglich.

Einschränkungen Arbeitsplatz Da die Entwickler an einem Arbeitsplatz sitzen, muss dieser auch dafür eingerichtet sein. Das bedeutet, dass es möglich sein muss bequem zu zweit an einem Rechner zu arbeiten, d. h. das Mobiliar muss hierfür hergerichtet sein.

Gemeinsame Entwicklungsplattform Weiterhin muss auch die Software, mit der entwickelt wird, von beiden Programmierern genutzt und akzeptiert werden. Wird XP eingesetzt, kann dies schon durch den dort praktizierten Standardisierungsprozess geschehen sein [BA04].

2.2 Verteilte Paarprogrammierung

Eine Erweiterung des Konzepts der PP ist die verteilte Paarprogrammierung. Hautptunterschied zur PP ist, dass die Programmierer sich nicht an einem Standort befinden müssen.

Daraus ergeben sich zwar weitere Vorteile, doch es ist auch teilweise schwierig, die von PP aufrecht zu erhalten.

2.2.1 Vorteile

Räumlich verteilt Wie bereits erwähnt, benutzen die Entwickler bei DPP getrennte Rechner, anstatt eines Einzelnen. Dadurch kann an verschiedenen Orten programmiert werden, was noch weitere Vorteile mit sich bringt [Mau02]:

- Die Entwickler sind flexibler.
- Die Entwicklungskosten können reduziert, Arbeitskräfte können ausgelagert und es muss weniger ausgebildet werden.

Erhöhter Grad an Parallelität Da der *Observer* einen eigenen Rechner bedient, kann er zusätzliche Werkzeuge verwenden und den *Driver* bei seiner Arbeit unterstützen [SWB+03]. Dies kann zum Beispiel das Nachschlagen von Dokumentation sein.

Es werden verschiedene Ebenen der Parallelität unterschieden [DOS07]:

Parallelität auf Programmebene Neben dem Werkzeug welches die DPP-Sitzung ermöglicht, können noch weitere bedient werden.

Parallelität auf Sichtebene Während der *Driver* an einen bestimmten Codeabschnitt arbeitet, kann der *Observer* sich anderen relevanten Teilen des Quelltextes widmen.

Parallelität auf Schreibebene Es können in einer Sitzung mehrere *Driver* existieren, die gleichzeitig schreiben können.

Bequemere Arbeitsplätze Die mögliche Einschränkung des Arbeitsplatzes bei PP kann durch DPP reduziert werden, da die Entwickler separate Arbeitsplätze benutzen [SWB+03].

2.2.2 Nachteile

Die vorher genannten Nachteile von PP kommen beim verteilten Programmieren weniger oder fast gar nicht zum tragen. Da der Zwang der physischen Nähe entfällt, entstehen wie schon erwähnt neue Vorteile, jedoch auch ein großer Nachteil. Es kann keine direkte interpersonelle Kommunikation[1] mehr stattfinden [KJCL01]. Diesen verlorenen und komplexen Kommunikationskanal muss das DPP-Werkzeug so gut wie möglich ersetzen können [SWB+03].

2.3 Technische Ansätze für verteiltes kollaboratives Arbeiten

Um verteiltes kollaboratives Arbeiten mithilfe von Computern zu ermöglichen existieren zwei populäre Ansätze im Forschungsgebiet der *Computer Supported Cooperative Work* (CSCW) [Han05, RSVW94], welche hier nun genauer vorgestellt werden sollen[2].

2.3.1 *Screensharing*

Screensharing[3] erlaubt es einem Benutzer seinen Bildschirminhalt, oder auch Teile davon, mit anderen Leuten zu teilen und sie ggf. damit interagieren zu lassen. Diese Technik wird häufig zur Fernwartung und Fernbedienung von Rechnern genutzt. Damit lassen sich auch Anwendungen, die nur für die Benutzung eines Anwenders konzipiert wurden,

1 damit ist alles gemeint was die zwischenmenschliche Kommunkation betrifft: Reden, Mimiken, Gestikulieren ...

2 Es existieren auch Mischformen aus beidem, z. B. wird in [RSVW94] ein Framework vorgestellt, mit dem Einzelbenutzeranwendungen durch *Screensharing* von mehreren Anwendern benutzt werden können. Dabei handelt es sich also um kontextsensitives *Screensharing*.

3 auch *Applicationsharing* oder *Desktopsharing* genannt

von mehreren benutzen [RSVW94]. Dies ist auch bekannt als *collaboration-transparent applications* [RSVW94].

Ein Beispiel für die Anwendung dieses Ansatzes für DPP findet sich in Abschnitt 4.1.

2.3.2 Kontextsensitive Anwendungen

Im Gegensatz zum *Screensharing* wissen kontextsensitive Anwendungen von ihrer verteilten Anwendung durch mehrere Benutzer. Dies erfordert zwar eine Anpassung vorhandener Einzelbenutzeranwendungen, ermöglicht aber die Integration von Kontextsensitivität[1] in die Benutzerschnittstelle sowie eine rollenbasierte Darstellung und Interaktion [RSVW94].

Kontextsensitivität wurde von [ADB+99] definiert:

> A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.

2.3.3 Vergleich

Ein großer Vorteil von *Screensharing* ist die Möglichkeit beliebige Anwendungen zu teilen. Da jedoch die gesamte Arbeitsfläche als Bild repliziert wird, können Darstellungsprobleme bzgl. der Größe und Übertragungsprobleme bzgl. der großen Menge an Daten entstehen. Es ist weiterhin nicht möglich die Benutzeroberfläche parallel zu nutzen.

Kontextsensitive Anwendungen müssen zwar speziell entwickelt und entworfen werden, weisen jedoch einen Vorteil in ihrer Benutzbarkeit auf [GG98].

Einen ausführlichen Vergleich beider Varianten, hauptsächlich bezogen auf DPP, hat Riad Djemili in seiner Diplomarbeit [Dje06] dargestellt. Die Ergebnisse des Vergleichs werden in Tabelle 2.1 aufgeführt.

1 engl. *collaboration awareness*

geteilte Anwendung	kontextsensitive Anwendung
+ unterstützt fast jede Software	- speziell angepasste Software nötig
+ für Demonstrationen geeignet	- nicht für Demonstrationen geeignet
- keine bzw. geringe Parallelität	+ hohe Parallelität möglich
- hohe Netzlast	+ niedrige Netzlast
+ evtl. Darstellungsprobleme	- optimale Darstellung
+ guter Lerneffekt	- geringer Lerneffekt
- gesamter Desktop wird freigegeben	+ mehr Datenschutz

Tabelle 2.1: Gegenüberstellung der Vor- und Nachteile der technischen Ansätze für verteiltes kollaboratives Arbeiten [Dje06]

3 Saros

Saros ist ein *open source* Werkzeug, welches verteilte kollaborative Echtzeitprogrammierung in Eclipse ermöglicht[1]. Den Grundstein hat Riad Djemelli mit seiner Diplomarbeit [Dje06] gelegt und es wird seitdem kontinuierlich weiterentwickelt.

In diesem Kapitel werde ich beschreiben was Saros genau ist und wie man damit arbeitet. Zudem werde ich hier die Motivation für diese Arbeit erläutern.

3.1 Arbeit mit Saros

Wie schon erwähnt ist Saros ein Werkzeug um DPP zu ermöglichen. Doch es existieren noch mehr Szenarien in denen es eingesetzt werden kann [SOBS10]

- Codeeinführung
- Codedurchsicht
- *Side-by-Side* Programmierung

Um mit mit Saros zu arbeiten benötigt jeder Teilnehmer einen *Extensible Messaging and Presence Protocol* (XMPP)-Account. Einer der Teilnehmer, der auch das Projekt besitzt mit welchem gearbeitet werden soll, initiiert eine Sitzung und lädt alle Verbleibenden ein. Dadurch wird er zum *Host* dieser Sitzung und erhält zu Beginn die Rolle des *Drivers*, während weitere Teilnehmer *Observer* sind. Während es dem *Driver* möglich ist alle Dateien des Projektes zu bearbeiten, sind die *Observer* nicht in der Lage etwas zu editieren. Die vergebenen Rollen sind nicht starr, sie können nach Belieben gewechselt werden. Es ist sogar möglich mehrere *Driver* in einer Sitzung zu haben.

1 Saros ist nicht das einzige, z. B. gibt es noch Sangam[San] und XPairtise[XPa].

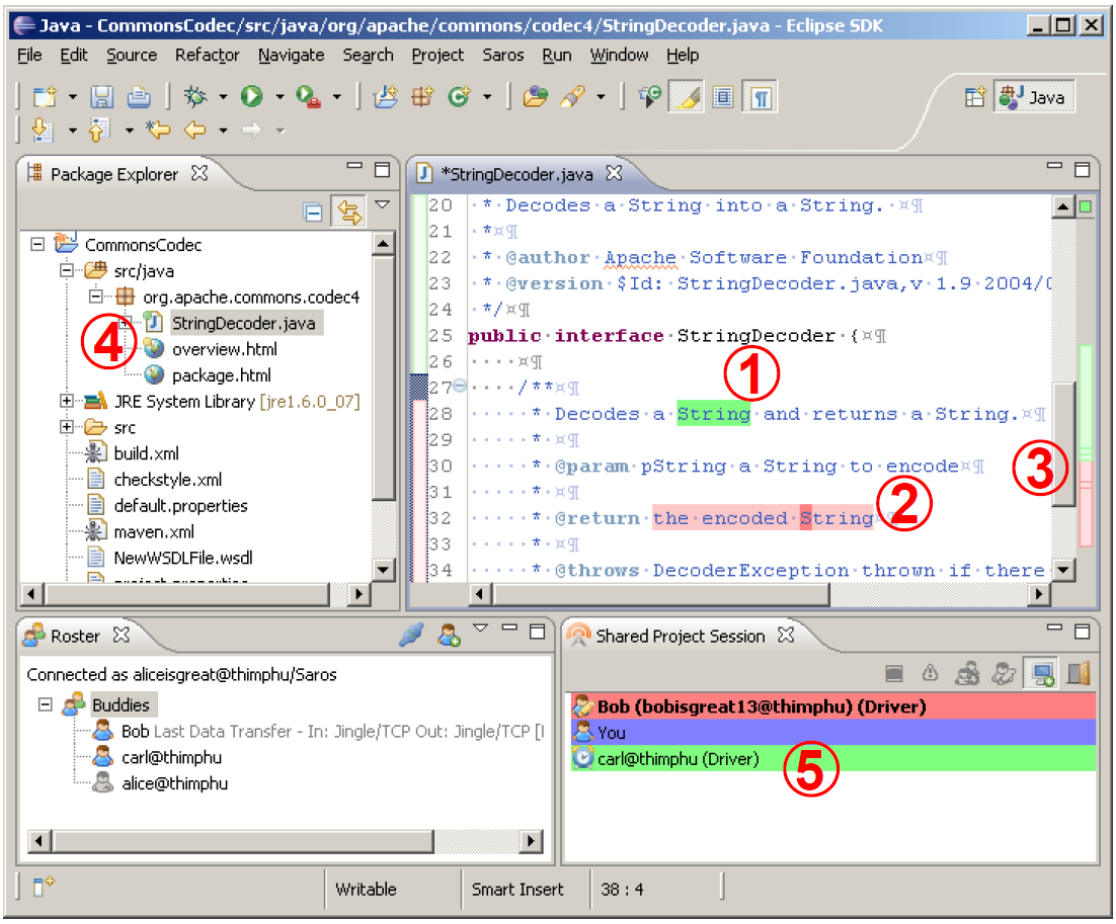

Abbildung 3.1: Eine exemplarische Saros-Sitzung mit markierten Annotationen [SOBS10]

Jeder Teilnehmer bekommt eine eindeutige Farbe zugewiesen, denn damit werden in Eclipses Text-Editoren die jeweiligen Aktionen der Teilnehmer annotiert. Die bisher verwendeten Annotationen sind in Abbildung 3.1 verdeutlicht:

1. markierter Text
2. editierter Text
3. Darstellungsfeld[1], also der Ausschnitt der Datei die betrachtet wird
4. geöffnete (gelb) und aktive Dateien (grün)

1 *Viewport*

5. Statusicon beim Benutzer: Eclipse im Vordergrund (Männchen) oder Hintergrund (Uhr)

Wenn der *Followmode* auf einen Benutzer aktiviert wurde, wird nun immer der aktuell geöffnete Editor im Vordergrund gehalten und zu der Stelle gesprungen die der verfolgte Benutzer auch sieht [SOBS10].

3.2 Probleme

Wie im vorherigen Abschnitt gezeigt, können die *Observer* dem *Driver* lediglich bei der Benutzung von Text-Editoren folgen. Interagieren sie mit anderen Editoren oder Views[1] oder sogar mit Werkzeugen oder Artefakten ausserhalb von Eclipse, kann ihnen nicht mehr gefolgt werden.

3.3 Motivation

Um die vorher genannten Probleme zu umgehen, könnte Saros für die Unterstüzung anderer Editoren oder spezieller *Views* noch erweitert werden. Dies ist jedoch zu unpraktikabel, da es aufwändig ist und eine unüberschaubare Anzahl an Komponenten existiert für die eine Anpassung nötig wäre. Ausserdem existieren noch externe Werkzeuge und Artefakte, für die solch eine Unterstützung nicht möglich ist.

Aus diesen Gründen bietet es sich an *Screensharing* in Saros zu integrieren, denn die Vorteile von kontextsensitiven Anwendungen bleiben erhalten und die des *Screensharings* kommen hinzu. Wie Djemelli in seiner Diplomarbeit schon feststellte [Dje06]:

> Ein ideales DPP-Werkzeug würde womöglich den Ansatz der Collaboration Awareness, mit der Möglichkeit zum Screen-Sharing bestimmter frei wählbarer Fenster, kombinieren. Ein derartiges Werkzeug würde die Vorteile beider Ansätze kombinieren.

Screensharing kann auch die Benutzbarkeit von Saros in möglichen Anwendungensszenarien verbessern. Allem voran sind die Szenarien, bei denen es darauf ankommt Ursache und Wirkung genau erkennen zu können. So kann zum Beispiel Codedurchsicht oder

1 das sind eigenständige Bereiche in Eclipse die nicht für das Editieren gedacht sind, wie z. B. die *Console*

Codeeinführung auch am laufenden Code, sowohl am Kompilat als auch zum Beispiel beim *Debugging*, vorgenommen werden. Dies kann die Verständlichkeit verbessern, da der Beobachter Ursache und Wirkung besser erkennen kann als es bei einer statischen Codedurchsicht möglich ist.

Bei Szenarien ausserhalb von Saros' Anwendungsmöglichkeiten kann *Screensharing* auch helfen. So kann es immer wieder passsieren das ein Entwickler diverse Probleme oder Fragen hat, bei denen die Erklärung schwieriger als bloßes Zeigen ist.

Auch wenn es möglich ist eine externe *Screensharing*-Lösung zu verwenden, so bringt doch die Integration dieser Funktion in Saros eine einfachere Benutzbarkeit mit. Dadurch kann der Entwicklungsprozess insgesamt flüssiger ablaufen und ein besseres Kontextgefühl vermitteln [CSH+04].

4 Verwandte Arbeiten

Es wurden vorher schon wissenschaftliche Arbeiten publiziert, die sich mit dem Thema *Screensharing* befassen. Zwei davon, die sich nah an dem Thema dieser Arbeit befinden, werde ich hier genauer vorstellen.

Die erste Arbeit befasst sich mit der Umsetzung DPP durch *Screensharing*. Ich werde die Realisierung sowie eine praktische Untersuchung dazu vorstellen.

In der zweiten Arbeit steht hingegegen die Vermittlung von Präsenz durch *Screensharing* im Vordergrund. Es wird hier vor allem gezeigt, dass die gewonnenen Informationen nicht nur für die Präsenz an sich nutzbar sind, sondern auch die Kollaborationsmöglichkeiten verbessert.

4.1 Verteilte Paarprogrammierung mit VNC

2002 begann Brian F. Hanks sich mit DPP zu beschäftigen. Technisch hat er es mittels *Screensharing* realisiert.

Dafür diente ihm ein modifiziertes VNC [RSFWH98], welches folgende zusätzliche Eigenschaften aufwies [Han02]:

Social Floor Control Die Flusskontrolle, also wer die Steuerung von Maus und Tastatur übernimmt, ist nicht technisch realisiert. Die Eingaben aller Teilnehmer werden an die virtualisierte Arbeitsfläche weitergereicht. Sie müssen sich untereinander absprechen wer nun die Kontrolle der Eingabegeräte übernimmt. Obwohl dies chaotisch erscheint, meint Hanks, dass dies bei kleineren Gruppen noch effektiv sei.

Telepointing Während der *Driver* die Kontrolle über Tastatur und Maus innehält, muss der *Observer* die Möglichkeit haben sich mitzuteilen. Er könnte zwar mit der Maus gestikulieren, jedoch würde dies den *Driver* bei seiner Arbeit unterbrechen. Deshalb hat Hanks einen zweiten an- und abschaltbaren Mauszeiger, den *Telepointer*, eingeführt (Abbildung 4.1).

Um nun die Auswirkungen und Effektivität von DPP und PP vergleichen zu können, hat er ein einjähriges Experiment durchgeführt. Dabei hat er die Prüfungsleistungen von Studenten verglichen, die für ihre Übungsaufgaben entweder klassisches PP oder sein DPP verwendet haben, und ihre Rückmeldungen ausgewertet [Han04].

Im Ergebnis haben sich die Prüngsleistungen statistisch nicht verändert. *Screensharing* wirkt sich also nicht negativ, aber auch nicht positiv auf die Ergebnisse aus. Bei DPP jedoch war für die Studenten die Zusammenarbeit einfacher, da sie weniger Zeitkonflikte bzgl. der Sitzungen hatten und deswegen auch seltener allein arbeiteten [Han04, Han05].

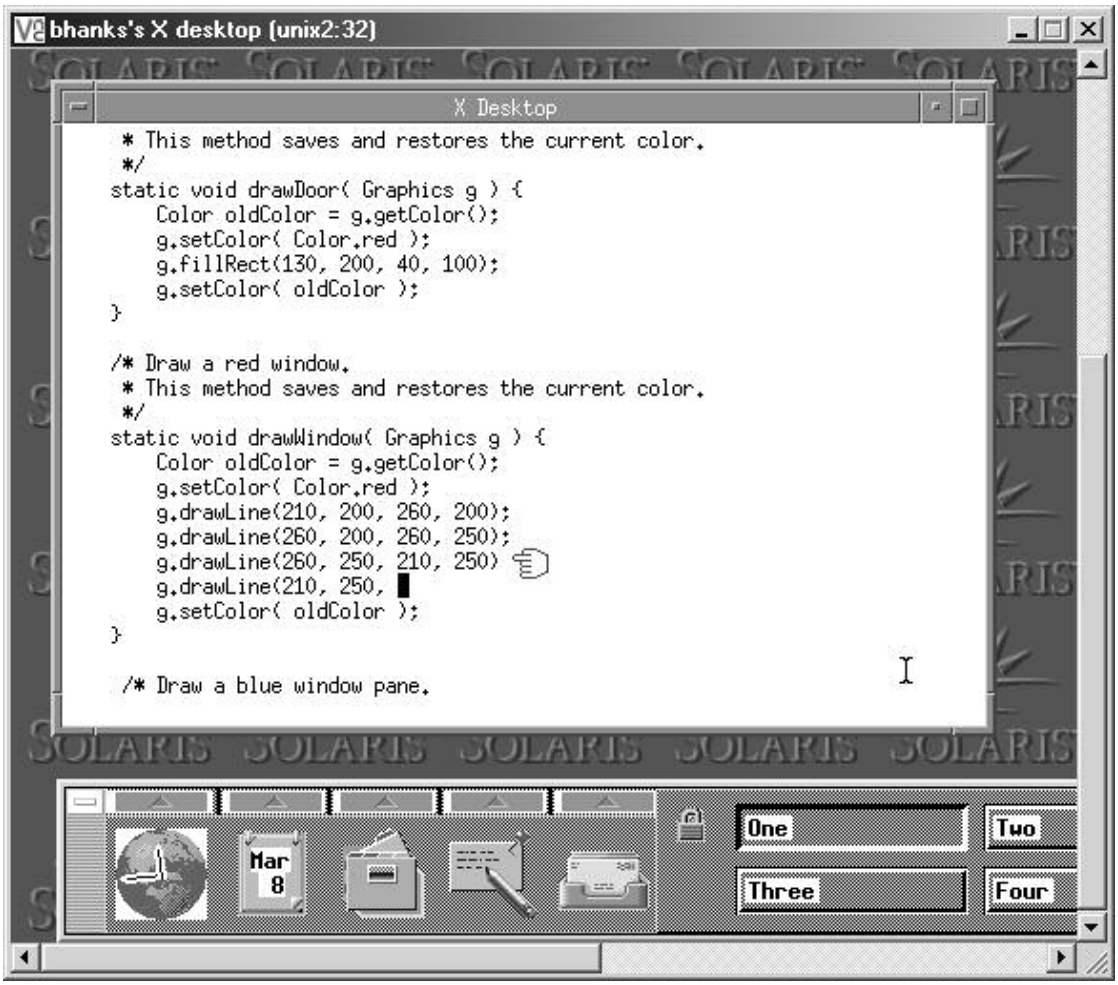

Abbildung 4.1: Hanks' modifiziertes VNC mit aktiviertem Telepointer

4.2 *Screensharing* in der Community Bar

In diesem Abschnitt werde ich die Community Bar und deren Anwendung von *Screensharing* vorstellen, wie es in den Publikationen dazu beschrieben wird [TGG06, TGGM06].

4.2.1 Community Bar

Die Community Bar (CB) ist eine *Groupware*, die für den schnellen Überblick und beiläufige Kommunikation in kleineren Gruppen gedacht ist. Jeder Benutzer ist Mitglied in einem *Place*[1] und er wird in dieser durch einem oder mehrere *Presence Items* dargestellt, welches z. B. ein simples Bild oder Text sein kann sowie das aktuelle Bild einer PC-Kamera. Damit kann den anderen Nutzern der aktuelle Status, d. h. Erreichbarkeit, Pläne etc., mitgeteilt werden.

Den Mitgliedern einer Gruppe ist es möglich über *Chat Items* miteinander zu kommunizieren und mittels *Media Items* Informationen an die Gruppe zu senden, wie z.B. Fotos, Notizen oder Webseiten.

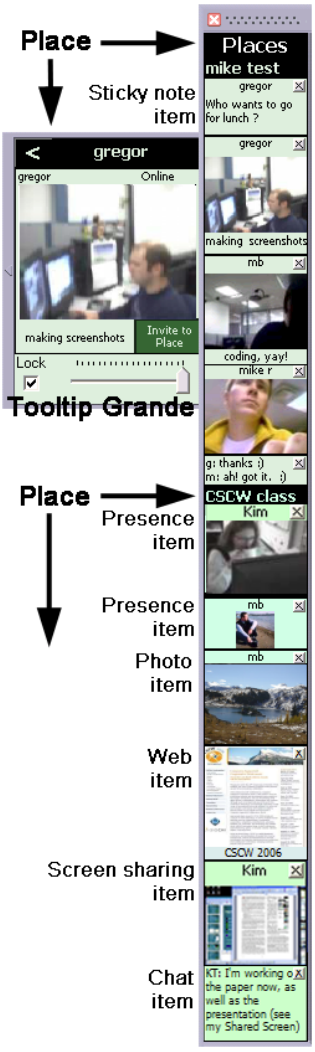

Abbildung 4.2: Die Benutzerschnittstelle der Community Bar [TGG06]

4.2.2 *Screensharing Item*

Die bisherigen *Presence Items* teilen im besten Fall[2] mit ob jemand den Rechner bedient, jedoch nicht was genau diese Person dort tut. Damit dies möglich ist, wurde das *Screensharing Item* entwickelt.

4.2.2.1 Funktion

Das *Screensharing Item* zeigt einen vom Besitzer[3] frei wählbaren Ausschnitt seines Bildschirms. Andere Mitglieder in seiner Gruppe sehen dann ein *Thumbnail* davon in ihrer CB. Falls weiteres Interesse besteht und der Bildschirm genau

1 so nennen sich die Gruppen in CB
2 wenn das Bild der PC-Kamera gezeigt wird und auf den Rechnernutzer gerichtet ist
3 der Nutzer der durch das *Presence Item* symbolisiert wird

betrachtet werden soll, besitzt das *Item* eine Vollansicht (Abbildung 4.3), in dem in das aktuelle Bild je nach Bedarf verkleinert und vergrößert werden kann. Anderen Nutzern ist es möglich bei Bedarf einen eigenen Mauszeiger auf der Arbeitsfläche des Besitzers anzufordern, dieser kann jedoch nur bewegt werden.

Um die Privatsphäre des Besitzers zu schützen, kann er in vielfältiger Hinsicht Einfluss auf das Verhalten des *Screensharing Items* nehmen.

Was wird gezeigt? Es kann eingestellt werden welcher Ausschnitt des Bildschirms gezeigt wird.

Wieviel wird gezeigt? Der Besitzer kann festlegen wie weit ein Nutzer in das Bild hineinzoomen darf, also wie viele Details zu erkennen sind. Besonders sensible Bereiche können mit verschiedenen Effekten[1] maskiert werden.

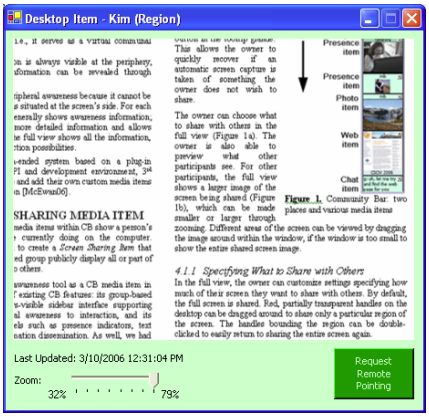

Abbildung 4.3: Die Vollansicht des *Screensharing Items* [TGG06]

Wann wird er gezeigt? Die Aktualisierung des Ausschnittes lässt sich manuell auslösen, kann aber auch automatisch in einem frei wählbarem Intervall erfolgen. Bei automatischer Aktualisierung wird der Besitzer 5 Sekunden vor Auslösung visuell darauf hingewiesen.

4.2.2.2 Anwendung

Um erste Erfahrungen und mögliche Anwendungsfälle zu sammeln, wurde das *Screensharing Item* zwei Wochen lang in der Forschungsgruppe von Kimberly Tee et al. eingesetzt.

In dieser Zeit haben sich folgende Anwendungsmöglichkeiten ergeben:

Artifact Awareness Es kann erkannt werden, was der Andere an seinem Rechner tut, welche Programme benutzt werden und an welchem Artefakten gearbeitet wird.

1 wie z.B. Unschärfe, Verpixelung

Presence Awareness Bei Personen, die ihre Präsenz nicht durch *Presence Items* mit PC-Kamera visualisieren, kann erkannt ob sie grad arbeiten und somit am Rechner sitzen.

Opportunistic Interactions Während der Untersuchung kam es häufiger zu spontanen Diskussionen über Artefakte, die auf dem Bildschirm gesehen wurden.

Focused Collaboration Die gezielte Zusammenarbeit, z. B. an einem Text bei dem unterschiedliche Abschnitte bearbeitet werden, ist bis zu einem bestimmtem Maß möglich. Wenn man sehen kann an welchen Teilen der Andere jeweils arbeitet, ist es möglich seine Arbeit dementsprechend anzupassen bzw. den Anderen bei seiner zu unterstützen.

5 Anforderungen

Die Anforderungen für *Screensharing* in Saros stammen aus der Aufgabenstellung meiner Betreuer[1] und den Erkenntnissen während der Entwicklungsphase.

Das hier betrachtete *Screensharing* wird in folgendem Szenario eingesetzt:

Der Host stellt seinen Bildschirminhalt einem oder mehreren Clients zur Verfügung. Der Bildschirminhalt besteht dabei aus einer Sequenz von Bildern, die beim Host kodiert werden und zum Client übertragen werden. Die kodierte Bildsequenz wird vom Client dekodiert und bei ihm dargestellt. Bis hierhin ist das *Screensharing* noch passiv, d. h. der Client kann den Bildschirm des Hosts nur betrachten. Der Client erhält jedoch noch die Möglichkeit in der Darstellung der Arbeitsfläche des Hosts Eingaben zu tätigen, sprich die Maus und Tastatur in diesem Kontext zu nutzen.

5.1 Funktionale Anforderungen

Host und Client haben rollenbedingt unterschiedliche funktionale Anforderungen an das *Screensharing*.

5.1.1 Host

1. Die lokale Arbeitsfläche dem Client zur Verfügung stellen.
    - Einzelbilder der Arbeitsfläche aufnehmen.
    - Die Bilder als Datenstrom kodieren, damit sie über das Netzwerk verschickt werden können.
2. Die Eingaben des Clients ausführen.
3. Eine mögliche Überschreitung der Netzwerkkapazität verhindern, also die genutzte Bandbreite adaptiv anpassen.

1 siehe hierzu Abschnitt A.1

5.1.2 Client

1. Den Datenstrom vom Host dekodieren und darzustellen, wobei die Größe der Ausgabebilder je nach Darstellung angepasst wird.

2. Eingaben in die virtuelle Arbeitsfläche tätigen.

5.2 Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen betreffen das *Screensharing* generell:

1. Der zeitliche Unterschied zwischen der Arbeitsfläche des Hosts und derer Darstellung beim Client muss so klein wie möglich gehalten werden, also die Kodierung, Dekodierung und Darstellung möglichst schnell ablaufen.

2. Die Bildquelle sollte austauschbar sein, da noch andere Quellen als die Arbeitsfläche[1] sinnvoll sind.

3. Es sollte möglich sein verschiedene Kodierverfahren zu verwenden. Hintergrund dessen ist die unterschiedliche Effizienz von javabasierten und nativen Encodern. Eine genaue Erläuterung zu dieser Problematik bietet Unterunterabschnitt 6.2.3.1.

4. Die Eingaben, die ein Client in die Arbeitsfläche des Hosts tätigen kann, müssen autorisiert werden, d. h. nicht jeder Client darf dies.

5. Laut Aufgabenstellung muss die Netzwerkschicht von Saros verwendet werden.

1 wie z.B. PC-Kamera, siehe Abschnitt 9.6

6 *Videosharing Framework*

Das *Videosharing Framework* (VSF) bildet die Basis für die Erweiterung von Saros um *Screensharing*, denn dabei handelt es sich lediglich um das Senden einer Bildsequenz vom Host zum Client. Dies wird mit dem VSF abstrahiert. Zudem soll es einfach erweiterbar, flexibel und modular sein, damit es auch als Grundlage für ähnlich Aufgaben dienen kann.

Vor allem steht beim VSF die Möglichkeit im Vordergrund, ein beliebiges Kodierverfahren implementieren zu können, da diese je nach Anwendungsfall unterschiedliche Anforderungen haben.

Neben der Erläuterung des Designs und der Funktionalität des VSF werden in diesem Kapitel die implementierten Komponenten für das *Screensharing* vorgestellt.

6.1 Überblick

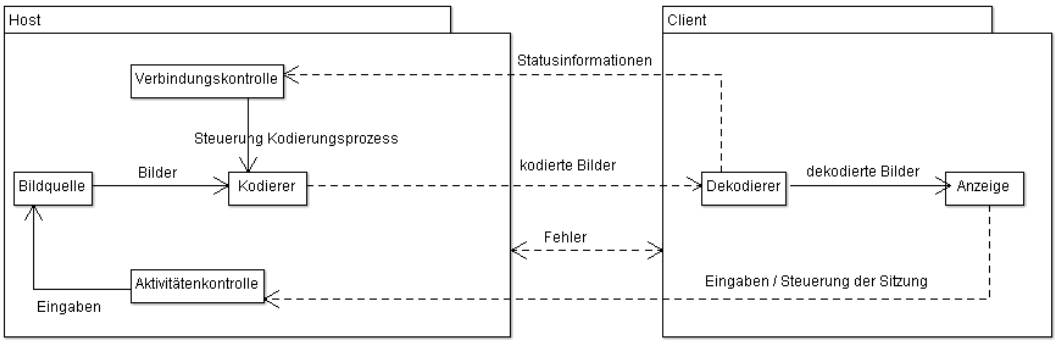

Abbildung 6.1: Grobe Übersicht des *Videosharing Framework*. Hier erkennt man die Komponenten die beim Host (stellt eine Bildquelle zur Verfügung) sowie beim Clienten (Betrachter) benötigt werden. Die gestrichelten Pfeile sind gerichtete Datenströme.

Zur bestmöglichen Abstraktion wird ein Framework geschaffen, welches unabhängig von verwendeter Bildquelle und Kodierungverfahren, Bilder eines Systems zu einem Anderen überträgt.

Basisparameter für jegliche Kodierung sind die verwendete Bildauflösung, *Frames per Second* (FPS), sowie die zu nutzende Bandbreite[1].

Die Architektur dieses Frameworks ist *Model View Controller* (MVC), eine grobe Übersicht hierzu bietet Abbildung 6.1.

6.1.1 Bildquelle – Modell

Das Modell, also die Datenbasis, ist die verwendete Bildquelle. Diese wird zunächst der Desktop sein. Als Erweiterung wäre z. B. die PC-Kamera, einzelne Teile von Eclipse oder Fenster auf dem Desktop denkbar.

Die Bildquelle kann Eingaben in Form von Mausklicks auf einem Teil des Bildes oder durch die Tastatur entgegennehmen. Dadurch kann der Benutzer die Bildquelle verändern. Passives Screensharing kann damit „aktiv“ werden, d. h. der Beobachtende kann in dem gesehenen Desktop, wie auf seinem eigenen, Eingaben tätigen.

6.1.2 Steuerung

Bei einer vorhanden Sitzung müssen zwei Aspekte gesteuert und kontrolliert werden:

Die Bandbreitennutzung: Die vorhandene Bandbreite darf nicht überschritten werden. Dies kann zur sehr großen Verzögerung in der Darstellung führen. Ausgelöst werden kann dies z. B. durch anderweitige Nutzung der verfügbaren Bandbreite, z. B. Downloads, oder Verschlechterung des Routings. Es muss also der Dekodierungprozess beim Clienten überwacht und der Kodierungsprozess beim Host dementsprechend gesteuert werden.

Auswertung von Aktionen des Clienten: Wie schon erwähnt, ist es dem Client möglich in dem übertragenen Bild Aktionen auszuführen. Dies kann je nach Bildquelle nicht unbedingt vom Host gewünscht werden, sodass hier eine Autorisierung stattfinden muss.

1 auch *Bitrate* genannt

6.1.3 Darstellung der Bilder – Präsentation

Die Anzeige beim Client hat die Aufgabe die dekodierten Bilder darzustellen, die Interaktionen von ihm entgegenzunehmen und an den Host weiterzuleiten. Zudem kann hier der Client die Sitzung soweit wie möglich steuern, z. B. Stoppen und Pausieren.

6.1.4 Kodierung der Bilder

Die übertragenen Bilder sollten natürlich von einem Kodierer komprimiert werden, um möglichst viel Bandbreite bei der Übertragung zu sparen. Es sollte möglich sein das Kodierungsverfahren zu wechseln, da Rahmenbedingungen, wie verwendete Bildquelle oder installiertes System, dieses uneffektiv oder sogar unbenutzbar machen können.

Der Kodiervorgang sollte soweit beeinflussbar sein, dass es möglich ist die verbrauchte Bandbreite zu regulieren sowie den Vorgang zu pausieren. Dies kann geschehen indem dem Encoder explizit die zu verwendende Bandbreite vorgegeben wird oder sie implizit durch die Wahl der Qualität oder der kodierten Bilder pro Sekunde reguliert werden kann.

6.2 Komponenten

6.2.1 Bildquelle

Eine Bildquelle implementiert das Interface `ImageSource`, wie in Abbildung 6.2 zu sehen. Ein Aufruf der Method `toImage()` sollte nun ein aktuelles Bild des Zustands liefern. Beeinflusst bzw. verändert werden kann die Bildquelle durch Aktivitätsobjekte (`processActivity(Activity)`). Sie kann auch noch unterschiedliche Modi bieten, die mittels `switchMode()` gewechselt werden können.

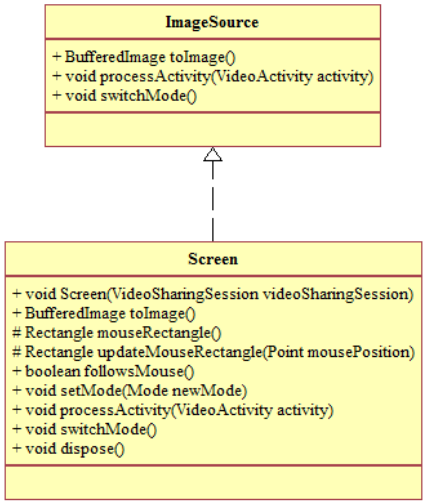

Abbildung 6.2: Das Interface `ImageSource` mit seiner Implementierung `Screen`.

6.2.1.1 `Screen`

Die Klasse `Screen` repräsentiert den aktuellen Desktop. Er besitzt zwei unterschiedliche Modi, wodurch `toImage()` unterschiedliche Bilder liefert.

Vollbild: Der gesamte Desktop, genauer gesagt der Inhalt des primären Bildschirms, wird erfasst.

Mausverfolgung: Ein definierter Bereich um den Mauszeiger wird erfasst.

6.2.2 Aktivitäten

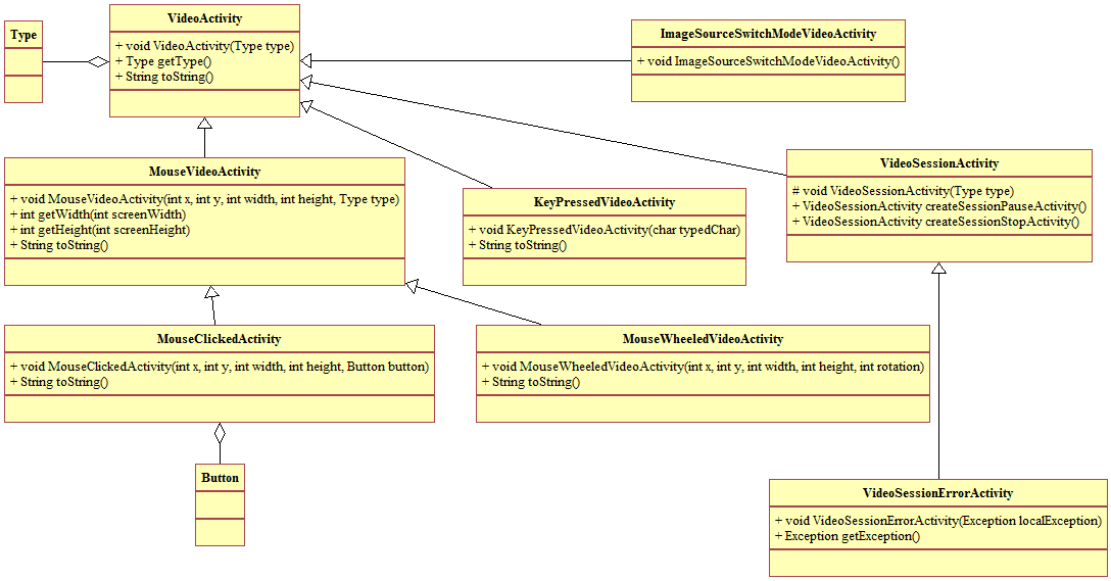

Abbildung 6.3: Klassendiagramm der Aktivitäten die ein Client an den Host senden kann.

Eine Aktivität sind alle Aktionen, die ein Client ausführen kann. Sie können die Sitzung direkt beeinflussen, also Stoppen und Pausieren dieser. Eingaben in das implementierte `VideoDisplay` sind ebenfalls Aktivitäten. Dies können Tastatur- und Mausaktionen sein. Letztere beziehen sich immer auf eine bestimmte Koordinate in dem Bild. Eine Übersicht der implementierten Aktivitätsklassen bietet Abbildung 6.3.

6.2.3 Kodierung

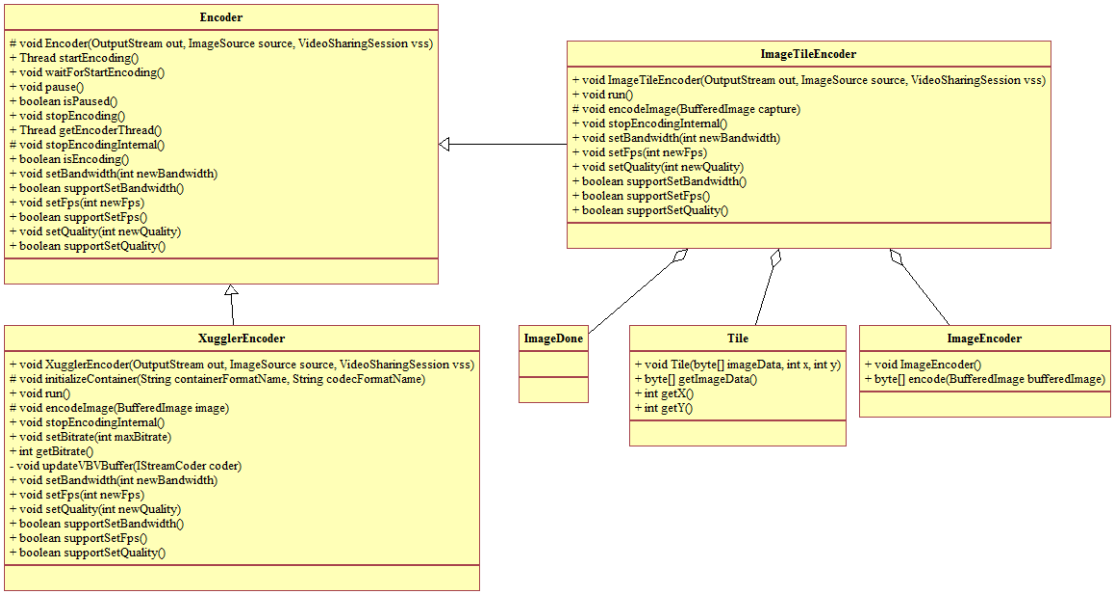

Abbildung 6.4: Klassendiagramm der Kodierer. `Encoder` ist eine abstrakte Klasse mit der Kodierverfahren implementiert werden können, hier `XugglerEncoder` und `ImageEncoder`.

Eine Implementierung eines bestimmten Kodierers erbt von der abstrakten Klasse `Encoder`. Damit erhält sie die nötige Umgebung, die zum Kodieren nötig ist. Diese besteht aus

- den Grundparametern der Kodierung
  - FPS
  - Bitrate
  - Auflösung
- Zugriff auf Saros' Einstellungen, falls für den Encoder extra welche angelegt wurden und Zugriff benötigt wird
- dem Stream in dem die kodierten Bilder geschrieben werden
- der Bildquelle

Der Kodierungprozess läuft in einem Thread, den der Kodierer selber implementiert. Er wird mit `startEncoding()` gestartet. Falls andere Komponenten darauf warten wollen, müssen sie die Methode `waitForEncoding()` aufrufen, welche bis zum Start blockiert. Während kodiert wird, können FPS, Bitrate und Qualität geändert werden, falls der Encoder dies unterstützt. Die Kodierung kann zeitweise (`pauseEncoding()`) oder auch endgültig (`stopEncoding()`) gestoppt werden. Eine Übersicht der beteiligten Klassen sowie implementierter Encoder ist in Abbildung 6.4 zu finden.

6.2.3.1 `XugglerEncoder`

Diese Implementierung eines Encoders kodiert die gegebene Bildsequenz zu digitalen Videodaten.

Er basiert auf Xuggler[Xug], welches ein *Java-Wrapper* für die Benutzung von FFMPEG[Bel] ist. FFMPEG ist eine Sammlung von Programmen und Bibliotheken, mit denen Video- und Audiodaten in verschiedenen Formaten verarbeitet werden können. Leider ist FFMPEG plattformabhängig, d.h. es muss im Gegensatz zu Java für das Zielsystem kompiliert werden, doch dadurch bietet es enorme Geschwindigkeitsvorteile. Java ist zwar eine funktionsreiche, plattformunabhängige Sprache, doch

> The same features that make Java so attractive, however, come at the expense of very slow performance. [KCSL00]

Vor allem wenn, wie bei der Videokodierung und -dekodierung, viele Daten verarbeitet und berechnet werden müssen, kann dieser Vorteil sich bemerkbar machen. Messungen haben ergeben, dass Bytecode, im Gegensatz zum kompiliertem C, 10 bis 50 mal langsamer ist [KCSL00]. Zudem nutzen einige Komponenten von FFMPEG die Erweiterungen moderner Prozessoren, wie z. B. *Multi Media Extension* (MMX), *Streaming SIMD Extensions* (SSE) oder *3DNow!*, wodurch Multimediaanwendungen im Mittel 3 mal schneller ausgeführt werden können [SS02, Table 5].

FFMPEG unterstützt verschiedenste Multimedia-Containerformate und -Codecs. Für die Anwendung hat sich *flv*[FLV07] als Container und als Codec endweder

H.264[Kal06] wegen sehr hoher Komprimierung, aber leider auch sehr rechenintensiv [OBL+04] oder

VP6[On205] bei vergleichsweise hoher Komprimierung und niedriger Rechenzeit

als sinnvoll erwiesen.

Beide komprimieren mit variabler Bitrate [Fog96, VBR], sie schwankt also je nach Veränderung der zu kodierenden Bildsequenz. Der Encoder hält unter Umständen die spezifizerten Beschränkungen der Bitrate nicht hundertprozentig ein. Bei sehr niedrigen Bitraten und hoher Auflösung/FPS können zeitweise Spitzen entstehen, die darüber hinaus gehen. Um dem entgegenzuwirken, kann man noch den *Video Buffer Verifier* (VBV) aktivieren, welcher dies vermeidet, jedoch eventuell kurze Aussetzer oder Bildverschlechterung nach sich zieht.

Der VBV wird folgendermaßen beschrieben [Fog96, VBV]

> MPEG concept defined in ISO/IEC 13818-2 (MPEG-2, Annex C) which employs a fixed size buffer to handle the transition of the channel bit rate to the rapidly flucuating coded bit rate of individual MPEG pictures. The scope of the VBV is only within a sequence. The VBV is built upon a framework of several axioms of decoder behavious which are unfortunately not very well described in the spec.

Obwohl der VBV dort nur im Zusammenhang mit der *MPEG-2*-Spezifikation erwähnt wurde, funktioniert dieser auch mit den vorher genannten Codecs.

6.2.3.2 `ImageTileEncoder`

Dieser Encoder stammt aus einer Beispielimplementierung von *Screensharing* für jingle-basierte Mediendienste in Smack [Jiva].

Der `ImageTileEncoder` teilt das zu kodierende Bild in Kacheln ein. Sobald sich eine Kachel ändert (im Vergleich zum vorherigen Bild) wird diese übertragen, ansonsten ist dies nicht nötig. Er kann auch die Qualität der Kacheln verringern, indem z. B. die Anzahl der Farben verringert wird. Das Bild verschlechtert sich jedoch nur optisch, die Bilddaten haben aber eine ähnliche Größe.

6.2.4 Dekoder

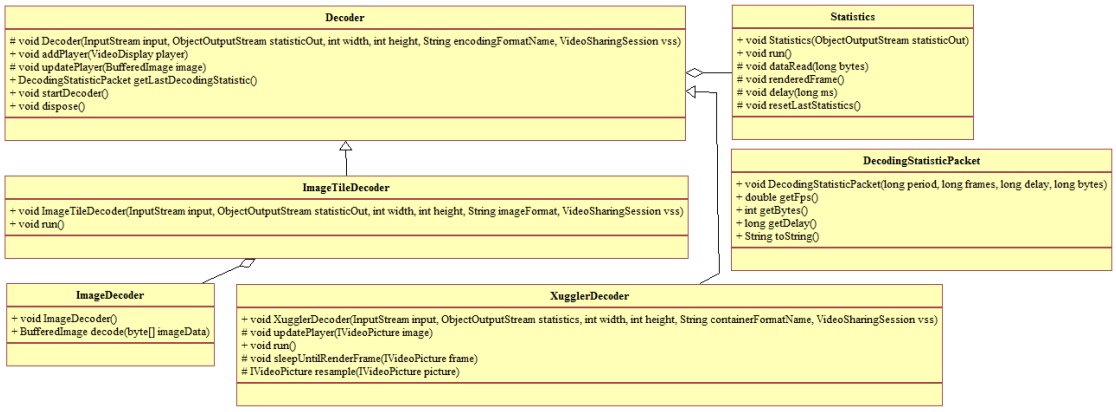

Abbildung 6.5: Klassendiagramm der Dekodierer.

Genau wie beim Encoder stellt auch hier die abstrakte Klasse `Decoder` die Basis der Implementierung eines Dekodierers dar, wie in Abbildung 6.5 zu sehen ist. Der Dekodierprozess läuft auch in seinem eigenen Thread. Damit die Statistiken über die Dekodierung generiert werden können, muss jedes dekodiertes Bild sowie seine Verzögerung, falls bekannt, gemeldet werden. Die für die Statistik ebenfalls wichtige Anzahl gelesener Bytes/s wird mithilfe des `java.io.InputStream` bestimmt. Dem Dekoder ist ein `VideoDisplay` bekannt, dem er die dekodierten Bilder übermitteln muss. Falls der Decoder auch die Bildgröße ändern kann, sollte er vor der Darstellung die optimale Größe `getImageDimension(Dimension)` erfragen, das Bild anpassen und es danach weiter geben.

6.2.4.1 `XugglerDecoder`

Er hat die Aufgabe den vom `XugglerEncoder` kodierten Videostream zu dekodieren.

6.2.4.2 `ImageTileDecoder`

Dieser dekodiert die vom `ImageEncoder` kodierten Kacheln, konstruiert ein Gesamtbild und gibt diese an das `VideoDisplay` weiter.

6.2.5 Wiedergabe

Die Anzeige beim Clienten hat die Aufgabe die dekodierten Bilder darzustellen, die Interaktionen von ihm entgegenzunehmen und an den Host weiterzuleiten. Zudem kann hier die laufende Sitzung pausiert oder gestoppt werden.

Eingaben in das `VideoCanvas` werden über den gegebenen `ObjectOutputStream` (`setActivityOutput(ObjectOutputStream)`) an den Host gesendet.

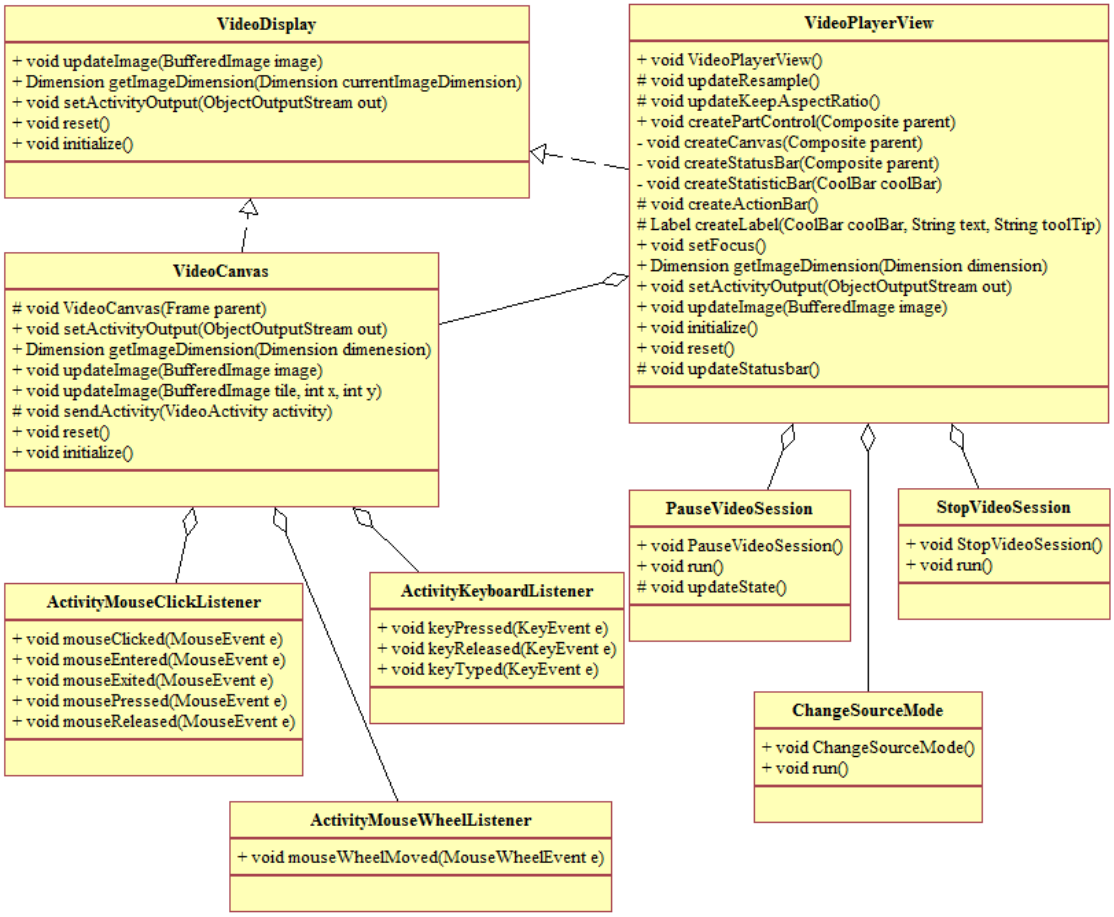

Abbildung 6.6: Interface `VideoDisplay` mit seiner Implementierung `VideoPlayerView` und den *Eingabe-Listenern* für das `VideoCanvas`

Je nach Konfiguration gibt die Methode `getImageDimension(Dimension)` die gewünschte Größe des darzustellenden Bildes zurück. Dies ist dann entweder vom `Decoder` anzupassen, wenn möglich, oder von der `VideoDisplayView` selbst.

6.2.6 Steuerung

6.2.6.1 Bandbreitensteuerung

Während der Sitzung sendet der Client ständig Informationen über den Status des Dekodierens. Diese enthält (`DecodingStatisticPacket`)

- FPS
- bytes/sec
- durchschnittliche Verzögerung der Bilder (Abweichung von der Synchronisation)

Diese Daten können bei der Überwachung und Regulierung der verwendeten Bandbreite helfen. Sollte die Verbindung überlastet werden, kann dies an dem allmähnlich voll laufenden Datenpuffer des Video-Ausgangs-Stroms erkannt werden.

Ein Pseudoalgorithmus, um die Bandbreite zu regulieren, lässt sich aus dem bei *Transmission Control Protocol* (TCP) verwendeten Überlaststeuerungsalgorithmus *Slow Start und Congestion Avoidance*[APS99] ableiten:

1. Beobachte Puffer. Wenn er voll ist (bzw. stetig voller wird), gehe zum nächten Schritt, ansonsten wiederhole diesen Schritt.
2. Reduziere Bandbreite die durch den Kodierer verbraucht wird.
   Ändere einen der Parameter Bitrate, FPS, Qualität (Priorität absteigend), je nach dem was der Kodierer unterstützt, auf die Hälfte des aktuellen Wertes oder den letzten Wert den der Dekodierer gesendet hat falls dieser niedriger ist.
   Sollte der Parameter zu niedrig werden, dann Kodierer pausieren.
3. Beobachte Puffer einige Zeit. Wenn er nicht leerer wird, wiederhole vorherigen Schritt.
4. Mache Schritt 2 rückgängig. Wenn Startwert erreicht, gehe zu Schritt 1.
5. Beobachte Puffer. Wird er leerer, Schritt 4 wiederholen, ansonsten gehe zu Schritt 3.

Fehlende Leistungsfähigkeit des Clienten kann auch als Verbindungsproblem diagnostiziert werden. In diesen Fällen läuft jedoch nicht der Puffer voll, da die kodierten Daten zwar empfangen, jedoch nicht schnell genug verarbeitet werden. Es kann in solchen Fällen entweder FPS oder *Bitrate* reduziert werden, um den Client zu entlasten.

6.2.6.2 Aktionssteuerung

Der `ActivityManager` empfängt die Eingaben, die vom `VideoDisplay`, also dem Clienten, getätigt werden. Seine Aufgabe ist es diese ggf. zu autorisieren und an die `ImageSource` weiterzuleiten.

7 Integration von *Screensharing* in Saros

In diesem Abschnitt werde ich beschreiben wie das entwickelte VSF in Saros integriert wurde. Erst hat es sich als nötig erwiesen die Netzwerkschicht von Saros zu erweitern, damit es möglich ist sitzungsbasiert Daten auszutauschen. Dann gehe ich genauer auf die Integration der Funktionalität von *Screensharing* ein, vor allem wie dies in der Benutzerschnittstelle umgesetzt wurde.

7.1 *Stream Service Framework*

Der Austausch von Aktivitäten, Dateien etc. in Saros erfolgt durch XMPP-Nachrichten oder *Peer-to-Peer* (P2P)-Verbindungen mittels Jingle[SOBS10]. Implementiert wird XMPP[SAST09] durch die Bibliothek Smack[Jiva].

Saros abstrahiert dies nochmal, indem es nur eine Methode anbietet Daten zu senden: ein `byte[]` bestimmter Größe von A nach B (`DataTransferManager.sendData( TransferDescription, byte[], SubMonitor)`), egal welche Verbindungsart (*In-Band Bytestreams* (IBB) oder Jingle) gerade verfügbar ist. Die Ankunft der Datenpakete sowie deren Reihenfolge wird garantiert.

Beim *Screensharing* handelt es sich um einen echtzeitsensitiven Datenstrom. Am besten geeignet hierfür ist das *Real-Time Transport Protocol* (RTP), welches auf dem *User Datagram Protocol* (UDP) basiert. Es existieren zwar Standards für direkte RTP[LSAE$^+$] oder UDP[BSAL$^+$, BLSA$^+$] Sitzungen mittels XMPP, doch Smack unterstützt diese leider nicht [Jivb].

Meine erste Idee für die Herstellung einer direkten Verbindung zwischen zwei Teilnehmern bestand darin, die bei der Erstellung der P2P-Verbindungen benutzten Sockets selbst zu verwenden. Dazu habe ich eine abstrakte Klasse geschrieben, die den Socket erstellt und für die Unterklassen bereitstellt. Diese Idee wurde jedoch von den Saros-Entwicklern abgelehnt, woraufhin das *Stream Service Framework* (SSF) entstanden ist.

Für das SSF muss also auf die vorher beschriebene paketbasierte Netzwerkschicht von

Saros zurückgegriffen werden. Da deren Eigenschaften denen von TCP-Streams sehr ähneln, liegt es nahe auch solche für das VSF zu benutzen.

Es wurde eine Erweiterung entwickelt, die *Streaming*, also die Verwendung von `java.io.InputStream` und `java.io.OutputStream`[Sun], möglich macht. Die Funktionalität der Implementierung wird hier kurz beschrieben.

7.1.1 `StreamService`

Damit Streaming durch andere Komponenten genutzt werden kann, müssen sie sich in dem bei Saros vorhandenen *Container*[1] eintragen.

Dafür müssen sie die Klasse `StreamService` implementieren, siehe Abbildung 7.1. Dort werden verschiedene Charakteristika definiert, wie

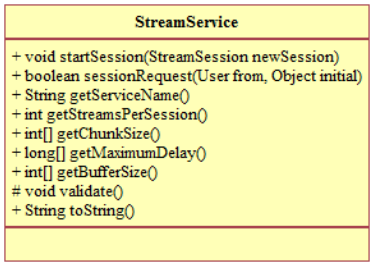

Abbildung 7.1: Klasse `StreamService`, welche zur Definition neuer Streaming-Dienste implementiert wird.

- eindeutiger Name des Dienstes (`getServiceName()`)
- Anzahl der Streams (`getStreamsPerSession()`)
- für jeden Stream
  - Größe des Datenpuffers (`getBufferSize()`)
  - minimale Größe der versendeten Pakete (`getChunkSize()`)
  - Verzögerung zwischen dem Versenden zweier Datenpakete, falls die minimale Größe für ein Paket nicht erreicht wird (`getMaximumDelay()`)

Weiterhin enthält die Implementierung Callback-Funktionen welche aufgerufen werden, wenn

- ein Benutzer eine Sitzung anfordert (`sessionRequest(User, Object)`).
- die vorher angefragte Sitzung erzeugt wurde (`startSession(StreamSession)`).

1 Der *Container* ist eine Sammlung von Objekten oder deren Klassen, die in Saros nur als einzelne Exemplare (*Singleton*[GHJV09]) vorhanden sein dürfen. Die *Container* werden von PicoContainer[Pic] zur Verfügung gestellt. Es ist ein *Dependency Injection Framework*[Fow04], mit dem die Abhängigkeiten der *Singletons* untereinander automatisch aufgelöst werden.

7.1.2 `StreamServiceManager`

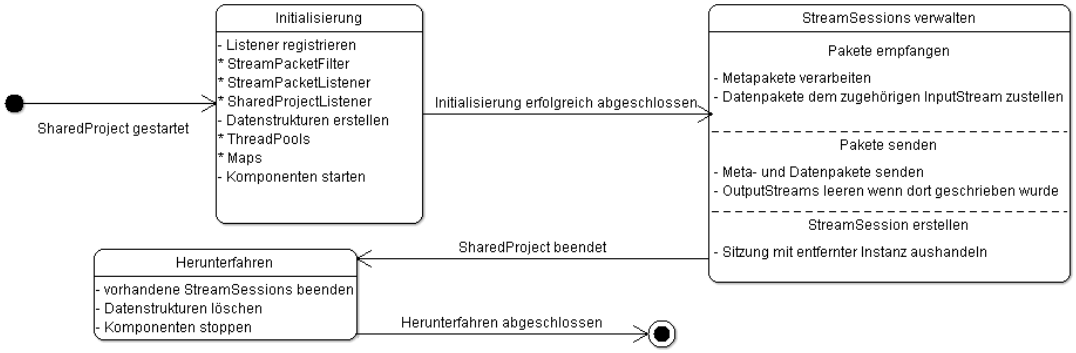

Abbildung 7.2: Grobes Zustandsdiagram der Klasse `StreamServiceManager`

Der `StreamServiceManager` verwaltet alle bekannten Dienste sowie laufende Sitzungen. Der Lebenszyklus des StreamServiceManagers ist in Abbildung 7.2 zu sehen. Erst nachdem ein Dienst auf beiden Seiten dem *Container* hinzugefügt wurde, können Sitzungen mittels `createSession(StreamService, User, Serializable, StreamSessionListener)` erstellt werden, falls der andere Benutzer diese annimmt. Ein Sequenzdiagramm dazu stellt Abbildung 7.3 dar.

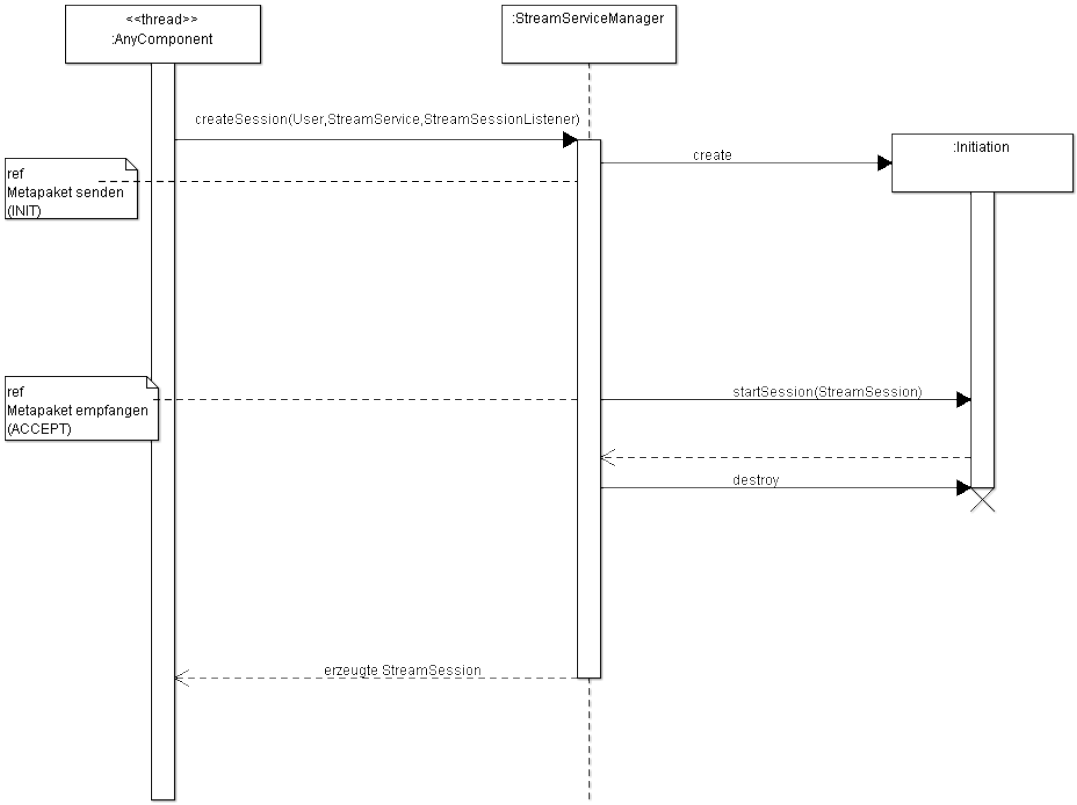

Abbildung 7.3: Sequenzdiagramm der Erstellung einer neuen `StreamSession`

Der `StreamServiceManager` hat auch die Aufgabe sich um die Verarbeitung ankommender Pakete zu kümmern (Abbildung 7.4). Dies können sowohl Datenpakete für den *Stream* einer Sitzung sein, als auch *Metapakete*, mit denen Sitzungen verwaltet werden (aushandeln, stoppen, *Stream* schließen).

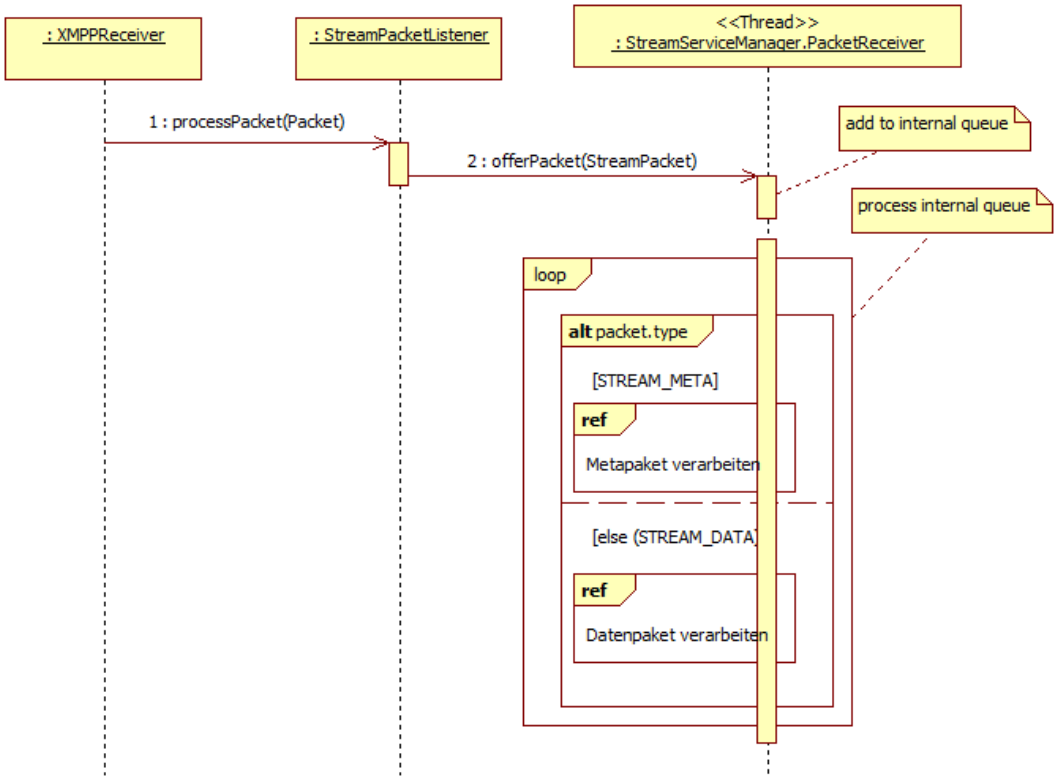

Abbildung 7.4: Sequenzdiagramm des Empfangs von Paketen und deren Verarbeitung im `StreamServiceManager`

Die Übersicht in Abbildung 7.5 zeigt den `StreamServiceManager` sowie alle direkt beteiligten Klassen.

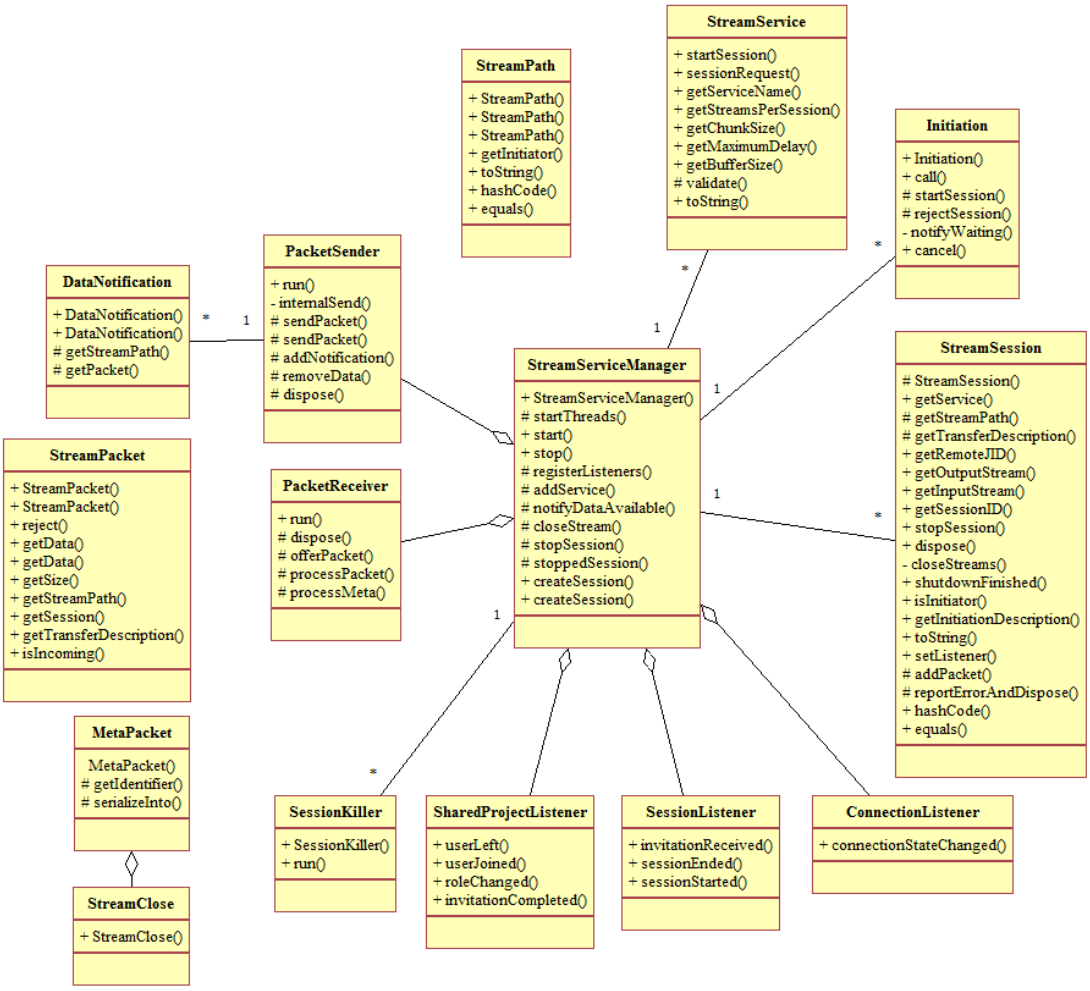

Abbildung 7.5: Klassendiagramm des SSF. Zur besseren Übersicht wurden einige Methoden und Parameter bzw. Rückgabewerte ausgelassen.

7.1.3 `StreamSession`

Eine Sitzung ist ein Exemplar von `StreamSession`. Wenn eine Sitzung gestartet wird, kann sie optional einen Listener (`StreamSessionListener`) besitzen. Dieser wird aufgerufen, wenn ein Fehler auftritt (z. B. Verbindungverlust) und die Session beendet werden muss oder wenn die andere Seite die Sitzung beenden möchte. Die laufende Sitzung kann mit `stopSession()` gestoppt werden. Die genaue Darstellung des Lebenszykluses ist in Abbildung 7.6 zu finden.

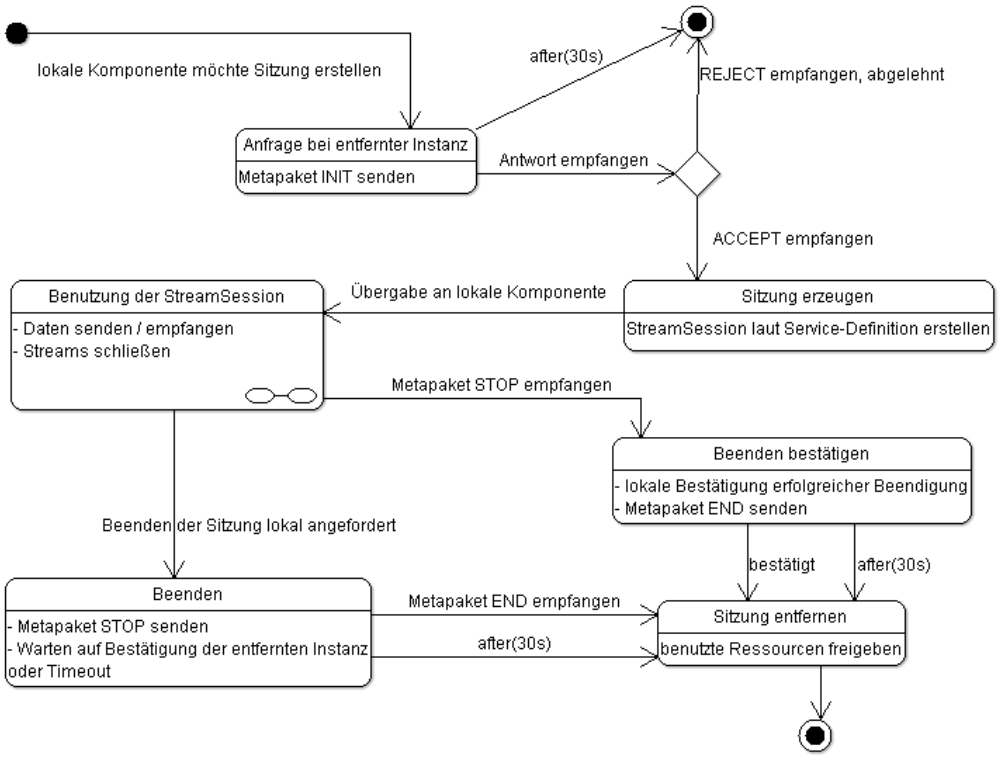

Abbildung 7.6: Zustandsdiagramm des Lebenszykluses einer `StreamSession`

Natürlich enthält sie auch die benötigte Anzahl an Streams, jeweils für Lese- und Schreiboperationen. Diese virtuellen Streams verhalten sich wie die javaeigenen [Sun, `java.io.InputStream`, `java.io.OutputStream`]. Dazu gehört nicht nur das Senden von Daten und der Empfang dieser auf der Gegenseite, sondern auch

- wenn einer der Streams geschlossen wird, wird auch der der Gegenseite geschlossen. Noch vorhandene Daten können bis zum Ende gelesen werden.
- das erst wieder Daten geschrieben werden können, wenn der lokale Puffer genügend freien Platz bietet.
- das der Inhalt des lokalen Puffers erst an den Empfänger gesendet wird, wenn dessen Puffer ausreichend freien Speicher besitzt.

- das ein Beenden der Sitzung zum Abbruch aller noch laufenden Lese-/Schreiboperationen führt.

Eine Übersicht der Klassen befindet sich in Abbildung 7.7.

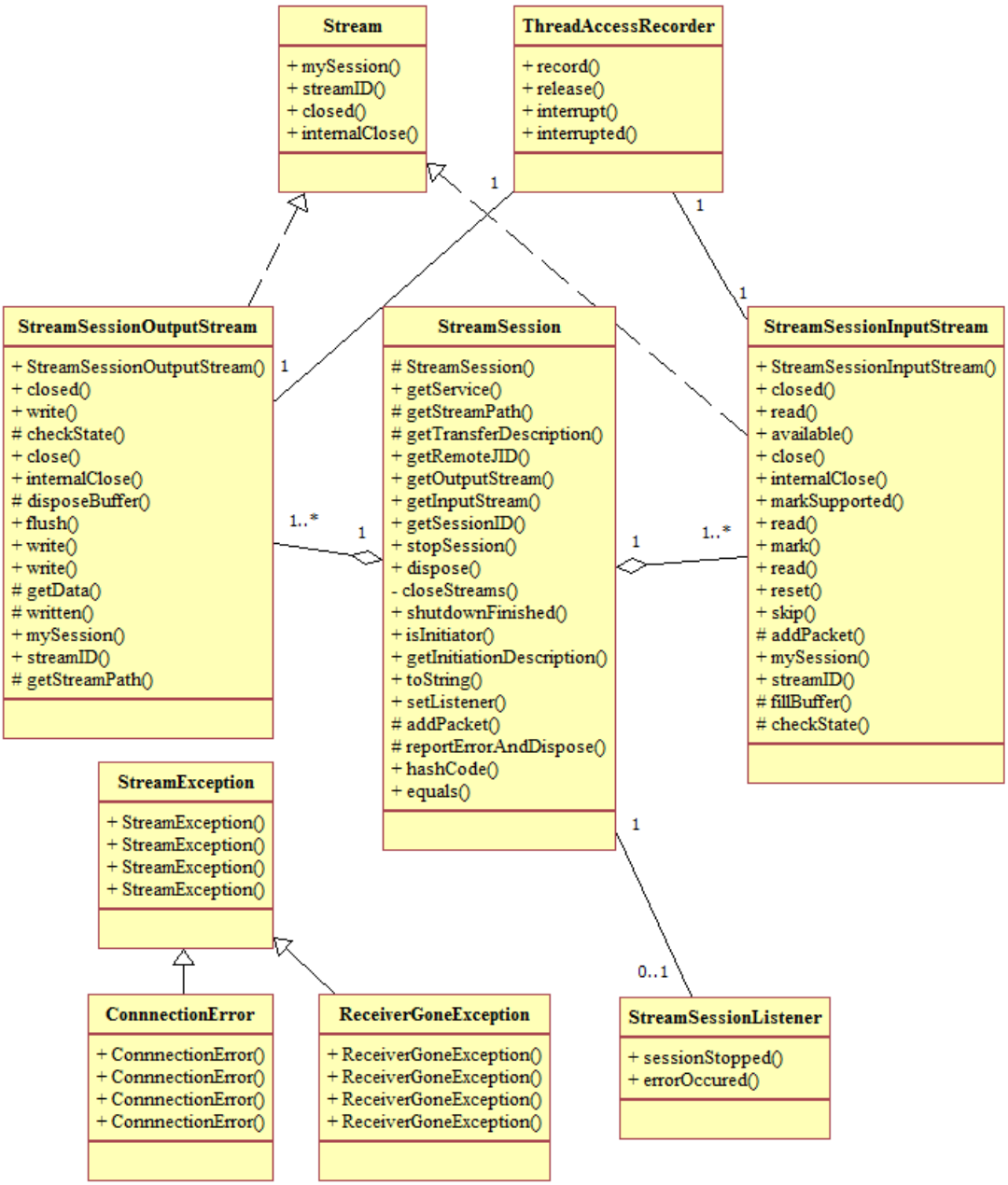

Abbildung 7.7: Klassendiagramm von `StreamSession` mit den beiden *virtuellen Streams*

7.1.4 Musterimplementierung eines Dienstes

Zur besseren Veranschaulichung für weitere Dienstimplementierungen habe ich ein funktionsfähiges Beispiel programmiert.

Es veranschaulicht folgende Aspekte der Benutzung einer `StreamSession`

- Benutzung der Datenströme[1]
- Reaktion auf Fehler- bzw. Abbruchsituationen
- korrektes Terminieren der Sitzung

Die Musterimplementierung dient der Übertragung einzelner Dateien zwischen zwei Teilnehmern einer Saros-Sitzung. Dabei ist es beiden an allen Stellen des Prozesses möglich die Übertragung abzubrechen.

Jede Übertragung wird in der *Progress View* von Eclipse visualisiert, siehe Abbildung 7.8.

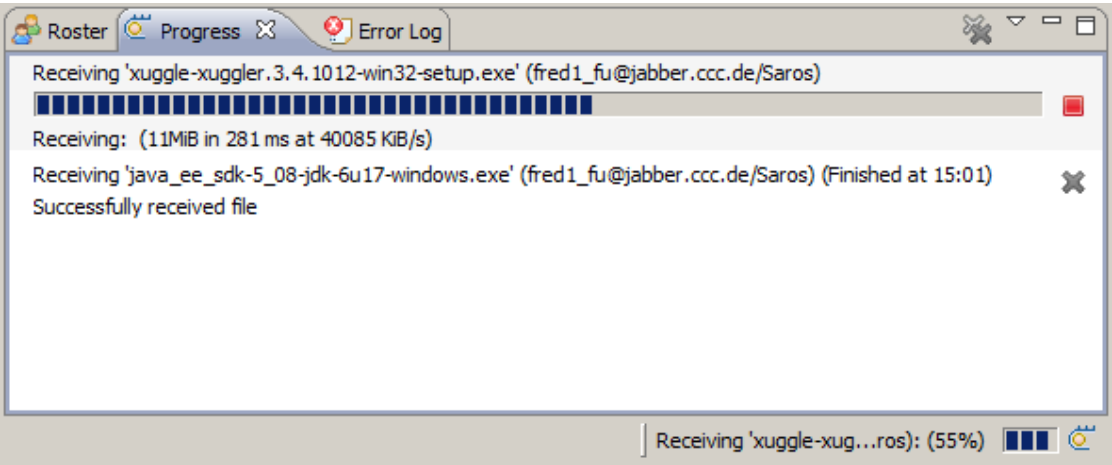

Abbildung 7.8: *Progress View* während des Empfangs einer Datei

Um die Funktionsfähigkeit zu gewährleisten, wurden einige manuelle Tests konzipiert, die in Abschnitt A.2 zu finden sind. Diese Zusatzfunktion ist seit der Veröffentlichung von Saros *10.3.26.r2105* im Praxiseinsatz.

1 Eigentlich gibt es hier ja nichts besonderes zu beachten, da sie sich, wie schon gesagt, wie normale Datenströme in Java behandeln lassen.

7.2 Komponenten

Das VSF wurde mit den Ziel entwickelt, möglichst unabhängig von Saros bzw. Eclipse zu sein[1]. In diesem Abschnitt wird die Integration des VSF und der Funktion des *Screensharings* in Saros beschrieben.

7.2.1 Einstellungen

Um die Konfigurationsoptionen der einzelnen Komponenten des VSF speichern und abrufen zu können, wurde das Einstellungsmodell, welches in Eclipse integriert ist[CR06], gewählt. Neue Einstellungsseiten in Eclipse's globalen Einstellungen erlauben es den Endbenutzer die Konfiguration einzusehen und nach belieben zu verändern. Jedes Modul, welches individueller Konfiguration bedarf, besitzt eine eigene Seite. Die einzige Ausnahme hierbei ist die Hauptseite der Einstellungen des *Screensharings*, welche komponentenübergreifende Einstellungen erlaubt. Damit kann der Benutzer schnell und einfach die wichtigsten Parameter ändern, ohne durch verschiedene Seiten navigieren zu müssen.

In den folgenden Abschnitten wird beschrieben welche Einstellungsseiten es gibt und was die Einstellungen genau bewirken.

7.2.1.1 Hauptseite

Die Hauptseite gibt eine Übersicht über die wichtigsten Einstellungen. Diese sind:

Encoder Hier kann man eines der implementierten Kodierverfahren für die nächste Sitzung zu wählen. Falls der ausgewählte Encoder nicht genutzt werden kann, wie das z. B. bei `XugglerEncoder` der Fall sein kann, weil Xuggler[Xug] nicht installiert ist, wird ein Fehler auf der Einstellungsseite angezeigt.

Video resolution Die Auflösung, die der ausgewählte Encoder verwendet.

Bandwidth Die vorhandene Bandbreite in Senderichtung, die dem Host zur Verfügung steht und der Encoder einhalten muss.

Captured area in follow mouse mode Die Auflösung des Bildschirmausschnitts im Modus *Mausverfolgung*. Die verwendete Einheit ist die Vergrößerung in Relation zu der

1 Einzige Ausnahme bildet hierbei der Zugriff auf persistente Einstellungen. Es wäre möglich gewesen einen *Adapter*[GHJV09] zu verwenden um diese Entkopplung zu erreichen.

gewählten Video-Auflösung. Damit wird die Größe des dargestellten Ausschnittes und implizit die Detailgenauigkeit festgelegt[1].

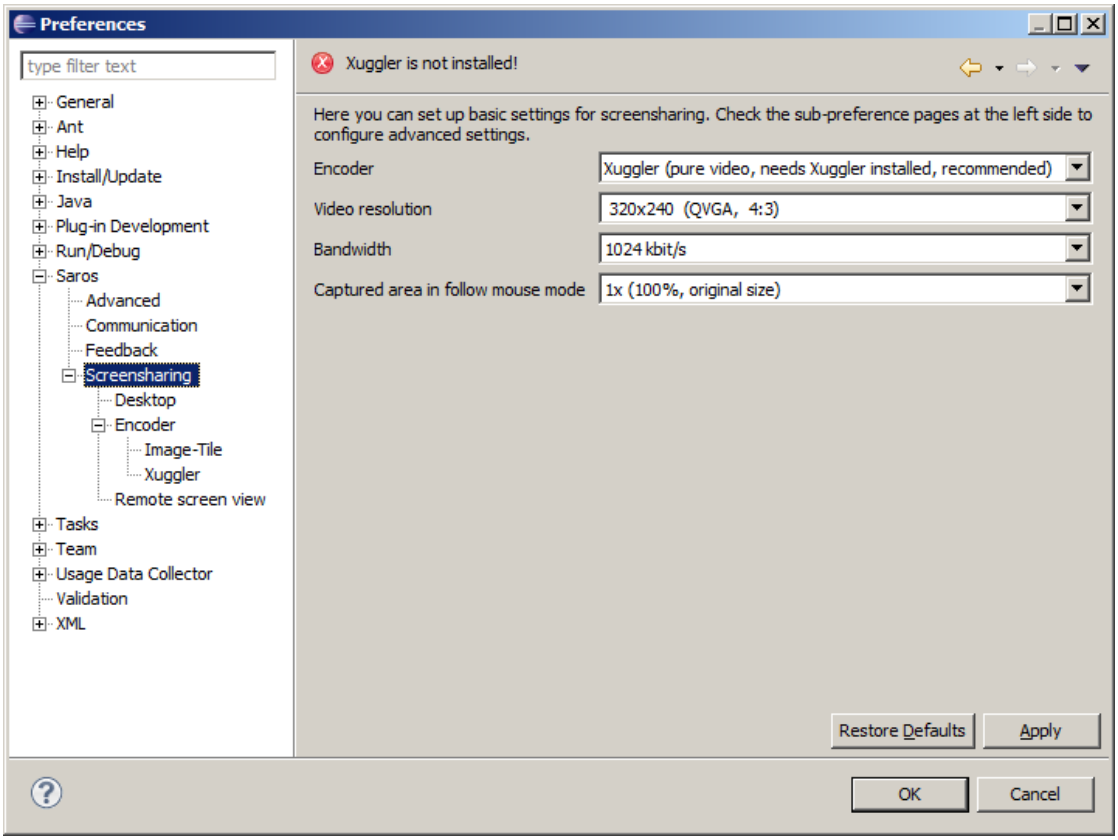

Abbildung 7.9: Die Hauptseite der Einstellungen für *Screensharing*.

7.2.1.2 Desktop

In dieser Unterseite lässt sich das Verhalten der Klasse `Screen` konfigurieren.

Initial mode Der *Modus* in dem die Erfassung startet.

Mouse area resolution (width, height) Hiermit lässt die Größe des erfassten Bereiches im Modus *Mausverfolgung* explizit bestimmen. Dies überschreibt den definierten Wert

1 Wird z. B. *2x* ausgewählt, dann ist der Bereich um den Mauszeiger, der erfasst wird, doppelt so groß wie die Auflösung in der kodiert wird. Damit werden jeweils immer vier Pixel des Ursprungsdesktop zu einem zusammengefasst, was einer Detailgenauigkeit von 50% entspricht.

für *Captured area in follow mouse mode*, der auf der Basiseinstellungsseite gesetzt wurde.

Show mouse pointer Leider wird bei Erfassung des Desktops der Mauszeiger nicht gezeigt[1]. Wenn diese Einstellung aktiviert ist, wird an der Stelle, wo der Mauszeiger sich befinden würde, einer nachgezeichnet.

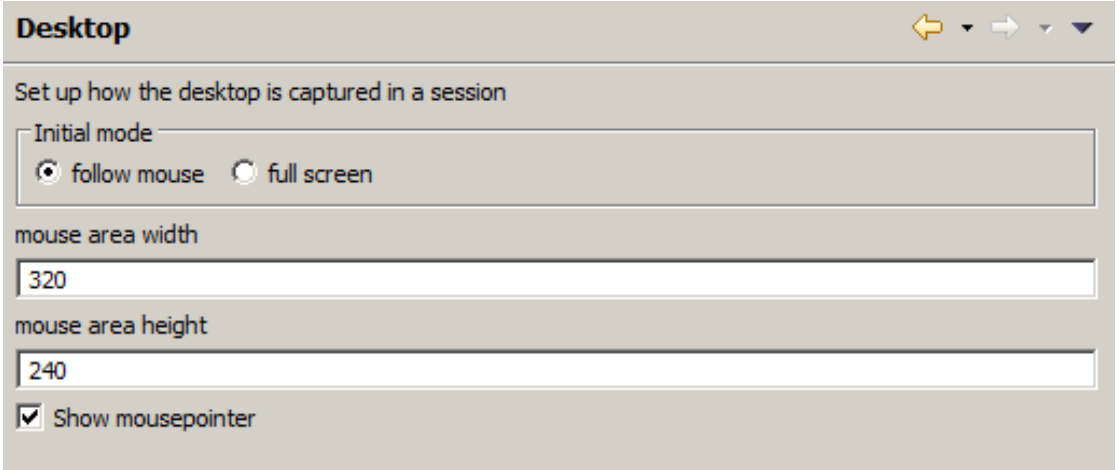

Abbildung 7.10: Die Einstellungseite für die Erfassung des Desktops

7.2.1.3 Encoder

Die Basisparameter für den Kodierprozess werden hier festgelegt.

Frames per second Kodierte Bilder pro Sekunde.

Resolution (width, height) Auflösung der Bildsequenz die der Client empfängt.

Max. bitrate Maximale Anzahl der Bits pro Sekunde des kodierten Datenstroms. Leider unterstützt derzeit nur `XugglerEncoder` die Umsetzung dieser Einstellung.

Encoder Hier kann man eines der implementierten Kodierverfahren für die nächste Sitzung zu wählen. Falls der ausgewählte Encoder nicht genutzt werden kann, wie

1 Die Klasse `java.awt.Robot` ist für die Erfassung des Desktops zuständig. Laut *Application Programming Interface* (API) wird bei der Erfassung mit der Methode `createScreenCapture(Rectangle)` der Mauszeiger ausgelassen.

das z. B. bei `XugglerEncoder` der Fall sein kann, weil Xuggler[Xug] nicht installiert ist, wird ein Fehler auf der Einstellungsseite angezeigt.

7.2.1.4 *Remote Screen View*

Die Ausgabeeigenschaften für die *Remote Screen View* werden auf dieser Seite konfiguriert. Die Möglichkeiten umfassen:

Resample input Ist diese Einstellung aktiviert, werden die Bilder, bevor sie angezeigt werden, auf die Größe des Sichtbereiches der *View* konvertiert.

Keep aspect ratio Diese Funktion kann nur aktiviert werden, sofern *Resample input* aktiv ist. Ist sie aktiviert, wird bei der Größenänderung das Seitenverhältnis beibehalten.

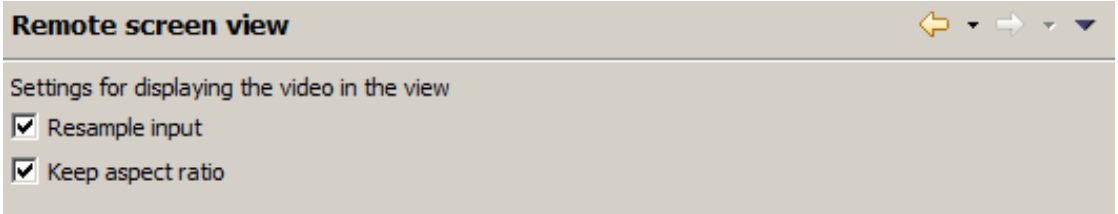

Abbildung 7.11: Der Konfigurationsdialog für die *Remote Screen View*

7.2.2 *Remote Screen View*

Die Visualisierung der Sitzung findet in einer *View* statt. Die *Actionbar*[1] enthält die Aktionen zum Stoppen[2] und Pausieren[3] der Sitzung, sowie dem Moduswechsel der Bildquelle[4]. Da die Bilder vom Typ `BufferedImage` sind, welche ein SWT-Canvas nicht darstellen kann, wurde ein AWT-Canvas (`VideoCanvas`) in einem SWT-Canvas eingebettet.

Im unteren Bereich der *View* befindet sich eine Informationsleiste, die den Status des Dekodierens visualisiert. Dieser wird alle drei Sekunden aktualisiert. Der Status enthält die selben Daten, die im `DecodingStatisticPacket` an den Host gesendet werden[5].

1 *Toolbar* (Funktionsleiste) einer *View*
2 Siehe Unterabschnitt 7.3.3
3 Siehe Unterabschnitt 7.3.4
4 Siehe Unterabschnitt 7.3.6
5 Siehe hierzu Unterunterabschnitt 6.2.6.1

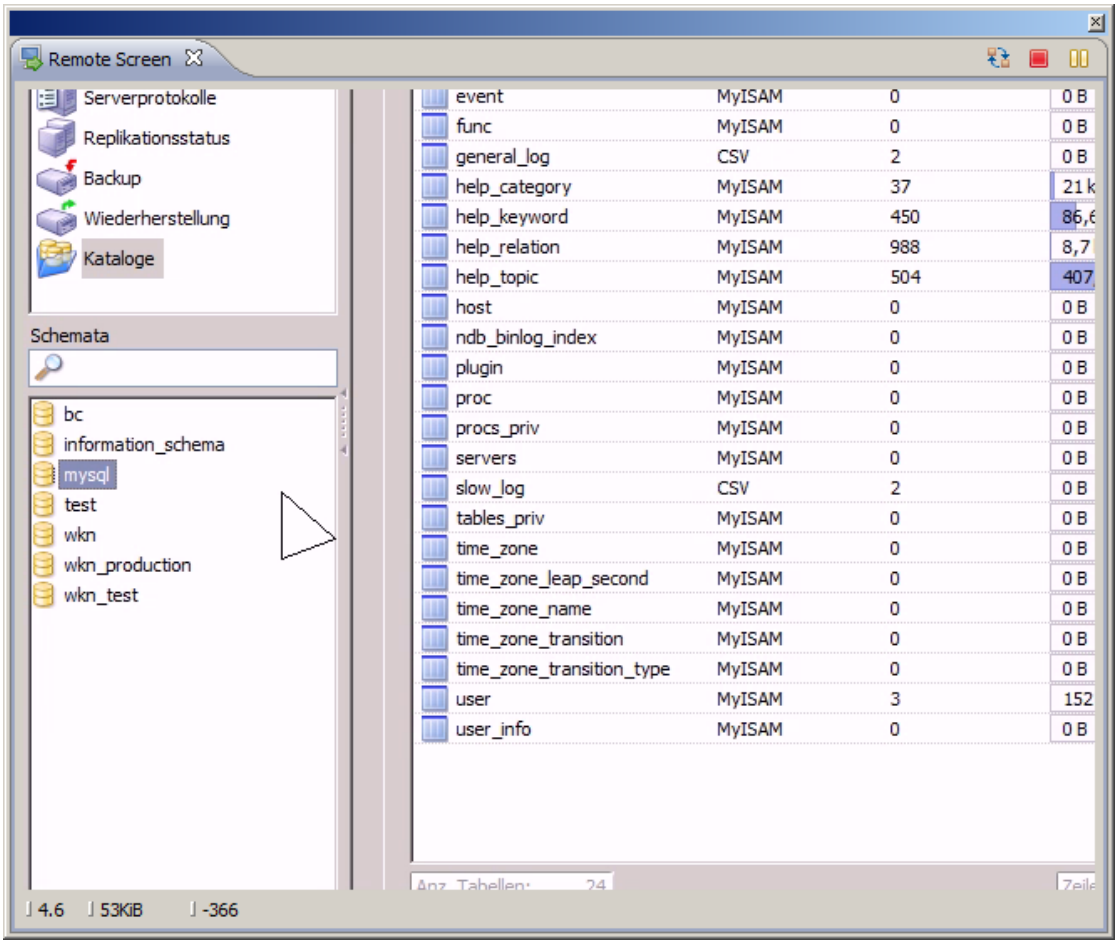

Abbildung 7.12: Die *Remote Screen View* während einer *Screenshare*-Sitzung

7.2.3 *Shared Project Session View*

In der *Shared Project Session View* werden alle Teilnehmer der aktuellen Saros-Sitzung angezeigt und ihre Rollen sowie die Sitzung selbst verwaltet. Diese Stelle wurde daher ausgewählt um auch die *Screensharing*-Sitzungen zu verwalten.

In die *Actionbar* wurde eine neue Funktion eingefügt, mit der *Screensharing* gestartet und gestoppt werden kann. Das Icon symbolisiert den derzeitigen Status wie folgt

: Eine Sitzung mit dem ausgewählten Teilnehmer kann aufgebaut werden. Dieses Icon ist deaktiviert falls niemand ausgewählt wurde. Siehe hierzu auch Unterabschnitt 7.3.1 und Abbildung 7.13.

: Der lokale Teilnehmer befindet sich schon in einer Sitzung und kann daher keine neue starten, nur die vorhandene beenden. Dieses Icon ist immer aktiv. Siehe hierzu auch Unterabschnitt 7.3.2 und Abbildung 7.14.

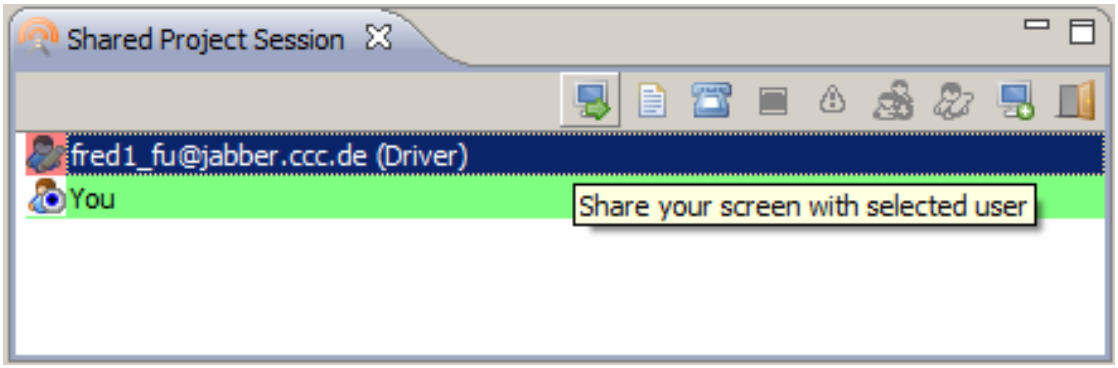

Abbildung 7.13: Die *Shared Project Session View* mit der erweiterten *Actionbar*. Mit dem ausgewählten Teilnehmer könnte die lokale Arbeitsfläche geteilt werden

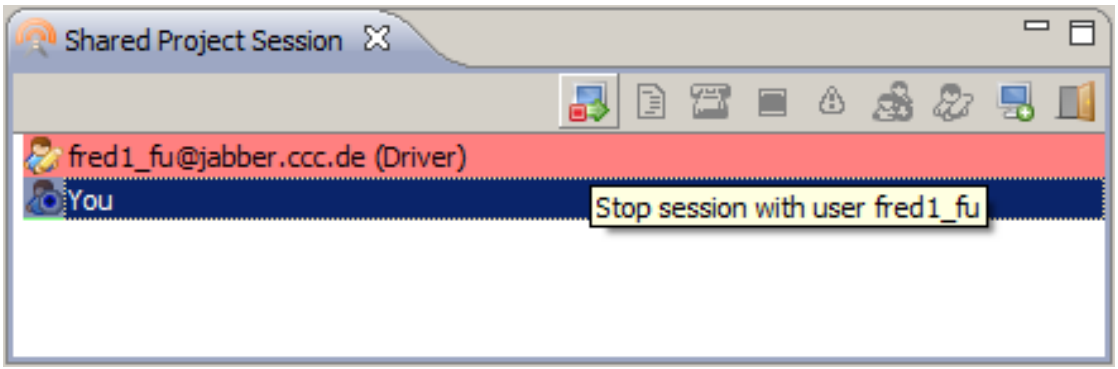

Abbildung 7.14: Die *Shared Project Session View* mit der erweiterten *Actionbar*. Hier wurde schon eine *Screenshare*-Sitzung gestartet die beendet werden kann.

7.3 Benutzerszenarien

7.3.1 B1: Teilen eines Bildschirms

Beschreibung:	Ein Teilnehmer einer Saros-Sitzung (Host) möchte einem anderen Teilnehmer (Client) den Inhalt seines Bildschirms zeigen.
Hauptakteur:	Host

Level:	Benutzerinteraktion
Beteiligte und Interessen:	*Host* Seinen Bildschirminhalt dem Client zur Verfügung stellen. *Client* Die Arbeitsfläche des Hosts betrachten und evtl. mit ihr interagieren.
Vorbedingungen:	• Erfolgreich installiertes Eclipse mit Saros-Plugin. • Host und Client sind in einer Saros-Sitzung. • Direkte Verbindung mit Jingle ist zwischen beiden möglich. Es ist auch mit allen anderen verfügbaren Verbindungsarten möglich, doch die Latenz und der geringe Durchsatz machen diese für *Screensharing* ungeeignet.
Minimalgarantie:	Der Host bekommt eine Rückmeldung im Fehlerfall, die Initiierung wird geloggt.
Erfolgsgarantie:	Beide befinden sich in einer *Screenshare*-Sitzung.

Haupterfolgsszenario:	1. Der Host wählt den Client in der *Shared Project Session*-View aus
	2. Der Host klickt auf die Schaltfläche in der *Toolbar* der *Shared Project Session View*
	3. Der Client bestätigt den Beginn in einem Ja-Nein-Dialog
	  • Erweiterung *E1*: Client bestätigt die Sitzung nicht.
	4. Als visuelle Bestätigung wandelt sich die Schaltfläche in
	  • Erweiterung *E2*: Einer der beiden kann das *Screensharing* nicht initialisieren.

Erweiterungen:	*E1* Der Host bekommt eine Fehlermeldung, dass der Client die Sitzung nicht annehmen möchte. Der Vorgang wird daraufhin abgebrochen. *E2* Eine benötigte Komponente kann auf einer der beiden Seiten nicht initialisiert werden. Der Fehler wird auf beiden Seiten als Fehlermeldung ausgegeben und der Vorgang abgebrochen.

7.3.2 B2: Beenden des *Screensharings*

Beschreibung:	Eine laufende *Screenshare*-Sitzung wird von einem der beiden Teilnehmer beendet.
Hauptakteur:	Host oder Client
Level:	Benutzerinteraktion
Beteiligte und Interessen:	*Host oder Client:* Beenden der Sitzung
Vorbedingung:	Eine *Screenshare*-Sitzung wurde aufgebaut, siehe *B1: Teilen eines Bildschirms.*
Minimalgarantie:	Die Sitzung wird beendet.
Erfolgsgarantie:	Die Sitzung wird beendet.
Haupterfolgsszenario:	1. Der Teilnehmer klickt auf die Schaltfläche in der *Toolbar* der *Shared Project Session View.*

7.3.3 B3: Client beendet das *Screensharing*

Beschreibung:	Eine *Screenshare*-Sitzung wurde aufgebaut und der Client möchte sie beenden.
Hauptakteur:	Client
Level:	Benutzerinteraktion
Beteiligte und Interessen:	*Client* Sitzung beenden.
Vorbedingung:	Eine *Screenshare*-Sitzung wurde aufgebaut, siehe *B1: Teilen eines Bildschirms.*

Minimalgarantie:	Die Sitzung wird beendet.
Erfolgsgarantie:	Die Sitzung wird beendet.
Haupterfolgsszenario:	1. Der Client öffnet die *Remote Screen View*, falls sie nicht schon geöffnet ist. 2. Der Client klickt in der *Toolbar* der *View* auf die Schaltfläche ■.

7.3.4 B4: Client pausiert das *Screensharing*

Beschreibung:	Eine *Screenshare*-Sitzung wurde aufgebaut und der Client möchte sie pausieren.
Hauptakteur:	Client
Level:	Benutzerinteraktion
Beteiligte und Interessen:	*Client* Sitzung pausieren, d.h. sie kurzfristig stoppen und ohne neue Initiierung wieder starten..

Vorbedingung:	Eine *Screenshare*-Sitzung wurde aufgebaut, siehe *B1: Teilen eines Bildschirms*.
Minimalgarantie:	Die Sitzung wird im Fehlerfall beendet.
Erfolgsgarantie:	Die Sitzung wird erfolgreich pausiert.
Haupterfolgsszenario:	1. Der Client öffnet die *Remote Screen View*, falls sie nicht schon geöffnet ist. 2. Der Client klickt in der *Toolbar* der *View* auf die Schaltfläche ⏸. Zur visuellen Bestätigung ändert sich die geklickte Schaltfläche zu ▶.

7.3.5 B5: Client setzt das *Screensharing* nach einer Pause fort

Beschreibung:	Eine *Screenshare*-Sitzung wurde aufgebaut und befindet sich im Pausezustand. Der Client möchte die Sitzung nun fortsetzen.
Hauptakteur:	Client
Level:	Benutzerinteraktion

Beteiligte und Interessen:	*Client* Die Sitzung entpausieren.
Vorbedingungen:	• Eine *Screenshare*-Sitzung wurde aufgebaut, siehe *B1: Teilen eines Bildschirms*. • Die laufende Sitzung wurde pausiert, siehe *B4: Client pausiert das* Screensharing.
Minimalgarantie:	Die Sitzung wird im Fehlerfall beendet.
Erfolgsgarantie:	Die Sitzung wird erfolgreich entpausiert, d.h. sie läuft normal weiter.
Haupterfolgsszenario:	1. Der Client öffnet die *Remote Screen View*, falls sie nicht schon geöffnet ist. 2. Nun klickt der Client in der *Toolbar* der *View* auf die Schaltfläche . Zur visuellen Bestätigung ändert sich die geklickte Schaltfläche zu .

7.3.6 B6: Client wechselt zwischen Vollbilddarstellung und Mausverfolgung

Beschreibung:	Der Client möchte den Anzeigemodus der virtuellen Arbeitsfläche ändern, d.h. zwischen Vollbilddarstellung und Mausverfolgung wechseln.
Hauptakteur:	Client
Level:	Benutzerinteraktion
Beteiligte und Interessen:	*Client* Anzeigemodus der virtuellen Arbeitsfläche wechseln.
Vorbedingung:	Eine *Screenshare*-Sitzung wurde aufgebaut, siehe *B1: Teilen eines Bildschirms*.
Minimalgarantie:	Die Sitzung wird im Fehlerfall beendet.
Erfolgsgarantie:	Der jeweils andere Modus ist aktiv.

Haupterfolgsszenario:	1. Client öffnet die *Remote Screen View*, falls sie nicht schon geöffnet ist. 2. Client klickt in der *Toolbar* der *View* auf die Schaltfläche . Zur visuellen Bestätigung erscheint oben rechts in der *View* für kurze Zeit der Name des nun aktiven Modus.

7.3.7 B7: Host öffnet die Konfiguration für *Screensharing*

Beschreibung:	Der Host möchte die Parameter einer Sitzung ändern und muss dafür den entsprechenden Dialog öffnen.
Hauptakteur:	Host
Level:	Benutzerinteraktion
Vorbedingung:	Erfolgreich installiertes Eclipse mit Saros-Plugin.
Erfolgsgarantie:	Dialog wurde geöffnet.
Haupterfolgsszenario:	1. Der Host öffnet die Einstellungen von Eclipse (i. d. R. *Window* → *Preferences*). 2. Er öffnet den Eigenschaftsbaum für *Saros* und klickt dort auf *Screensharing*.

7.3.8 B8: Host wechselt das Kodierverfahren

Beschreibung:	Der Host möchte das verwendete Kodierverfahren für *Screensharing* ändern.
Hauptakteur:	Host
Level:	Benutzerinteraktion
Beteiligte und Interessen:	*Host* Ändern des verwendeten Encoders, z. B. zu *Tile-Encoding* weil *Xuggler* nicht installiert ist.
Vorbedingung:	Der Konfigurationsdialog ist geöffnet, siehe *B7: Host öffnet die Konfiguration für* Screensharing.
Erfolgsgarantie:	Neue Einstellung ist gespeichert und der gewählte Encoder wird für die nächste *Screenshare*-Sitzung verwendet.

Haupterfolgsszenario:

1. Host wählt in der Liste *Encoder* einen aus.
   - Erweiterung *E1*: Der gewählte Encoder ist nicht installiert.
2. Er klickt auf *Apply* oder *Ok* um die Einstellungen zu speichern.

Erweiterungen:

E1 Es wird eine Fehlermeldung in dem Einstellungsdialog angezeigt, die den Host darauf hinweist, dass der gewählte Encoder nicht installiert ist. Die neue Einstellung kann trotzdem gespeichert werden.

7.3.9 B9: Host ändert die Basiseinstellungen für die Kodierung

Beschreibung:	Der Host möchte Parameter der Kodierung ändern.
Hauptakteur:	Host
Level:	Benutzerinteraktion
Beteiligte und Interessen:	*Host* Der Host möchte Parameter der Kodierung ändern, um sie an seine verfügbare Rechenleistung bzw. Bandbreite anzupassen.
Vorbedingung:	Der Konfigurationsdialog ist geöffnet, siehe *B7: Host öffnet die Konfiguration für* Screensharing.
Erfolgsgarantie:	Neue Einstellung ist gespeichert und die geänderten Parameter werden für die nächste *Screenshare*-Sitzung verwendet.

Haupterfolgsszenario:

1. Auf der Hauptseite der Einstellungen vom *Screensharing* kann der Host folgende Basiseinstellungen ändern

   Bildauflösung Die Auflösung der gesendeten Bilder lässt sich in der Liste des Feldes *Resolution* auswählen.

   Bandbreite Die verwendete Bandbreite des Encoders wird aus dem Feld *Bandwidth* ausgewählt.

2. Klick auf *Apply* falls einer der Werte auf *custom* gesetzt wurde oder die FPS geändert werden soll, ansonsten *Ok* und Anwendungsfall beenden.

3. Ein anderer Wert als einer der Vorgegebenen wird über die Einstellungsunterseite *Encoder* geändert.

   Bildauflösung In den Feldern *Resolution width* (Breite) und *Resolution height* (Höhe) kann die genaue Bildauflösung bestimmt werden.

   FPS Die Aktualisierungsrate der Bilder kann in dem Feld *Frames per second* eingegeben werden.

   Bandbreite Die verwendete Bandbreite (bit/s) kann genau in dem Eingabefeld *Max. bitrate* eingegeben werden.

   Jedes der Felder verlangt die Eingabe einer Zahl. Falls keine oder eine ungültige eingegeben wurde wird eine entsprechende Meldung angezeigt und die geänderten Einstellungen werden nicht gespeichert.

4. Die geänderten Parameter werden mit einem Klick auf *Ok* oder *Apply* gespeichert.

7.3.10 B10: Host konfiguriert ein ausgewähltes Kodierverfahren

Beschreibung:	Der Host möchte Parameter eines ausgewählten Encoders ändern.
Hauptakteur:	Host
Level:	Benutzerinteraktion

Beteiligte und Interessen:	*Host* Die Parameter des Encoders ändern, um sie an seine verfügbare Rechenleistung bzw. Bandbreite anzupassen.
Vorbedingungen:	• Der Konfigurationsdialog ist geöffnet, siehe *B7: Host öffnet die Konfiguration für* Screensharing. • Der zu konfigurierende Encoder ist in *B8: Host wechselt das Kodierverfahren* ausgewählt worden.
Erfolgsgarantie:	Neue Einstellung ist gespeichert und die geänderten Parameter werden für die nächste *Screenshare*-Sitzung verwendet.
Haupterfolgsszenario:	1. In dem Unterpunkt *Encoder* der *Screensharing*-Einstellungen befindet sich für jeden Encoder der sich konfigurieren lässt eine Einstellungsseite. Der Host wählt eine davon aus. 2. Er passt die dargestellten Optionen nach seinen Wünschen an. 3. Die Einstellungen werden gespeichert mit einem Klick auf *Apply* oder *Ok*.

7.3.11 B11: Host ändert die Detailgenauigkeit im Modus Mausverfolgung

Beschreibung:	Der Host möchte die Detailgenauigkeit im Modus Mausverfolgung ändern. Das ist die Größe des Bereichs der bei Mausverfolgung gesendet wird.
Hauptakteur:	Host
Level:	Benutzerinteraktion
Beteiligte und Interessen:	*Host* Der Host möchte die Detailgenauigkeit im Modus Mausverfolgung ändern.

Vorbedingungen:

- Der Konfigurationsdialog ist geöffnet, siehe *B7: Host öffnet die Konfiguration für* Screensharing.
- Der zu konfigurierende Encoder ist in *B8: Host wechselt das Kodierverfahren* ausgewählt worden.

Erfolgsgarantie: Neue Einstellung ist gespeichert und die geänderten Parameter werden für die nächste *Screenshare*-Sitzung verwendet.

Haupterfolgsszenario:

1. Auf der Hauptseite der Einstellungen vom *Screensharing* das Feld *Captured area in follow mouse mode* wählt der Host die gewünschte Skalierung.
2. Der Host klickt auf *Apply* falls der Wert auf *custom* gesetzt wurde, ansonsten *Ok* und Anwendungsfall beenden.
3. In der Unterseite *Desktop* kann die genaue Auflösung des aufgenommenen Bereichs eingestellt werden.
4. Die Einstellungen werden gespeichert mit einem Klick auf *Apply* oder *Ok*.

8 Arbeit im Saros-Projekt

Den Großteil meiner Arbeitszeit habe ich in dem eigens für die Arbeit am Sarosprojekt bereitgestelltem Raum 017 in der Takustr. 9 verbracht. Es herrschte ein sehr angenehmes und ruhiges Arbeitsklima dort, wenn auch die Luft mitunter nach längerer Zeit etwas stickig wurde.

Im Großen und Ganzen hatte ich das Gefühl mich in die Prozesse des Saros-Teams gut integrieren zu können, obwohl ich leider zugeben muss, dass ich meiner Arbeit am *Screensharing* mehr Zeit gewidmet hatte, als es von Christopher Oezbek erwartet wurde[1].

Während meiner Anwesenheit habe ich die meiste Zeit mit den folgenden Studenten, die auch an Saros mitgewirkt haben, verbracht (alphabetisch sortiert):

- Olaf Loga [Log10]
- Marc Rintsch [Rin]
- Henning Staib [Staa]
- Eike Starkmann [Stab]
- Sandor Szücs [Szü]

Ein großes Problem bei Saros ist, dass das meiste Wissen über die Interna nicht schriftlich festgehalten ist. Vor allem Marc und Sandor konnten mir diesbezüglich vieles zeigen und erklären, da sie bevor meinem Anfang schon einige Zeit Erfahrungen mit Saros gesammelt hatten.

---

1 Zitat aus einer Mail von Christopher (Betreff: `[Saros] Bye Bye Saros List...` gesendet am Mo, 8.03.2010, 14:56 Uhr an die interne Mailingliste von Saros)

> Think about it like this: 50% of your time in your thesis should be spent on working with the team – Reviewing patches, discussing the future of Saros, explaining technology, fixing bugs, making releases. 25% should be spent on developing features and writing test cases, but not more. The rest of the time is necessary to write the thesis.

Ich habe hierbei gefühlte 20 statt 50% meiner Zeit mit der Teamarbeit verbracht.

Neben der alltäglichen Arbeit an unserem eigenen Projekt, hatten wir, wie schon erwähnt, die Aufgabe uns um Saros zu kümmern. Darunter fielen folgende Sachen

1. Saros zu verbessern, also Bugs zu beheben und nach Möglichkeit zu erweitern.
   Patches, die von anderen *Sarosianern* eingereicht wurden, mussten bevor sie in dem Quellcode von Saros integriert wurden, möglichst sorgfältig überprüft werden. Dies schließt die statische Durchsicht auf Fehler und eventuelle Defekte, sowie Einhaltung der *Codeconventions*[Sar] ein. Eine Übersicht der Patches die ich überprüft habe sind in Tabelle 8.2 zu finden. Patches, die ich selber erstellt habe, beziehen sich auf die Integration der von mir entwickelten Funktionen sowie Verbesserungen dieser, siehe Tabelle 8.4.

2. Regelmäßig eine neue Version von Saros zu veröffentlichen.
   Anfangs wurde alle 3 Woche eine neue Version veröffentlicht, doch aufgrund der Umstrukturierung der Projektleitung von Saros wurde dies sehr unregelmäßig, bis Karl Beecher einen fixen Zyklus von 4 Wochen festgelegt hat[1]. Ich selbst habe bei den Releases *9.10.30.r1833* als *Releasemanager* (RM) und *9.12.4.r1878* als *Assistant Releasemanager* (ARM) mitgewirkt. Bei den Releases von *9.9.11.r1706* bis *10.1.29.r1970*, sowie *10.3.26.r2105* habe ich auch als Tester teilgenommen.

1 Siehe Mail `Plans for this release cycle` gesendet am Di, 2.02.2010, 12:27 Uhr an die interne Mailingliste von Saros

Titel des Patches (Betreff der Mail in der er eingereicht wurde, sortiert nach Datum)	zum Quellcode des Hauptentwicklungszweigs in Revison x hinzufügt
#2869752 Viewport follows refactoring	1825
UndoTest	1844
error message in Saros#connect	1844
exception handling	1844
Path before diff output in log	1848
Remove a special case for the Host in User	1856
Removed unnecessary XStream annotations	1851
Saros enabled filter deactivated by default	1857
Project synchronization confirmation	1966
fix renaming folders in project root	1862
refactor JingleFileTransferSession	1895
DTM should not know what XMPP Packets/Messages are	1893
fix Race condition while connect disconnect	1894
fix ByteBuffer.array not supported on some platforms	1933
BinaryChannel speed-up + unit test	1959
Unnest blocks in Util...	1946
Patch review - StatisticsAggregator	1957
VoIP	2089
Archive streaming	– (noch nicht hinzugefügt)

Tabelle 8.2: Von mir überprüfte Patches für Saros.

Titel des Patches (Betreff der Mail in der er eingereicht wurde, sortiert nach Datum)	zum Quellcode des Hauptentwicklungszweigs in Revison x hinzufügt
creating sockets for jingle media sessions	leider abgelehnt, siehe Abschnitt 7.1
handle EOF in BinaryChannel	1965
streaming services	2065
sendfile and some fixes for streaming	2078
Screensharing	noch nicht vollständig überprüft

Tabelle 8.4: Von mir eingereichte Patches für Saros.

9 Ausblick

Die Entwicklung des *Screensharings* für Saros ist noch nicht abgeschlossen. Leider konnten die genannten Anforderungen nicht vollständig umgesetzt werden. Mit der aktuellen Implementierung ist derzeit nur passives *Screensharing* möglich und die genutzte Bandbreite passt sich nicht dynamisch den Gegebenheiten an.

Die Möglichkeit der statistischen Auswertung ist auch noch nicht möglich, da keine Daten über den Verlauf der Sitzung aufgezeichnet werden. Dies ist jedoch dringend notwendig um den Nutzen von *Screensharing* für Saros und DPP untersuchen zu können. Um eine tiefgehende Untersuchung zu ermöglichen, ist es auch von Vorteil, wenn die Sitzung komplett aufgezeichnet werden kann.

Einige Verbesserungs- und Erweiterungsmöglichkeiten werden ich auch noch vorstellen.

9.1 Aktives *Screensharing*

Wie schon erwähnt, ist das derzeitig implementierte Screensharing nur passiv, d.h. der Client kann den Desktop des Hosts nur betrachten. Die derzeitige Realisierung erlaubt es schon Eingaben in die *Remote Screen View* zu tätigen, welche auch an den Host gesendet werden[1].

Wie bei den Anforderungen erläutert sollte es jedoch nicht jedem Client erlaubt sein auch tatsächlich die virtuelle Arbeitsfläche zu bedienen, von daher muss noch eine benutzerbasierte Autorisierung stattfinden. Es bietet sich hierbei an die möglichen Rechte zu unterscheiden nach

Keine Eingaben: Keine Eingaben des Benutzers werden ausgeführt.

Passive Eingaben: Es werden zwar keine Eingaben getätigt, der Client erhält jedoch die Möglichkeit Stellen des Desktops mittels Mausklick zu markieren. Dies ist ähnlich

1 Derzeit nur Mausklicks, Bewegungen mit dem Mausrad und Tastaturanschläge ohne Tastenkombinationen oder Sondertasten.

der Funktion des *Telepointers* von Hanks, siehe hierzu Abschnitt 4.1. Dieses Konzept könnte noch erweitert werden auf die Anzeige der Tastendrücke.

Aktive Eingaben: Volle Eingabemöglichkeit für den Client.

Um den Endbenutzer diese Rechte so unkompliziert wie möglich ändern zu lassen, sollten in dem Kontextmenü des jeweiligen Benutzers, endweder in der *Shared Project* oder *Roster View*, diese Rechte selektierbar sein.

Nachdem diese dann in den Eclipse-Einstellungen persistent gespeichert wurden, kann die Klasse `ActivityManager` die Autorisierung der empfangenen Aktivitäten während des *Screensharings* vornehmen.

9.2 Aufnahme und Wiedergabe einer Sitzung

Um DPP-Sitzungen für unsere Forschungen besser auswerten zu können, bietet es sich an diese aufzuzeichnen. Sei es nun um die Benutzung des neuen Features *Screensharing* im speziellen oder Saros im allgemeinen zu untersuchen.

Die Speicherung der Bildsequenz des Desktops vom Host ist jetzt schon prinzipiell möglich, sofern der `XugglerEncoder` benutzt wird. Er kodiert die virtuelle Arbeitsfläche als Video, somit besitzt jedes Bild einen Zeitstempel wann es gezeigt werden soll. Bei anderen Encodern oder den Aktivitäten müssen noch Zeitstempel integriert werden, damit diese synchron laufen.

Da alle Daten über Datenströme verschickt werden, bietet es sich an diese beim Host oder Client direkt in eine Datei zu speichern.

Zur späteren Wiedergabe der gespeicherten Sitzung müsste nun noch eine Komponente entwickelt werden, die diese Dateien einliest, einen entsprechenden Dekoder zur Wiedergabe startet, diesen mit Daten versorgt und die Aktivitäten des Clienten, soweit vorhanden, zur entsprechenden Zeit ausgibt.

9.3 Statistische Auswertung

Lisa Dohrmann hat in ihrer Arbeit [Doh09] ein Statistik-Framework für Saros entwickelt. Es bietet sich an die Daten über die Benutzung von *Screensharing*, wie Dauer und Häufigkeit, in die Sammlung der Statistiken zu integrieren.

9.4 Anpassung der genutzten Bandbreite

Um *Screensharing* in bester Qualität zu ermöglichen, wird bei der Konfiguration des Encoders die maximal vorhandene Bandbreite angegeben[1]. Es kann jedoch passieren das die verfügbare Bandbreite um einiges niedriger ist, wie in Unterunterabschnitt 6.2.6.1 beschrieben. Der dort erwähnte Algorithmus ist noch nicht implementiert, da die Zeit für eine ausreichende Evaluation gefehlt hat.

9.5 Weitere Encoder

Die zwei schon implementierten Kodierverfahren funktionieren zwar, doch sie haben noch einige Mängel.

`XugglerEncoder` bietet zwar eine gute Performance, ist jedoch abhängig vom Betriebssystem und benötigt daher eine extra Installation von Xuggler[Xug]. Benutzern könnte es zu aufwendig oder vielleicht auch gar nicht möglich sein dies zu installieren. Diese wären dann gezwungen den anderen Encoder, `ImageTiles`, zu verwenden, der aber sehr rechenintensiv ist. Bei Tests waren sogar auf sehr schnellen modernen Rechnern nur zwei FPS bei guter Auflösung möglich.
Es scheint als ob die Komprimierung der Bilder, wie sie in Java integriert ist [Sun, javax.imageio], nicht sehr effizient ist. Die aktuelle Implementierung könnte noch verbessert werden, in dem die Bildkomprimierung auf mehrere Threads verteilt wird, wenn Mehrkernprozessoren verwendet werden.

In Zukunft sollten jedoch Encoder anderer Java-Programme für *Screensharing* in das VSF integriert werden. Geeignete Kandidaten wären zum Beispiel *Dayon!*[Pol] oder *Java Remote Control*[Jav], da sie unter freien Lizenzen stehen und funktionsfähig sind. Nach dem Studium des nahezu undokumentierten Quelltextes beider Programme schien es jedoch zu zeitaufwendig den Encoder zu extrahieren, woraufhin der `ImageTileEncoder` entstanden ist.

1 siehe hierzu B9: Host ändert die Basiseinstellungen für die Kodierung

9.6 Andere Bildquellen

Anstelle des Desktops kann auch jede andere beliebige Bildquelle übertragen werden. Es bietet sich zum Beispiel an die Bilder einer PC-Kamera zu übertragen[1].

Um dies zu erreichen, ist es nötig eine Klasse zu programmieren die das Interface `ImageSource` implementiert und Bilder der PC-Kamera liefern kann. Ähnlich dem Wechseln des Anzeigemodus[2] muss dem Client die Möglichkeit gegeben werden, zwischen virtuellem Desktop und PC-Kamera wechseln zu können. Der Host muss die PC-Kamera-Funktion konfigurieren können, also sie endweder ganz deaktivieren und, falls mehrere PC-Kameras o. ä. angeschlossen sind, eine PC-Kamera auswählen können.

9.7 Besserer Schutz der Privatsphäre

Einer der großen Nachteile von *Screensharing* für die Anwendung in DPP ist der fehlende Schutz der Privatsphäre (Unterabschnitt 2.3.3). Dies ist dadurch bedingt das immer der gesamte Bildschirm übertragen wird. Für den Benutzer sollte es jedoch möglich sein, sensible Bereiche unkenntlich zu machen oder gar nicht zu übertragen.

Die vorhandene Implementierung bietet schon einen rudimentären Schutz der Privatsphäre:

- Es wird stets nur der Inhalt des primären Bildschirms übertragen. Damit ist jedoch nur ein Schutz möglich, wenn mehrere Monitore eingesetzt werden.
- Durch die konfigurierbare Auflösung, sowohl die der Übertragung an sich als auch der im *Mausverfolgungsmodus*, lässt sie die Detailgenauigkeit der gesendeten Bilddaten regulieren.

In der CB (Abschnitt 4.2) wurde gute Prinzipien zum Schutz der Privatsphäre eingeführt. Einige dieser Ideen lassen sich auch hier verwenden:

- Limitierung des Bildschirmausschnittes der übertragen wird. Besser wäre sogar die Beschränkung auf einzelne Fenster bzw. Prozesse, doch auch nach langer Recherche habe ich keinen Weg gefunden, wie es in Java möglich ist die Position von Fenstern anderer Prozesse zu bestimmen.

1 In Verbindung mit der *Voice over IP* (VoIP)-Funktion[Log10] ist hiermit sogar Bildtelefonie möglich.
2 siehe hierzu B6: Client wechselt zwischen Vollbilddarstellung und Mausverfolgung

- Maskierung sensibler Bereiche durch grafische Effekte.
- Beschränkung der maximalen Vergrößerung eines Ausschnittes, sofern die Möglichkeit des Vergrößerns auch in dieser Umsetzung implementiert wird.

9.8 Mehrere Empfänger

Da eine Saros-Sitzung auch mehrere Teilnehmer haben kann, sollte es möglich sein, eine oder mehrere Arbeitsflächen unter den Sitzungsteilnehmern zu verteilen. Derzeit kann leider nur der lokale Desktop an einen anderen Teilnehmer gesendet werden. Mit mehreren Teilnehmern wird es komplizierter die gegebenen Anforderungen umzusetzen.

Ich werde hier einige Möglichkeiten vorstellen wie dies doch gelingen kann und die Vor- und Nachteile diskutieren.

Alle Empfänger haben keine Möglichkeit zur Interaktion. Bei diesem Szenario kann der Desktop nur betrachtet werden. Es können keine Eingaben getätigt werden, sowie der Anzeigemodus kann nicht gewechselt werden. Der Host legt hierbei den initialen Anzeigemodus fest.

Ein ausgewählter Empfänger kann interagieren, alle Anderen nicht. Im Gegensatz zur vorherigen Möglichkeit kann ein Teilnehmer wie gewohnt interagieren, die Anderen betrachten dies nur.

Alle Empfänger können interagieren. Jedem Teilnehmer steht die gewohnte Funktionalität zur Verfügung.

Diese Szenarien haben alle den Nachteil, dass der Host jedem seinen Bildschirminhalt übertragen muss. Dies verringert in der Regel die Qualität, da die Clienten sich die verfügbare Bandbreite teilen und diese in Senderichtung meist sehr niedrig ist.

In den ersten beiden Szenarien muss sich der Encoder an die Empfangs- und Verarbeitungsleistung des *schlechtesten* Clienten orientieren. Die Anderen empfangen dadurch den Desktop in einer schlechteren Qualität als es eigentlich möglich wäre.

Wenn alle Empfänger interagieren können, muss beim Host für jeden Empfänger ein eigener Encoder laufen. Dadurch kann zwar jeder Client den Bildschirm in fast optimaler Qualität empfangen, doch beim Host kann, je nach Teilnehmerzahl und Einstellungen der Kodierung, der Rechner an seine Leistungsgrenzen stoßen.

9.8.1 Verteilen der Videostreams mit einem Flash Media Server

Im Gegensatz zu den vorher genannten Szenarien, bei denen der Host an alle Clients seinen Bildschirminhalt überträgt, gibt es die Möglichkeit die Videodaten mit einem *Flash Media Server* (FMS)[1] zu verteilen[2]. Von Vorteil ist hierbei, dass der FMS die Videoströme sofort an alle Clienten verteilen und bei Bedarf, wenn ein Client eine niedrigere Empfangsbandbreite hat als das Video, rekodieren kann.

Es werden mehrere Szenarien der Live-Kommunikation unterstützt [Ado]

One2many Ein Host sendet sein Video dem FMS, welcher dieses unter den Clients verteilt (Abbildung 9.1).

Many2many Alle Teilnehmer tauschen Daten über den FMS aus (Abbildung 9.2). Dies könnte es ermöglichen, dass die Teilnehmer einer Saros-Sitzung eine verkleinerte Version ihres Desktops untereinander verteilen können und bei Bedarf in eine direkte *Screenshare*-Sitzung wechseln können, wie es in der CB (Abschnitt 4.2) gemacht wird.

Nachteilig ist in jedem Fall, dass ein dedizierter Server mit genügend Ressourcen und Bandbreite benötigt wird.

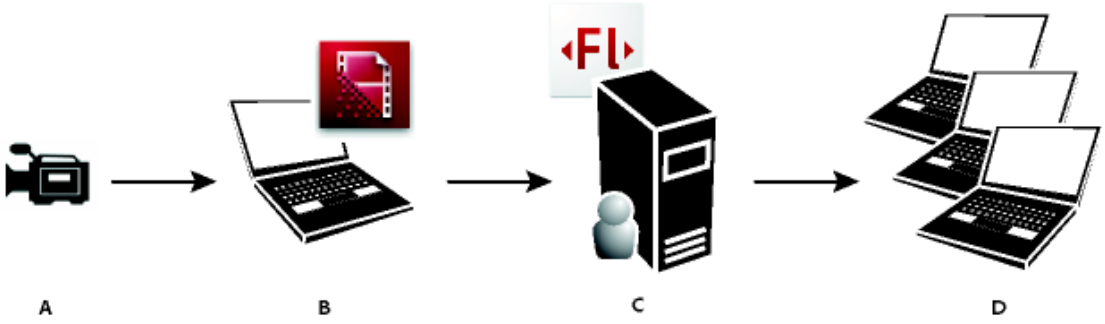

Abbildung 9.1: One2many-Verteilung des FMS[Ado]

1 Weitere Implementierungen sind der *Wowza Media Server*[Wow] (auch kommerziell) und *Red5*[Red] (frei). Ein kurzer Vergleich der drei Varianten ist unter `http://askmeflash.com/article/10/comparison-wowza-vs-fms-vs-red5` zu finden.

2 Es werden auch Audio- und Binärdaten unterstützt.

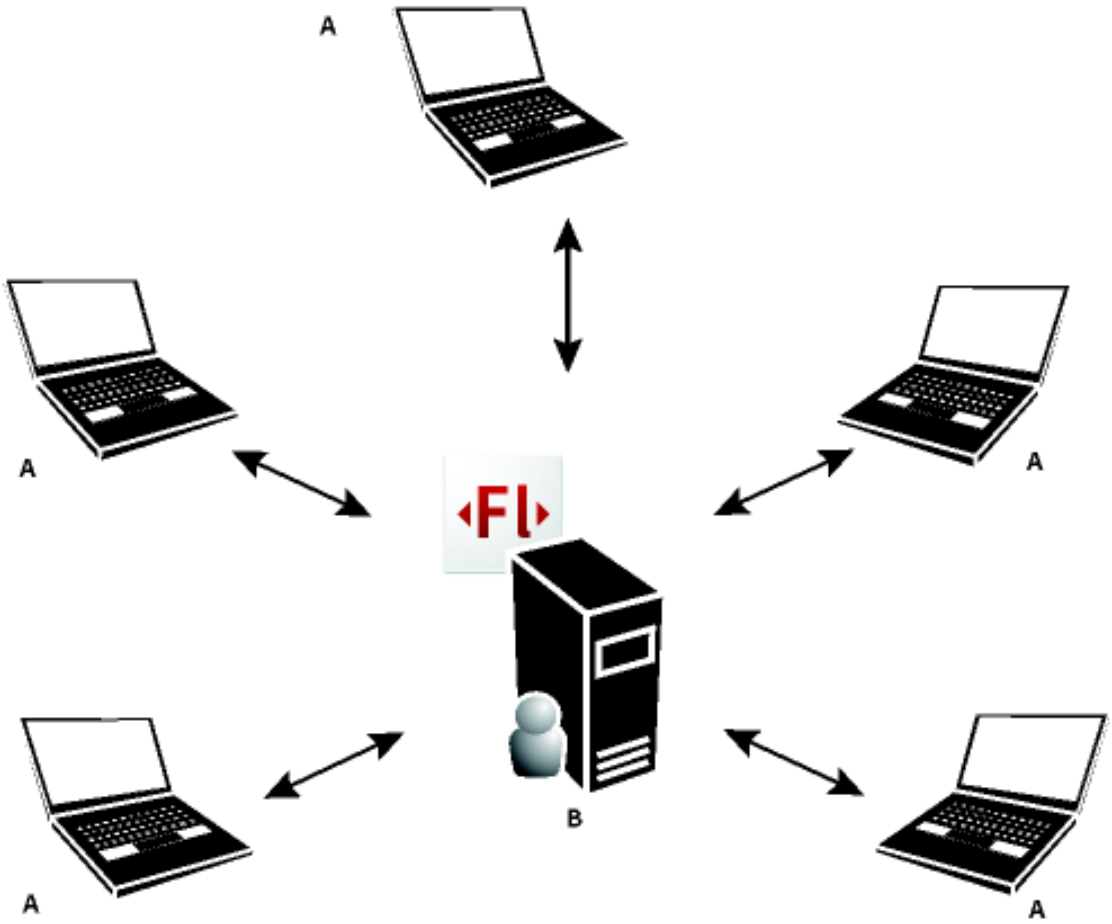

Abbildung 9.2: Many2many-Verteilung des FMS[Ado]

10 Schluss

In dieser Arbeit wurde Saros um die Zusatzfunktion *Screensharing* erweitert. Vor allem die Erkenntnis, dass die Entwicklung an einem Projekt auch mehr Werkzeuge involviert als Eclipse und auch dieses nur unvollständig unterstützt wird, führt zu diesem notwendigen Schritt.

Dazu wurde erst abstraktes Framework entworfen, mit dem es möglich ist Bildsequenzen vom einem zum anderen Rechner zu übertragen und beliebige Kodierverfahren hierbei einzusetzen. Mithilfe dieses Frameworks wurde nun *Screensharing* in Saros integriert. Es ist nun auch möglich ähnliche Anwendungen, wie zum Beispiel Bildtelefonie, einfach zu implementieren.

Diese Integration war jedoch nicht ohne weiteres möglich. Es musste vorher die Möglichkeit geschaffen werden, dass die Teilnehmer einer Saros-Sitzung Daten mittels Datenströme austauschen können, bisher war dies nur blockweise möglich. Deswegen wurde noch das *Stream Service Framework* entwickelt und implementiert, welches nicht nur für das *Screensharing*, sondern auch für andere datenstrombasierten Dienste, von Nutzen ist. Neue Funktionen wie VoIP benutzen dies schon, weitere sind geplant.

Nun ist es den Teilnehmern einer Saros-Sitzung möglich ihren Bildschirm untereinander zu teilen. Aufgrund der Komplexität des SSF und dem daraus resultierenden hohen Entwicklungsaufwand konnten leider nicht alle Punkte der Anforderungen an das *Screensharing* umgesetzt werden. Am gravierendsten ist, dass das *Screensharing* sich nur auf die Anzeige des geteilten Bildschirms beschränkt. Interaktion mit diesem ist zwar prinzipiell schon möglich, dies wurde jedoch noch nicht implementiert.

Nach einem ersten Test in der Gruppe der Sarosentwickler wurde die in dieser Arbeit entwickelte *Screensharing*-Funktion positiv aufgenommen und dem baldigen Einsatz in der Praxis steht nun nichts mehr im Wege.

A Anhang

A.1 Anhang A: Das Aufgabendokument

```
X-Arbeit Stephan Lau

---+ Verbesserte Präsenz durch Screensharing für ein Werkzeug zur Verteilten
     Paarprogrammierung

  * Ziel der Arbeit:

    * Saros soll um Screensharing Unterstützung erweitert werden

  * Forschungsfragen:

    * Welche Vor- und Nachteile lassen sich zwischen Screensharing
      und Collaboration Awareness lassen sich aus der Literatur und
      theorethischen Überlegungen ableiten?
    * Empirisch:
      * Welcher Anteil einer PP-Sitzung wird Screensharing bevorzugt?
      * Kann man sagen in welchen Situationen dies insbesondere ist?
    * Wie ermöglicht man ein möglichst flüssiges Umschalten zwischen SS und CA?

  * Deine konkreten Aufgaben:

    * Eine möglichst fertige Lösung zum Streamen von Screen capture Shots in
      Saros integrieren.
      * Die Übertragung der Daten muss hierbei über einen von uns
        bereitgestellten Stream erfolgen
        DataTransferManager.send()
```

```
    * Ein bißchen Nicht-Funktionale Anforderungen:
      * Die Framerate/Qualität für Screensharing muss etwas dynamisch
        angepasst werden, je nachdem wieviel Bandbreite zur Verfügung steht.
        => Screensharing soll nicht die Saros-Activity-Übertragung blockieren.
    * Wann immer möglich sollte existierende Software/APIs/Tools verwendet werden.
      => 1te konkrete Aufgabe - Überblick über existierende Software
Java/Platformunabhängig gewinnen.
    * Im Netzwerkmodul sind entsprechende Anpassungen durchzuführen um auch
      beim Fehlen einer Jingle-Verbindung (d.h. dann ist XMPP FileTransfer
      bzw. IBB möglich) ein Screensharing zu ermöglichen

    * Wenn möglich sollte der Benutzer auswählen können, ob er nur Eclipse oder
      den ganzen Desktop teilen möchte.

    * Am Ende Deiner Arbeit sollte eine kleine empirische Evaluation Deines
      Arbeitsergebnisses stehen.
      * Minimal: Demonstration bei Deinen Betreuern
      * Mittel: Du lässt mehrere Studenten hier am Institut Dein Feature testen
      * Maximal: Echte Nutzer werden von Dir in der freien Wildbahn befragt
```

A.2 Anhang B: Manuelle Testfälle für SendFileAction

Dies sind die manuellen Testfälle wie sie in unser internes Testverwaltungsprogramm Testlink[TEA] eingetragen wurden.

```
1. Successful transmission

Pre-condition:
Shared project with 2 users Alice and Bob.
Alice has a file with atleast 20MB to transmit.

Alice selects Bob in the Shared Project Session view
Alice clicks the file icon in the Shared Project Session view's toolbar
Alice chooses any file with at least 20MB
Bob hits yes at the appearing question dialog
```

```
Bob chooses any location for saving the file

Expectation:
File successfully transmitted (check size, optional: verify with checksum),
job is completed (see Progress view) at Alice and Bob


2. Receiver does not accept file

Pre-condition:
Shared project with 2 users Alice and Bob.

Alice selects Bob in the Shared Project Session view
Alice clicks the file icon in the Shared Project Session view's toolbar
Alice chooses any file
Bob hits no at the appearing question dialog

Expectation:
Job is canceled (see Progress view) at Alice and Bob and
Alice sees an error dialog.


3. Receiver cancels file dialog

Pre-condition:
Shared project with 2 users Alice and Bob.
Alice has a file with at least 20MB to transmit.

Alice selects Bob in the Shared Project Session view
Alice clicks the file icon in the Shared Project Session view's toolbar
Alice chooses any file
Bob hits yes at the appearing question dialog
Bob cancels the file dialog

Expectation: Job is canceled (see Progress view) at Alice and Bob
```

```
4. Receiver cancels the transmisson

Pre-condition:
Shared project with 2 users Alice and Bob.
Alice has a file with at least 20MB to transmit.

Alice selects Bob in the Shared Project Session view
Alice clicks the file icon in the Shared Project Session view's toolbar
Alice chooses any file,
 it must be big enough that Bob has the chance to cancel the transfer
 (depends on connection)
Bob hits yes at the appearing question dialog
Bob chooses any location for saving the file
Bob stops the transmission by canceling the operation in the Progress view

Expectation:
Job is canceled (see Progress view) at Alice and Bob


5. Sender cancels the transmisson

Pre-condition:
Shared project with 2 users Alice and Bob.
Alice has a file with at least 20MB to transmit.

Alice selects Bob in the Shared Project Session view
Alice clicks the file icon in the Shared Project Session view's toolbar
Alice chooses any file,
 it must be big enough that Alice has the chance to cancel the transfer
 (depends on connection)
Bob hits yes at the appearing question dialog
Bob chooses any location for saving the file
Alice stops the transmission by canceling the operation in the Progress view
```

```
Expectation:
Job is canceled (see Progress view) at Alice and Bob
```

Literaturverzeichnis

[ADB+99] Abowd, Gregory D. ; Dey, Anind K. ; Brown, Peter J. ; Davies, Nigel ; Smith, Mark ; Steggles, Pete: Towards a Better Understanding of Context and Context-Awareness. In: *Handheld and Ubiquitous Computing* Bd. 1707, Springer Berlin / Heidelberg. – ISBN 978–3–540–66550–2, 304–307

[Ado] Adobe Systems Inc.: *Adobe Flash Media Server 3.5 Technical Overview*. `http://help.adobe.com/en_US/FlashMediaServer/3.5_TechOverview/`. – Online–Ressource, Abruf: 30. April 2010

[APS99] Allman, M. ; Paxson, V. ; Stevens, W.: *TCP Congestion Control*. RFC 2581. `http://www.ietf.org/rfc/rfc2581.txt`. Version: 4 1999

[BA04] Beck, Kent ; Andres, Cynthia: *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004. – ISBN 0321278658

[Bel] Bellard, Fabrice: *FFmpeg multimedia system*. `http://www.ffmpeg.org`. – Online–Ressource, Abruf: 30. April 2010

[BLSA+] Beda, Joe ; Ludwig, Scott ; Saint-Andre, Peter ; Hildebrand, Joe ; Egan, Sean ; McQueen, Robert: *XEP-0176: Jingle ICE-UDP Transport Method*. `http://xmpp.org/extensions/xep-0176.html`. – Online–Ressource, Abruf: 30. April 2010

[BSAL+] Beda, Joe ; Saint-Andre, Peter ; Ludwig, Scott ; Hildebrand, Joe ; Egan, Sean: *XEP-0177: Jingle Raw UDP Transport Method*. `http://xmpp.org/extensions/xep-0177.html`. – Online–Ressource, Abruf: 30. April 2010

[CR06] Clayberg, Eric ; Rubel, Dan: *Eclipse: Building Commercial-Quality Plug-ins (2nd Edition) (Eclipse)*. Addison-Wesley Professional, 2006. – ISBN 032142672X

[CSH+04] Cheng, Li-Te ; Souza, Cleidson R. ; Hupfer, Susanne ; Patterson, John ; Ross, Steven: Building Collaboration into IDEs. In: *Queue* 1 (2004), Nr. 9, S. 40–50. `http://dx.doi.org/10.1145/966789.966803`. – DOI 10.1145/966789.966803. – ISSN 1542–7730

[CW01] Cockburn, Alistair ; Williams, Laurie: The costs and benefits of pair programming. (2001), S. 223–243. ISBN 0–201–71040–4

[Dje06] Djemili, R.: *Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierung verteilter Paarprogrammierung*, Freie Universität Berlin, Diplomarbeit, 2006

[Doh09] Dohrmann, Lisa: *Erhebung von Benutzerfeedback aus der Nutzung eines Werkzeugs zur verteilten Paarprogrammierung*, Freie Universität Berlin, Bachelorarbeit, 2009

[DOS07] Djemili, Riad ; Oezbek, Christopher ; Salinger, Stephan: Saros: Eine Eclipse-Erweiterung zur verteilten Paarprogrammierung. In: *Software Engineering (Workshops)*, 2007, S. 317–320

[FLV07] *Encoding von Flashvideos.* Springer Berlin Heidelberg (X.media.press). – 135–148 S. – ISBN 978–3–540–37894–5 (Print) 978–3–540–37895–2 (Online)

[Fog96] Fogg, C.: *DVD Technical Notes – Glossary.* Version: 1996. `http://www.mpeg.org/MPEG/DVD/General/Glossary.html`. – Online–Ressource, Abruf: 30. April 2010

[Fow04] Fowler, Martin: *Inversion of Control Containers and the Dependency Injection pattern.* Version: 2004. `http://www.martinfowler.com/articles/injection.html`. – Online–Ressource, Abruf: 30. April 2010

[GG98] Gutwin, Carl ; Greenberg, Saul: Effects of awareness support on groupware usability. In: *CHI '98: Proceedings of the SIGCHI conference on Human factors in computing systems.* New York, NY, USA : ACM Press/Addison-Wesley Publishing Co., 1998. – ISBN 0–201–30987–4, S. 511–518

[GHJV09] Gamma, E. ; Helm, R. ; Johnson, R. ; Vlissides, J.: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software.* Addison Wesley Verlag, 2009

[Han02] Hanks, Brian F.: Tool Support for Distributed Pair Programming. In: *„Knowledge Management Support for Distributed Agile Software Processes“, Submitted for the final Proc. of the Fourth International Workshop on Learning Software Organizations (LSO)*, 2002. – Position Paper, Workshop „Distributed Pair Programming, Extreme Programming and Agile Methods“ in XP/Agile Universe 2002, August 6

[Han04] Hanks, Brian F.: Distributed Pair Programming: An Empirical Study. In: *Extreme Programming and Agile Methods - XP/Agile Universe 2004* Bd. 3134, Springer Berlin / Heidelberg. – ISBN 978–3–540–22839–4, 81–91

[Han05] Hanks, Brian F.: *Empirical studies of distributed pair programming.* Santa Cruz, CA, USA, Diss., 2005. – Chair-Mcdowell, Charlie

[Jav] Java Remote Control Team: *Java Remote Control Homepage.* `http://code.google.com/p/java-remote-control/`. – Online–Ressource, Abruf: 30. April 2010

[Jiva] Jive Software: *Smack API.* `http://www.igniterealtime.org/projects/smack/`. – Online–Ressource, Abruf: 30. April 2010

[Jivb] Jive Software: *Smack Extensions Manual.* `http://www.igniterealtime.org/builds/smack/docs/latest/documentation/extensions/index.html`. – Online–Ressource, Abruf: 30. April 2010

[Kal06] Kalva, Hari: The H.264 Video Coding Standard. In: *IEEE MultiMedia* 13 (2006), S. 86–90. `http://dx.doi.org/10.1109/MMUL.2006.93`. – DOI 10.1109/MMUL.2006.93. – ISSN 1070–986X

[KCSL00] Kazi, Iffat H. ; Chen, Howard H. ; Stanley, Berdenia ; Lilja, David J.: Techniques for obtaining high performance in Java programs. In: *ACM Comput. Surv.* 32 (2000), Nr. 3, S. 213–240. `http://dx.doi.org/10.1145/367701.367714`. – DOI 10.1145/367701.367714. – ISSN 0360–0300

[KJCL01] Kircher, Michael ; Jain, Prashant ; Corsaro, Angelo ; Levine, David: Distributed eXtreme Programming. In: *Second international conference on eXtreme Programming and Agile Processes in Software Engineering*, 2001, S. 66–71

[Kra07] Kraif, Ursula: *Duden, das Fremdwörterbuch.* Dudenverlag, 2007 (Der Duden in zwölf Bänden ; 5). – 1104 S. S. – ISBN 978–3–411–04059–9, 3–411–04059–9

[KSA09] Karneges, Justin ; Saint-Andre, Peter: *XEP-0047: In-Band Bytestreams.* Version: 2009. `http://xmpp.org/extensions/xep-0047.html`. – Online–Ressource, Abruf: 30. April 2010

[Log10] Loga, Olaf: *Verbesserung der Kommunikationsmöglichkeiten in Saros.* Version: 2010. `https://www.inf.fu-berlin.de/w/SE/ThesisDPPXV`. – Online–Ressource, Abruf: 30. April 2010

[LSAE+] Ludwig, Scott ; Saint-Andre, Peter ; Egan, Sean ; McQueen, Robert ; Cionoiu, Diana: *XEP-0167: Jingle RTP Sessions.* `http://xmpp.org/extensions/xep-0167.html`. – Online–Ressource, Abruf: 30. April 2010

[Mau02] Maurer, Frank: Supporting Distributed Extreme Programming. In: *Extreme Programming and Agile Methods – XP/Agile Universe 2002* Bd. 2418, Springer Berlin / Heidelberg. – ISBN 978–3–540–44024–6, 95–114

[OBL+04] Ostermann, J. ; Bormans, J. ; List, P. ; Marpe, D. ; Narroschke, N. ; Pereira, F. ; Stockhammer, T. ; Wedi, T.: Video Coding with H.264/AVC: Tools, Performance and Complexity. In: *IEEE Circuits and Systems Magazine* (2004), April, Nr. 4(1), S. 7–28

[On205] On2 Technologies Inc.: *On2 VP6 for Flash 8 Video.* Version: 2005. `http://www.on2.com/file.php?127`. – Online–Ressource, Abruf: 30. April 2010

[Pic] PicoContainer Committers: *PicoContainer Homepage.* `http://www.picocontainer.org/`. – Online–Ressource, Abruf: 30. April 2010

[Pol] Polizzi, Marc: *Dayon! Homepage.* `http://dayonhome.sourceforge.net/`. – Online–Ressource, Abruf: 30. April 2010

[Red] Red5 Project: *Red5 Homepage.* `http://red5.org/`. – Online–Ressource, Abruf: 30. April 2010

[Rin] Rintsch, Marc: *Technische Betreuung von Saros in einem betieblichen Umfeld.* `https://www.inf.fu-berlin.de/w/SE/ThesisDPPXIV`. – Online–Ressource, Abruf: 30. April 2010

[RSFWH98] Richardson, Tristan ; Stafford-Fraser, Quentin ; Wood, Kenneth R. ; Hopper, Andy: Virtual Network Computing. In: *IEEE Internet Computing* 2 (1998), Nr. 1, S. 33–38. http://dx.doi.org/10.1109/4236.656066. – DOI 10.1109/4236.656066. – ISSN 1089–7801

[RSVW94] Reinhard, Walter ; Schweitzer, Jean ; Völksen, Gerd ; Weber, Michael: CSCW Tools: Concepts and Architectures. In: *Computer* 27 (1994), Nr. 5, S. 28–36. http://dx.doi.org/10.1109/2.291293. – DOI 10.1109/2.291293. – ISSN 0018–9162

[San] Sangam Project Community: *Sangam Homepage*. http://sangam.sourceforge.net/. – Online–Ressource, Abruf: 30. April 2010

[Sar] Saros Team: *Saros Developer Guide*. https://www.inf.fu-berlin.de/w/SE/DPPHowToDevel. – Online–Ressource, Abruf: 30. April 2010

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

[SOBS10] Salinger, Stephan ; Oezbek, Christopher ; Beecher, Karl ; Schenk, Julia: *Saros: An Eclipse Plug-in for Distributed Party Programming*. 2010. – Submitted for CHASE 2010, Workshop at the ICSE 2010 May 2nd 2010

[SS02] Slingerland, Nathan ; Smith, Alan J.: Measuring the Performance of Multimedia Instruction Sets. In: *IEEE Trans. Comput.* 51 (2002), Nr. 11, S. 1317–1332. http://dx.doi.org/10.1109/TC.2002.1047756. – DOI 10.1109/TC.2002.1047756. – ISSN 0018–9340

[Staa] Staib, Henning: *Verbesserung einer XMPP-Bibliothek für den Einsatz in verteilter Paarprogrammierung*. https://www.inf.fu-berlin.de/w/SE/ThesisDPPX. – Online–Ressource, Abruf: 30. April 2010

[Stab] Starkmann, Eike: *Verteilte Paarprogrammierung in Open Source Projekten*. https://www.inf.fu-berlin.de/w/SE/ThesisDPPIX. – Online–Ressource, Abruf: 30. April 2010

[Sun] Sun Microsystems Inc.: *Java™2 Platform Standard Edition 5.0 API Specification*. http://java.sun.com/j2se/1.5.0/docs/api/. – Online–Ressource, Abruf: 30. April 2010

[SWB+03] Stotts, David ; Williams, Laurie ; Baheti, Prashant ; Jen, Dennis ; Jackson, Anne: Virtual teaming: experiments and experiences with distributed pair programming. 2003. – Technischer Bericht

[Szü] Szücs, Sandor: *Behandlung von Netzwerk- und Sicherheitsaspekten in einem Werkzeug zur verteilten Paarprogrammierung.* `https://www.inf.fu-berlin.de/w/SE/ThesisDPPVI`. – Online–Ressource, Abruf: 30. April 2010

[TEA] TEAMST: *Testlink Homepage.* `http://www.teamst.org/`. – Online–Ressource, Abruf: 30. April 2010

[TGG06] Tee, Kimberly ; Greenberg, Saul ; Gutwin, Carl: Providing artifact awareness to a distributed group through screen sharing. In: *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work.* ACM. – ISBN 1–59593–249–6, 99–108

[TGGM06] Tee, K. ; Greenberg, S. ; Gutwin, C. ; McEwan, G.: Shared Desktop Media Item: The Video. In: *Demonstration and short paper, Adjunct Proceedings ACM CSCW 2006*, 2006. – Video and two page paper, duration 4:00

[Wil00] Williams, Laurie A.: *The collaborative software process*, The University of Utah, Diss., 2000. – Adviser: Kessler, Robert R.

[WK02] Williams, Laurie ; Kessler, Robert: *Pair Programming Illuminated.* Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2002. – ISBN 0201745763

[Wow] Wowza Media Systems: *Wowza Media Systems Homepage.* `http://www.wowzamedia.com/`. – Online–Ressource, Abruf: 30. April 2010

[XPa] XPairtise Team: *XPairtise – Pair Programming for Eclipse.* `http://xpairtise.sourceforge.net/`. – Online–Ressource, Abruf: 30. April 2010

[Xug] Xuggle Inc.: *Xuggler.* `http://www.xuggle.com/xuggler/`. – Online–Ressource, Abruf: 30. April 2010