1 Einleitung

1.1 Motivation

Das Peer-to-Peer-Computing ist nicht völlig neu. Es gab schon Überlegungen und Arbeiten zu Architekturen, die man heute als Peer-to-Peer-Architekturen bezeichnen würde. Peer-to-Peer basiert eigentlich auf der ursprünglichen Idee von verteilten Systemen. Diese wurden zunächst häufig in Form von Client/Server genutzt. Es gibt aber auch schon diverse verteilte Lösungen ohne zentralen Aspekt, z.B. verteilte Dateisysteme, MBone-Anwendungen. Nun sind heutzutage Computerleistung, Bandbreite und Speicherkapazität preiswert. Dieser Faktor hat der Peer-to-Peer-Entwicklung einen neuen Aufschwung gegeben. **Peers** können Desktop PCs (Personal Computers), Laptops, Handys oder PDAs (Personal Digital Assistant) u.s.w. sein. In Peer-to-Peer verteilten Systemen sind Peers gleichberechtig, selbst organisiert und selbständig. D.h. Benutzer können direkt miteinander kommunizieren und interagieren. Es existiert keine zentrale Serverrolle.

Im Bereich der vernetzten Zusammenarbeit ist es denkbar, dass ein Peer-to-Peer-System die gemeinsame Nutzung von Anwendungen (Application Sharing) ohne zentralen Server z.B. bei ad hoc Treffen ermöglicht.

Diese Diplomarbeit befasst sich mit der Erstellung einer Peer-to-Peer-Architektur für die gemeinsame Nutzung von Anwendungen. D.h. es wird ein dezentrales System bzw. ein P2P-Framework für die gemeinsame Nutzung von Anwendungen erstellt und darauf werden Peer-to-Peer-Anwendungen aufgebaut.

1.2 Aufgabenstellung

Die Aufgabenstellung wurde folgendermaßen festgelegt:

"Im Gegensatz zur Situation in Vorlesungen steht das Szenario einer Gruppe von Studierenden, die bestenfalls über Notebooks verfügen, von denen möglicherweise keines eine Serverrolle übernehmen kann. Im Rahmen der Diplomarbeit soll daher untersucht werden, wie eine Architektur für die gemeinsame Nutzung von Anwendungen zu gestalten ist, falls kein zentraler Ser-

ver zur Verfügung steht, sondern die entsprechende Funktionalität statt dessen durch eine Reihe gleichberechtigter Komponenten erbracht wird. Dazu ist zunächst herauszufinden, in welchen Punkten sich die beiden Architekturvarianten unterscheiden und wie gut die für eine m:n-Kommunikation zur Verfügung stehenden Java-Mechanismen für diesen Fall geeignet sind. Auf dieser Basis sollen geeignete Komponenten entworfen und prototypisch umgesetzt werden. Soweit möglich, sollen dabei die bereits erstellten, generischen Komponenten aus dem Projekt SASCIA eingesetzt werden und die Kompatibilität erhalten bleiben. Der Prototyp ist anhand ausgewählter Szenarien und Messungen zu bewerten."

1.3 Überblick

Im folgenden Abschnitt wird eine kurze Übersicht über den Inhalt der Kapitel gegeben:

Die Szenarien werden im **Kapitel 2** vorgestellt. Aus den Szenarien wird die Analyse der allgemeinen und speziellen Anforderungen abgeleitet.

Im **Kapitel 3** wird das Peer-to-Peer-Computing kurz vorgestellt und erklärt. Es wird untersucht, in welchen Punkten sich die Client/Server- und P2P-Architektur unterscheiden.

Um die Entwicklungszeit der Software zu verkürzen, werden im **Kapitel 4** anhand der Anforderungen die Eignung des PROEM und JXTA Systems sowie die Wiederverwendbarkeit der SASCIA-Komponenten untersucht.

Im **Kapitel 5** wird die Untersuchung der Replikationsfaktoren für einzelne Anwendungsschichten hinsichtlich der Darstellung einer Anwendung durchgeführt. Dieses Ergebnis und das Ergebnis der Eignungsanalyse von existierenden Systemen im Kapitel 4 führen zur Entwurfsentscheidung. Aus der Entwurfsentscheidung werden die wichtigsten Komponenten der Peer-to-Peer-Architektur für die gemeinsame Nutzung von Anwendungen entworfen.

Anschließend wird im **Kapitel 6** die Implementierung der Komponenten in der Programmiersprache Java beschrieben. Dieses Kapitel beschreibt auch die Benutzungsoberfläche.

Im **Kapitel 7** werden die Messumgebung und -szenarien beschrieben sowie die Messergebnisse analysiert. Um die Funktionalität des P2P-Framework zu testen, wird eine einfache Beispielanwendung implementiert und das Testergebnis wird bewertet.

Mit der Zusammenfassung der Ergebnisse dieser Diplomarbeit sowie einem Ausblick schließt im **Kapitel 8** die vorliegende Ausarbeitung.

2 Anforderungsanalyse

In diesem Kapitel werden Szenarien vorgestellt und darauf untersucht, welche Anforderungen ein Peer-to-Peer-System für die gemeinsame Nutzung von Anwendungen erfüllen muss.

Die Grundlagen für die Erarbeitung der Anforderungen sind dabei sowohl die Anforderungen an das System aus der Aufgabenerstellung und die Charakteristiken von mobilen, ad hoc Peer-to-Peer-Netzwerken als auch Überlegungen, die sich aus den Szenarien einer gemeinsamen Nutzung von Anwendungen ergeben.

2.1 Szenarien

2.1.1 Szenario 1: Zusammenarbeiten bei ad hoc Treffen

In diesem Fall treffen sich **spontan** einige Mitarbeiter in einem Projekt oder einige Studenten. Die Mitarbeiter wollen an einem Dokument zusammenarbeiten. Jeder ist für ein Kapitel des Dokuments zuständig. Die Studenden wollen einen Bericht über ein bestimmtes Thema zusammenschreiben. Jeder hat andere Informationen für den Bericht gefunden. Für die Zusammenarbeit können die Studenten diese Informationen austauschen und gemeinsam verwenden. Jedoch ist **kein Server** verfügbar. Der Ablauf ist wie folgt beschrieben:

Beginn der Sitzung:

1. Ein Benutzer erzeugt eine Peergruppe/Sitzung.
2. Die anderen Benutzer müssen zuerst die Peergruppe finden und können dann an der Peergruppe teilnehmen. Wer an dieser Peergruppe/Sitzung teilnimmt, erhält alle Informationen der Peergruppe (z.B. Name der Teilnehmer, Anwendungen sowie Daten, die für diese Gruppe angeboten werden).
3. Ein Teilnehmer bietet die Anwendung für die Zusammenarbeit z.B. ein Programm für Textbearbeitung für die Peergruppe an und öffnet die Dokumentendatei.

Anmelden an der Sitzung und Zusammenarbeit:

4. Die Teilnehmer können die Anwendung für die Zusammenarbeit nach Beitreten zu der Peergruppe anfragen.
5. Ein Teilnehmer kann die Rolle des Moderators übernehmen, der die Sitzung steuert.

Er gibt einem Teilnehmer das Recht (als Floor bezeichnet), Inhalte zu ändern oder zu blättern. Durch die Unterstützung des Systems kann die Rechtevergabe auch ohne Moderator geschehen.

6. Wer den Floor besitzt, kann Aktionen durchführen. Diese Aktionen (das Blättern oder die Änderungen) werden an alle Teilnehmer übertragen.
7. Kommt ein Teilnehmer nicht vom Beginn der Sitzung, kann er die Anwendung für die Zusammenarbeit und den aktuellen Inhalt abfragen. Wenn der Anbieter aus irgendeinem Grund nicht zur Verfügung steht, kann ein anderer Teilnehmer die Antwort an den Nachzügler senden.

Abmelden von der Sitzung:

8. Jeder Teilnehmer kann sich jeder Zeit von der Gruppe/Sitzung abmelden. Der Benutzer beendet die Anwendung. Die Verbindung zu den anderen Teilnehmern wird abgebaut.
9. Verläßt ein Teilnehmer die Peergruppe, sollte dies die anderen Teilnehmer nicht beeinträchtigen.

2.1.2 Szenario 2: Vorlesung oder Seminar

In diesem Fall hält der Professor/in oder der Dozent/in eine Vorlesung oder ein Seminar. Die Lehrinhalte werden durch eine Anwendung für die Zusammenarbeit z.B. ein Whiteboard oder ein Präsentationsprogramm für Folien präsentiert. Der Ablauf der Sitzung ist ähnlich wie im Szenario 1 beschrieben. Aber der Professor oder der Dozent ist der Moderator.

2.1.3 Zusätzliche Funktionen des Frameworks

Das P2P-Framework unterstützt die Szenarien 1 und 2, die gemeinsame Nutzung von Anwendungen. Daneben ermöglicht das Framework auch das Angebot der zusätzlichen Daten und Anwendungen, die der Zusammenarbeit oder der Sitzung dienen. Z.B. das Anbieten der kompletten Aufzeichnungsdaten für Person, der nicht in der Sitzung war. Oder der Professor gibt den Studenten ein zusätzliches Dokument mit Lehrinhalten oder Übungsaufgaben usw., damit die Studenten es zu Hause anschauen oder bearbeiten können. Diese Daten oder/und Anwendungen können für die Zusammenarbeit in der Peergruppe verwendet werden.

2.1.4 Gemeinsamkeit der beschriebenen Szenarien

Die beschriebenen Szenarien haben Gemeinsamkeit der Floor Control und der Zusammenar-

beit. Die Option "Moderator" ist für die Floor Control. D.h. die Floor Control kann mit oder ohne Moderator stattfinden. Für die Zusammenarbeit gibt es zwei Optionen: "DataScroll" und "DataChange". Diese Optionen werden beim Anlegen von Applikation Sharing durch den Anwendungsanbieter festgelegt. Jenach Kontext kann der Anwendungsanbieter die entsprechenden Optionen auswählen. Die Optionen werden im folgenden Abschnitt erklärt.

- **Moderator**: ist jemand, der die Sitzung steuert, einem Teilnehmer das Rederecht geben und wieder entziehen kann.
  Diese Option ist für die Peergruppen mit Vorsitzenden geeignet.
- **DataScroll**: Es geht hier um die synchronisierte Anzeige bzw. um die Übertragung des Blätterns. Diese Option bedeutet, dass das **Blättern** der Person, die den Floor besitzt, an alle Teilnehmer übertragen wird. Das ist das Prinzip von WYSIWIS (What You See Is What I See). Die Form der Kopplung von Sichten wie striktes WYSIWIS oder verschiedene Abschwächungen von WYSIWIS (siehe [BURGER 97]) ist von der Anwendungen abhängig.
  Diese Option ist für Vorlesungen geeignet.
- **DataChange**: Es geht hier um die Berechtigung zum Bearbeiten der Daten bzw. um die Übertragung des Änderns der Daten. Diese Option bedeutet, dass die **Änderungen** der Daten an alle Teilnehmer übertragen werden. Die Person, die den Floor besitzt, darf die Daten ändern.
  Diese Option ist für Gruppenarbeit und für Vorlesungen geeignet.

Arbeitet ein Teilnehmer an einer Stelle, wo die Änderungen durchgeführt werden, wird der Teilnehmer die Änderungen gleich sehen. Arbeitet einer anderen an andere Stelle der Änderung, sieht er die Änderungen nicht aber die Änderungen werden bei ihm gespeichert. Wenn er an die geänderte Stelle blättert, wird er die Änderungen sehen.

Man kann die beiden Optionen für Daten auswählen. D.h. das Blättern und die Änderungen werden an alle Teilnehmer übertragen. Wie sinnvoll die Auswahl der Optionen ist, kommt auf den Kontext an.

Z.B. für die Vorlesungen sind die beiden Optionen "DataScroll" und "DataChange" geeignet. Blättert oder ändert der Professor oder ein Student die Inhalte, werden das Blättern und die Änderungen an alle Zuhörer übertragen und alle Teilhemer können deshalb das Blättern oder die Änderungen gemeinsam und gleichzeitig sehen.

Für die Seminare oder die Vorträge ist die Option "DataScroll" geeignet, da der Vortragende

oder ein Zuhörer die Folien blättert und alle andere Zuhörer können die Präsentationsfolien gleichzeitig anschauen. Änderungen sind in diesem Fall nicht notwendig.

2.2 Spezielle Anforderungen

Aus den oben beschriebenen Szenarien ergeben sich die speziellen Anforderungen.

Floor Control

Floor Control bestimmt, wer wann welche Aktionen einer Anwendung durchführen darf. Das System muss sicherstellen, dass die Rechte an der Benutzung von Anwendungen richtig gesteuert werden können.

Sitzungs- und Teilnehmerverwaltung

Das System muss die Sitzungs- und Teilnehmerverwaltung unterstützen.

Dauerhaftigkeit

Abgesehen von der Einschränkung der Stabilität von Funknetz und von Batterien bei tragbaren Geräten muss das System garantieren, dass die Interaktionen zwischen Peers einige Stunden dauern können.

Dateitransfer

Für die gemeinsame Nutzung von Anwendungen ist eine Möglichkeit zur Dateiübertragung nützlich. Diese wird benutzt, um die Daten vom Besitzer-Peer auf andere Peers zu übertragen. Z.B. für die Verteilung einer Vorlesungsmitschrift oder eines Protokolls.

Daten-Konsistenz

Das System muss gewährleisten, dass die Daten einer Anwendung nach der Bearbeitung durch Peers konsistent sind. D.h. die Daten, die nach Änderungen lokal gespeichert werden, sind alle gleich. Das System garantiert nicht, dass die Daten semantisch korrekt bearbeitet werden, sondern garantiert nur, dass auf den Daten die richtige Reihenfolge von Operationen durchgeführt wird.

Daten-Persistenz

Das System muss nach Möglichkeit sicherstellen, dass die Daten für andere Peers immer noch verfügbar sind, wenn der Besitzer-Peer aus irgendeinem Grund nicht mehr zur Verfügung steht.

Daten-Integrität

Das System muss die vollständige und korrekte Übertragung von Daten unterstützen.

2.3 Allgemeine Anforderungen

Nach den Betrachtungen der Szenarien können die allgemeinen Anforderungen wie foglt formuliert werden:

Offenheit

Durch das System muss die Erweiterung bestehender Funktionalitäten unterstützt werden. Das bedeutet auch, dass der Zugriff auf systeminterne Informationen möglich sein sollte.

Plattformen

Das System muss plattformunabhängig sein. D.h. die Software muss auf den unterschiedlichen beteiligten Plattformen verfügbar sein, wie beispielsweise auf Unix, Microsoft Windows usw. Dies umfaßt sowohl mobile Geräte (wie z.B. Laptops, PDAs usw.) als auch Desktop PCs.

Peer-Auffinden

Das Auffinden von Peers soll ohne Einschränkung der physikalischen Entfernung von Peers stattfinden. D.h. es gibt keine Beschränkung der physikalischen Lokation von Peers.

Stabilität

Die Stabilität des Systems ist für den Erfolg der gemeinsamen Nutzung von Anwendungen sowie für die unterbrechungsfreie Verbindung zwischen Peers von entscheidender Bedeutung.

Skalierbarkeit

Das System muss Anwendungen mit mehreren teilnehmenden Endgeräten unterstützen, um im Bereich der vernetzten Zusammenarbeit (wie Vorlesungen mit bis zu 20 Studenten) sinnvoll einsetzbar zu sein.

Sicherheit

Das System muss die Authentification von Benutzern unterstützen, um die Teilnahme an einer bestimmten Peergruppe einschränken zu können.

2.3.1 Lizenz-Copyright-Problem

Im Abschnitt 5.1.2 werden Fälle betrachtet, bei denen der Code der Anwendung an alle Peers übertragen wird. Es entsteht ein Problem der Lizenzvergabe im P2P-Framework. Die normalen Methoden für die Lizenzvergabe wie Bezahlung beim Kauf oder Lizenz-Server können im P2P-Framework nicht eingesetzt werden. Entsprechend ist die Problematik bei Copyright geschützten Daten. Deshalb wird eine Lösung für das Lizenz- und Copyright-Problem benötigt. Diese Probleme werden im Rahmen dieser Arbeit jedoch nicht weiter betrachtet.

3 Peer-to-Peer-Computing

Im letzten Kapitel wurde auf Anforderungen eingegangen, die teilweise auf den Eigenschaften von mobilen, ad hoc Peer-to-Peer-Netzwerken basierten. In diesem Kapitel wird deshalb das Peer-to-Peer-Computing kurz vorgestellt und das Peer-to-Peer-Modell charakterisiert.

3.1 Was ist Peer-to-Peer-Computing?

3.1.1 Definition von Peer-to-Peer-Computing

Das Peer-to-Peer-Computing ist die gemeinsame Nutzung von Computerressourcen und Dienstleitungen durch einen direkten Austausch unter den Systemen. Zu diesen Ressourcen und Dienstleitungen gehört der Austausch von Informationen, Verarbeitungszyklen, Cachespeicherung und Plattenspeicherung von Dateien. Das Peer-to-Peer-Computing nutzt die Vorteile der vorhandenen Desktop-Computerleistung und der Netzwerkkonnektivität, so dass preiswerte Clientcomputer ihre kollektive Stärke zum Nutzen des gesamten Unternehmens einbringen (siehe [INTEL]).

3.1.2 Warum Peer-to-Peer?

Peer-to-Peer-Computing bringt Vorteile z.B.:

- **Verfügbarkeit**: Ein großer Vorteil des Peer-to-Peer-Computing ist die Verfügbarkeit von Informationen für Benutzer, die mit mobilen, tragbaren Geräten wie Laptops oder PDAs unterwegs arbeiten. D.h. wenn Server nicht zur Verfügung stehen, kann der Benutzer durch die dynamischen Peer-to-Peer-Verbindungen die anderen Benutzer nach Informationen fragen.
- **Flexibilität**: Es geht hier um die Schnelligkeit und die Flexibilität für Benutzer. D.h. der Benutzer ist unabhängig von Administratoren und anderen Entwicklern. Er kann jeder Zeit die Funktionalität einer Anwendung ändern oder erweitern. Wenn z.B. bei einem Applikation-Server die Mitarbeiter/Benutzer die Funktionalität der Anwendung erweitern wollen, muss ein Entwicklungsteam zuerst eine Erweiterung für den zentralen Server entwickeln. Danach muss der Administrator die Erweiterungssoftware auf

dem Server installieren. Dies führt zu einer Verzögerung zwischen Anforderung und Erfüllung. In Peer-to-Peer-Computing Umgebung kann diese Funktionalität direkt ins Netzwerk hinzugefügt werden. D.h. der Benutzer kann Funktionen lokal auf dem eigenen Rechner implementieren und erweitern und den anderen Benutzern zur Verfügung stellen.

Die Client-Server-Architektur ist bekannt (siehe Lehrnbücher z.B. Moderne Betriebssystemme von Tannenbaum, usw.) und deshalb wird hier nicht erklärt. Hingegen wird die Peer-to-Peer-Architektur für diese Arbeit genauer betrachtet.

3.2 Peer-to-Peer-Architektur

Die Peer-to-Peer-Architektur hat folgende Eigenschaften bzw. Charakteristiken:

- **Dezentralisierte Architektur**: Die Peer-to-Peer-Architektur ist eine dezentralisierte Architektur, da alle Peers autonom und gleichberechtigt miteinander interagieren. Jeder Peer kann Dienste anbieten und anfragen.
- **Symmetrische Beziehung**: Die Peer-to-Peer-Beziehung ist symmetrisch, da alle Peers gleichzeitig Dienste den anderen Peers anbieten **und** Dienste anfragen können.
- **Existenz des Netzwerks**: Ein Peer-to-Peer-Netzwerk existiert, wenn mindestens ein Peer aktiv ist. Das Netzwerk ist nur dann nicht mehr verfügbar, wenn **alle** Peers nicht aktiv sind.
- **Kommunikation/Netzwerkkapazität**: Im Peer-to-Peer-Modell ist die Kommunikation eine m:n-Kommunikation. Die Netzwerkkapazität steigt in der Peer-to-Peer-Architektur mit der Menge von Peers. D.h. im Netzwerk ist jeder Peer ein aktiver Teilnehmer, der zum Netzwerk bestimmte Ressourcen beiträgt. Je mehr Peers hinzukommen, desto höher ist die Netzwerkkapazität. Dadurch erhöht sich die Stabilität und die Zuverlässigkeit des Netzwerks. Voraussetzung ist jedoch eine hohe Redundanz und Verteilung der Daten auf viele Peers. Dadurch kann es jedoch zu hohen Speicherplatzanforderungen auf den einzelnen Peers kommen. Außerdem wird die Netzwerkkapazität für das Verteilen der Daten benötigt.
- **Netzwerk-Management**: Im Vergleich zum Client-Server-Modell ist die Verwaltung des Peer-to-Peer-Netzwerks schwierig, da die Administrationsaufgaben (wie Konfigu-

ration, Backup usw.) bei jedem Peer durchgeführt werden müssen.

- **Sicherheit**: Das Peer-to-Peer-Konzept baut darauf auf, dass Benutzer ihren Computer für fremde Anwender öffnen. Dies kann zu Problemen wegen Viren oder Mißbrauch führen. Es ist erheblich schwieriger und aufwendiger, ein Sicherheitssystem (bzgl. physikalischer und Software-Sicherheit) für jeden Peer einzurichten.
- **Verteilte Informationen**: Verteilte Informationen in Peer-to-Peer-Architektur bringen Vorteile. Erstens können Benutzer andere Benutzer nach Informationen fragen. Zweitens sind verteilte Informationen redundant. Das ist auch nützlich, da die Informationen immer noch auf Rechnern von anderen Benutzern verfügbar sind, wenn ein Benutzer aus irgendeinem Grund nicht mehr zur Verfügung steht. Trotz Nützlichkeit haben verteilte Informationen den Nachteil, dass die Übersicht fehlt. Z.B. ein Projektleiter will den aktuellen Zustand des Projekts wissen. Er muss sämliche Rechner des Teams durchschauen, damit er die Übersicht über die aktuellen Daten hat. Ist ein Mitarbeiter nicht anwesend, sind die Daten auch nicht notwendigerweise vollständig.

3.3 Fazit

Jede Architektur hat Vorteile und Nachteile. Generell ist die Client/Server-Architektur effektiv und einfacher als die Peer-to-Peer-Architektur. P2P-Architektur ist ein dezentrales System. Jeder Peer muss Aufgaben eines Servers übernehmen, selbst organisieren und selbständig arbeiten. Von daher ist die Peer-to-Peer-Architektur aufwendiger als die Client/Server-Architektur.

Die Peer-to-Peer-Architektur ist jedoch für bestimmte Anwendungen geeignet, besonders bei Echtzeit-Anwendungen oder in Situationen, in denen kein zentraler Server verfügbar ist und ad hoc Treffen, z.B. Arbeiten mit Laptops oder PDAs unterwegs.

4 Eignungsanalyse von existierenden Systemen

In diesem Kapitel werden drei Systeme (SASCIA, PROEM, JXTA) anhand der Kriterien der Anforderungsanalyse (siehe Kapitel 2) untersucht und analysiert, um festzustellen, welches System oder welche Komponenten für die Aufgaben dieser Diplomarbeit geeignet sind und evtl. übernommen oder erweitert werden können.

SASCIA ist bisher ein client-server-basierendes Framework, durch das Anwendungen für Lehrveranstaltungen gemeinsam genutzt werden können.

PROEM ist eine Peer-to-Peer-Plattform, die Anwendungsentwicklern eine Plattform für Peer-to-Peer-Anwendungen im Bereich der Zusammenarbeit in mobilen ad hoc Netzwerken bietet.

JXTA ist eine Peer-to-Peer-Plattform, die einfache und flexible Mechanismen für die Entwicklung von Peer-to-Peer-Anwendungen zur Verfügung stellt.

4.1 Wiederverwendung von SASCIA Komponenten

Die Gesamtarchitektur von SASCIA sowie die wichtigen Komponenten werden vorgestellt. Die Ausführungen basieren dabei auf dem Paper "Application sharing in teaching context with wireless networks" [BURGER 01] sowie den Studienarbeitenen "Vergabe von Zugangsberechtigungen (Floor Control) für die gemeinsame Nutzung von SMARTboards in der Vorlesung" [OBER 01], "Prototypische Erstellung einer Protokollierung für den Einsatz in Lehrveranstaltungen" [SCHMID 01], "Prototypische Erstellung einer Sitzungsverwaltung für den Einsatz in Lehrveranstaltungen" [SOMMER 01] und der Diplomarbeit "Eine Architektur zur gemeinsamen Nutzung von Anwendungen in der Lehre mit prototypischer Realisierung von Tafel und Leinwand" [OBER 02].

4.1.1 Beschreibung

Das Projekt SASCIA ist ein Teilprojekt des Projekts FESTIVAL in der Abteilung Verteilte Sy-

steme der Universität Stuttgart. SASCIA steht für **S**ystem **A**rchitecture **S**upporting **C**ooperative and **I**nteractive **A**pplications. Das Ziel des Projekts SASCIA ist die Entwicklung eines Frameworks, auf dem Anwendungen (z.B. Whiteboard, Chat usw.) für die Lehrveranstaltungen aufgebaut werden. Das System ermöglicht es, Anwendungen gemeinsam zu nutzen, Lehrinhalte zu präsentieren und zu kommentieren, sowie die Lehrinhalte mit den Teilnehmern gemeinsam zu erarbeiten und zu diskutieren. Das System besteht aus mehreren Komponenten (Abb. 4.1). Die wichtigen Komponenten sind:

- Kommunikationssystem
- Session Management und Session Directory
- Floor Control
- Application frame
- Protokollierungsdatenbanken

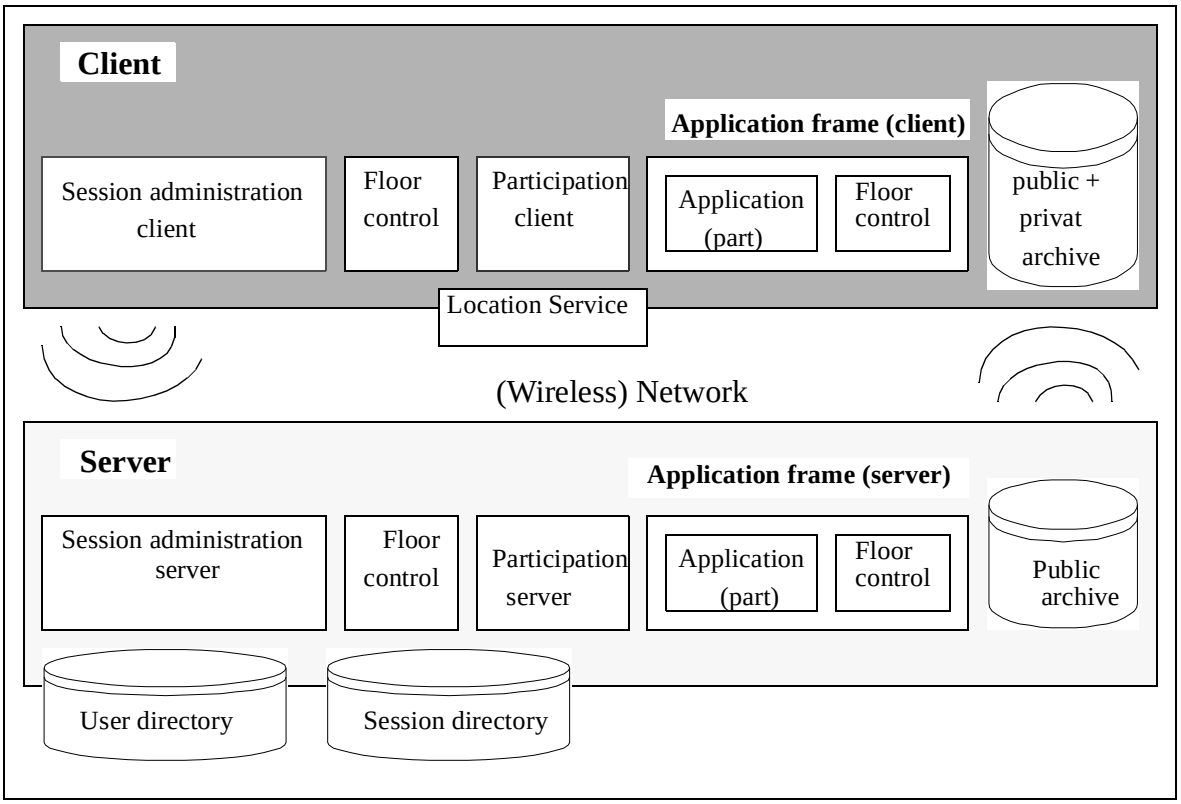

Abbildung 4.1: SASCIA-Architektur

Das **Kommunikationssystem** ermöglicht es, dass die Teilnehmer miteinander kommunizieren sowie Daten austauschen können. Es bietet den anderen Komponenten eine Schnittstelle, über

die sie eine Verbindung zu den Teilnehmern starten und beenden können. Auf Serverseite wird ein Kanal beim Verbindungsaufbau geöffnet und auf Clientseite wird der Kanal betreten, über den der Nachrichtenaustausch stattfindet. Der Nachrichtenversand erfolgt dann entweder an alle Teilnehmer außer an sich selbst oder an alle inklusive sich selbst oder nur an spezielle Teilnehmer. Die empfangenen Nachrichten werden an einen Konsumer weitergegeben, der die Nachrichten dann verarbeitet. Das Kommunikationssystem kapselt das darunter liegende System, das zur Zeit JSDT (Java Share Data Toolkit) (siehe [JSDT]) ist. JSDT kann so jederzeit durch ein anderes System ersetzt werden (siehe [OBER 02]).

Das **Session Directory** bietet eine Übersicht über die verschiedenen Veranstaltungen. Es dient als Anlaufstelle für die Teilnehmer, um stattfindende Veranstaltungen aufzufinden. Es bietet dazu eine Liste der Sitzungen mit entsprechenden Zusatzinformationen an, beispielsweise die Verbindungsdaten zum Präsentationssystem bzw. Sitzungsserver einer Veranstaltung oder inhaltliche Hinweise zu einer Veranstaltung. Anhand dieser Zusatzinformationen kann sich der Teilnehmer die Veranstaltung aussuchen, an der er teilnehmen möchte und sich an der Sitzung anmelden. Die Liste der stattfindenden Sitzungen wird aktualisiert, wenn sich Sitzungsserver an- bzw. abmelden.

Das **Session Management** (siehe [SOMMER 01]) ist für den Start und das Beenden registrierter Anwendungen und erforderlicher Komponenten verantwortlich. Es bildet die Verwaltung einer Sitzung und der verfügbaren Anwendungen.

Mit der **Floor Control** wird das Rede- bzw. Schreibrecht überwacht. Durch die Floor Control wird kontrolliert, wer von den Teilnehmern im Moment aktiv sein kann und wer passiv bleiben muss. Wenn jeder Teilnehmer unkontrolliert Aktionen durchführt bzw. Annotationen zu Lehrinhalten macht, würde niemand mehr mitbekommen, von wem eine Anmerkung gemacht wurde und in welchem Zusammenhang. Mit der Floor Control wird der Ablauf kontrollierbar. Das Rederecht muss von einem Teilnehmer beantragt werden, bevor er einen Dienst bzw. eine Anwendung nutzen kann, um Anmerkungen zu machen oder Aktionen durchzuführen.

Dabei gibt es drei unterschiedliche Szenarien, auf welche Weise über die Vergabe des Rederechts entschieden wird (siehe [OBER 01]):

1. *Zentrale Vergabe*: Wenn die Sitzung durch einen Vorsitzenden (z.B. ein/e Professor/in oder ein/e Dozent/in) geleitet wird, hat er die alleinige Kontrolle über die Vergabe des Rederechts. Er entscheidet, ob ein Teilnehmer, der das Rederecht beantragt, dieses erhält oder nicht. Er kann einem Teilnehmer das Rederecht auch wieder entziehen.

Beispiel: Vorlesung mit Professor und Studenten.

2. *Dezentrale Vergabe*: Wenn es keinen expliziten Vorsitzenden gibt, sondern alle Teilnehmer gleiche Priorität besitzen, erfolgt die Vergabe des Rederechts nach dem "first come, first serverd"-Prinzip. Der Teilnehmer, der das Rederecht bekommen hat, behält es solange, bis er es wieder freigibt. Dann kann sich der nächste Teilnehmer das Rederecht nehmen.
   Beispiel: Gruppenübung mit Studenten.
3. *Delegierte Vergabe*: Falls kein Teilnehmer mit höherer Priorität anwesend ist oder wenn der Person mit höherer Priorität nicht moderieren will, kann dennoch eine zentrale Vergabe-Strategie erreicht werden. Dazu stimmen die Teilnehmer ab, wer den Vorsitz übernehmen soll. Jeder Teilnehmer hat dafür eine Stimme mit einem bestimmten Gewicht. Das Gewicht der Stimme hängt von der Priorität des Teilnehmers ab. Die Stimme eines Teilnehmers mit höherer Priorität als alle anderen Teilnehmer zählt dabei mehr als alle Stimmen der Teilnehmer mit niedriger Priorität. Nach der Abstimmung entscheidet der Vorsitzende über die Rederechtvergabe.
   Beispiel: Gruppenübung mit Studenten und einem studentischen Übungsleiter.

Für jeden Dienst bzw. für jede Anwendung gibt es eine eigene Floor Control. Innerhalb einer Anwendung kann man u.U. (abhängig von der Anwendung) noch eine feinere Rechte-Vergabe einführen durch sogenannte Sub-Floors. D.h. wenn eine Gruppe von Teilnehmern das Recht bekommen hat, eine Anwendung zu benutzen, kann durch Vergabe der Sub-Floors innerhalb der Gruppe entschieden werden, wer welche Aktionen der Anwendung durchführen darf (siehe [BAI 01]).

Die **Protokollierungsdatenbanken** (siehe [SCHMID 01]) zeichnen die mit den Anwendungen erstellten Daten auf. Die Datenbank ist aufgeteilt jeweils in einen privaten und einen öffentlichen Bereich. Im öffentlichen Bereich werden alle Daten gespeichert, die über das Kommunikationssystem beim einzelnen Teilnehmer ankommen. Diese Daten sind beispielsweise Daten der elektronischen Präsentationsfolien, Annotationen des Dozenten, Annotationen anderer Teilnehmer, sofern diese ein Rederecht haben und Java Applets oder Java Applikationen. Hingegen enthält der private Bereich nur Daten, die vom Teilnehmer selbst erzeugt wurden und nicht für alle anderen Teilnehmer sind. Durch eine permanente Abspeicherung der Daten inklusive Reihenfolgeindex in einer Datenbank beim einzelnen Teilnehmer wird die Wiedergabe auch nach der Lehrveranstaltung möglich. Selbst eine Wiedergabe in Echtzeit wird durch die Datenbank unterstützt. Zusätzlich zur Abspeicherung der zur Zeit übermittelten Daten ermöglicht die Da-

tenbank auch die Abfrage bereits übermittelter Daten, um diese auch noch abspeichern zu können. Dies ist vor allem dann von Nutzen, wenn ein Teilnehmer eine Vorlesung bzw. einen Vortrag nicht von Anfang an mitverfolgen konnte oder wenn Daten von der Datenbank nicht korrekt empfangen wurden.

Überprüfung der Wiederverwendung von SASCIA-Komponenten:

SASCIA verwendet JSDT (siehe [JSDT]) für die Kommunikation. JSDT ist ein Client/Server-Architektur. Jeder Verbindungsaufbau in JSDT läuft über den Server. Im Peer-to-Peer-Framework existiert kein Server, deshalb kann JSDT für die Kommunikation im P2P-Framework nicht verwendet werden.

Die **Sitzungs- und Teilnehmerverwaltung** sowie die **Floor Control** Komponenten von SASCIA sind client-server konzipiert. Der zentrale Server verwaltet alle Informationen über Floor Control sowie über Sitzungen und Teilnehmer. Im P2P-System werden die Sitzungs- und Teilnehmerverwaltung und die Floor Control verteilt konzipiert. Es existiert kein zentraler Server. Von daher können die Sitzungs- und Teilnehmerverwaltung und die Floor Control von SASCIA nicht direkt übernommen werden.

Die **Protokollierungsdatenbank** von SASCIA kann wiederverwendet werden. Es ist sinnvoll, dass eine Anwendung eine Aufzeichnungsdatenbank für die Wiedergabe einer Veranstaltung hat. Wenn die Datenbank von SASCIA für das P2P-Framework wiederverwendet wird, muss sie an das P2P-Framework angepaßt sowie einige Komponenten neugeschrieben werden.

Im Bild 4.2 kann man sehr klar sehen, dass die Datenbank (siehe [SCHMID 01]) zwei Teile hat: Serverseite und Clientseite. Es gibt im P2P-Framework keinen zentralen Datenbankserver. Die Daten werden bei jedem Peer lokal gespeichert. Deshalb können die Module bei der Serverseite nicht übernommen werden. Die Module bei der Clientseite werden für die Datenbank des P2P-Frameworks übernommen.

Ein Vorschlag für die Aufzeichnungsdatenbank für das P2P-Framework wird beim Entwurf im Kapitel 5 genauer beschrieben.

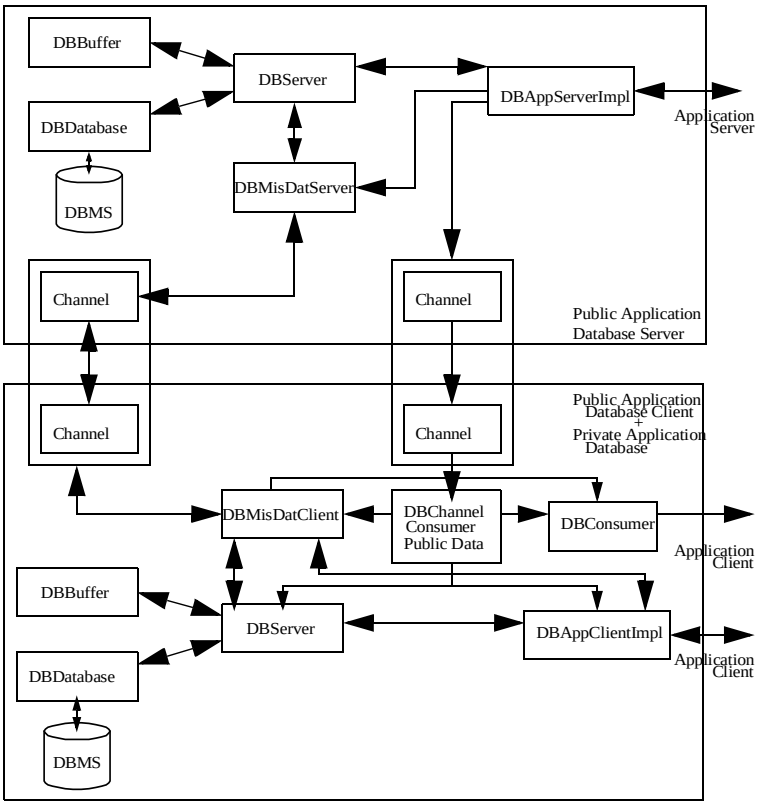

Abbildung 4.2: Architektur der Protokollierungsdatenbanken von SASCIA

4.1.2 Bewertung

4.1.2.1 Spezielle Anforderungen

Vollständig erfüllt:

Dauerhaftigkeit

Die Dauer von Interaktionen zwischen Benutzern ist nicht beschränkt (üblicherweise mehrere Studen). Es ergibt sich die Dauerhaftigkeit des Systems.

Dateitransfer

Die Datenübertragung ist durch einen integrierten Dateitransfer möglich.

Daten-Konsistenz

Die öffentlichen Daten sind konsistent.

Daten-Persistenz

Die Daten sind auf dem Server und zusätzlich auf jedem Client gespeichert und für Teilnehmer verfügbar.

Daten-Integrität

Bei den bereits durchgeführten Tests wurden die Daten vollständig und korrekt übertragen.

Teilweise erfüllt:

Floor Control

Eine Floor Control wurde vollständig implementiert. D.h. die Rechtevergabe pro Benutzer/Anwendung sowie das Sperren der Eingabe ist möglich. Die Zuordnung von Rechten zu Teilnehmern ist dynamisch möglich. Aber die Floor Control ist client-server konzipiert, sie ist nur teilweise geeignet für das P2P-Framework.

Sitzungs- und Teilnehmerverwaltung

Weil die Sitzungs- und Teilnehmerverwaltung client-server konzipiert ist, ist sie nur teilweise geeignet für das P2P-Framework.

4.1.2.2 Allgemeine Anforderungen

Vollständig erfüllt:

Offenheit

Eine Integration von neuen Tools bzw. die Erweiterung der Funktionalität ist möglich, da die internen Schnittstellen der Software zugänglich sind.

Plattformen

Das System SASCIA ist für verschiedene Betriebssysteme beispielsweise Unix, Micrsoft Windows, sowie auf unterschiedlichen Rechnertypen wie Desktops und Laptops verfügbar.

Stabilität

Das System SASCIA ist in der Praxis eingesetzt. Es ergibt eine gute Stabilität des Systems.

Skalierbarkeit

Das SASCIA System wird in Vorlesungen mit bis zu 20 Studenten verwendet. Das ist

akzeptabel für die Verwendung auf Laptops über Funk-LANs.

Sicherheit

Die Authentifizierung erfolgt durch Benutzername und Passwort. Für ein prototypisches System bietet dieses Vorgehen eine ausreichende Sicherheit.

Peer-Auffinden

Diese Anforderung ist erfüllt, da alle Clients vom Server an Clients gemeldet werden.

4.1.3 Zusammenfassung

Wie oben erklärt können die Komponenten wie die Floor Control und die Sitzungs- und Teilnehmerverwaltung für die Erstellung einer dezentralen Peer-to-Peer-Architektur für Application Sharing nicht vollständig übernommen werden. Durch die unterschiedlichen Architekturen ergeben sich unterschiedliche Protokolle und Implementierungen.

4.2 Eignungsanalyse PROEM

4.2.1 Beschreibung

PROEM ist ein Projekt der Universität Oregon in den USA [KSPTFS 01]. Das Projekt liegt im Bereich des Collaborative Peer-to-Peer-Computings in mobilen ad hoc Netzwerken. In einem mobilen ad hoc Netzwerk wird die Verbindung durch die Distanz zwischen Peers bestimmt. Wenn Peers ihre physikalische Lokation ändern, erzeugen sie paarweise Kommunikationsverbindungen, die auf der "mutual proximity" basieren. PROEM ist ein Peer-to-Peer-System sowie eine Entwicklungsplattform für mobile ad hoc Anwendungen.

Ziele von PROEM

- **Allgemeinheit**: PROEM ist eine Infrastruktur für den Aufbau von verschiedenen mobilen Gemeinschaften, z.B. von der gemeinsamen Nutzung von MP3-Musikdateien bis zu Nachrichtenaustausch. PROEM stellt für umfangreiche Anwendungen die Peer-to-Peer-Plattform zur Verfügung.
- **Interoperabilität**: PROEM ermöglicht es, dass diverse Hardware- und Software-Plattformen zusammenarbeiten können.

- **Plattformunabhängigkeit**: PROEM ist unabhängig von Programmiersprachen (z.B. C, C++ oder Java usw.), von Systemplattformen (wie Unix, Microsoft Windows, usw.) und von Netzwerkplattformen (wie TCP/IP, usw.). Um dieses Ziel zu erreichen, wurden offene Web-Standards und Technologie wie http, xml, mime usw. verwendet.
- **High-level Entwicklungsunterstützung**: Ein wichtiges Ziel von PROEM ist die Entwicklung einer einfachen aber mächtigen Plattform, die die Implementierung von mobilen Peer-to-Peer-Anwendungen erleichtern kann.

Fokus von PROEM

- PROEM bietet eine **generelle** Plattform für das Aufbauen von beliebigen mobilen Peer-to-Peer Anwendungen an.
- PROEM wurde für **völlig dynamische ad hoc** Netzwerke entworfen, bei denen Verbindungen und Verfügbarkeit von Resourcen konstant ändern.
- Der Schwerpunkt von PROEM ist die **Kooperationsunterstützung**.

4.2.1.1 Das PROEM Konzept

Protokolle und Nachrichten

PROEM definiert vier Protokolle: ein low-level Transport-Protokoll und drei higher-level Protokolle.

Das **Transport-Protokoll** ist ein verbindungsloses und asynchrones Kommunikation-Protokoll. Daten werden von einem Peer an einen anderen Peer in einer atomartigen Einheit gesendet. Dieses Protokoll verwendet XML für die Darstellung von Nachrichten und wird durch existierende Protokolle wie TCP/IP, UDP oder HTTP implementiert. Wenn ein unzuverlässiges Transport-Protokoll verwendet wird, können Nachrichten mehrfach oder in falscher Reihenfolge ankommen oder sogar verlorengehen. Der Empfang einer Nachricht wird nicht bestätigt, außer wenn es im Protokoll explizit definiert wird.

Das **Presence-Protokoll** enthält Nachrichten, die die Ankündigung der Anwesenheit von Peers und die Verfügbarkeit von Elementen in Netzwerken unterstützen. Der primitive Nachrichttyp des Presence-Protokolls ist Profile.

Das **Data-Protokoll** enthält Nachrichten, die Peers die gemeinsame Nutzung und die Synchronisation von Daten hinsichtlich Data Spaces erlauben.

Das **Community-Protokoll** enthält Nachrichten, die für das Beantragen, die Garantie und die Kontrolle der Community-Mitgliedschaft zuständig sind.

Nachrichten sind die fundamentale Einheit der Kommunikation zwischen Peers. Nachrichten werden als XML-Dokumente verschlüsselt und von einem Peer an einen anderen Peer gesendet. Dies erlaubt, dass PROEM Peers in verschiedenen Programmiersprachen implementiert werden können und dafür keine spezielle Transport-Protokolle erforderlich sind.

4.2.1.2 Die PROEM Architektur

Peerlets und Peerlet Engine

Die PROEM Anwendungsumgebung beruht auf der Idee von Peerlets (siehe Abb. 4.3). **Peerlets** sind einfache strukturierte Peer-to-Peer-Anwendungen, die ein event-basierendes Programmierungsmodell erlauben. Peerlets werden in dem Peerlet Engine gehosted.

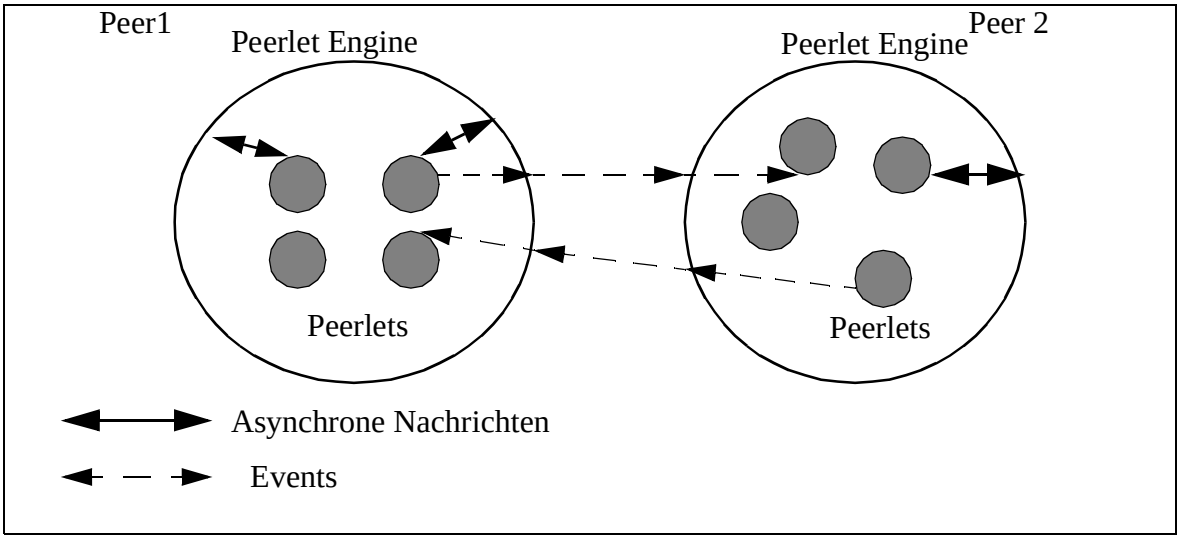

Abbildung 4.3: Die Beziehung zwischen Peers, Peerlets und Peerlet Engine

Peerlet Engine ist für den Start, die Ausfürhung und das Beenden von Peerlets zuständig. Peerlets wurden als drop-in Modulen entworfen und deshalb können zur Laufzeit in den Peerlet Engine eingefügt und vom Peerlet Engine entfernt werden.

Peerlets reagieren auf Ereignisse und kommunizieren über Ereignisse miteinander. Die Peerlet Engine sendet Ereignisse an Peerlets als eine Reaktion auf die Änderung ihres internen Zustandes oder als eine Reaktion auf den Empfang von enfernten Nachrichten. Peerlets werden über Ereignisse asynchron benachrichtigt und behandeln sie auch asynchron.

4.2.2 Bewertung

4.2.2.1 Spezielle Anforderungen

Vollständig erfüllt:

Sitzungs- und Teilnehmerverwaltung

Die PROEM Plattform ist ein dezentralisiertes System. Ein Mechanismus wurde entworfen, um ständig festzustellen, welcher Peer neu ins Netzwerk hinzugefügt wird oder vom Netzwerk weggeht.

Dateitransfer

Die Datenübertragung ist durch das Transport Protokoll möglich.

Daten-Konsistenz

Durch die Speicherung von "shared data" Objekten bei Peers entsteht ein Problem der Inkonsistenz von Daten, da die Objekte unabhängig und lokal geändert werden. Deshalb wurden dafür Synchronisationsmechanismen entworfen.

Daten-Persistenz

Daten sind persistent, da die Data Spaces bei allen Peers in einer Gruppe gespeichert werden. D.h. jeder Peer hat eine lokale Kopie von jedem "shared data" Objekt.

Daten-Integrität

Daten wurden vollständig und korrekt übertragen.

Nicht erfüllt:

Floor Control

Diese Anforderung ist nicht erfüllt, da PROEM nur eine Infrastruktur bzw. eine generelle Plattform für die Entwicklung von Peer-to-Peer-Anwendungen bietet. Floor Control ist jedoch anwendungsspezifisch.

Dauerhaftigkeit

Die Interaktionen zwischen Peers in PROEM dauert nur einige Sekunden bis einige Minuten. Deshalb ist diese Anforderung nicht erfüllt.

4.2.2.2 Allgemeine Anforderungen

Vollständig erfüllt:

Plattformen

Die PROEM Plattform ist unabhängig von Betriebsystemen und Netzwerkplattformen.

Stabilität

PROEM wurde in verschiedene Szenarien (siehe [KSPTFS 01]) erfolgreich verwendet.

Teilweise erfüllt:

Skalierbarkeit

PROEM wurde in MP3 File Sharing Szenarien und im Software Engineering Lehrgang der Universität Oregon verwendet. D.h. PROEM wurde in Systemen mit zwei bis zu mehreren Peers eingesetzt. Das ist akzeptabel für die Face-to-Face-Interaktionen. Es ist nicht sicher, ob PROEM diese Anforderung für Vorlesungen mit mehr als 20 Studenten erfüllt.

Peer-Auffinden

Durch eine nähe-basierende Verbindung geschieht das Auffinden eines Peers. D.h das Auffinden von Peers ist durch die Entfernung zwischen Peers beschränkt. Aus dem Paper geht nicht hervor, dass das Konzept von PROEM oder die Abhängigkeit vom darunterliegenden Transportsystem diese Beschränkung verursacht.

Nicht erfüllt:

Offenheit

Die Software ist keine Open-Source. Deshalb kann die Erweiterung der Funktionalität des Systems nicht gemacht werden. D.h diese Anforderung ist nicht erfüllt.

Sicherheit

Es ist schwierig, effiziente verteilte Authentifikation-Protokolle für mobile ad hoc Netzwerke zu entwerfen. PROEM hat noch keine richtige Authentifikation von Benutzern.

4.2.3 Zusammenfassung

PROEM erfüllt nur teilweise die allgemeinen Anforderungen wie Peer-Auffinden, Skalierbarkeit. PROEM erfüllt vollständig die speziellen Anforderungen wie Sitzungs- und Teilnehmerverwaltung, Datenkonsistenz, Datenpersistenz, Datenintegrität und Dateitransfer. Nicht erfüllt sind die Regelung der Rechtevergabe, die Dauerhaftigkeit, Offenheit und Sicherheit.

4.3 Eignungsanalyse JXTA

4.3.1 Beschreibung

JXTA ist eine open-source Peer-to-Peer-Plattform von Sun Microsystems [JXTA]. JXTA bietet eine generelle Plattform für Peer-to-Peer-Anwendungsentwicklung.

Ziele von JXTA

- *Interoperabilität*: JXTA Technologie ermöglicht es, dass Peers einander auffinden und kommunizieren können, dass Peers an beliebigen Peergruppen teilnehmen können und dass Peers Dienste für andere Peers über verschiedene Peer-to-Peer-Netzwerken anbieten können.
- *Plattformunabhängigkeit*: JXTA ist unabhängig von Programmiersprachen (z.B. C, C++ oder Java usw.), von Systemplattformen (wie Unix, Microsoft Windows, usw.) und von Netzwerkplattformen (wie TCP/IP, usw.).
- *Allgegenwart (Ubiquity)*: JXTA wurde für Implementierung auf jedem digitalen Gerät entworfen. Z.B. Sensoren, Unterhaltungselektronik, PDAs, Vorrichtungen oder Elektrogeräte, Netzwerk-Routers, Desktop Computers, Laptops, Data-Center-Server und Speicherungssysteme.

Mit JXTA soll jeder Peer unabhängig von Software- und Hardware-Plattform sein, sich mit anderen Peers verbinden und ihre Dienste nutzen können.

4.3.1.1 Das JXTA Konzept

Einige JXTA-Terminologien[KRISHNAN 01]

Peers

Peers sind vernetzte Geräte (z.B. Sensoren, Handys, Laptops, PDAs, Desktops, Servers, Supercomputers, usw.), die ein oder mehrere JXTA Protokolle implementieren. Jeder Peer arbeitet

asynchron und unabhängig voneinander. Ein Peer kann Dienste für andere Peers zu Verfügung stellen bzw. Dienste von anderen Peers benutzen. Jeder Peer hat eine eindeutige Peer-ID. Ein Peer kann einen oder mehrere Vermittler-Peers benutzen, um Nachrichten an andere Peers weiterzuleiten.

Peer groups

Eine Peer-Gruppe ist eine Gruppe von Peers, die gemeinsame Interessen haben. Jede Peer-Gruppe hat eine eindeutige Peer-Group-ID. Eine Peer-Gruppe stellt für Peer-Mitglieder eine Menge von Diensten "peergroup Services" zur Verfürgung. Wenn ein Peer in eine Peer-Gruppe hinzukommen möchte, muss er zuerst ein Mitglied finden und dann kann er an der Gruppe teilnehmen. Ein Peer kann zu einer oder mehreren Peer-Gruppen gehören.

Messages

Die Kommunikation im JXTA-Netzwerk wird durch Senden und Empfang von Nachrichten (Messages) erreicht. Diese Nachrichten werden JXTA-Messages genannt, die sich an ein Standard-Format halten, dies unterstützt die JXTA-Interoperabilität. Nachrichten definieren eine XML-Hülle für die Übertragung von unterschiedlichen Datentypen. JXTA-Protokolle werden als eine Menge von XML-Nachrichten, die zwischen Peers ausgetauscht werden, spezifiziert.

Pipes

Pipes erzeugen virtuelle Kommunikationskanäle in der JXTA-Umgebung. Peers benutzen sie für das Senden und den Empfang von JXTA-Nachrichten. Pipes sind asynchron und unidirektional.

Advertisements

Um JXTA-Ressourcen (Peers, Peer-Gruppen, Pipes, Dienste, usw.) zu beschreiben, zu veröffenlichen und aufzufinden, werden Ankündigungen (Advertisements) verwendet. Ankündigungen werden als XML-Dokumente dargestellt. Es gibt verschiedenen Ankündigungen z.B. Peergroup-, Peer-, Pipe-Ankündigung usw.

JXTA Protokolle

JXTA hat sechs Protokolle. Sie sind:

- ***Peer Discovery Protocol (PDP)***: Peers können dieses Protokoll für das Auffinden von Advertisements z.B. Peer-, Peergruppen-, Pipe-Advertisement benutzen. Dadurch kann

ein Peer Peergruppen oder anderen Peers finden.

- ***Peer Resolver Protocol (PRP)***: Das PRP Protokoll ist ein Mechanismus, durch den ein Peer generelle Anfragen nach Diensten an andere Peers senden und Antworten empfangen kann.
- ***Peer Information Protocol (PIP)***: Mit diesem Protokoll kann sich ein Peer durch das Senden einer Ping-Nachricht über Status-Informationen (z.B. Zustand, Kapazität, usw.) anderer Peers informieren. Wenn ein Peer eine solche Nachricht empfängt, hat er mehrere Antwort-Möglichkeiten. Er kann die Nachricht ignorieren oder eine einfache Bestätigung oder eine vollständige Antwort inklusiv seiner Ankündigung senden.
- ***Peer Membership Protocol (PMP)***: Dieses Protokoll ermöglicht es, dass Peers an einer Peer-Gruppe teilnehmen oder die Gruppe verlassen.
- ***Pipe Binding Protocol (PBP)***: Das PBP Protokoll ist ein Mechnismus, durch den sich ein Peer einen virtuellen Kommunikation-Kanal (Pipe) zwischen einem oder mehreren Peers etablieren kann. Ein Peer benutzt dieses Protokoll dafür, dass er sich mit zwei oder mehreren Pipe-Endpoints (Input- und Output-Pipes) verbinden oder ein neues Pipe erzeugen oder von einer Pipes lösen kann.
- ***Endpoint Routing Protocol (ERP)***: Das ERP-Protokoll definiert eine Menge von Anfragenachrichten, um einem Peer zu ermöglichen, dass der Peer durch Peer-Routers seine Nachrichten ans Ziel weiterleiten kann. Dieses Protokoll ermöglicht wiederum den Peer-Routers, dass sie andere Peer-Router nach verfügbaren Routern für das Senden der Nachrichten fragen können.

Mechanismen zum Auffinden(Discovery Mechanisms)

In JXTA Version 1.0 stehen die folgenden Mechanismen zum Auffinden zur Verfügung:

- **LAN-basiertes Auffinden**: Das Auffinden geschieht über Brodcast im LAN.
- **Auffinden durch Einladung**: Wenn ein Peer eine Einladung, die Informationen über einen Peer enthält, bekommt, kann er den Peer darüber finden.
- **Auffinden durch Vermittler**: Wenn ein Peer einen anderen Peer gefunden hat, kann er sich durch den Peer nach neuen anderen Peers, Gruppen oder Diensten erkundigen und sie dadurch auffinden.
- **Auffinden via Rendezvous-Peer**: Durch einen Rendezvous-Peer kann ein Peer die

Anwesenheit von anderen Peers erfahren und sie auffinden.

4.3.1.2 Die JXTA Architektur

Die JXTA Architektur enthält drei Schichten (Abb. 4.4): Plattform oder JXTA Core-Schicht, Services- und Applikation-Schicht.

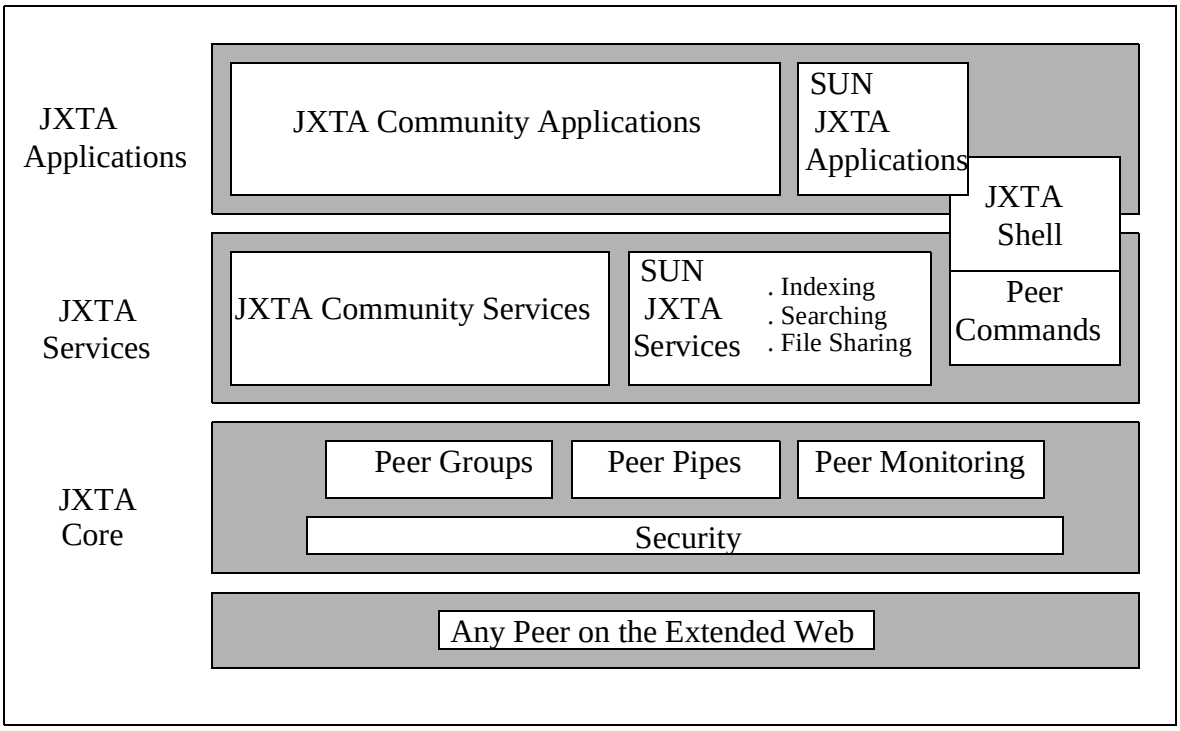

Abbildung 4.4: JXTA Architektur [in JXTA Technology Overview]

Plattform/JXTA Core

Diese Schicht befasst sich mit Kommunikationsverwaltung wie Routing bzw. mit wichtigen Primitiven für allgemeine Peer-to-Peer-Netzwerke, z.B. Peers, Peer-Gruppe, Auffinden, Überwachung, Sicherheit usw. D.h. die Protokolle werden in JXTA Core enthalten.

Services-Schicht

Diese Schicht befaßt sich mit übergeordneten Konzepten wie indexing, file sharing, searching, Speichrungssystemen, Ressourcen-Aggregation, Authentifizierung usw.

Applikation-Schicht

Diese Schicht enthält Anwendungen wie Peer-to-Peer-EMail-Systeme, verteilte Auktions-Systeme, usw.

4.3.2 Bewertung

4.3.2.1 Spezielle Anforderungen

Vollständig erfüllt:

Dauerhaftigkeit

Diese Anforderung ist erfüllt, da die Dauer von Interaktionen zwischen Benutzern nicht beschränkt ist.

Dateitransfer

Die Datenübertragung ist durch die JXTA Protokolle möglich (Pipe Binding Protocol und Endpoint Routing Protocol).

Teilweise erfüllt:

Sitzungs- und Teilnehmerverwaltung

Diese Anforderung ist nur teilweise erfüllt, da die Mitgliedschaft in JXTA noch nicht realisiert ist. JXTA unterstützt noch nicht, dass man Teilnehmer in einer Peergruppe bestimmen kann oder dass man feststellen kann, welcher Peer online oder offline ist.

Nicht erfüllt:

Floor Control

Diese Anforderung ist nicht erfüllt, da JXTA nur eine generelle Plattform für die Entwicklung von Peer-to-Peer-Anwendungen zur Verfügung stellt. Floor Control ist jedoch anwendungsspezifisch.

Daten-Konsistenz

JXTA unterstützt die Konsistenz von Daten nicht, da JXTA eine generelle Plattform für die P2P-Anwendungen anbietet.

Daten-Integrität

JXTA unterstützt die Integrität von Daten nicht, da JXTA eine generelle Plattform für die P2P-Anwendungen anbietet.

Daten-Persistenz

JXTA unterstützt die Persistenz von Daten nicht, da JXTA eine generelle Plattform für die P2P-Anwendungen anbietet.

4.3.2.2 Allgemeine Anforderungen

Vollständig erfüllt:

Offenheit

Diese Anforderung ist völlig erfüllt, da die JXTA Software eine Open Source ist und daher die Erweiterung bestehender Funktionalitäten möglich ist.

Plattformen

JXTA ist plattformunabhängig, da die Software auf die unterschiedlichen beteiligten Plattformen, wie beispielsweise auf Unix, Microsoft Windows usw. und auf verschiedenen Geräte wie z.B. Laptops, Desktop PCs usw. laufen kann. JXTA kann in verschiedene Programmiersprachen z.B. c/c++, Java, Perl, usw. implementiert werden.

Peer-Auffinden

Diese Anforderung ist vollständig erfüllt, da das Auffinden von Peers ohne Einschränkung der Entfernung von Peers stattfindet. D.h. es gibt keine Beschränkung der physikalischen Lokation von Peers.

Stabilität

Die JXTA-Anwendung "InstantP2P" (siehe [INSTANTP2P]) enthält die Funktionen wie File Sharing, Chat für Gruppe und für one-to-one, usw. Dies ergibt die Stabilität des Systems.

Skalierbarkeit

Das System JXTA unterstützt Anwendungen mit mehr als 20 teilnehmenden Endgeräten.

Nicht erfüllt:

Sicherheit

Diese Anforderung ist nicht erfüllt, da JXTA keine Authentifizierungsmöglichkeit für Teilnehmer anbietet, um die Teilnahme an einer Peergruppe einzuschränken.

4.3.3 Zusammenfassung

Das System JXTA erfüllt zwei spezifische Anforderungen wie Dauerhaftigkeit, Datentransfer und alle allgemeinen Anforderungen bis auf die Sicherheit für die Authentifikation von Teil-

nehmern. Teilweise erfüllt ist die Sitzungs- und Teilnehmerverwaltung. Nicht erfüllt ist die Regelung der Rechtevergabe, Datenkonsistenz, Datenpersistenz und Datenintegrität.

4.3.4 Fazit

Anforderung	SASCIA	PROEM	JXTA
Offenheit	+	-	+
Plattformen	+	+	+
Peer-Auffinden	+	0	+
Stabilität	+	+	+
Skalierbarkeit	0/+	0/+	+
Sicherheit	+	-	-
Floor Control	0	-	-
Session-Management	0	+	0
Dauerhaftigkeit	+	-	+
Dateitransfer	+	+	+
Daten-Konsistenz	+	+	-
Daten-Persistenz	+	+	-
Daten-Integrität	+	+	-

Tabelle 4.1: Gesamtbewertung der existierenden Systeme

Aus der Gesamtbewertung (Tabelle 4.1) sieht man anhand der Anforderungen die Vorteile und Nachteile der Systeme SASCIA, PROEM und JXTA.

JXTA wird für die Arbeit der Diplomarbeit übernommen und eingesetzt, da JXTA eine generelle Plattform für die Entwicklung von Peer-to-Peer-Anwendungen anbietet. JXTA wird das Peer-Auffinden und die Kommunikation zwischen Peers im entworfenem System unterstützen.

Eine Implementierung auf Basis des Systems PROEM ist nicht zu empfehlen, da erstens die Dauerhaftigkeit der Interaktionen zwischen Benutzern kurz ist, zweitens das Peer-Auffinden von der Entfernung abhängt, drittens der Quellcode nicht verfügbar ist.

Die Aufzeichnungsdatenbank von SASCIA kann für das P2P-Framework übernommen und eventuell noch an das P2P-Framework angepasst werden.

5 Entwurf

In diesem Kapitel werden die Replikationsfaktoren für einzelne Anwendungsschichten untersucht, um heraus zu finden, welcher Teil oder welche Schicht einer Anwendung für die Zusammenarbeit auf andere Peers kopiert werden soll und bei welcher Schicht der Anwendung die Schnittstellen angelegt werden.

Dieses Ergebnis und die Ergebnisse aus dem letzten Kapitel führen zur Entscheidung des Entwurfs. D.h. die eigentliche dezentrale Peer-to-Peer-Architektur für die gemeinsame Nutzung von Anwendungen wird entworfen. Die Funktionalität der Komponenten der Architektur werden beschrieben und die Schnittstellen werden vorgestellt.

5.1 Untersuchung der Replikationsfaktoren für einzelne Anwendungsschichten

Eine Anwendung wird in vier Schichten dargestellt (Abb. 5.1):

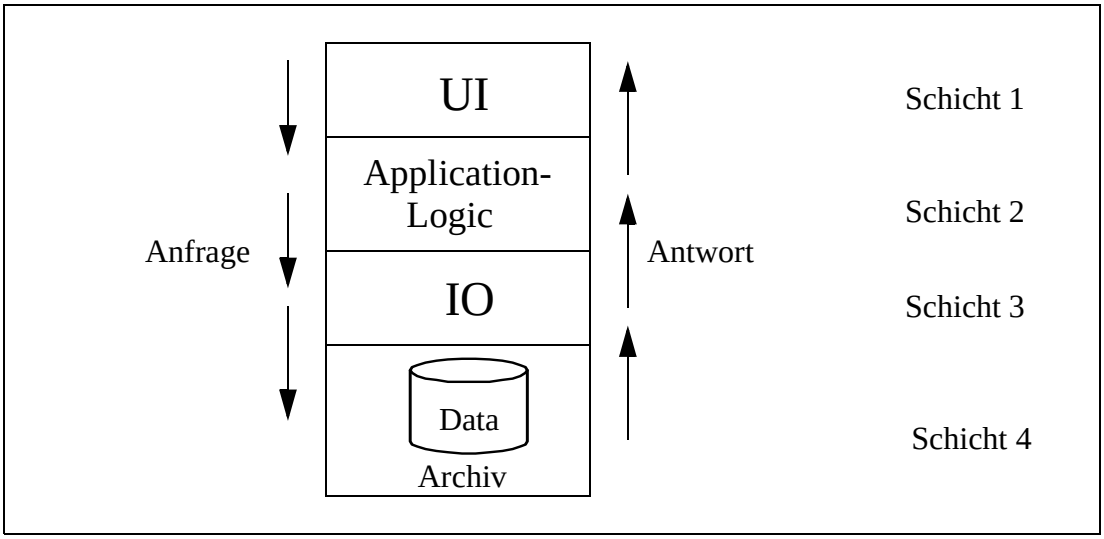

Abbildung 5.1: Schichtendarstellung einer Anwendung

Bei der Schicht **UI (User Interface)** läuft der Dialog zwischen Benutzer und Anwendungssystem nach bestimmten Regeln ab. Diese Regeln werden durch die Benutzungsschnittstelle festgelegt. Eine Eingabe des Benutzers wird an die weiteren Schichten weitergeleitet und dort

bearbeitet und ausgeführt. Das Ergebnis wird dem Benutzer angezeigt.

Die Schicht **Application-Logic** umfasst die eigentlichen Funktionen der Anwendung.

Die Schicht **IO (Input/Output)** umfasst die Umsetzung zwischen der Applikationslogik Schicht und der Archiv Schicht. Die Transaktionsfunktion wird hier bereitgestellt.

Die Schicht **Archiv** umfasst die Speicherung von Daten. Das Lesen sowie das Schreiben von Daten finden auf dieser Schicht statt.

5.1.1 Aufgaben der betrachteten Szenarien

Von den Szenarien im Kapitel 2 können die Aufgaben für diese Szenarien kurz gefasst werden:

- wie bekommen Studenten die Folien oder die Mitarbeiter die Anwendung und die Daten,
- wie wird die Darstellung oder das Dokument gesteuert,
- was passiert, wenn der Rechner des Anwendungsanbieters defekt ist und daher die Daten nicht mehr zur Verfügung stehen, wie können die Studenten oder die Mitarbeiter die Anwendung weiter zusammen nutzen usw.

5.1.2 Analyse der Lösungsmöglichkeiten

Durch Kopieren können die Studenten die Folien und die Mitarbeiter das Dokument bekommen. Es gibt mehrere Möglichkeiten für das Kopieren. Die Möglichkeiten werden in zwei Klassen eingeteilt: **zentrale Datenhaltung** und **lokale Datenhaltung**. Die Möglichkeiten lassen sich wieder in mehrere Fälle unterteilen. Die Anzahl der Fälle hängt von der Lage der Schnittstelle ab, über die die Kommunikation zwischen Peers stattfindet. D.h. bei welcher Schicht (bzgl. der Schichtendarstellung einer Anwendung) die Schnittstelle gelegt wird. Die Frage ist, welche Möglichkeit bzw. welcher Fall effektiv und effizient ist, um im Entwurf einsetzbar zu sein. Sie werden im folgenden Abschnitt analysiert und bewertet.

5.1.2.1 Zentrale Datenhaltung

Bei der zentralen Datenhaltung werden Daten an eine Stelle (in diesem Kontext ist die Zentrale der Besitzer-Peer, der Daten besitzt) verwaltet und gespeichert. Wenn ein Benutzer einen Knopf drückt, wird das Ereignis an den Besitzer-Peer gesendet und dort verarbeitet. Das Ergebnis wird an alle anderen Peers übertragen. Die Kommunikation zwischen Peers inklusiv dem Besitzer-Peer erfolgt über eine Schnittstelle, die bei einer Schicht der Anwendung gelegt wird.

In dieser Klasse gibt es drei Fälle:

- entweder nur die Benutzungsschnittstelle UI
- oder die UI und die Applikationslogik
- oder die UI und die Applikationslogik und die IO Schicht der Anwendung

werden an alle anderen Peers kopiert.

1. Fall: *Kopieren der UI der Anwendung*

In diesem Fall wird nur die Benutzungsschnittstelle (UI) der Anwendung vom Besitzer-Peer an alle anderen Peers kopiert (Abb. 5.2).

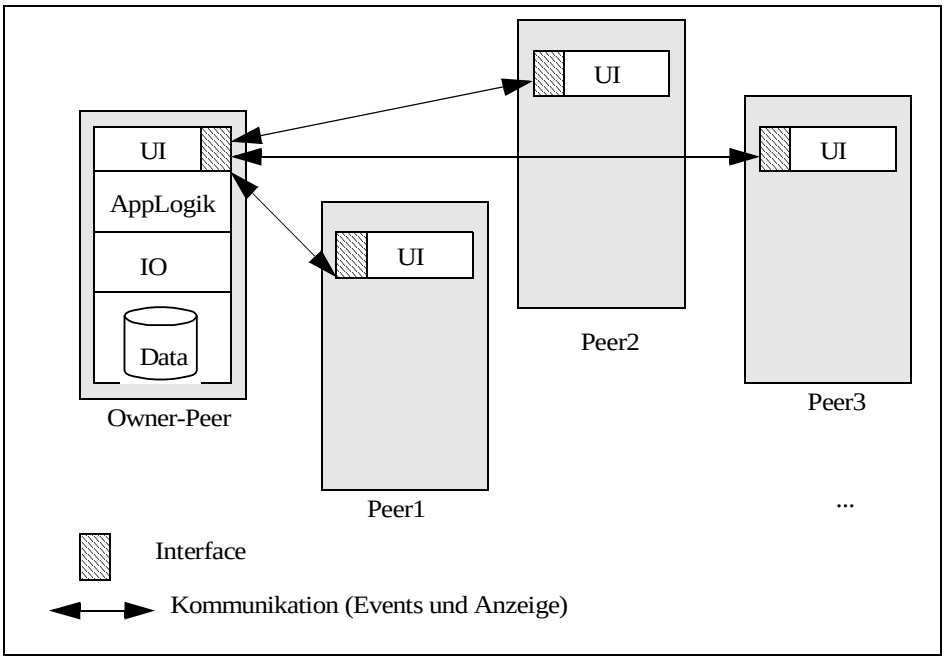

Abbildung 5.2: Zentrale Datenhaltung mit Interface bei UI Schicht

Das ist das Prinzip WYSIWIS (What You See Is What I See) [BURGER 97]. Die Kommunikation zwischen Peers inklusiv dem Besitzer-Peer erfolgt über die Schnittstelle bei UI. Wenn ein Ereignis (z.B. ein Knopf wird von einem Benutzer gedrückt) bei einem Peer stattfindet, wird das Ereignis an den Besitzer-Peer über die Schnittstelle gesendet (1:1-Verbindung). Und dort wird das Ereignis durch die anderen Schichten bis zur Archiv Schicht weitergeleitet (Falls Daten gebraucht werden) und lokal verarbeitet. Das Ergebnis wird wieder durch alle Schichten bis zu der lokalen UI gegeben und von dort an alle Peers übertragen (1:m-Verbindung). D.h. die anderen Peers bekommen eine aktuelle Anzeige vom Besitzer-Peer, obwohl die Änderungen bzw. die Ereignisse nicht von ihnen sind. Dieses Prinzip ist eine Art wie Fernsteuerung einer Anwendung mittels des Besitzer-Peer bzw. eines Servers (siehe [NGUYEN 00]).

Dieser Fall hat Vorteile und Nachteile, die im folgenden Abschnitt erläutert werden.

Vorteile:

- Daten sind konsistent, da die Daten nur an einer Stelle, bei Besitzer-Peer, bearbeitet und dort gespeichert werden.
- Die Schnittstelle (Interface) ist einfach zu implementieren, da die Schnittstelle hauptsächlich nur für Ereignis- und Anzeigeübertragung (als Ergebnisse) entworfen wird.

Nachteile:

- Ein großer Nachteil ist, dass die Daten nicht lokal persistent sind. Wenn der Besitzer-Peer nicht mehr anwesend ist, sind die Daten auch nicht mehr verfügbar und daher können alle anderen Peers nicht weiter arbeiten.
- Die Übertragungszeit der aktuellen Anzeige an jeden Peer kann sehr lang sein.
  Zum Betrachten ist eine typische Bildschirmauflösung von:
  1024 * 768 * 16 bit = ca. 1,5 MByte gerechnet.
  Als Ergebnis eines Ereignisses (neue Folie anzeigen) wird die UI mit der Größe von 1,5 MB vom Besitzer-Peer an jedem Peer übertragen. Das ist 1:m-Verbindung, m ist z.B. 20 Studenten/Peers. Bei einem Funk-LAN mit einem Durchsatz von 2Mbit/s ergibt sich die Übertragungszeit:
  (1,5 MByte * 8 bit) / (2 Mbit/s / 20 Studenten) = 120 Sekunden = 2 Minuten
  Das ist eine zu lange Antwortzeit für interaktive Benutzung.

2. Fall: *Kopieren der UI und der Applikationslogik Schicht der Anwendung*

In diesem Fall werden die obersten zwei Schichten der Anwendung vom Besitzer-Peer an alle Peers übertragen (Abb. 5.3). Die Kommunikation zwischen Peers inklusiv dem Besitzer-Peer erfolgt durch die Schnittstelle bei der Applikationslogik. Ein Ereignis eines Peers wird an die Applikationslogik Schicht lokal weitergeleitet. Die **betreffende Funktion** wird an den Besitzer-Peer gesendet (1:1-Verbindung) und dort ausgeführt. Das Ergebnis wird vom Besitzer-Peer an alle Peers über die Schnittstelle bei der Applikationslogik Schicht gesendet (1:m-Verbindung). Und von dort wird es an die Benutzungsschnittstelle geliefert. Die Daten werden wie in dem ersten Fall beim Besitzer-Peer gespeichert.

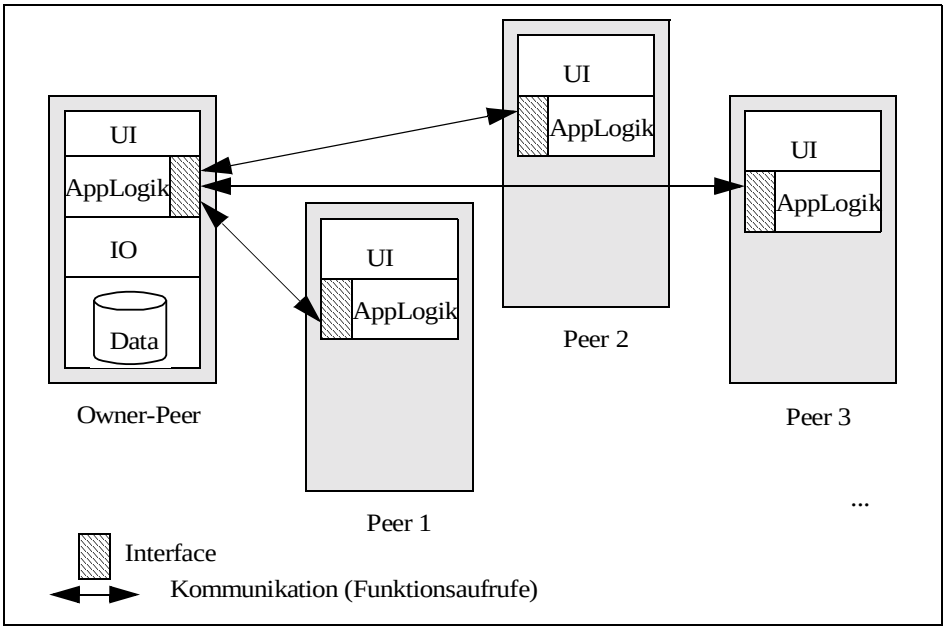

Abbildung 5.3: Zentrale Datenhaltung mit Interface bei Applikationslogik Schicht

Vorteile:

- Daten sind konsistent, da die Daten nur an einer Stelle, bei originalem Peer, bearbeitet und dort gespeichert werden.
- Die Schnittstelle bei der Applikationslogik Schicht ist anwendungsspezifisch. Das ist ein Vorteil besonders für Anwendungen, die spezifische Eigenschaften haben, da die Schnittstelle spezifische Funktionen enthalten kann. Z.B. kann die Kommunikationslast minimiert werden, wenn anwendungsspezifische intelligente Funktionen benutzt werden, die die Übertragung weniger Daten erfordern. Dann ist die Übertragungszeit kurz.

Nachteile:

- Die Daten sind nicht lokal persistent. Wenn der Besitzer-Peer nicht mehr anwesend ist, sind die Daten nicht mehr verfügbar und daher können alle anderen Peers nicht weiter arbeiten.
- Die Schnittstelle bei der Applikationslogik Schicht ist anwendungsabhängig. Das ist ein Nachteil, da eine Schnittstelle nicht alle Anwendungen abdecken und daher auch nicht standardisiert werden kann. Eine Schnittstelle für alle allgemeine Anwendungen ist nicht möglich.

- Die Kommunikationslast ist anwendungsabhängig. Wenn eine Anwendung Funktionen hat, die große Datenmengen enthalten, ist die Kommunikationslast wegen der Parameter-Übertragung groß.

3. Fall: *Kopieren der UI und der Applikationslogik und der IO Schicht der Anwendung*

In diesem Fall werden die obersten drei Schichten der Anwendung vom Besitzer-Peer an alle Peers übertragen (Abb. 5.4).

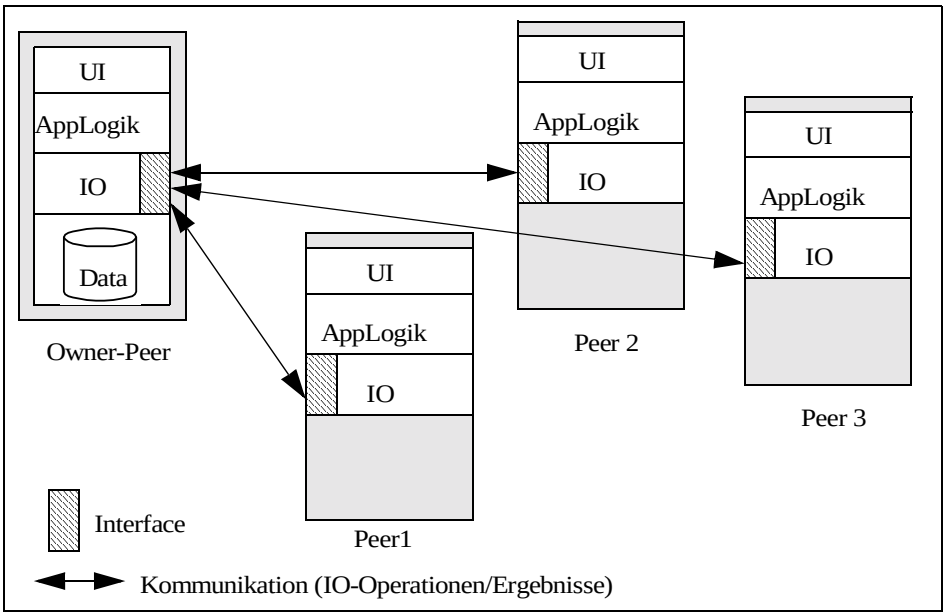

Abbildung 5.4: Zentrale Datenhaltung mit Interface bei Input/Output Schicht

Die Kommunikation zwischen Peers inklusiv dem Besitzer-Peer erfolgt durch die Schnittstelle bei der Input/Output Schicht. Diese Schnittstelle ist normalerweise für die Kommunikation mit z.B. einer entfernten Datenbank zuständig, in diesem Kontext mit dem Besitzer-Peer. Wenn ein Ereignis bei einem Peer stattfindet, wird das Ereignis an die weiteren Schichten geleitet und dort verarbeitet. Auf der IO Schicht werden die Input/Output-Operationen, Schreib/Lese-Operationen, an den Besitzer-Peer gesendet und von dort ausgeführt. Also z.B. ein Datensatz wird von einem Peer geholt, geändert und wieder an den Besitzer-Peer gesendet. D.h. der Read-Befehl wird an den Besitzer-Peer gesendet und das Read-Ergebnis an den Peer zurück geschickt. Der Peer hat den Datensatz geändert und schickt die Änderungen, den Schreib-Befehl, an den Besitzer-Peer. Ein Transaktionsmechanismus für die Konsistenz von Daten wird benötigt. Wenn ein Peer mit dem Datensatz, der gerade von einem anderen Peer geändert wurde, arbeitet, muss er die Änderung bzw. die aktuellen Daten auch bekommen.

Vorteile:

- Daten sind konsistent, da die Daten nur an einer Stelle, beim originalem Peer, bearbeitet und dort gespeichert werden.
- Die Schnittstelle ist einfach zu implementieren, da die Schnittstelle haupsächlich für die Übertragung von IO-Operationen entworfen wird.

Nachteile:

- Die Daten sind nicht lokal persistent. Wenn der Besitzer-Peer nicht mehr anwesend ist, sind die Daten auch nicht mehr verfügbar und daher können alle anderen Peers nicht weiter arbeiten.
- Wenn die IO-Operationen große Datensätze enthalten, ist die Übertragungszeit lang.

5.1.2.2 Lokale Datenhaltung

Bei der lokalen Datenhaltung werden Daten bei jedem Peer lokal bearbeit und gespeichert. Peers kommunizieren über eine noch zubestimmende Schnittstelle miteinander. Wenn ein Ereignis eines Peers stattfindet, wird das Ereignis über die Schnittstelle an alle anderen Peers gesendet. Das ist eine n:m-Verbindung. Die Aktion wird an die weiteren lokalen Schichten bei jedem Peer weitergeleitet und dort bearbeitet und durchgeführt. Das Ergebnis wird lokal bei jedem Peer angezeigt. Es existiert deshalb keine explizite Ergebnisübertragung zwischen Peers.

Bei der lokalen Datenhaltung bringt die Übertragung der kompletten Anwendung einen großen Vorteil gegenüber der zentralen Datenhaltung. Dieser Vorteil ist, dass die Daten lokal pesistent sind. Wenn ein Peer nicht mehr anwesend ist, sind die Daten immer verfügbar, da sie lokal gespeichert und verwaltet werden. Es ergibt sich wiederum das Problem der Konsistenz von Daten. Dieses Problem wird im folgenden Abschnitt betrachtet.

Nicht betrachtete Konfiguration

1. Fall: Kopieren der IO und der Archiv Schicht und

2. Fall: Kopieren der UI und der IO und der Archiv Schicht (siehe Abbildung 5.5)

UI und Daten sind für eine Anwendung notwendig. Wenn eine Zwischenschicht (Applikationslogik/IO) fehlt, ist eine lokale Verarbeitung nicht möglich. Deshalb machen diese Fälle keinen Sinn, da die Anwendung beim ersten Fall keine Benutzungsschnittstelle und keine Applikationslogik und beim zweiten Fall keine Applikationslogik hat. Im P2P-Framework muss jeder

Peer ohne Server selbständig arbeiten. Wenn es keine Applikationslogik bei einem Peer gibt, kann dieser Peer nicht selbständig arbeiten. Deshalb werden die beiden Fälle nicht weiter betrachtet.

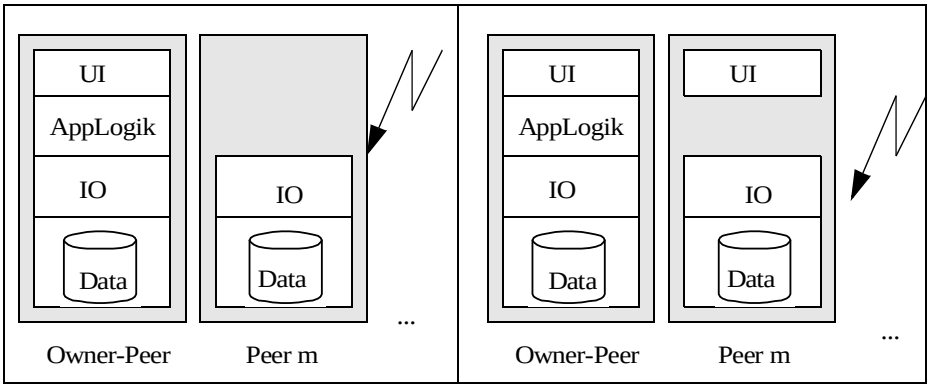

Abbildung 5.5: Die nicht betrachteten Fälle der lokalen Datenhaltung

In der lokalen Datenhaltung Klasse gibt es vier Fälle zu betrachten, um festzustellen, welche Schnittstellen für die Kommunikation zwischen Peers benötigt werden.

- Die komplette Anwendung (vier Schichten) mit Schnittstelle bei der UI Schicht
- oder die komplette Anwendung mit Schnittstelle bei der Applikationslogik Schicht
- oder die komplette Anwendung mit Schnittstelle bei der IO Schicht
- oder die komplette Anwendung mit Schnittstelle bei der Archiv Schicht

wird vom Besitzer-Peer an alle Peers kopiert. Von jetzt an existiert die Rolle des Besitzer-Peers (hinsichlich des Eigentumsrechts von Daten) nicht mehr, sondern alle Peers inklusiv des Besitzer-Peers können gleichberechtig (hinsichtlich der Datenhaltung) miteinander interagieren sowie kommunizieren. Deshalb werden die folgenden Abbildungen ohne Besitzer-Peer beschriftet. Die vier Fälle werden im folgenden Abschnitt analysiert und bewertet.

1. Fall: *Kopieren der kompletten Anwendung mit Schnittstelle bei der UI Schicht*

In diesem Fall wird die komplette Anwendung mit Schnittstelle bei der UI Schicht an alle anderen Peers übertragen (Abb. 5.6). Peers kommunizieren über die Schnittstelle miteinander. Es gibt zwei Möglichkeiten der Kommunikation über diese Schnittstelle bei der UI: entweder das **Ereignis** oder das **Ergebnis** wird an alle Peers übertragen. Mit Floor Control kann nur ein Peer etwas ändern oder bearbeiten.

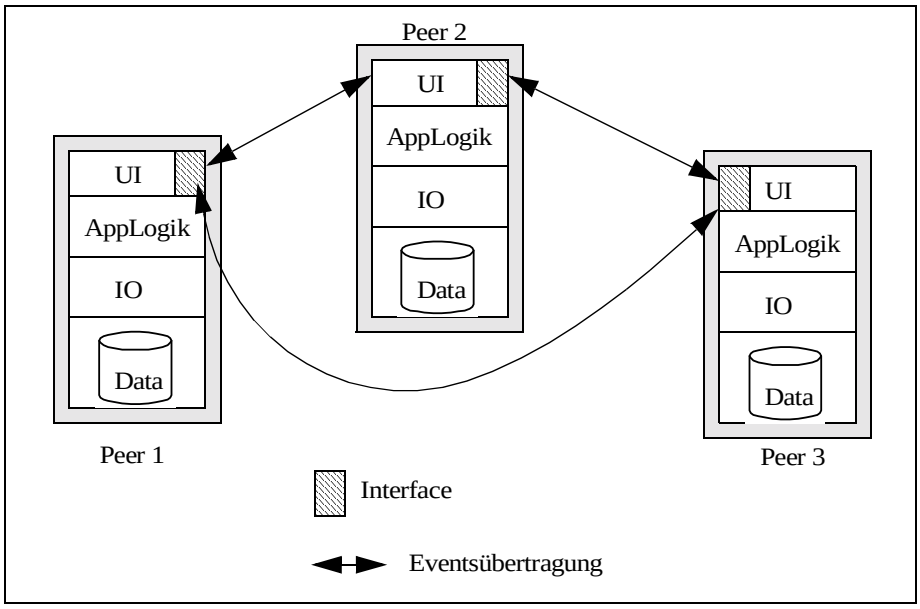

Abbildung 5.6: Lokale Datenhaltung mit Interface bei UI Schicht

a) **Erste Möglichkeit:** Wenn ein Ereignis eines Peers stattfindet, wird das **Ereignis** über die Schnittstelle an alle anderen Peers gesendet. Das Ereignis läuft bei jedem Peer alle Schichten durch, wird dort bearbeitet. Das Ergebnis wird bei jedem Peer wieder durch alle Schichten bis an die UI Schicht gegeben und angezeigt. Es ergibt sich keine explizite Ergebnisübertragung zwischen den Peers.

Vorteile:

- Daten sind lokal persistent, da die Daten lokal gespeichert und verwaltet werden.
- Das Interface ist einfach zu realisieren, da die Schnittstelle hauptsächlich nur für Ereignisübertragung entworfen wird.
- Die Kommunikationslast ist geringt, wenn z.B. ein Knopf gedrückt wird, wird das Ereignis an alle anderen Peers übertragen. Das ist die [1: (n-1)]-Verbindung (n = 21 Peers). Die Übertragungszeit des Signals/Ereignisses mit Größe z.B. von 3 Bytes (für die Indentifikation des Knopfs) von einem Peer an einen anderen Peer ist:
  Daten zum Übertragen: [3 Byte . 8 bit . n . (1: n-1)]
  Übertragungszeit: [3 Byte .8 bit . 21 . 20] / [2 Mbit : 20] = 0.10 Sekunden

Nachteil:

- Die Daten sind inkonsistent bei Funktionen mit Seiteneffekten (z.B. bei Einbeziehung

der Uhrzeit), da auf jedem Peer eine getrennte Verarbeitung stattfindet. Von daher würde jeder Peer zu einem anderen Ergebnis kommen, obwohl jeder Peer das gleiche Ereignis erhalten hat.

b) **Zweite Möglichkeit**: Drückt ein Benutzer einen Knopf, wird das Ereignis bei ihm zuerst verarbeitet und dann das **Ergebnis** bzw. die ganze UI über die Schnittstelle an alle andere Peers übertragen.

In diesem Fall sind die Daten inkonsistent trotz der Floor Control, da die Daten nur bei dem Peer, der das Ereignis verarbeitet, geändert werden und bei allen anderen Peers die alten Daten noch zur Verfügung stehen. Die zweite Möglichkeit macht keinen Sinn und wird deshalb nicht weiter betrachtet.

2. Fall: *Kopieren der kompletten Anwendung mit Schnittstelle bei der Application-Logik Schicht*

Bei diesem Fall erfolgt die Kommunikation zwischen Peers durch die Schnittstelle bei der Applikationslogik Schicht (Abb. 5.7).

Wenn ein Ereignis bei einem Peer stattfindet, wird das Ereignis an die Applikationslogik Schicht weitergeleitet. Von hier wird das Ereignis in eine Funktion umgewandelt und diese Funktion wird an alle anderen Peers über die Schnittstelle bei der Applikationslogik Schicht gesendet. Das ist eine n:m-Verbindung. Die Funktion wird bei jedem Peer lokal verarbeitet und durchgeführt. Es gibt keine Übertragung von Ergebnissen.

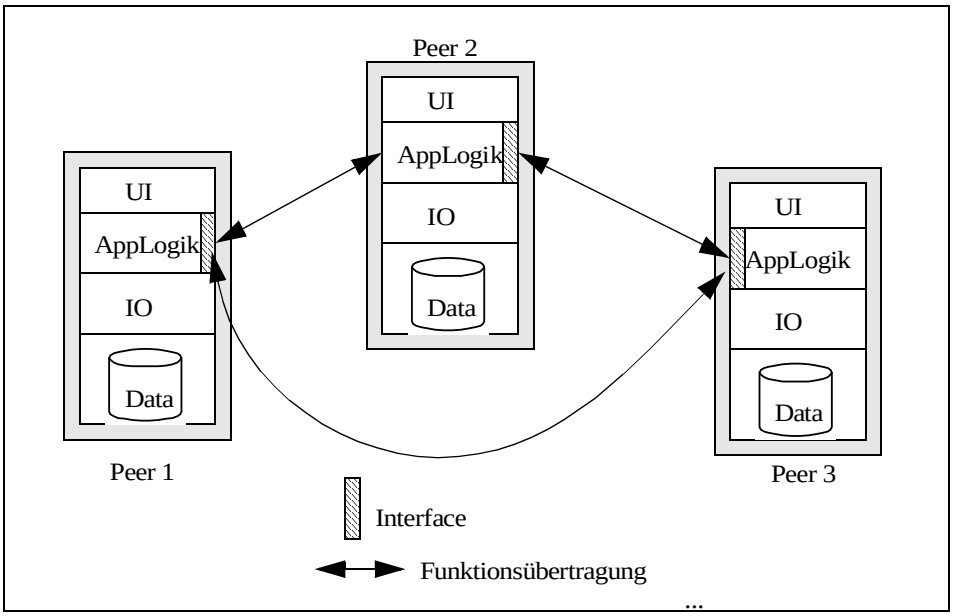

Abbildung 5.7: Lokale Datenhaltung mit Interface bei Applikationslogik Schicht

Vorteile:

- Daten sind lokal persistent, da die Daten lokal gespeichert und verwaltet werden.
- Die Schnittstelle bei der Applikationslogik Schicht ist anwendungsspezifisch. Das ist ein Vorteil besonders für Anwendungen, die spezifische Eigenschaften haben, da die Schnittstelle spezifische Funktionen enthalten kann. Z.B. kann die Kommunikationslast gering sein, wenn die Schnittstelle intelligente Funktionen hat, die kleine Datenmenge benötigt.

Nachteile:

- Die Schnittstelle ist nicht einfach zu implementieren, da die Schnittstelle anwendungsabhangig ist. Eine Schnittstelle kann deshalb nicht alle Anwendungen abdecken und daher auch nicht standardisiert werden. Eine Schnittstelle für allgemeine Anwendungen ist nicht möglich.
- Es ist nicht einfach, die Konsistenz von Daten zu erhalten. Ein Synchronisationsmechanismus z.B. die Floor Controll wird dafür benötigt.

3. Fall: *Kopieren der kompletten Anwendung mit Schnittstelle bei der IO Schicht*

Bei diesem Fall erfolgt die Kommunikation zwischen Peers über die Schnittstelle bei der IO Schicht (Abb. 5.8).

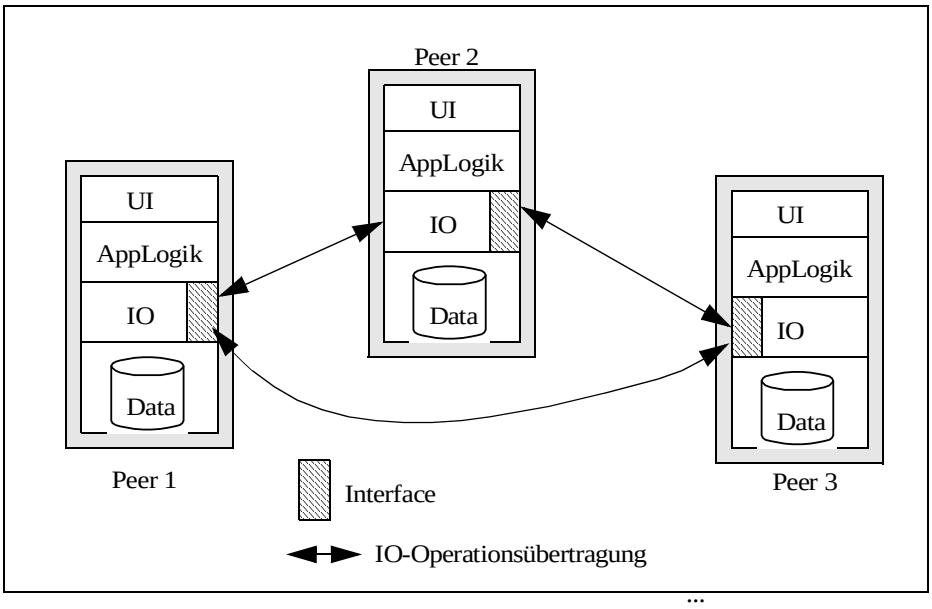

Abbildung 5.8: Lokale Datenhaltung mit Interface bei IO Schicht

Wenn ein Ereignis bei einem Peer stattfindet, wird das Ereignis lokal bis der IO Schicht verarbeitet. Die **IO-Operation (Schreiben/Lesen)** werden an jeden Peer über die Schnittsstelle gesendet. Das Schreiben oder Lesen wird bei jedem Peer lokal durchgeführt. Das Ergebnis wird lokal bei jedem Peer gezeigt. Es gibt keine Ergebnisübertragung zwischen Peers.

Vorteile:

- Daten sind lokal persistent, da die Daten lokal gespeichert und verwaltet werden.
- Die Schnittstelle ist einfach zu implementieren, da die Schnittstelle hauptsächlich für Lesen/Schreiben-Operationen entworfen wird.

Nachteil:

- Die Kommunikationslast ist hoch, wenn der Schreib-Befehl große Daten enthält. Das ist anwendungsabhängig.

4. Fall: *Kopieren der kompletten Anwendung mit Schnittstelle bei Archiv Schicht*

Bei diesem Fall erfolgt die Kommunikation zwischen Peers über die Schnittstelle bei der Archiv Schicht (Abb.5.9).

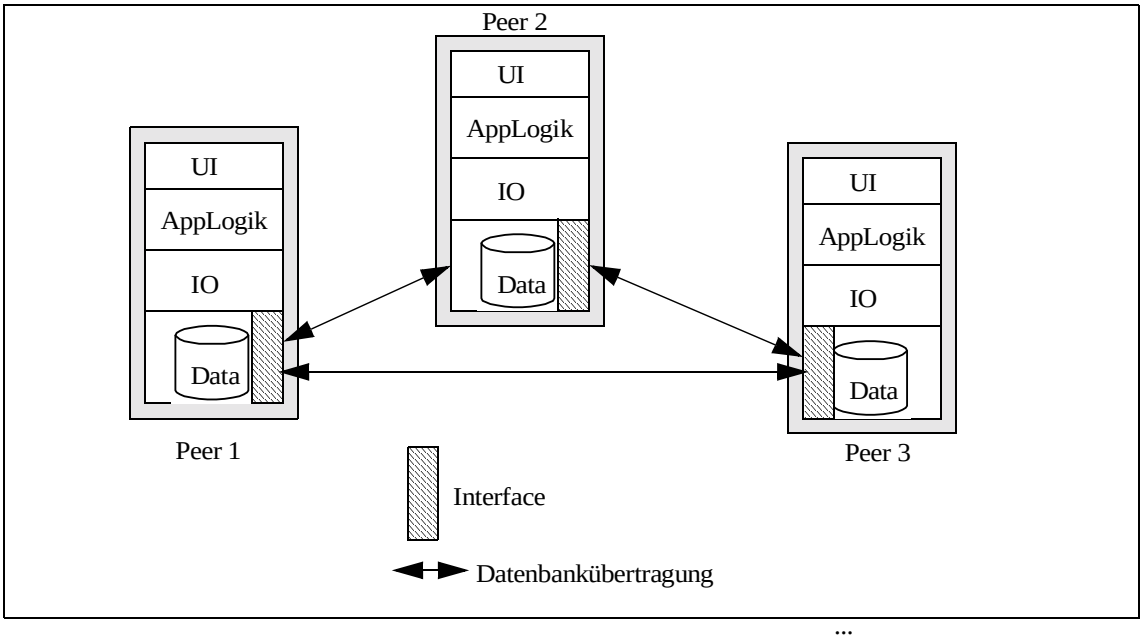

Abbildung 5.9: Lokale Datenhaltung mit Interface bei Archiv Schicht

Wenn ein Ereignis bei einem Peer stattfindet, wird das Ereignis zuerst bei ihm vollständig verarbeitet. Der gesamte Inhalt der Datenbank wird an alle anderen Peers übertragen. Die anderen Peers bekommen die Daten.

Vorteile:

- Daten sind lokal persistent, da die Daten lokal gespeichert und verwaltet werden.
- Die Interface ist einfach, da die Schnittstelle hauptsächlich für Datentransport entworfen wird.
- Kein Synchronisationsmechanismus für die Erhaltung der Datenkonsistenz wird benötigt.

Nachteil:

- Die Kommunikationslast ist groß, da die gesamte Datenbank an alle anderen Peers übertragen wird, wenn ein Knopf bei einem Peer gedrückt wird.

5.1.3 Zusammenfassung und Fazit

Die Ergebnisse der Analyse der verschiedenen Lösungsmöglichkeiten werden in der Tabelle 5.1 zusammengefaßt. Die Bedeutung der Zeichen + +, +, 0, -, -- wird wie folgt erklärt:

+ + : sehr gut

+ : gut

0 : neutral

- : nicht gut

-- : schlecht

Ein Vorgehen wird neutral bewertet, wenn es sowohl Vorteile als auch Nachteile in einem Bewertungsfaktor hat.

	Interface bei	**Interface Implementierung**	**Daten Konsistenz**	**Daten Persistenz**	**Kommunikations -last**
Zentrale Datenhaltung	UI Schicht	+	+ +	- -	0
	App-Logik-Schicht	0	+ +	- -	0
	IO Schicht	+	+ +	- -	0
Lokale Datenhaltung	UI Schicht	+	-	+ +	+
	App-Logik Schicht	0	-	+ +	0
	IO Schicht	+	+	+ +	0
	Archiv Schicht	+	+	+ +	- -

Tabelle 5.1: Gesamtbewertung

Ein großer Vorteil bei der zentralen Datenhaltung ist, dass die Daten konsistent sind. Trotzdem sind die drei Fälle bei der zentralen Datenhaltung für den Entwurf nicht zu empfehlen, da es großes Problem der Datenpersistenz gibt. Die Persistenz von Daten ist eine wichtige Anforderung des Peer-to-Peer-Konzepts. Und diese Fälle haben diese Anforderung nicht erfüllt. Von daher werden sie für den Entwurf nicht genommen.

Das Vorgehen der Schnittstelle bei UI Schicht (bei der lokale Datenhaltung) hat jedoch einen großen Nachteil. Der Nachteil ist, dass die Erhaltung der Datenkonsistenz von schwierig bis zu

nicht lösbar ist. Man hätte Probleme der Funktionen mit Seiteneffekten, da auf jedem Peer eine getrennte Verarbeitung stattfindet. Von daher würde ein anderes Ergebnis bei jedem Peer kommen, obwohl jeder Peer gleiches Ereignis erhalten hat. Deshalb wird dieser Fall für den Entwurf nicht ausgewählt.

Die lokale Datenhaltung ist für den Entwurf zu empfehlen. Ein großer Vorteil dieser Klasse ist, dass die Daten lokal persistent sind. Die drei Fälle: das Interface bei Applikationlogik-, IO- und bei Archivschicht werden auf jeden Fall für den Entwurf genommen, da jeder Fall für bestimmte Aufgaben zuständig ist.

- Die Schnittstelle für Peer-to-Peer-Computing wird auf der Applicationslogik Schicht implementiert. Die Schnittstelle ist anwendungsabhängig. Dies bedeutet, dass auf der Seite des Frameworks allgemeine Funktionen für Kommunikation und Synchronisation realisiert werden und auf der Seite der Anwendung mehr Implementierungsaufwand benötigt wird. Dadurch sind jedoch intelligente anwendungsspezifische Lösungen möglich. Z.B. wird für die Synchronisation im Framework eine Floor Controll implementiert, jedoch erfolgt im Framework nur die Aushandlung wer den Floor erhält. Auf der Anwendungsseite muss die eigentliche Steuerung des Zugangs implementiert werden.
- Die Schnittstelle der IO Schicht ist für die Kommunikation mit entfernter Datenbank zuständig. Diese Schnittstelle ist für das Peer-to-Peer-Computing nicht notwendig, sondern nur für bestimmte Anwendungsfälle, bei denen eine Anwendung eine entfernte Datenbank benötigt. Die Implementierung des Interfaces entspricht einem Transaktionsystem für entfernte Datenbankzugriffe. Der Synchronisationsmechanismus ist anwendungsunabhängig.
- Die Schnittstelle der Archiv Schicht ist für das Kopieren von Daten zum lokalen Arbeiten zuständig. Der Vorteil dieses Vorgehens ist, dass der Synchronisationsmechanismus der Erhaltung der Daten anwendungsunabhängig ist. Es hat jedoch einen großen Nachteil, dass die Kommunikationslast groß ist, da die gesamte Daten an alle Peers übertragen werden.

5.2 Entwurfsentscheidung

Das Ergebnis der Eignungsanalyse von existierenden Systemen im letzten Kapitel und die Ergebnisse in 5.1 führen zur Entwurfsentscheidung für die dezentrale P2P-Architektur für die gemeinsame Nutzung von Anwendungen. Zuerst werden die Komponenten und ihre Beziehungen vorgestellt. Dann werden die Abläufe im Detail erklärt.

5.2.1 Die Peer-to-Peer-Architektur für Application Sharing

Die Peer-to-Peer-Architektur (siehe Abb. 5.10) bietet Anwendungsentwicklern ein Framework an, auf dem P2P-Anwendungen aufgebaut werden. Das Framework ermöglicht es, Peers und ihre Dienste aufzufinden und die Dienste gemeinsam zu nutzen. Das System hat zwei Hauptmodule: Session Manager und Data/Application Sharing. Der Peer-Manager ist für die Verwaltung von Peers z.B. Peer-Informationen, Peer-Auffinden, Teilnahme an einer Peergroup, usw. zuständig. Die Data/Application Sharing ist für die gemeinsame Nutzung einer Anwendung zuständig, z.B. für die Zugriffsrechte auf Daten sowie Anwendungen oder für die Erhaltung der Daten-Konsistenz, usw.

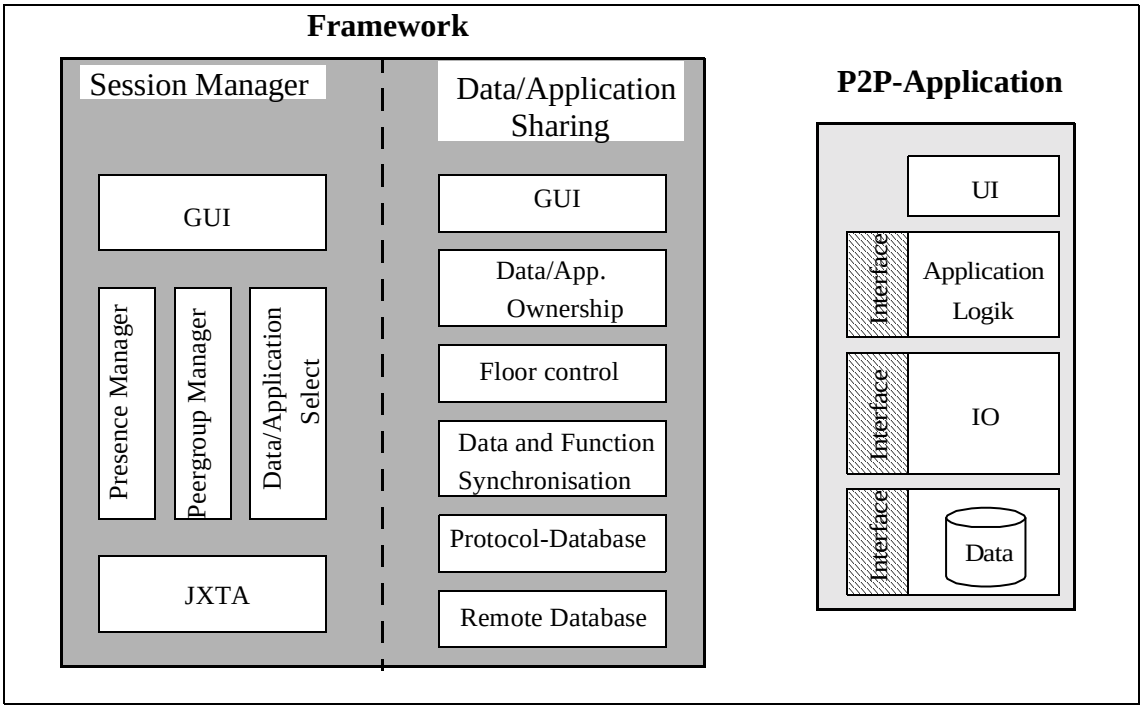

Abbildung 5.10: P2P-Software-Architektur für Application Sharing

Bevor die Komponenten des Frameworks erklärt werden, wird die Begriffskonvention vorgestellt, um die Mehrdeutigkeit der Begriffe zu erläutern, da in Peer-to-Peer Systemen die Begriffe wie Peer, Peergruppe, usw. häufig benutzt werden:

Peer: Ein Rechner PC oder mobile Geräte wie Laptop, PDA usw.

Peergruppe: Ist eine Gruppe, an der Peers jeder Zeit teilnehmen können. Peergruppe ist quasi wie eine Sitzung oder eine Veranstaltung, an der Zuhörer oder Teilnehmer teilnehmen können.

Join: Bedeutet für den Kontext einer Sitzung oder einer Veranstaltung das Teilnehmen oder das Anmelden.

Leave: Bedeutet für den Kontext einer Sitzung oder einer Veranstaltung das Abmelden oder das Verlassen.

D.h. ein Peer kann an einer Peergruppe teilnehmen sowie die Peergruppe verlassen.

5.2.2 Kommunikation in einer Peergruppe

JXTA (wie im Abschnitt 4.3 vorgestellt) unterstützt das Peer-Auffinden und die Kommunikation zwischen Peers. Dafür liefert JXTA eine API für die Implementierung. JXTA bietet sogenannte Pipes für die Kommunikation. Pipes (wie in Kap 4.3 erwähnt) sind virtuelle Kanäle, über die Peers Nachrichten senden und empfangen können.

Eigenschaften der Kommunikation

1. Jeder Peer einer Peergruppe kann mit jedem anderen Peer der Peergruppe über eine direkte Verbindung Daten austauschen.
2. Die Kommunikation ist zuverlässig.

Für die **Kommunikation** zwischen Peers in einer Peergruppe werden zwei Arten von Pipe verwendet: Propagate Pipe und Point-to-Point Pipe.

5.2.2.1 Propagate-Kommunikation

Propagate Pipe funktioniert wie Multicast. D.h. wenn ein Peer eine Nachricht sendet, empfangen alle Peers in Peergruppe inklusive ihm diese Nachricht. Für die Erzeugung einer Propagate Pipe wird ein Propagate Pipe Advertisement (wie in Kap 4.3 erwähnt) benötigt. Jede Peergruppe hat genau ein eigenes Propagate Pipe Advertisement. Dies bedeutet, dass jede Peergruppe einen eigenen Propagate-Kommunikationskanal hat, über den die Peers in der Peergruppe Nachrichten senden und empfangen können. Der Propagate-Kommunikationskanal ist das

Herzstück von einer Peergruppe, da fast alle "Gespräche" zwischen Peers über diesen Kanal geschehen.

Dieser Kanal wird verwendet für:

- Sitzungs- und Teilnehmerverwaltungsnachrichten usw.,
- Floor Control Nachrichten,
- Datensynchronisation z.B. die Übertragung von Ereignissen bei Applikation Sharing.

Jeder Peer in der Peergruppe muss genau dieses Propagate Pipe Advertisement verwenden, um den Propagate-Kommunikationskanal zu erzeugen und damit Nachrichten an alle Peers senden, sowie Nachrichten von den anderen Peers in der Peergruppe empfangen zu können.

5.2.2.2 Point-to-Point-Kommunikation

Ein Point-to-Point-Pipe verbindet exakt zwei Peers miteinander. Dieser Kanal wird verwendet, um den Propagate-Kommunikationskanal der Peergruppe zu entlasten. D.h. Nachrichten, die nicht unbedingt für alle Teilnehmer sind, werden über diesen Kanal gesendet. Z.B. für die Übertragung der Peergruppeninformationen an einen neuen Teilnehmer oder des Codes der Anwendung. Um den Kanal zu erzeugen, wird ein sogenanntes Secure Unicast Pipe Advertisement benötigt. Jeder Peer erzeugt ein eigenes Secure Unicast Pipe Advertisement. Ein Peer, der mit einem anderen Peer reden will, muss genau das Secure Unicast Pipe Advertisement des anderen Peers verwenden, um Nachrichten an ihn zu senden.

Die Konzepte von *Channel* in JSDT (siehe [JSDT]) und *Pipe* in JXTA scheinen weitgehend identisch zu sein. Aber hinsichtlich des Verbindungsausbau gibt es Unterschied zwischen *Channel* und *Pipe*. JSDT ist ein Client/Server System. Für das Erzeugen von *Channels* wird ein *Session-Objekt* benötigt. Für das Erzeugen des *Session-Objekts* wird wiederum ein Server benötigt. Jeder Verbindungsausbau läuft über den Server. D.h. ein Server wird für jeden Verbindungsausbau in JSDT benötigt. In JXTA hingegen wird kein Server für jeden Verbindungsausbau benötigt.

5.2.3 Session-Manager

Hierarchische Struktur in das P2P-Framework:

Ein Peer hat eine eindeutige Identität, die PeerID. Jeder Peer kann an mehreren Peergruppen teilnehmen. Jede Peergruppe hat eine eindeutige Identität, die PeerGroupID und kann mehrere

Anwendungen für die Zusammenarbeit enthalten. Jedes Applikation Sharing hat eine eindeutige Indentität, die ApplicationSharingID. Die Abbildung 5.11 zeigt diese hierarchische Struktur für einen Peer im P2P-Framework.

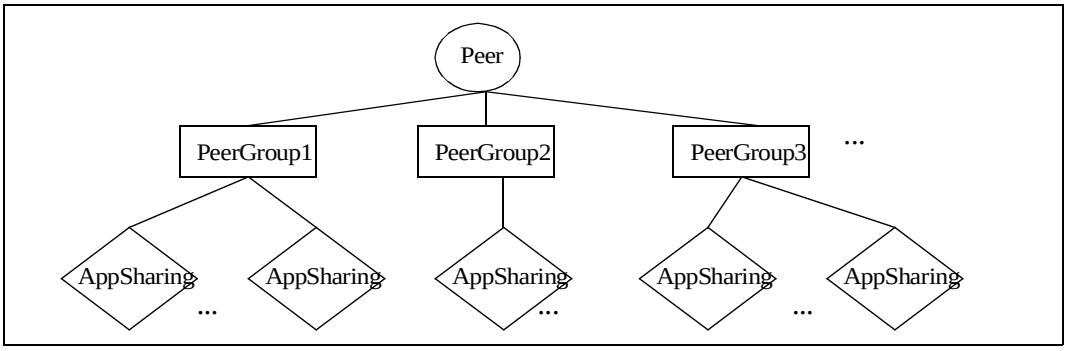

Abbildung 5.11: Hierarchische Struktur im P2P-Framework

Die Komponenten des Session-Managers sind: das System JXTA, Presence Manager, Peergroup Manager, Data/Application Select und GUI. Das System JXTA wurde in 4.3 beschrieben, deshalb wird es in diesem Abschnitt nicht im Detail erklärt. Der Peergroup Manager wird im Abschnitt 5.2.3.1 genau beschrieben.

Presence-Manager ist für das Auffinden von Pipe-Advertisements und Peergruppen-Advertisements zuständig.

Wenn er ein Peergruppen-Advertisement gefunden hat, kann er die Informationen wie den Namen und die Beschreibung der erzeugten Peergruppe ermitteln. Er gibt diese Informationen an den Peergroup Manager weiter. Dort werden sie bearbeitet, z.B. die Namen von erzeugten Peergruppen werden in die Peergruppenliste eingetragen und bei jedem Peer angezeigt.

Wenn er Pipe-Advertisements gefunden hat, kann er diese Advertisements lokal speichern und für die Erzeugung von Kommunikationskanälen benutzen.

Die Peergruppe- und Pipe Advertisements werden regemäßig wieder veröffenlicht (published), damit die späteren Teilnehmer die Peergruppen auffinden und die Kommunikation starten können. Das Problem ist, dass man weiss nicht, wann und welcher Peer abstürzen wird. Deshalb wird die Wiederholung der Veröffenlichung von Advertisements bei jedem Peer realisiert, um das Auffinden von Peergruppen zu gewährleisten. Von daher werden Advertisements beim Vorgang des Auffindens gefiltert, um das mehrfache Erhalten von Advertisements zu vermeiden.

Data/Application Select ist für die Auswahl von Diensten (Daten und Anwendungen) zustän-

dig. Hier findet das Setzen der Zugriffsrechte auf Daten statt. Wenn ein Teilnehmer bestimmte Daten und Anwendungen ausgewählt hat, kann er erfahren, ob er die Daten nur anschauen oder auch ändern bzw. bearbeiten darf.

GUI steht für Graphical User Interface. Die GUI des Peer-Managers ermöglicht es, dass Benutzer Peergruppen sowie angebotene Daten und Anwendungen auswählen können, dass er Namen von Mitgliedern der Peergruppe sehen kann. Im Kapitel 6 wird diese GUI genauer beschrieben.

5.2.3.1 Peergroup-Manager

Peergroup-Manager ist nicht nur für das Anmelden/Anwesenheit sowie Abmelden/Abwesenheit von Peer-Mitgliedern zu Gruppen, sondern auch für Informationen über angebotene Daten und Anwendungen zuständig. Wie oben erwähnt hat jede Peergruppe einen eigenen Kommunikationskanal, d.h. in dem JXTA Netzwerk können mehrere Peergruppen existieren und parallel arbeiten. Ein Peer kann an mehreren Peergruppen teilnehmen. Z.B. zwei Übungsgruppen von Verteilte Systeme und Datenbank finden parallel statt. Entsprechend zweier Übungsgruppen werden zwei Peergruppen erzeugt. Jeder Student kann an einer von den Peergruppen oder an den beiden Peergruppen teilnehmen. Solche Kontexte werden in der Abbildung 5.12 dargestellt:

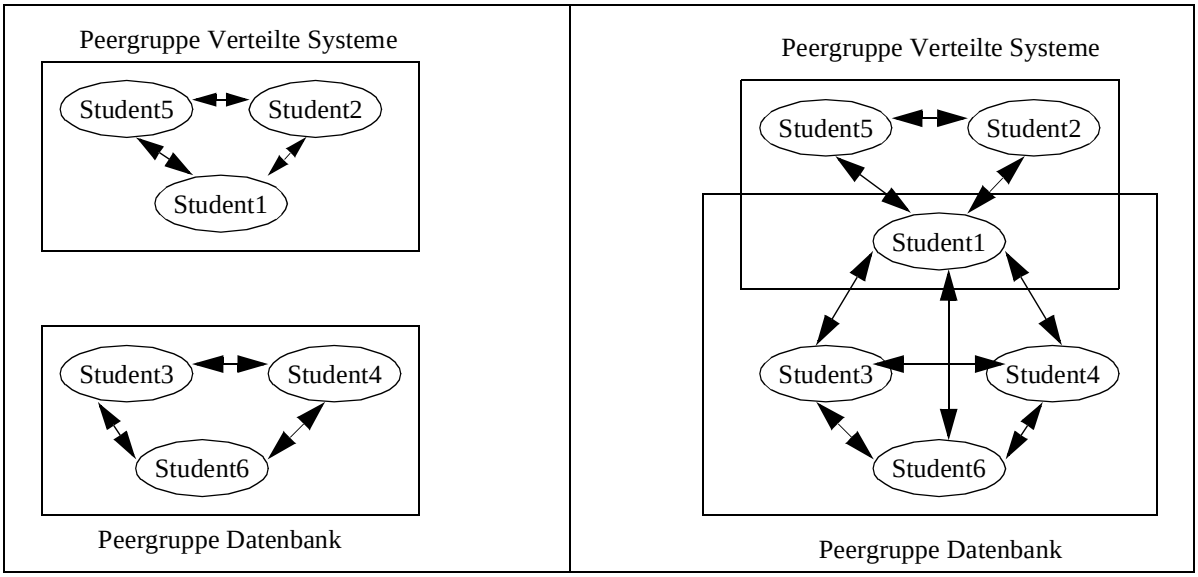

Abbildung 5.12: Verschiedene Kontexte von Peergruppen

- In dem ersten Kontext (links) hat jede Peergruppe/Übungsgruppe eigene Studenten, eigene Teilnehmer. Die Anwendung für die Zusammenarbeit (z.B. Whiteboard, usw.)

wird für die Übungsgruppe angeboten sowie angefragt. Nach dem Empfangen der Anwedung kann die Zusammenarbeit begonnen werden.

- In dem zweiten Kontext (rechts) nimmt der Student 1 an den beiden Übungsgruppen teil. D.h. er kann Daten und Anwendungen von beiden Übungsgruppen anfragen, bekommen und dann zusammen mit den anderen Teilnehmern von jeder Übungsgruppe arbeiten. Alles funktioniert als ob er nur an einer Übungsgruppe teilnehmen würde. In Wirklichkeit laufen auf seinen Bildschirm zwei Whiteboards statt ein Whiteboard mit unterschiedlichen Inhalten.

  Wenn er eine Änderung in der Übungsgruppe Datenbank durchführt, wird die Änderung an alle Teilnehmer von der Datenbank-Gruppe übertragen und umgekehrt bekommt er auch die Änderungen von den anderen in dieser Gruppe. Dies geschieht genau so für die Übungsgruppe Verteilte Systeme.

Das P2P-Framework kann durch die PeerGroupID und die Applikation Sharing ID unterscheiden, welche Änderungen zu welchem Whiteboard gehören.

Die **wichtigen Aktionen** sind Erzeugen (create), Teilnehmen/Anmelden (join), Abmelden/Verlassen (leave), Anbieten (offer) und Anfragen (access/request). Im folgenden Abschnitt werden diese Aktionen genau erklärt.

- **Erzeugen**: Eine Peergruppe wird von einem Peer erzeugt. Zum Zeitpunkt des Erzeugens der Peergruppe wird das Propagate Pipe Advertisement veröffenlicht (published), damit andere Peers es auffinden und benutzen können.
- **Teilnehmen/Anmelden:** Hat ein Peer eine Peergruppe gefunden, kann er an der Peergruppe teilnehmen. Zum Zeitpunkt seines Anmeldens bei dieser Gruppe wird die Kommunikation gestartet. D.h. der Peer kann Nachrichten an die anderen Peers in der Peergruppe senden und Nachrichten von den anderen Peers empfangen. Durch die Anmeldung wird sein Name in der Peergruppe bekannt gemacht und in die Mitgliedliste bei jedem Peer eingetragen. Er kann die angebotenen Daten und Anwendungen von der Peergruppe anfordern. Er kann auch Daten und Anwendungen für diese Peergruppe anbieten.
- **Abmelden/Verlassen:** Ein Peer kann jederzeit die Peergruppe verlassen und dies sollte die anderen Peers nicht beeinträchtigen. Zum Zeitpunkt seines Abmeldens wird er anonym. Die Kommunikation wird abgebaut, sein Name wird aus der Mitgliedliste und aus der Floor Control Liste entfernt. Er kann weiter an den anderen Peergruppen teilnehmen.

- **Anbieten:** Ein Peer kann Services (Applikation Sharing, Daten und Anwendungen) für die Peergruppe anbieten. Die Namen von den angebotenen Services werden in die Listen bei jedem Peer eingetragen. Beim Anbieten von Applikation Sharing muss er die Optionen (Moderator, DataScroll und DataChange), die in Kapitel 2.1.3 erklärt wurden, auswählen.
- **Anfragen:** Ein Peer kann die angebotenen Services anfragen. Wenn er die Anwendung für die Zusammenarbeit anfordert, wird die Anwendung vom P2P-Framework gestartet. Die Daten für die gemeinsame Nutzung wie z.B. Folien der Lehrinhalte oder Dokumente werden an alle Peers übertragen.

5.2.3.2 Anfordern eines Service

Fragt ein Peer einen Service an, wird der Ablauf des Empfangens eines angebotenen Service wie folgt beschrieben (siehe Abb. 5.13).

- Im Beispiel sendet Peer 4 die Serviceanfrage an alle Peers in der Peergruppe über das Propagate Pipe.
- Peer 1 ignoriert die Anfrage, weil er den Service nicht hat.
- Peer 2 und Peer 3 senden Antworten auf die Anfrage über das Propagate Pipe, dass sie den Service haben.

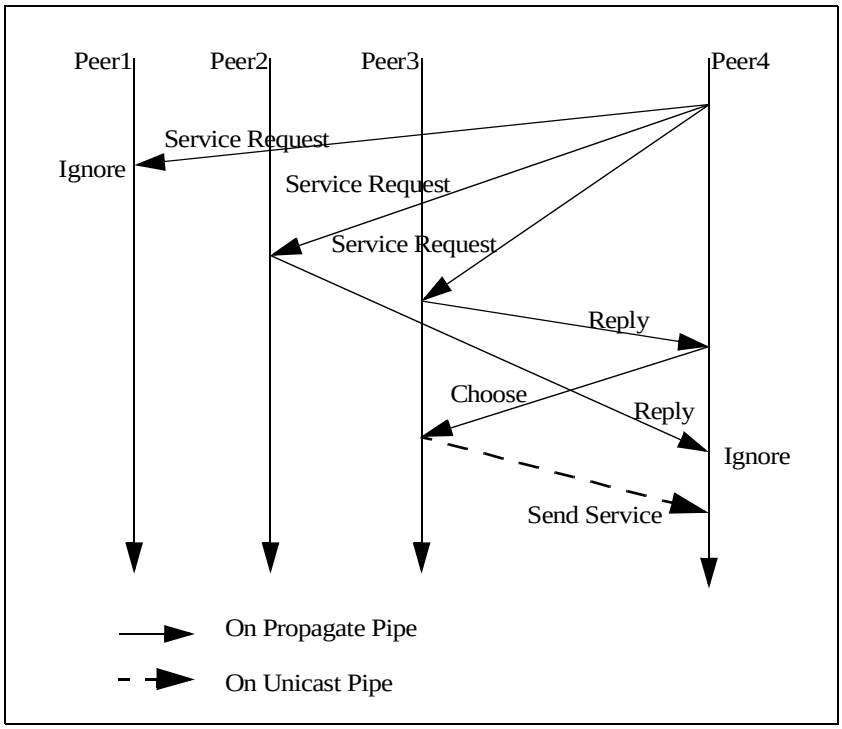

Abbildung 5.13: Ablauf des Anforderns eines Service

- Peer 4 wählt den ersten Peer in der Antwortliste aus, er ignoriert die anderen Antworten. Er sendet eine Bestätigungsnachricht an den ausgewählten Peer über das Propagate Pipe, dass er ihn ausgewählt hat. In diesem Fall ist der Peer 3 der ausgewählten Peer.
- Empfängt Peer 3 die Bestätigungsnachricht von Peer 4, sendet er den Service an den Peer 4 über das Unicast Pipe.

5.2.3.3 Teilnehmerverwaltung

Die Teilnehmerverwaltung setzt eine zuverlässige Kommunikation voraus.

Die Teilnehmerverwaltung ist in dem P2P-Framework verteilt zu konzipieren. Es gibt keinen zentralen Server, der die Informationen aller Teilnehmer in der Peergruppe verwaltet. Jeder Peer muss Teilnehmerinformationen selbst verwalten. Die Frage ist, wie ein Peer die Anwesenheit bzw. Abwesenheit von anderen Peers erkennt und wie ein Peer, der an der Peergruppe später teilnimmt, die vollständigen Peergruppeninformationen (z.B. Peername, PeerID, die angebotenen Services usw.) erhält.

- **Erkennen der Anwesenheit:** Wenn ein Peer an der Peergruppe teilnimmt, sendet er eine Anmeldungsnachricht an alle Peers, sagt sozusagen, dass er ein neues Mitglied ist. Wenn ein Peer die Anmeldungsnachricht eines Peers empfängt, trägt er den Namen des Peers in die Mitgliedliste ein.
- **Erkennen der Abwesenheit:** Wenn ein Peer die Peergruppe verlassen will, sendet er eine Abmeldungsnachricht an alle Peers, sagt sozusagen, dass er nicht mehr Mitglied der Peergruppe ist. Wenn ein Peer die Abmeldungsnachricht eines Peers empfängt, entfernt er den Namen des Peers von der Mitgliedliste sowie von der Floorliste. Die Verbindung des abgemeldeten Peers zur Peergruppe wird beendet.
- **Bekommen der vollständigen Peergruppeninformationen**: Zum Zeitpunkt der Teilnahme an der Peergruppe wird die Kommunikation gestartet. Ein Peer kann daher Anmeldungsnachrichten senden und empfangen. D.h. alle Anmeldungsnachrichten, die gesendet wurden bevor er an der Peergruppe teilgenommen hat, kann er nicht empfangen. Seine Mitgliedliste ist nicht vollständig, da diese Liste nicht alle Namen der Teilnehmer der Peergruppe enthält. Deshalb gibt es einen Mechanismus, der zum Anfragen der vollständigen Peergruppeninformationen (siehe Abb. 5.14) benötigt wird.

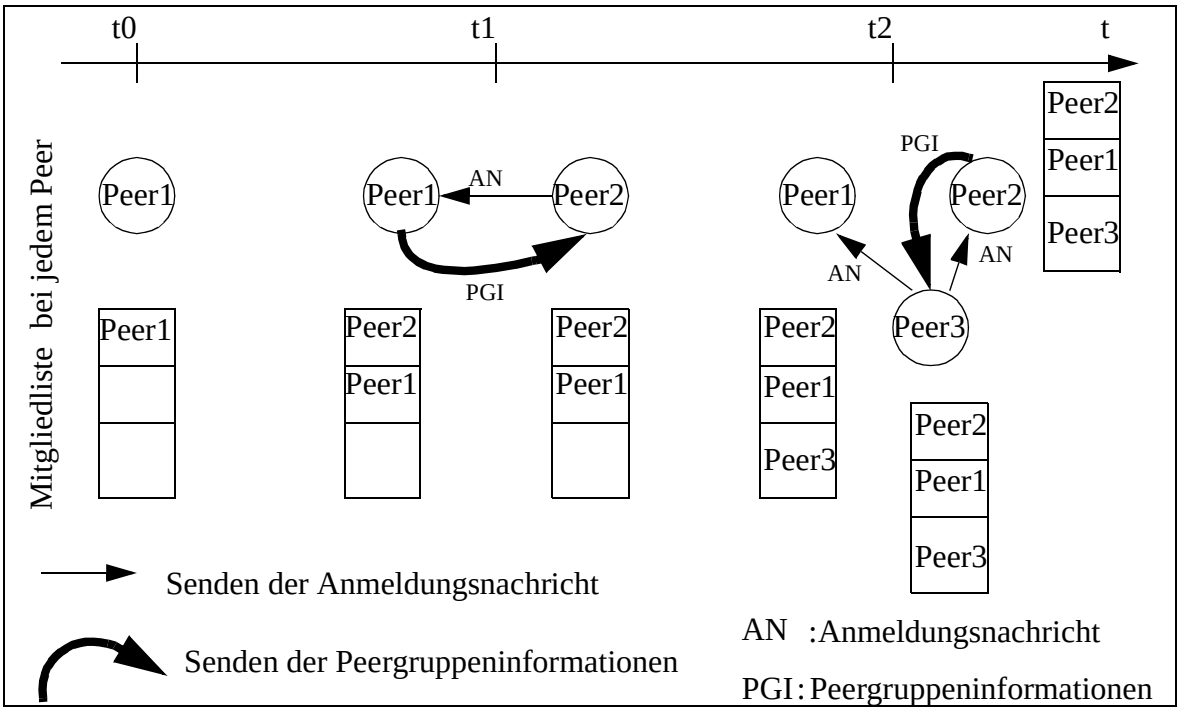

Abbildung 5.14: Mechanismus für das Empfangen der Peergruppeninformationen

Der Anlegende wird automatisch das erste Mitglied der Peergruppe.

Wenn ein Peer eine Anmeldungsnachricht eines anderen Peers empfängt, weiss er, dass der Peer ein neues Mitglied ist. Er überprüft seine Mitgliedliste, ob sie leer ist. Wenn nein, trägt den Namen des Peers in die Mitgliedliste ein und sendet er die Mitgliedliste über Unicast an den Peer. Die Mitgliedliste wird nach der PeerID sortiert.

Um das Netzwerk zu entlasten, senden nicht alle Peers mit nicht leeren Mitgliedlisten die Peergruppeninformationen an den Peer, sondern nur ein Peer. Das ist die Point-to-Point-Verbindung. Wer der erste in der Mitgliedliste ist, darf die Peergruppeninformationen an den neuen Peer senden. Falls der erste in der Midgliedliste der neue Peer ist, darf der nächste in der Mitgliedliste die Informationen an den Peer senden.

Empfängt ein Peer seine Anmeldungsnachricht selbst, ignoriert er sie.

Empfängt ein Peer beim Beitreten zur Peergruppe nach einer gewissen Zeit keine Peergruppens informationen, z.B. wenn der Peer, der für das Senden der Peergruppeninformationen zuständig ist, ausfällt, fragt der neue Peer nach der Peergruppeninformationen per Multicast. Der nächste Peer in der Liste wird das Senden der Peergruppensinformation an den neuen Peer übernehmen. Der Name des ausgefallen Peers wird bei jedem Peer aus der Mitgliedliste entfernt.

Zum Zeitpunkt t0 in der Abbildung 5.14 ist Peer1, der die Peergruppe erzeugt hat, der erste Teilnehmer in der Peergruppe.

Zum Zeitpunkt t1 sendet Peer2 eine Anmeldungsnachricht an alle Peers. Peer1 empfängt diese Nachricht und weiss, das Peer2 ein neues Mitglied ist. Er sendet die Peergruppeninformationen an den Peer2 und trägt den Namen des Peer2 in die Mitgliedliste ein. Besitzt Peer2 eine größere PeerID als Peer1, wird Peer2 vor Peer1 eingetragen.

Zum Zeitpunkt t2 sendet Peer3 eine Anmeldungsnachricht an alle Peers. Peer1 und Peer2 empfangen diese Nachricht und wissen, dass Peer3 ein neues Mitglied ist. Aber nur Peer2 sendet die Peergruppeninformationen an den Peer3, da Peer2 der erste in der Mitgliedliste ist.

Erhaltung der Konsistenz der Mitgliedlisten

Die Konsistenz der Mitgliedlisten wird benötigt für zwei kritische Fälle:

- Der Moderator-Peer fällt aus.
- Floor Control ohne Moderator

Für die beiden Fälle wird ein Abstimmungsverfahren (siehe Kapitel 5.2.4.1) verwendet. Die Konsistenz der Mitgliedlisten wird wiederum für das Abstimmungsverfahren benötigt. Ein Mechanismus wird für die Erhaltung der Datenkonsistenz benötigt.

Der erste Ansatz wäre, dass jeder Peer nach dem Teilnehmen an der Peergruppe periodisch eine Kontrollnachricht sendet. Wenn jeder Peer die Kontrollnachricht durch Multicast-Kommunikation an alle Peers senden würde, würde das Netzwerk belastet, da dies eine n:m-Kommunikation ist. Dadurch könnte die Kommunikation zwischen Peers für das Applikation Sharing beeinträchtigt werden. Um die Belastung des P2P-Netzwerks durch das ständige Senden der Kontrollnachricht an alle Peers zu vermeiden, könnte ein **Peer-Ring** für das Senden der Kontrollnachricht verwendet werden.

Das Prinzip des Sendens beim Peer-Ring ist eine Point-to-Point-Verbindung. Dies wird in Anhang genauer beschrieben.

Der Peer-Ring wird für eine unzuverlässige Kommunikation verwendet, um die Konsistenz der Mitgliedlisten zu erhalten. Wenn die Floor Control eine zuverlässige Kommunikation voraussetzt, wird der Peer-Ring nicht benötigt. Außerdem wird der Peer-Ring für Sitzungen mit Moderator nicht benötigt. Deshalb wird der Peer-Ring in dieser Arbeit nicht implementiert.

5.2.4 Data/Applikation Sharing

Nachdem die Aktionen wie Erzeugen, Teilnehmen, Anbieten und Anfragen durchgeführt werden, kommt die eigenliche gemeinsame Nutzung der Anwendung. Die Komponenten der Module Data/Application Sharing sind: Floor Control, Data und Function Synchronisation, Data/ Application Ownership, Aufzeichnungsdatenbank, Remote Database, und GUI. Die Floor Control wird im Abschnitt 5.2.4.1 im Detail erklärt.

Data Synchronisation ist für die Erhaltung der Daten-Konsistenz zuständig. Sie muss so synchronisieren, dass die Reihefolgen von Operationen von mehreren Peers richtig ausgeführt werden und damit die Daten konsistent sind.

Wenn beim Applikation Sharing mit Floor Control zusammengearbeitet wird, sind die Daten sicher konsistent, da nur der Peer, der den Floor besitzt, die Daten bearbeiten bzw. ändern darf. Dadurch werden die Reihenfolgen von Operationen bei jedem Peer richtig ausgeführt.

Funktion Synchronisation ist für die Übertragung von z.B. Annotation, Bild oder Video zustädig. Für diese Synchronisation genügt die einmalige Übertragung an alle Peers ohne Bestätigung. Es ist unzuverlässige Übertragung aber effizient.

Data/Application Ownership ist für das Setzen der Zugriffrechte auf Daten/Applikation für alle Teilnehmer zuständig. Der Daten/Applikation-Besitzer kann entscheiden, ob Daten nur angeschaut oder bearbeitet werden dürfen. Eine lokale Änderung der Daten sollte bei Anschauen und Bearbeiten bei jedem Peer immer möglich sein. Diese Änderung ist unsichtbar für andere Peers.

Remote Database bietet eine Schnittstelle zu einer entfernten Datenbank an, die ein Peer besitzt. Über diese Schnittstelle können andere Peers auf die Datenbank zugreifen.

Das **GUI** in Data/Applikation ermöglicht es, dass Teilnehmer in der Floor Control Liste aufgelistet werden und Zugriffrechte auf Data/Applikation für einzelnen Teilnehmer zur Laufzeit geändert werden können. Dieses GUI wird im Kapitel 7 genauer beschrieben.

5.2.4.1 Floor Control

Die Floor Control setzt eine zuverlässige Kommunikation voraus.

Die Floor Control ist im P2P-System verteilt zu konzipieren. Es existiert kein zentraler Server für die Floor Control. Jeder Peer verwaltet selbst die Informationen über die Floor Control.

Floor Control ist zur Laufzeit für die Steuerung der Zugriffrechte auf Daten/Applikation für die einzelnen Teilnehmer zuständig. Floor Control ist die Überwachung des Rede- bzw. Schreibrechts. Es gibt für diese Architektur zwei unterschiedliche Szenarien, auf welche Weise über die Vergabe des Rederechts entschieden wird (siehe [OBER 01]):

1. *Zentrale Vergabe*: Wenn die Sitzung durch einen Vorsitzenden (z.B. ein Professor oder ein Dozent/in) geleitet wird, hat er die alleinige Kontrolle über die Vergabe des Rederechts. Er entscheidet, ob ein Teilnehmer, der das Rederecht beantragt, dieses erhält oder nicht. Er kann einem Teilnehmer das Rederecht auch wieder entziehen.
   Beispiel: Vorlesung mit Professor und Studenten.
2. *Dezentrale Vergabe*: Wenn es keinen expliziten Vorsitzenden gibt, sondern alle Teilnehmer gleiche Priorität besitzen, erfolgt die Vergabe des Rederechts nach der Abstimmung (siehe Floor Control ohne Moderator). Der Teilnehmer, der das Rederecht bekommen hat, behält es solange, bis er es wieder freigibt. Dann kann sich der nächste Teilnehmer das Rederecht nehmen.
   Beispiel: Gruppenübung mit Studenten oder Zusammenarbeit bei ad hoc Treffen

In folgendem Abschnitt werden die **Abläufe** der Floor Control mit und ohne Moderator genauer erklärt:

Floor Control mit Moderator:

Wenn ein Teilnehmer das Applikation Sharing anbietet, muss er die Optionen (erwähnt im Kapitel 2.1.3) auswählen. Wenn er die "Moderator" Option auswählt, ist er der Moderator. Der Moderator kann jederzeit den Floor entziehen und entweder sich selbst oder einer anderen Person geben. Der Floor wird entweder von einem Teilnehmer freigegeben oder vom Moderator entzogen. Jeder Teilnehmer kann jederzeit durch die Listen beobachten, wer gerade den Floor besitzt und wer Floor-Kandidat ist. Der Ablauf der Floor Control mit Moderator wird wie folgt beschrieben:

1. Ein Teilnehmer sendet eine Floor-Anfrage an alle Teilnehmer in der Peergruppe. Empfängt ein Teilnehmer die Floor-Anfragennachricht, trägt er den Namen des Beantragers in die Floor-Kandidatenliste ein.
2. Empfängt **der Moderator** die Floor-Anfragennachricht eines Teilnehmers, trägt er den Namen des Teilnehmers in die Floor-Kandidatenliste ein und sendet eine **Wartenachricht** an den Teilnehmer zurück.
3. Der Moderator kann in seine Floor Kandidatenliste schauen und entscheiden, wem er

den Floor gibt. Dann sendet er die Floor-Erlaubnisnachricht an alle Teilnehmers.

4. Jeder Teilnehmer, der die Floor-Erlaubnisnachricht empfängt, entfernt den Namen des Peers, der den Floor erhält, aus der Floor-Kandidatenliste und trägt diesen Namen in die Floor-Besitzerliste ein. Außerdem überprüft er, ob diese Nachricht für ihn ist. Wenn ja, muss er die Nachricht, dass er den Floor besitzt, der Anwendung mitteilen, damit sie aktiviert wird.
5. Der Moderator kann jederzeit den Floor entziehen und entweder sich selbst oder einer anderen Teilnehmer geben.
6. Wenn ein Teilnehmer eine Floor-Freigaben- oder Floor-Entziehennachricht empfängt, entfernt er den Namen des Peers aus der Floor-Besitzerliste und überprüft, ob er selbst der getroffene Teilnehmer ist, wenn ja, muss er die Nachricht der Anwendung mitteilen, dass er keinen Floor mehr hat, damit die Anwendung deaktiviert ist.
7. Wenn die Floor-Besitzerliste leer ist, kann der Moderator einen von den Floorkandidaten auswählen und ihm den Floor geben.

Erkennen der Abwesenheit des Moderators

Es gibt zwei Fälle zu betrachten:

1. Fall: Der Moderator will sich abmelden. In diesem Fall entscheidet er sich, wer sein Nachfolger ist. Er gibt seinem Nachfolger die Moderatorrolle, bevor er das Framework beendet. Der neue Moderator kann die Sitzung weiter leiten. Z.B. weiss der Professor, dass er die Vorlesung nicht bis zum Ende führen kann, deshalb kommt sein Assistent auch in die Vorlesung und nimmt an der Peegruppe teil. Wenn der Professor gehen muss, übergibt er seinem Assistenten die Moderatorrolle. Der Assistent steuert die Vorlesung weiter.

In diesem Fall wird die Abwesenheit des Moderators die Zusammenarbeit der Peergruppe nicht beeinträchtigen.

2. Fall: Der Moderator-Peer stürzt ab. Empfängt der beantragende Teilnehmer nach einer gewissen Zeit keine Wartenachricht vom Moderator, geht er davon aus, dass der Moderator nicht mehr zur Verfügung steht. Dann sendet er eine Abwesenheitsnachricht an alle Teilnehme, um einen neuen Moderator auswählen zu können.

Für die Auswahl eines neuen Moderators wird das Abstimmungsverfahren verwendet. Dieses Verfahren entspricht dem Abstimmungsverfahren, das für die Floor Control ohne Moderator verwendet und im folgendem Abschnitt beschrieben wird.

Falls der Moderator immer noch da ist und die Abwesenheitnachricht auch empfängt, sendet er sofort eine Anwesenheitbestätigungsnachricht an alle Peers, damit die Abstimmung nicht begonnen werden kann oder abgebrochen wird.

Floor Control ohne Moderator:

In diesem Fall gibt es keinen Moderator für die Rechtevergabe, d.h. jeder Teilnehmer muss selbst entscheiden oder wissen, wem wann er den Floor gibt. Das Abstimmungsverfahren wird für die Rechtevergabe verwendet.

Die **relative Mehrheit** wird für die Abstimmungsberechnung benutzt. D.h. nachdem die Floor-Kandidaten die Stimmen von anderen Peers erhalten haben, werden sie die Anzahl der Stimmen miteinander vergleichen. Der Floor-Kandidat mit den meisten Stimmen nimmt den Floor.

Falls die Anzahl der Stimmen mehrerer Floor-Kandidaten gleich sind, wird der Kandidat mit der größten PeerID den Floor nehmen.

Eine Stimme hat drei mögliche Werte: **zustimmend**, **ablehnend** und **ungültig**.

Floor-Anträge an sich sind unfristet, solange keine Mehrheit dafür oder dagegen simmmt, steht er noch da für die weitere Abstimmung. Ein Floor-Antrag kann vom Antragssteller zurückgenommen werden.

Der Ablauf der Floor Control ohne Moderator wird wie folgt beschrieben:

1. Will ein Teilnehmer den Floor haben, sendet er die Floor-Anfragenachricht an alle Teilnehmers.
2. Empfängt ein Teilnehmer eine Floor-Anfragenachricht, trägt er den Namen des Teilnehmers in die Floor-Kandidatenliste ein. Besitzt der Teilnehmer gerade den Floor, sendet er an den Antragssteller eine Antwortnachricht.
3. Ein Teilnehmer reagiert auf eine Abstimmung, indem er seine Stimme einem Floor-Kandidaten vergibt. Er sendet eine Zustimmungsnachricht an den Teilnehmer und eine Ablehnungsnachricht an alle anderen Teilnehmer.
4. Reagiert ein Teilnehmer nach einer gewissen Zeit nicht auf die Abstimmung, wird seine Stimme als ungültig bewertet.
5. Die Stimmvergabe funktioniert nur einstufig, d.h. ein Teilnehmer kann nur seine eingene Stimme weitergeben, nicht aber die Stimmen, die er von anderen Teilnehmern erhalten hat.

6. Erhält ein Floor-Kandidat mehr Stimmen als die anderen Kandidaten, nimmt er automatisch den Floor ein. Dann sendet er die Floor-Besitzernachricht an alle Teilnehmer. Er teilt der Anwendung auch mit, dass er den Floor besitzt, damit die Anwendung aktiv wird.
7. Empfängt ein Teilnehmer eine Floor-Besitzernachricht, entfernt er den Namen des Teilnehmer aus der Floor-Kandidatenliste und trägt diesen Namen in die Floor-Besitzerliste ein.
8. Gibt ein Teilnehmer den Floor frei, sendet er eine Floor-Freigabenachricht an alle Teilnehmer. Er teilt der Anwendung auch mit, dass er den Floor nicht mehr besitzt, damit die Anwendung nicht mehr aktiv wird.
9. Empfängt ein Teilnehmer eine Floor-Freigabenachricht, entfernt er den Namen des Teilnehmers aus der Floor-Besitzerliste. Ist die Floor-Besitzerliste leer ist, vergibt der Teilnehmer seine Stimme einem anderen Teilnehmer.

Erkennen des Ausfalls eines Peers, der den Floor gerade besitzt

Empfängt ein Floor-Kandidat nach einer gewissen Zeit keine Antwort vom Floor-Besitzer, geht er davon aus, dass der Peer nicht mehr zur Verfügung steht. Dann sendet er eine Abwesenheitnachricht des Peers an alle Peers, um eine neue Abstimmung beginnen zu können.

Empfängt ein Peer eine Abwesenheitnachricht, macht er genauso wie er eine Floor-Freigabennachricht empfängt (siehe oben Schritt 9).

Falls der Floor-Besitzer immer noch da ist und die Abwesenheitnachricht auch empfängt, muss er verhalten wie er den Floor freigibt bis auf das Senden der Floor-Freigabennachricht. D.h. er entfernt seinen Namen aus der Floor-Besitzerliste und teilt der Anwendung mit, dass er den Floor nicht mehr hat. Er kann wieder kandidieren.

Die Mitgliedlisten werden durch die Floor-Nachrichten auch aktuallisiert. D.h. empfängt ein Peer eine Floor-Anfrage, fehlt der Namen des Peers der Mitgliedliste, trägt der Peer diesen Namen in die Mitgliedliste ein. Und umgekehrt, empfängt ein Peer eine Abwesenheitnachricht, entfernt er den Namen des Peers aus der Mitgliedliste.

5.2.4.2 Lokale Aufzeichungsdatenbank

Der Vorschlag für die lokale Aufzeichnungsdatenbank für das P2P-Framework basiert auf der bereits abgeschlossenen Studienarbeit [SCHMID 01]. Dabei wurden die Module bei der Client-

seite genau erklärt

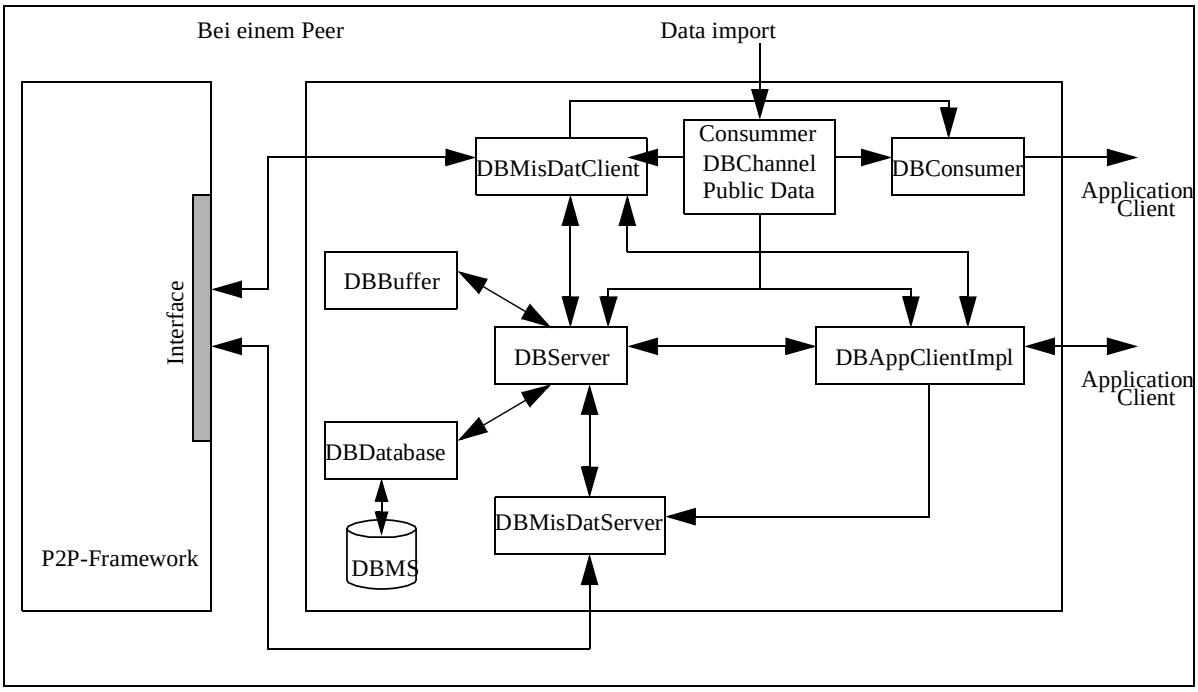

Abbildung 5.15: Lokale Aufzeichungsdatenbank

Die Funktionsweise dieser Module werden im folgenden Abschnitt zusammenfassend erläutert:

1. Das Objekt **DBAppClientImpl** repräsentiert die Schnittstelle der Datenbank nach außen. Über sie können Daten der Anwendung, die die Datenbank verwendet, gespeichert und Daten abgefragt werden.
2. Das Objekt **DBConsumer** wird von der Anwendung implementiert und beim DBAppClientImpl-Objekt registriert. Es dient dazu, die Anwendung über Ereignisse in der Datenbank zu informieren und entsprechende Reaktionen bei der Anwendung auszulösen.
3. Zu den Aufgaben des **DBChannelConsumerPublicData**-Objekts gehören das Speichern eines angekommenen Datensatzes und die Benachrichtigung der Datenankunft für den Application Client. Weil die Kommunikation *Channel* im P2P-Framework nicht existiert, muss dieses Objekt so umgeschrieben werden, dass es Daten über Import von Data Sharing bekommt. D.h. die komplette Aufzeichnungsdaten sollte von der Aufzeichnungsdatenbank in eine Datei geschrieben werden. Durch das Anbieten der Aufzeichnungsdaten kann ein Peer, der nicht in der Sitzung war, diese Daten anfordern. Dann kann die angeforderte Aufzeichnungsdatei in der Aufzeichnungsdatenbank bei dem Peer importiert werden.

4. Das Objekt **DBServer** übernimmt die Verwaltung der Daten. Es ist sozusagen die interne Ansprechstelle für die Zugriffe auf die protokollierten Daten.
5. Der DBServer verwaltet die Objekte **DBBuffer** und **DBDatabase** und vermittelt zwischen ihnen. Der DBBuffer hält die übergebenen Daten im Hauptspeicher und ermöglicht damit eine schnellere Verarbeitung. Für die Persistenz von Daten müssen die Daten auf der Festplatte gespeichert werden. Dazu werden die Daten aus dem DBBuffer entnommen und an die DBDatabase übergeben. Die DBDatabase ist die Schnittstelle zu einem Datenbank-Management-System (**DBMS**), das für die Speicherung und Verwaltung der Daten auf Festplatte eingesetzt wird. Über DBDatabase kann auf das DBMS und damit auf die persistent gespeicherten Daten zugegriffen werden.
6. Die Objekte **DBMisDatSever** und **DBMisDatClient** sind für die Beschaffung fehlender Daten zuständig. DBMisDatClient fordert fehlende Daten beim DBMisDatServer an. Der DBMisDatServer holt die geforderten Daten von seinem DBServer und sendet die Anfrage. Der anfordernde DBMisDatClient übergibt die gelieferten Daten an seinen DBServer.

Bei der Frameworkseite wird ein Interface festgelegt, über dieses Interface kann ein Peer die anderen Peers nach fehlender Daten fragen sowie auf Anfragen anworten. D.h. die Module **DBMisDatClient** und **DBMisDatServer** müssen für die Peer-Kommunikation, die über diese Schnittstelle stattfindet, umgeschrieben werden.

Eine weitere Änderung betrifft die Synchronisation der Daten. Bisher erfolgt die Synchronisation mittels globaler Zeitstempel. Für die Verwendung in einer Peer-to-Peer-Architektur muss eine Lösung mit relativen Zeitangaben entwickelt werden, da keine globale Zeitstempel möglich sind.

5.2.5 Abläufe

Welche Aktionen ein Benutzer vom Start des Programms bis zum Ende benötigt, werden im folgenden regulären Ablauf erklärt (siehe Abb. 5.16). In einer Peergruppe können sowohl Applikation Sharing als auch zusätzliche Daten und Anwendungen, die dem Applikation Sharing dienen, angeboten werden.

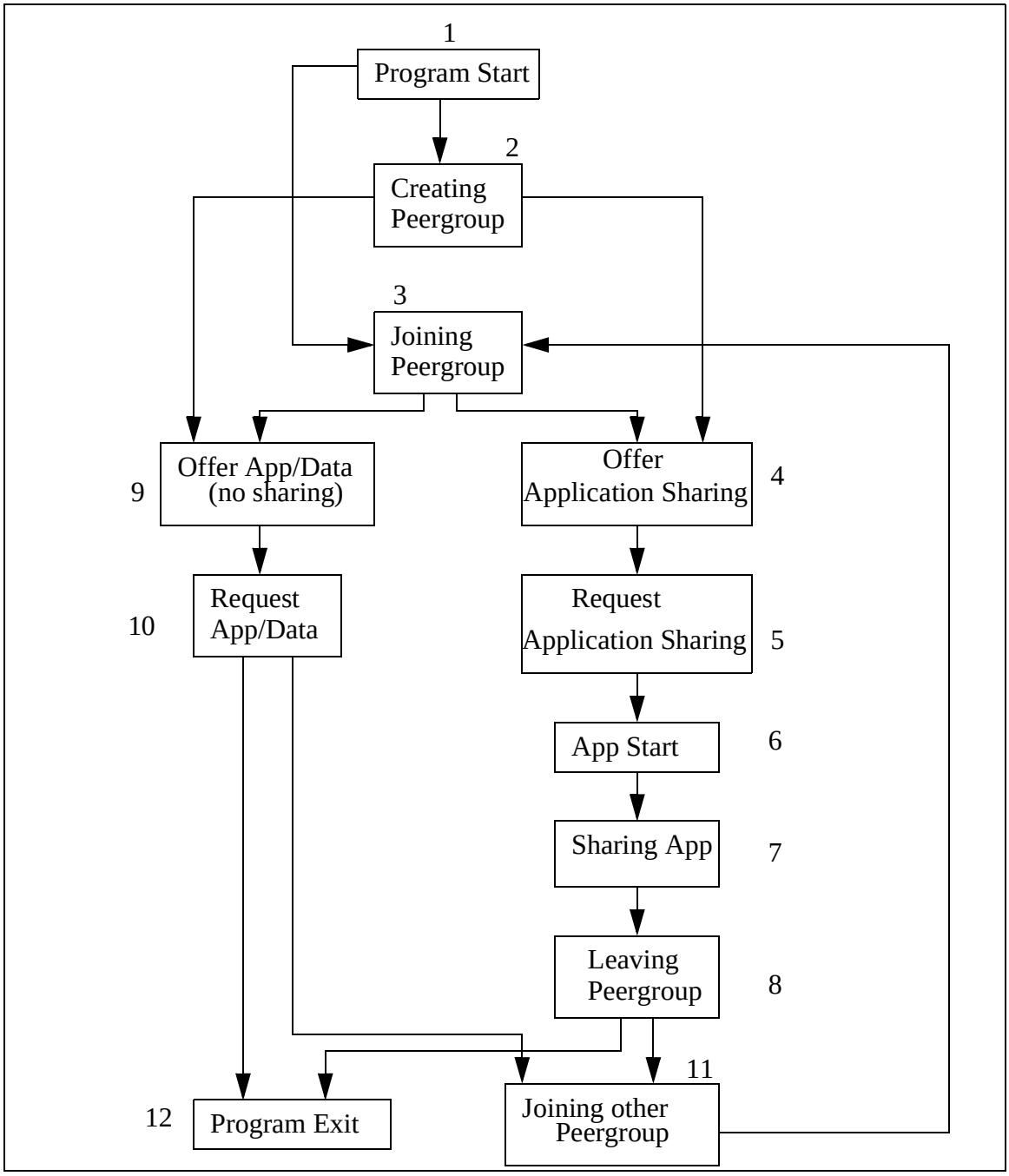

Abbildung 5.16: Regulärer Ablauf

1. Das Programm wird gestartet.

2. Ist noch keine Peergruppe vorhanden, legt ein Benutzer eine Peergruppe an.

3. Ein Benutzer findet eine Peergruppe , nimmt er an der Peergruppe teil.

4. Ein Teilnehmer in der Peergruppe bietet das Applikation Sharing an.

5. Die Teilnehmer in der Peergruppe fordern das angebotene Applikation Sharing an.

6. Nach der Übertragung des Applikation Sharing an die Teilnehmer wird das Applikation Sharing vom Framework gestartet.

7. Die Inhalte für die Zusammenarbeit werden entweder bei der Übertragung der Anwendung mitgesendet oder erst übertragen, wenn ein Teilnehmer eine Datei öffnet. Die Zusammenarbeit wird begonnen.

8. Ein Teilnehmer verläßt die Peergruppe. Durch das Verlassen eines Teilnehmers wird die Zusammenarbeit nicht beeinträchtigt.

9. Ein Teilnehmer in der Peergruppe bietet zusätzliche Daten oder/und Anwendungen an, die dem Applikation Sharing dienen.

10. Ein Teilnehmer fordert die zusätzlichen Daten oder/und Anwendungen an.

11. Ein Teilnehmer nimmt an einer anderen Peergruppe teil (nach dem Empfangen der zusätzlichen Daten oder Anwendungen oder nach dem Verlassen der Peergruppe).

12. Ein Teilnehmer beendet das Programm.

5.2.6 Schnittstellen

Das P2P-Framework und die Anwendung, die für die Zusammenarbeit ist, kommunizieren miteinander über die Framework-Schnittstellen (siehe Abbildung 5.17).

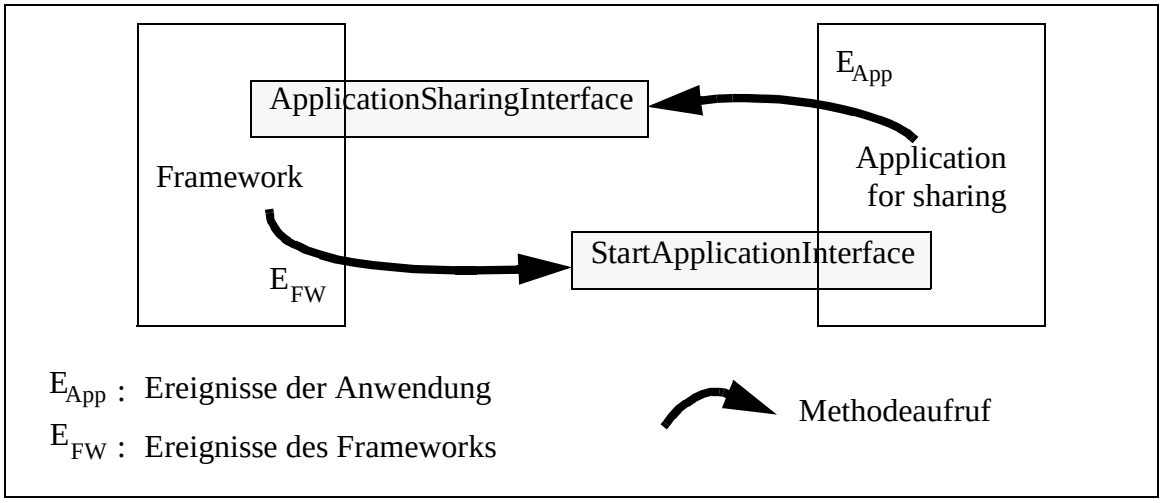

Abbildung 5.17: Schnittstellen zwischen dem P2P-Framework und der Anwendung

Die Schnittstelle **ApplicationSharingInterface** wird vom Framework implementiert und die Schnittstelle **StartApplicationInterface** wird von der Anwendung implementiert. Wenn ein Ereignis beim Framework geschieht (z.B. Floor Control Ereignis, usw.), gibt das Framework dieses Ereignis an die Anwendung über das StartApplicationInterface. Und umgekehrt wenn ein Ereignis bei der Anwendung geschieht (z.B. das Blättern, die Änderungen, usw.), gibt die Anwendung dieses Ereignis an das Framework über das ApplicationSharingInterface.

6 Implementierung

In diesem Kapitel wird die Implementierung der Hauptkomponenten des P2P-Frameworks wie die Kommunikation zwischen Peers, die Sitzungs- und Teilnehmerverwaltung, die Floor Control und die Benutzungsoberfläche beschrieben.

6.1 Kommunikation zwischen Peers in einer Peergruppe

Für die Propagate-Kommunikation wurde die Klasse **PropagatePipeComm** und für die Point-to-Point-Kommunikation wurden die Klassen **SecureUnicastPipeComm** und **SendServiceOnSecureUnicastPipe** implementiert. Das Objekt **SynchronSend** ist für die Synchronisation des Sendens von Nachrichten zuständig.

6.1.1 Propagate-Kommunikation

In der Klasse *PropagatePipeComm* ist die Methode **createIOPipe** für die Erzeugung der **Output**- und **Inputpipe** (siehe 4.3) zuständig. Über das *Outputpipe* werden Nachrichten an alle Peers gesendet und über das *Inputpipe* werden Nachrichten empfangen. Durch das Objekt **DiscoveryPropagatePipeAdvertisement** können andere Peers das *Propagate Pipe Advertisement* empfangen.

Für das Senden von Nachrichten werden die Methoden **sendByteServiceMessage** und **sendStringServiceMessage** verwendet. Um ein Objekt in einen Byte Array und umgekehrt einen Byte Array in ein Objekt umzuwandeln, werden die Methoden **convertObjectIntoByteArray** und **convertByteArrayIntoObject** verwendet.

Für das Empfangen von Nachrichten wird die Methode **pipeMsgEvent** benutzt. Wenn die *pipeMsgEvent* eine Nachricht empfängt, gibt sie die Nachricht weiter an das Objekt **AnalyseMessageOnPropagatePipe**. Dieses Objekt bearbeitet die empfangenen Nachrichten. Für die Bearbeitung der Nachrichten werden die Methoden **checkAndSendAnswerFor**... und **getPathUndSendService** usw. verwendet.

6.1.2 Point-to-Point-Kommunikation

Durch das Objekt **DiscoverySecureUnicastPipeAdvertisement** können andere Peers das *Secure Unicast Pipe Advertisement* finden.

Die Methode **createInputPipe** in der Klasse *SecureUnicastPipeComm* ist für die Erzeugung von *Inputpipe* zuständig. Über das *Inputpipe* kann ein Peer auf Nachrichten hören, die nur für ihn sind. Die empfangenen Nachrichten werden vom Objekt **AnalyseMessageOnUnicastPipe** bearbeitet.

Um Nachrichten von einem Peer an einen anderen Peer zu senden, wird ein *Outputpipe* verwendet. Um ein *Outputpipe* zu erzeugen, wird die Methode **createOutputSUPipe** in der Klasse *SendServiceOnSecureUnicastPipe* verwendet.

6.2 Session Manager

6.2.1 Sitzungsverwaltung

Durch das Objekt **CreatePeerGroup** kann eine Peergruppe angelegt bzw. erzeugt werden. Durch das Objekt **DiscoveryPeerGroup** können andere Peers das *Peer Group Advertisement* empfangen.

Für die Wiederholung der Veröffenlichung von Peergruppen ist der Thread **RepeatPublishingAdvertisements** zuständig.

In einer Peergruppe können sowohl Applikation Sharing als auch zusätzliche Daten und Anwendungen angeboten werden. Alle wichtige Informationen über die Angebotenen wie Name, IDs, Beschreibung usw. werden in Objekte gespeichert. Diese Objekte sind **StoreAppSharingInfo**, **StoreDataInfo** und **StoreAppInfo**. Die Objekte werden mit einer Anfragen oder Antwort mitgesendet und enthalten alle notwendigen Informationen für die Bearbeitung.

6.2.2 Teilnehmerverwaltung

Jeder Peer besitzt eine eigene Teilnehmerverwaltung und führt eine Mitgliedliste. Die Mitgliedliste wird durch das Objekt **CompareMemberMessageInfo** nach der Peer ID sortiert. Alle wichtige Informationen über ein Peer wie Peer Name, Peer ID, PeerGroup ID usw. wer-

den in das Objekt **StorePeerPresentInfo** gespeichert. Dieses Objekt ist in der An- und Abmeldungsnachricht enthalten.

Wenn ein Peer eine Anmeldungsnachricht empfängt, kann er die Informationen über den Peer durch das Objekt *StorePeerPresentInfo* auslesen. Für die Bearbeitung der Nachricht ruf er die Methode **doingForPresentMessage** in der Klasse *PropagatePipeComm* auf.

Wenn ein Peer eine Abmeldungsnachricht empfängt, verwendet er die Methode **doingForAbsenceMessage** in der Klasse *PropagatePipeComm* für die Bearbeitung der Nachricht.

Der Timeout für das Warten auf die Peergruppeninformationen beträgt 30 Sekunden.

6.3 Floor Control

6.3.1 Floor Control mit Moderator

Alle Informationen wie Moderatorzustand, Peer Name, Peer ID, Application Sharing ID usw. werden in das Objekt **StoreFCWithModeratorMessageInfo** gespeichert. Dieses Objekt wird mit Anfragen oder Antworten mitgesendet.

Empfängt ein Peer eine Floor-Anfrage, verwendet er die Methode "**doingForClaimFloorWithModerator**" für die Verarbeitung der Nachricht.

Empfängt ein Peer eine Floor-Erlaubnisnachricht, verwendet er die Methode **doingForYesMessage** für die Verarbeitung der Nachricht. Der Peer, der die Floor hat, ruf die Methode **setFloor** der Schnittstelle *StartApplicationInterface* auf und gibt der Anwendung die Floor (*setFloor(true)*).

Wenn ein Floor-Freigabennachricht oder Floor-Entziehennachricht empfangen wird, wird die Methode **releaseFloorWithModerator** für die Bearbeitung dieser Nachricht benutzt. Die Floor Listen werden durch die Methode **upToDateFCWithModeratorList** aktuallisiert. Der Peer, der die Floor freigegeben hat oder dem die Floor vom Moderator entzogen wurde, ruf die Methode *setFloor(false)* auf. Dadurch hat die Anwendung keine Floor mehr.

Der Timeout für das Erkennen des Ausfalls des Moderators beträgt 1 Minute.

6.3.2 Floor Control ohne Moderator

Alle Informationen wie Moderatorzustand, Peer Name, Peer ID, Application Sharing ID werden in das Objekt **StoreFloorControlMessageInfo** gespeichert. Dieses Objekt wird mit der Floor-Anfragen mitgesendet.

Empfängt ein Peer eine Floor-Anfrage, verwendet er die Methode **doingForClaimFloorNoModerator** für die Verarbeitung der Nachricht.

Durch die Methode **compareVotes** der Klasse *AnalyseMessageOnPropagatePipe* werden die Stimmen der Floor Kandidaten verglichen.

Empfängt ein Peer ein Floor-Freigabennachricht, verwendet er die Methode **releaseFloorNoModerator** für die Bearbeitung dieser Nachricht. Die Floor Listen werden durch die Methode **upToDateFCNoModeratorList** aktuallisiert.

Der Timeout für das Erkennen des Ausfalls eines Peers beträgt 1 Minute.

6.4 Interfaces

6.4.1 StartApplicationInterface

Diese Schnittstelle (siehe Anhang) wird von der Anwendung implementiert. Für den Start der Anwendung wird die Methode **starter()** verwendet. Durch die Methode **setFloor()** holt die Anwendung den Floor und reagiert darauf. Die Anwendung gibt bzw. holt die aktuellen Daten z.B. beim Start durch die Methode **giveActualData()** und **getActualData()**. Die Optionen "DataScroll" und "DataChange" können zur Laufzeit vom Moderator geändert werden. Durch die Methode **getActualDataOptions()** kann die Anwendung die aktuellen Information über die Optionen empfangen und darauf reagieren. Durch die Methode **receiveMessageFromOtherPeers()** kann die Anwendung z.B. die Änderungen oder das Blättern empfangen.

6.4.2 ApplicationSharingInterface

Diese Schnittstelle (siehe Anhang) wird vom P2P-Framework implementiert. Die Schnittstelle bietet die Methode **sendMessageToPeers()** an. Die Anwendung verwendet diese Methode für die Übertragung der Änderungen oder des Blätterns an alle Peers.

6.5 Benutzungsoberfläche

In diesem Kapitel werden die Benutzungsoberflächen für die JXTA-Konfiguration, für die Verwaltung von Peergruppen und für die Floor Control vorgestellt.

6.5.1 JXTA-Konfiguration

JXTA wird beim Start des P2P-Frameworks automatisch gestartet. Dafür wird eine JXTA-Konfiguration benötigt. Im folgenden Abschnitt wird erklärt, wie man JXTA für das Framework konfiguriert, weitere Informationen siehe [JUGEIDE].

In dem "**Basics**" Dialog gibt es zwei Elemente (siehe Abb. 6.1): Bei **Peer Name** muss ein String eingegeben werden. Dieser Name wird für die Teilnehmerverwaltung und für die Floor Control benutzt. Die **Web Proxy-Server** Einstellungen werden für das Framework nicht benötigt.

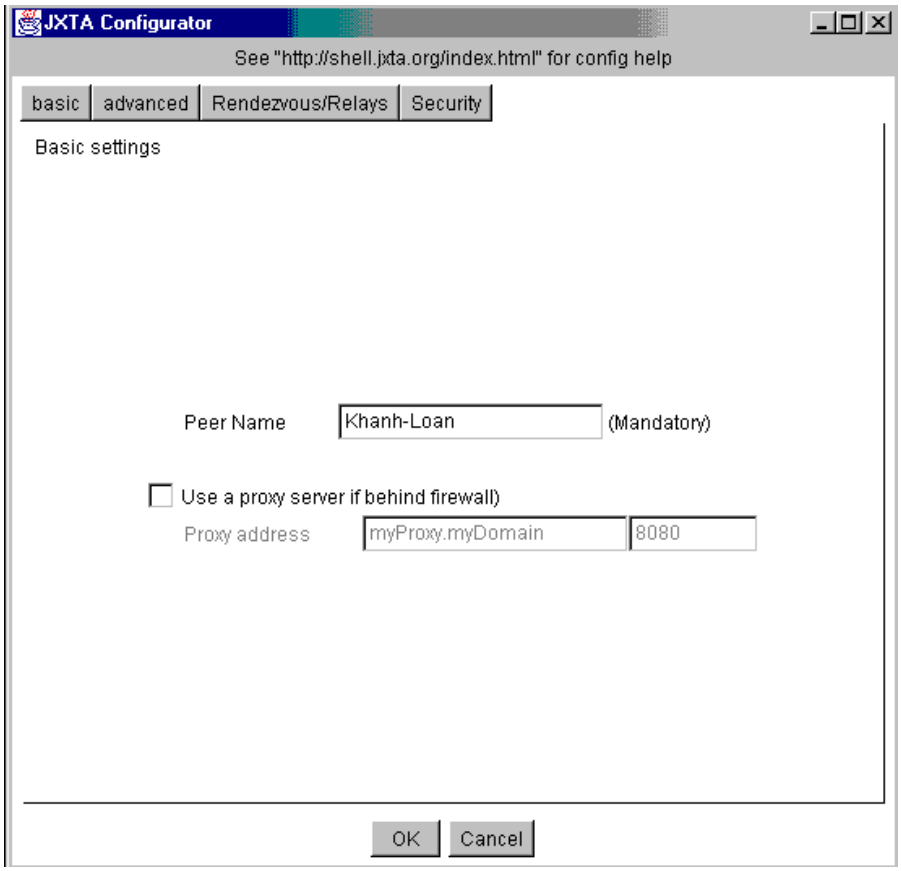

Abbildung 6.1: JXTA-Konfiguration: Basics Dialog

Im "**advanced**" Dialog wird die Option "**Enabled**" für TCP angekreuzt (siehe Abb. 6.2). Für HTTP wird diese Option nicht gesetzt, da dies für das Framework nicht benötigt wird. Wenn zwei Peers auf dem gleichen Rechner ausgeführt werden, müssen verschiedene Portnummern verwendet bzw. gesetzt werden.

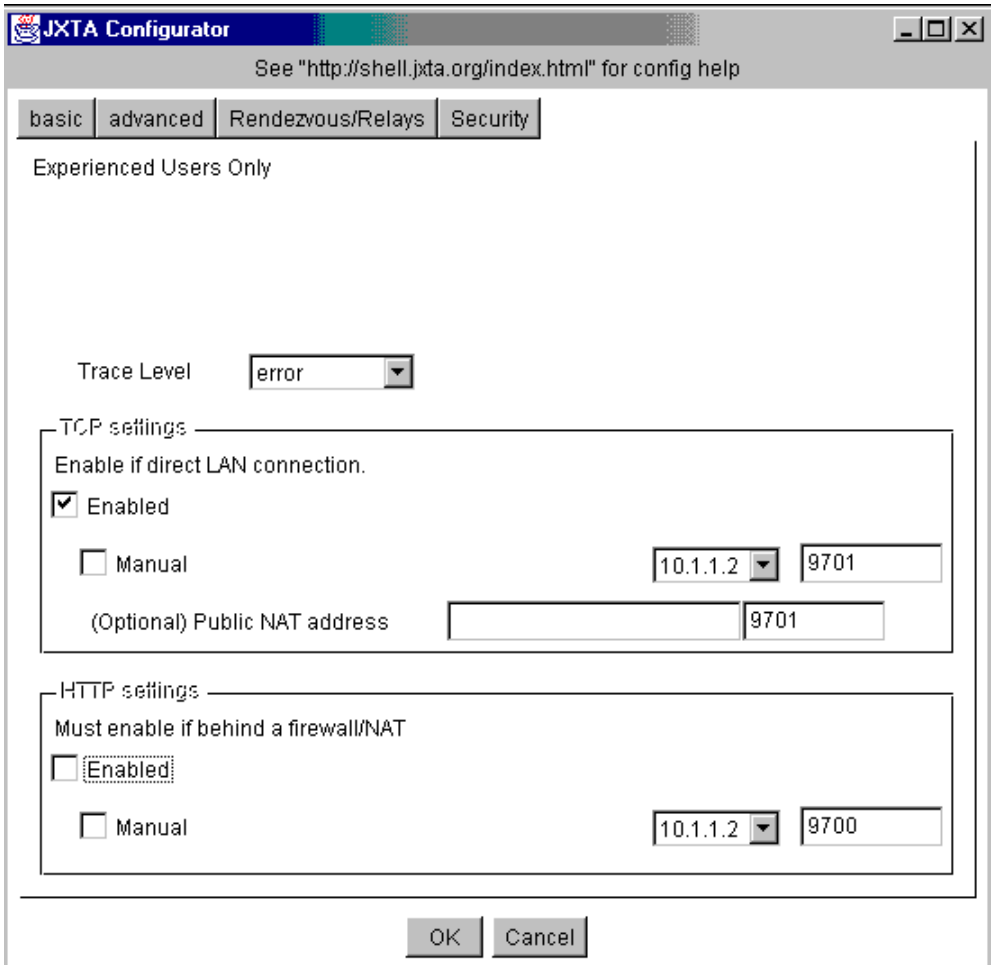

Abbildung 6.2: JXTA-Konfiguration: Advanced Dialog

Die Konfiguration für "**Rendezvous/Relays**" ist nicht nötig, da das Framework den Mechanismus zum Auffinden via Rendezvous-Peer nicht verwendet. Für das P2P-Framework wird der Mechanismus des LAN-basierenden Auffindens verwendet. Ein Peer wird daher nicht als Rendezvouz-Peer konfiguriert.

Im "**Security**" Dialog müssen der Benutzername und das Kenntwort angegeben werden (siehe Abb. 6.3). Diese Informationen werden für die Indentität des Peers benutzt. Wenn alle beschriebenen Einstellungen gemacht wurden, kann der Benutzer durch den OK-Knopf bestätigen.

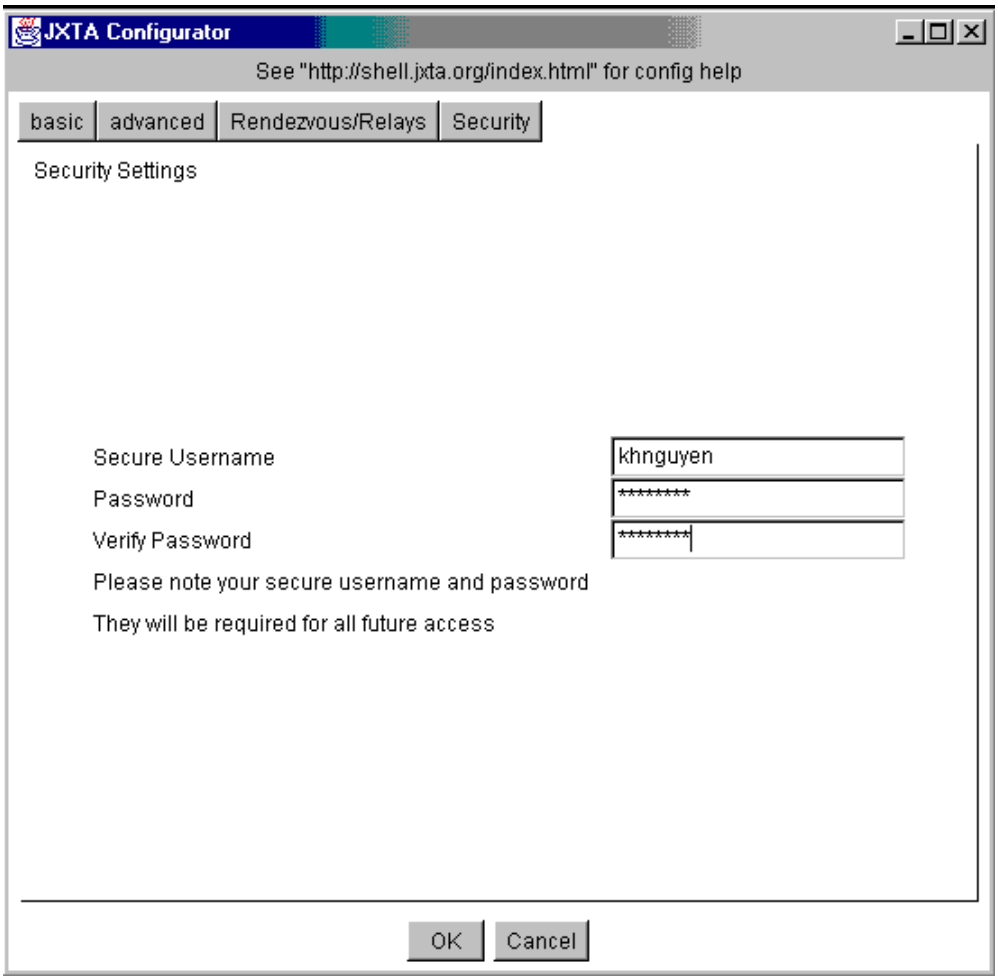

Abbildung 6.3: JXTA-Konfiguration: Security Dialog

6.5.2 Benutzungsoberfläche für Peergruppen/Sitzungen

Nach dem Start des Frameworks erscheint ein Dialog "**Creating, Joining and Leaving Peer Group**" (siehe Abb. 6.4) für

- das Erzeugen einer Peergruppe,
- das Teilnehmen an einer Peergruppe,
- das Anbieten von Applikation Sharing sowie Daten und Anwendungen, die dem Applikation Sharing dient,
- und für das Anzeigen der Mitgliedliste und der angebotenen Services in einer Peergruuppe.

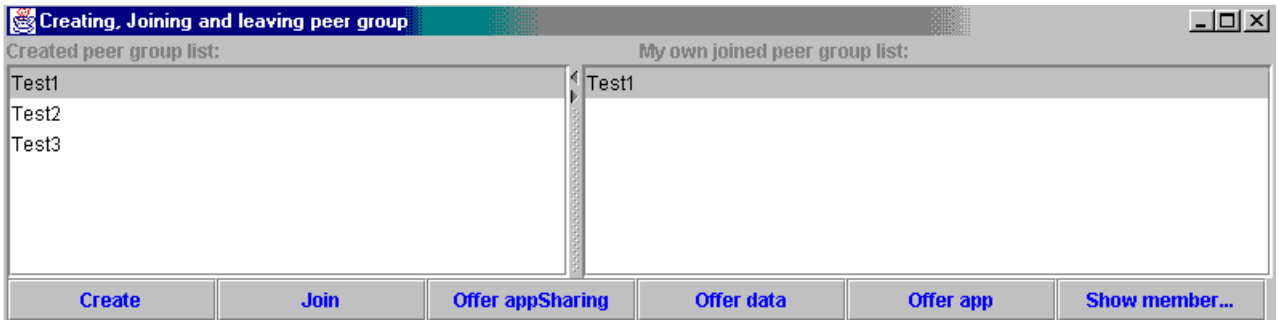

Abbildung 6.4: Erzeugen, Teilnehmen an einer Peergruppe

Links ist die Liste der erzeugten Peergruppen. Rechts ist die Liste der Peergruppen, an denen ein Benutzer teilnimmt.

Wenn ein Benutzer an einer Peergruppe teilnehmen möchte, wählt er den Namen der Peergruppe aus und drückt den Knopf "**Join**".

Wenn ein Teilnehmer das Framework beenden will, schließt er diesen Dialog.

6.5.2.1 Erzeugen einer Peergruppe

Wenn ein Benutzer eine Peergruppe erzeugen möchte, drückt er den Knopf "**Create**" (siehe Abb. 6.4), dann erscheint ein Fenster "**Required informations for creating peer group**" (siehe Abb 6.5). Der Name des Benutzers, der Name und die Beschreibung der Peergruppe müssen angegeben werden. Diese Informationen und die PeerGroupID werden in dem Objekt **StorePeerGroupInfo** gespeichert. Nach dem Anlegen der Peergruppe wird dieses Objekt an alle Peers übertragen. Bei jedem Teilnehmer wird der Name der erzeugten Peergruppe angezeigt.

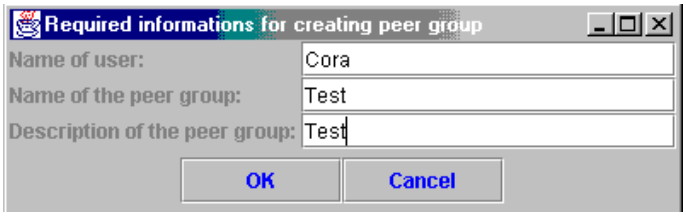

Abbildung 6.5: Die Informationen für das Erzeugen einer Peergruppe

6.5.2.2 Anbieten der Services

Ein Teilnehmer einer Peergruppe kann z.B. das Applikation Sharing anbieten, indem er den Knopf "**Offer AppSharing**" drückt, dann erscheint ein Dateidialog "**Required informations for Application Sharing**" (siehe Abb 6.6).

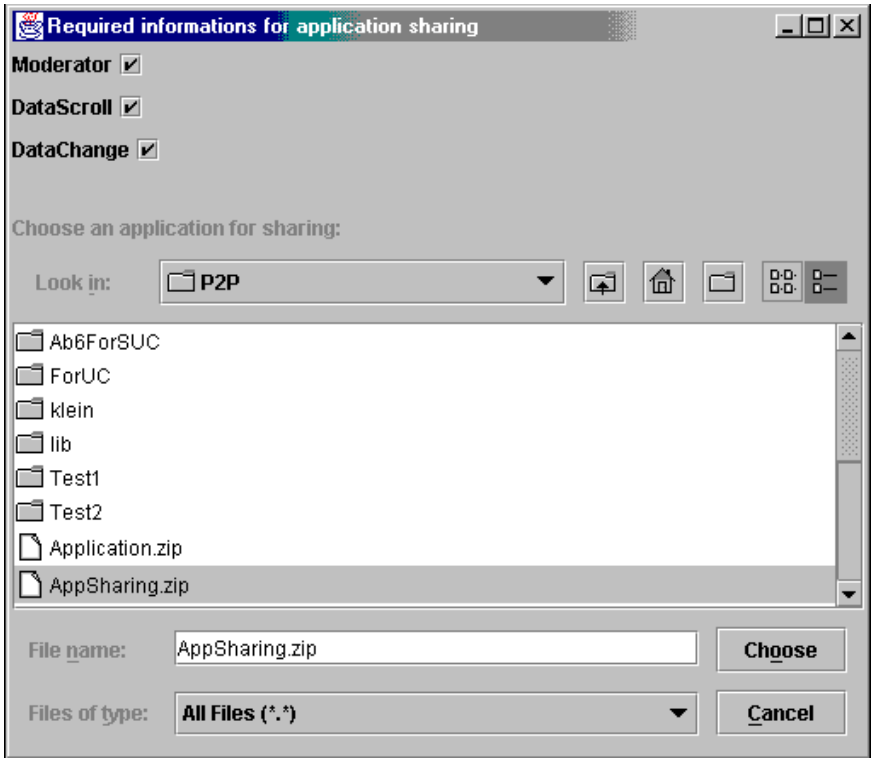

Abbildung 6.6: Das Erzeugen eines Applikation Sharing

Der Anbieter wählt die Optionen (siehe Kapitel 2.1.3) und die Datei, die den Code der Anwendung enthält, aus. Die ausgewählten Optionen, der Name und die ID des Applikation Sharing werden in dem **Objekt StoreAppSharingInfo** gespeichert. Wenn er den Knopf "**Choose**" drückt, wird dieses Objekt an alle Peers übertragen. Der Name der Anwendung wird bei jedem Teilnehmer angezeigt.

Das Anbieten der zusätzlichen Daten und Anwendungen, die dem Applikation Sharing dienen, geschieht so ähnlich wie beim Anbieten eines Applikation Sharing. D.h. ein Teilnehmer den Knopf "**Offer Data**" bzw. "**Offer App**" drückt, erscheint ein Dialog (siehe Abb 6.7). Er kann die Datei auswählen und durch den Knopf "*Choose*" bestätigen. Alle Informationen wie Name, IDs werden in dem Objekt **StoreDataInfo** bzw. **StoreApplicationInfo** gespeichert. Diese Objekte werden an alle Peers übertragen und angezeigt.

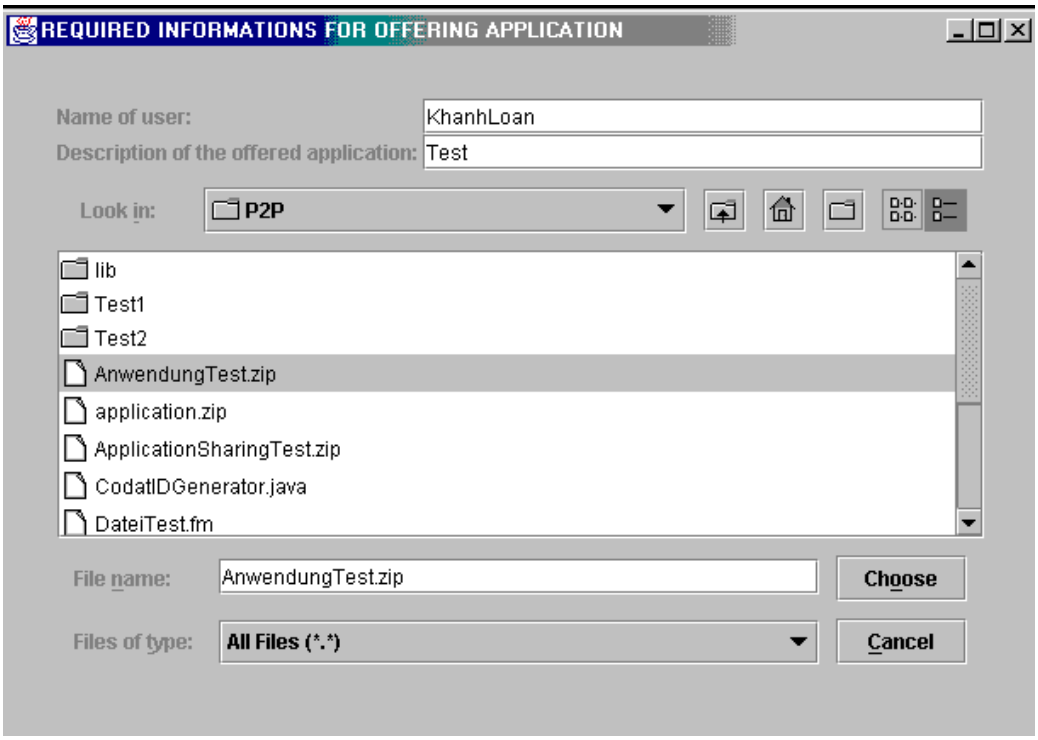

Abbildung 6.7: Das Anbieten der zusätzlichen Daten und Anwendungen

Für den Start der Anwendung für die Zusammenarbeit ist das Objekt **ThreadStartApplicationByFramework** zuständig. Dieses Objekt erzeugt wiederum ein Objekt **ServiceClassLoader**, das die Klassen der Anwedung in die Laufzeitumgebung lädt.

6.5.2.3 Anzeige der Teilnehmerliste und angebotenen Services

Wenn ein Teilnehmer die Teilnehmerliste sowie die angebotenen Services sehen will, drückt er den Knopf "**Show Members/App**" (siehe Abb. 6.4). Dann erscheint ein Dialog (siehe Abb. 6.8), durch den ein anderer Teilnehmer die Service auswählen kann und durch die Knöpfe "**Access AppSharing**" oder "**Access Application**" oder "**Access Data**" anfordern.

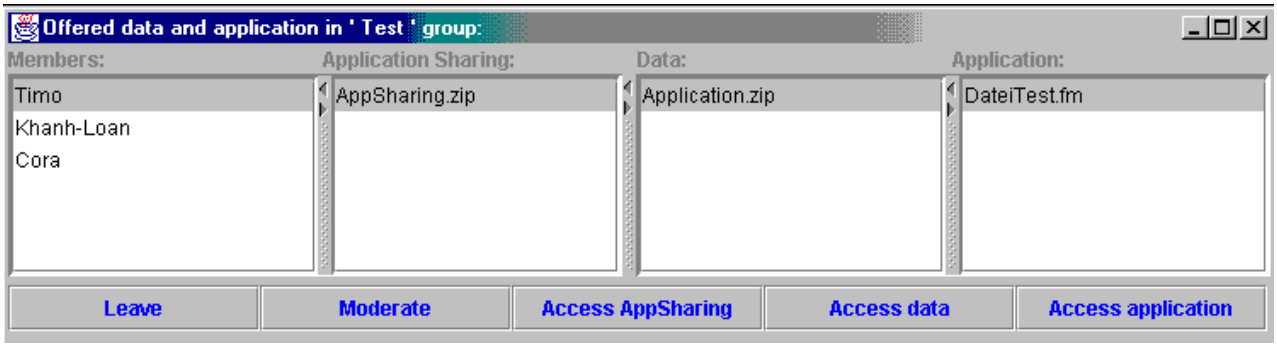

Abbildung 6.8: Die Mitgliedliste und die angebotenen Services in einer Peergruppe

Wenn ein Teilnehmer die Peergruppe verlassen will, kann er "**Leave**" drücken. Sein Name wird von allen Mitglied- und Floor Listen entfernt.

Will der Moderator sich abmelden, kann er seinen Nachfolger auswählen, indem er einen Teilnehmer in der Teilnehmerliste auswählt und den Knopf "**Moderate**" drückt. Bei dem neuen Moderator erscheint der Dialog für den Moderator (siehe 6.4.3.1) und dadurch kann er den Floor geben oder entziehen.

Speichern eines Service

Wenn ein Teilnehmer den Knopf "Access appSharing" z.B. für Applikation Sharing drückt, erscheint ein Dialog (siehe Abb. 6.9).

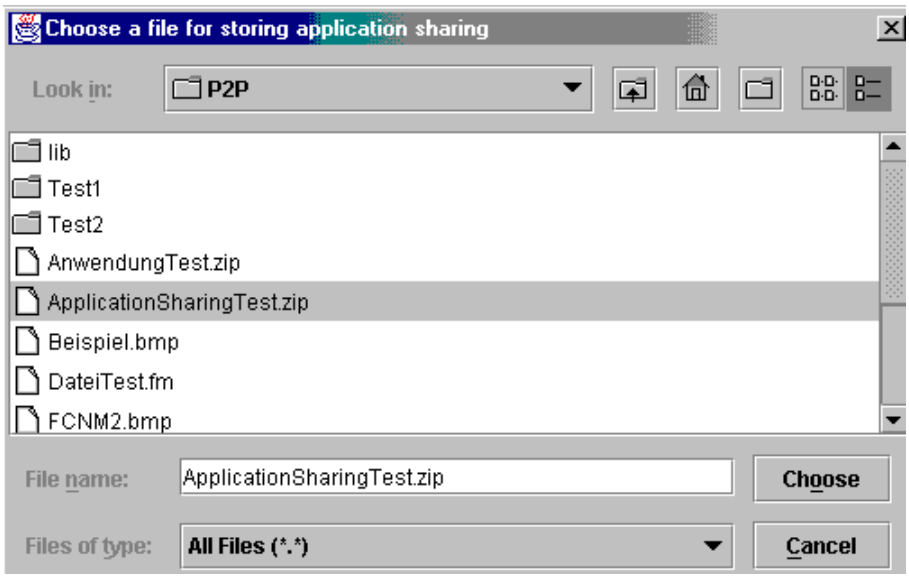

Abbildung 6.9: Das Speichern eines Service

Er wird gefragt, wo der Code des Service gespeichert sollte. Wenn er ein Verzeichnis auswählt und den Dateinamen eingibt, drückt er den Knopf "*Choose*" als Bestätigung. Dieser Pfad wird für spätere Antworten auf den Service gespeichert, die er schon bekommen hat, wenn ein anderen Teilnehmer nach diesem Service fragt. Durch die lokale Speicherung der Daten wird die Persistenz der Daten gesichert, wenn der Teilnehmer, der die Service angeboten hat, die Peergruppe verläßt. Ein anderer Teilnehmer, der den Code der Service hat, übernimmt die Übertragung des Service.

Services sind im Sinne nur Java-Programme für das P2P-Framework. Die Installation bedeutet deshalb nur das Kopien von Java-Klassen oder Archivdatei an Peers.

6.5.3 Benutzungsoberfläche für Floor Control

6.5.3.1 GUI für Floor Control beim Moderator

Die Abbildung 6.10 zeigt das GUI für Floor Control beim Moderator. Der Moderator kann den Floor einem Teilnehmer geben, indem er den Namen des Teilnehmers in der Floor-Kandidatenliste auswählt und den Knopf "**Grant**" drückt. Die Floor-Erlaubnisnachricht wird an alle Teilnehmer übertragen. Bei jedem Teilnehmer werden die **floorOwnerList** und **floorCandidateList** aktuallisiert.

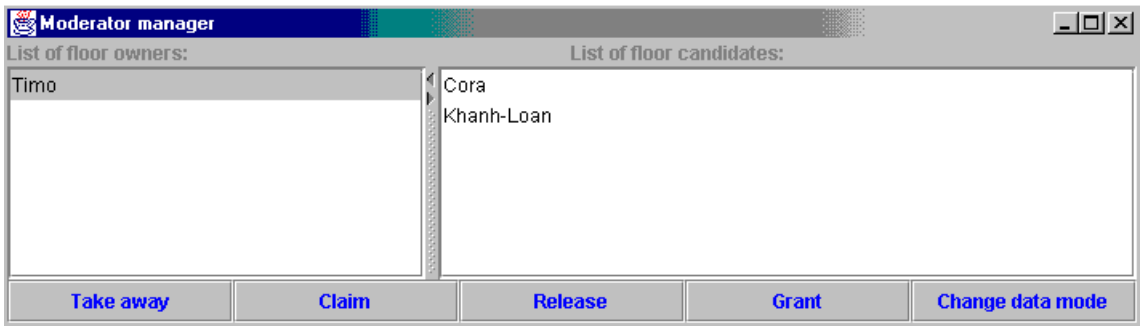

Abbildung 6.10: Floor Control UI bei Moderator

Der Moderator kann den Floor von einem Teilnehmer entziehen, indem er den Namen des Teilnehmers in der *floorOwnerList* auswählt und den Knopf "**Take away**" drückt.

Der Moderator kann die Optionen "**DataView**" und "**DataChange**" zur Laufzeit durch den Knopf "**Change data mode**" ändern. Drückt er "Change data mode", erscheinen die Optionen zur Auswahl.

"**Claim**" und "**Release**" werden im Abschnitt 6.4.3.2 erklärt.

6.5.3.2 GUI für Floor Control bei anderen Teilnehmern

Die Abbildung 6.11 ist für UI mit Moderator bei einem anderen Teilnehmer. Wenn ein Teilnehmer den Floor haben will, kann er den Floor beantragen, indem er den Knopf "**Claim**" drückt. Sein Name wird an alle Teilnehmer übertragen und in die *floorCandidateList* eingetragen. Wenn ein Teilnehmer den Floor (siehe 5.2.4.1) besitzt, wird sein Name in die *floorOwnerList* eingetragen und ist der Knopf "*Claim*" Rot makiert (siehe Abb. 6.11), damit der Teilnehmer weiss, dass er jetzt den Floor besitzt und bearbeiten kann.

Wenn der Teilnehmer den Floor freigeben will, drückt er den Knopf "**Release**". Die Floor-Freigabennachricht wird an alle Teilnehmer übertragen. Der Name des Teilnehmers wird

von der *floorOwnerList* entfernt.

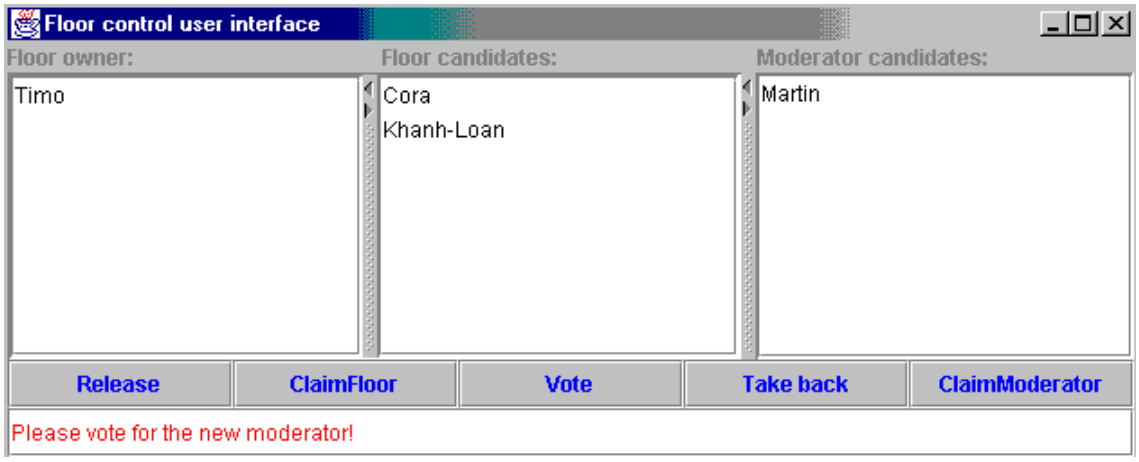

Abbildung 6.11: Floor Control mit Moderator UI bei einem anderen Teilnehmer

Der Knopf "**Vote**" ist für die Abstimmung über den Moderator, wenn der Moderator ausfällt. Das untere **Textfeld** ist für die Anzeige der Informationen z.B. "Please vote for the new moderator!".

Ein Floor- oder Moderator-Kandidat kann jederzeit seinen Antrag zurücknehmen, indem er seinen Namen in die Listen auswählt und den Knopf "**Take back**" drückt.

Das UI für Floor Control **ohne Moderator** bei anderen Teilnehmern sieht wie in der Abbildung 6.12 aus. Die Knöpfe funktionieren wie oben erwähnt.

Der Knopf "Vote" in diesem Fall ist für die Abstimmung über den Floor.

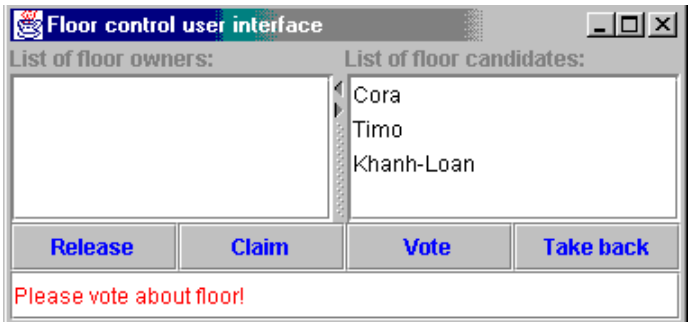

Abbildung 6.12: Floor Control ohne Moderator UI bei anderen Teilnehmern

6.5.4 Benutzungsoberfläche für Start einer Anwendung

Will ein Benutzer allein arbeiten, kann er die Anwendung durch den Dialog (siehe 6.13) starten, indem er die Anwendung auswählt und den Knopf "**Start application**" drückt.

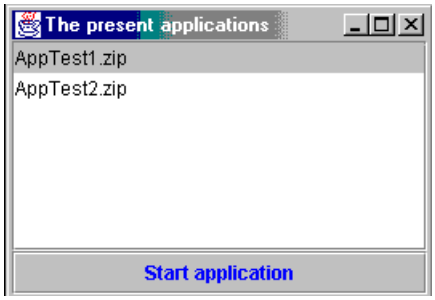

Abbildung 6.13: Start einer Anwendung

Wenn ein Teilnehmer eine angefragte Anwendung empfängt, wird der Name der Anwendung in die Liste **appList** eingetragen. Diese Liste wird auf dem Festplatte in die Datei "**FileStoresPresentApplications**" gespeichert. Jedes Mal wenn der Benutzer das Framework startet, erhält er diese Liste mit allen vorhandenen Anwendungen.

7 Test und Bewertung

7.1 Testumgebung und -szenarien

7.1.1 Testumgebung

Für den Test wurden Laptops verwendet. Die Netzwerkverbindung zwischen den Laptops erfolgt über Funk-LAN. Die Laptops haben die folgenden technischen Daten:

- Intel Pentium III 800 MHz
- 190 MB Hauptspeicher
- Betriebssystem Windows ME
- 2 Mbit/s Wireless LAN Funkkarte

7.1.2 Testszenarien

7.1.2.1 Kommunikationstest

Um die Funktionstüchtigkeit der Kommunikation des Frameworks zu überprüfen, wurden Datenpakete mit verschiedenen Größen über die Point-to-Point- und Propagate-Kommunikation des Frameworks gesendet. Drei Peers wurden für die Messungen benutzt. Die folgenden Datenübertragungszeiten wurden gemessen:

- Zeit vom Senden bis zum Empfangen eines Datenpakets über Point-to-Point-Kommunikation (Unicast). Dieser Kanal wird für die Übertragung von angefragten Services benutzt.
- Zeit vom Senden bis zum Empfangen eines Datenpakets über Propagate-Kommunikation (Multicast). Dieser Kanal wird für die Sitzungs- und Teilnehmerverwaltung, für die Floor Control, für die Übertragung der Änderungen der Anwendung usw. verwendet.
- Zeit vom Start des P2P-Frameworks bis zum Auffinden einer Peergruppe.
- Zeit vom Start des P2P-Frameworks bis zum Start einer Anwendung, die für die Zusammenarbeit ist.
- Zeit für die Übertragung einer Änderung der Anwendung von einem Peer an drei Peers.

7.1.2.2 Test der Funktionalität des P2P-Frameworks

Bei SASCIA gibt es eine bereits existierende Chat-Anwendung, die aus einem Chat-Client und einem Chat-Server besteht. Die Chat-Anwendung kann nicht für den Test des P2P-Frameworks übernommen werden, da die Chat-Anwendung einen Server benötigt und das P2P-Framework ohne Server funktionieren muss und die Chat-Anwendung keine Schnittstellen des P2P-Frameworks enthält. Von daher wurde eine einfache Beispielanwendung für dieses Framework implementiert, um die Grundfunktionen des Frameworks zu überprüfen. Die Beispielanwendung wird in der Abbildung 7.1 gezeigt.

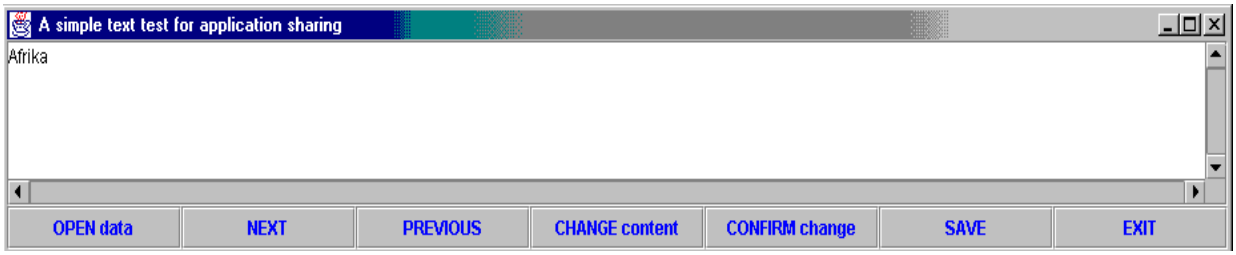

Abbildung 7.1: Die Beispielanwendung zum Test der Funktionalität des Frameworks

Diese Beispielanwendung präsentiert einige Funktionen wie das Blättern und das Ändern von Inhalten, um die Floor Control und die Daten Synchronisation zu überprüfen. Der angezeigte Inhalt ist z.B. Namen der Kontinente, die Strings sind.

- "**Open Data**" ermöglicht Öffnen der Datei, die eine Liste der Namen der Kontinente enthält.
- "**Next**" zeigt den nächsten Kontinent an.
- "**Previous**" zeigt den vorherigen Kontinent an.
- "**Change Content**" ermöglicht das Ändern den Namen eines Kontinents.
- "**Confirm Change**" bestätigt die Änderung und diese wird an alle Peers übertragen.
- "**Save**" speichert den Inhalt in einer Datei.
- "**Exit**" beendet die Beispielanwendung.

7.2 Messwerte und -ergebnisanalyse

7.2.1 Messwerte und -ergebnisanalyse des JXTA-Unicast-basierten Tests

Wenn ein Peer einen Service anfordert, wird die Zeit vom Datensenden bis zum Datenempfangen gemessen. Die Zeit für Hin- und Zurückübertragung des angeforderten Service wird gemessen. Und die Häfte davon wird für die Übertragungszeit des Service berechnet.

Jedes Datenpaket wird 10 Mal hin und zurück gesendet. Die Mittelwerte der Messungen werden in der Abbildung 7.2 gezeigt.

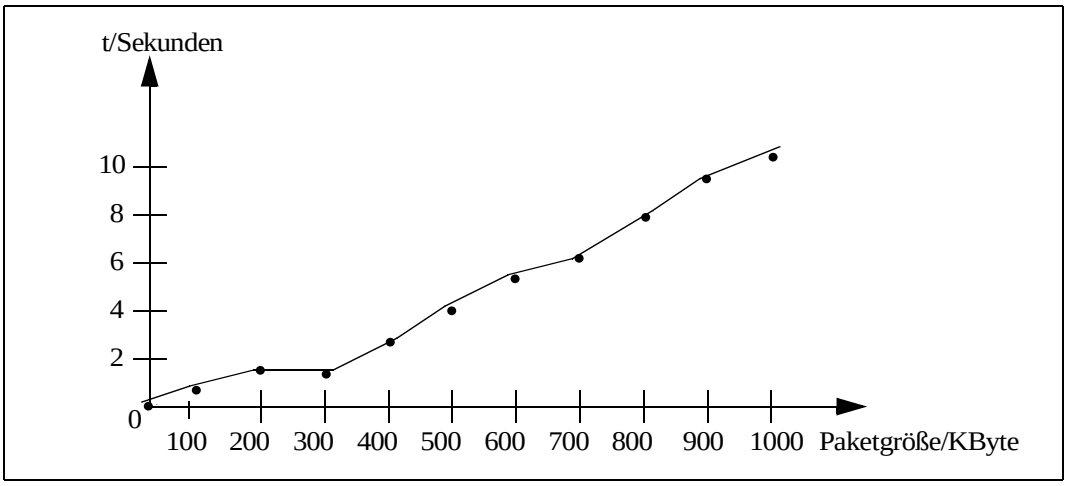

Abbildung 7.2: Antwortzeit-Diagramm bei JXTA-Unicast-Kommunikation

Die Datenübertragungszeit ist tendenziell linear in der Paketgröße bzw. in der Größe der Anwendungen. Die Unicast-Kommunikation wird nicht für die Interaktionen von Benutzern verwendet, sondern nur für die gelegentliche Übertragung des Codes der angeforderten Services.

7.2.2 Messwerte und -ergebnisanalyse des JXTA-Multicast-basierten Tests

In JXTA wurde das Propagate Pipe mit der IP-Multicast implementiert und deshalb ist die Datenpaketgröße auf ca. 16 KByte beschränkt. Die Datenpakete, die für den Test benutzt wurden, sind deshalb nur bis der Größe von 15 KByte.

Die Zeit für Hin- und Zurückübertragung eines Datenpaket wird gemessen. Und die Häfte

davon wird für die Übertragungszeit des Datenpakets berechnet.

Jedes Datenpaket wird 10 Mal hin und zurück gesendet. Die Mittelwerte der Messungen werden in der Abbildung 7.3 gezeigt.

Die Datenübertragungszeit ist tendenziell linear in der Paketgröße bzw. in der Größe der Anwendungen. Für die Sitzungs- und Teilnehmerverwaltung sowie für die Floor Control ist diese Zeit ausreichend, da die Größe der Nachrichtobjekten weniger als 1 KByte ist.

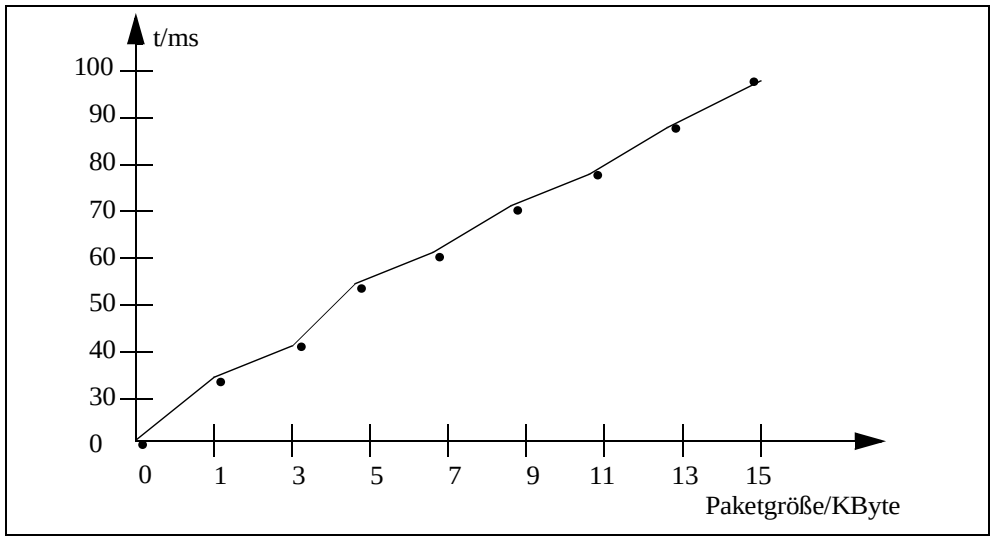

Abbildung 7.3: Übertragungszeit-Diagramm bei JXTA-Multicast-Kommunikation

7.2.3 Zeit vom Start des P2P-Frameworks bis zum Auffinden einer Peergruppe

Angenommen, dass eine Peergruppe schon angelegt ist. Das Auffinden einer Peergruppe kann nur stattfinden, wenn JXTA gestartet wird. Der JXTA- Start findet nach dem Start des Frameworks statt und benötigt die JXTA-Konfiguration. D.h. die Zeit vom Start des P2P-Frameworks bis zum Auffinden einer Peergruppe hängt davon ab, wie schnell ein Benutzer die Eingaben für die JXTA-Konfiguration eintippt und wie schnell das Auffinden an sich ist.

Die Zeit für den Start des JXTA wurde gemessen und liegt zwischen ca. 30 bis 40 Sekunden. Die Zeit für das Auffinden an sich, d.h. die Zeit nach dem Start des JXTA bis zum Auffinden einer Peergruppe liegt zwischen ca. 1 bis ca. 10 Sekunden.

Die Zeit für das Auffinden hängt von der Zeit (timeout) ab, die für die regelmäßige Wiederholung der Veröffenlichung von Peergruppen- und Pipe Advertisements eingestellt wird

(siehe 5.2.3). Das Timeout wurde auf 10 Sekunden eingestellt. Je mehrere Peers in der Peergruppe, desto kürzer ist die Zeit für das Auffinden, da die Wiederholung der Veröffenlichung von Advertisements durch mehrere Peers nacheinander geschieht.

Wenn man die obere Grenze der beiden Zeiten nimmt, ist die Zeit vom Start des P2P-Frameworks bis zum Auffinden einer Peergruppe ca. 50 Sekunden.

7.2.4 Zeit vom Start des P2P-Frameworks bis zum Start eines Applikation Sharing

Mehrere Aktionen werden vom Start des Frameworks bis zum Start einer Anwendung, die für die Zusammenarbeit ist, durchgeführt. Z.B.

- Eingabe für JXTA-Konfiguration,
- Anlegen einer Peergruppe,
- Anbieten eines Applikation Sharing (Auswahl des Verzeichnises und Datei, die den Codes der Anwendung enthält),
- Teilnehmen an der Peergruppe,
- Anfordern des Applikation Sharing (Auswahl des Verzeichnises und Eingabe des Dateinamen für das Speichern des Codes des Applikation Sharing).

Die Zeit hängt davon ab, wie groß die Anwendung ist und wie schnell der Benutzer die Eingaben eintippt.

Für den Test wurde die einfache Beispielanwendung, die im Abschnitt 7.3 beschrieben wird, benutzt. Das ist eine .zip Datei mit der Größe von 23 KByte. Die Zeit liegt zwischen 1,5 bis 2,5 Minuten. Dabei macht die Zeit für die Interaktionen der Benutzer ca. 80% der benötigten Zeit aus.

7.2.5 Zeit der Übertragung einer Änderung eines Applikation Sharing

Die Änderung eines Applikation Sharing beim Test ist das Ändern eines Strings der Beispielanwendung. Die Änderung und die zusätzlichen Informationen werden in dem Objekt *StoreActualDataInfo* gespeichert und über die Propagate-Kommunikation an alle Peers gesendet. Dieses Objekt ist ca. 180 Byte und deshalb ist die Zeit nur ungefähr 8 ms. Diese Zeit ist akzeptabel für die Interaktionen von Benutzern.

7.3 Test der Funktionalität des P2P-Frameworks

Wenn ein Peer den Floor besitzt, kann er den Inhalt blättern oder ändern und diese Aktionen werden an alle Peers übertragen, je nachdem ob der Inhalt nur angeschaut oder bearbeitet werden darf (siehe Kapitel 2). Die Aktionen wie Floor-Beantragen, -Freigabe, -Entziehen und -Geben haben richtig funktioniert.

7.4 Bewertung

Die Grundfunktionen des Systems wie Sitzungs- und Teilnehmerverwaltung, die Floor Control wird nachgewiesen. Die Datenübertragungszeit über die JXTA-Unicast-Kommunikation ist ausreichend für das System. Die Zeit für das Auffinden einer Peergruppe ist weniger als eine Minute, dies ist ausreichend für das System.

Über die JXTA-Multicast-Kommunikation wurde die Größe von Datenpaketen auf 15 KByte beschränkt. Dies ist nicht geeignet für Applikation Sharing, bei der die Nachrichten der Anwendungen z.B. die Änderungen größer als 15 KByte sind. Dieses Problem wird umgangen, indem die Nachrichten des Applikation Sharing überprüft werden, bevor sie an alle Peers gesendet werden. Für den Grenzwert der Datenpaketsüberprüfung wird der Wert von 10 KByte genommen, um die Übertragung über den JXTA-Multicast-Kanal zu sichern. Wenn eine Nachricht der Anwendung kleiner als 10 KByte ist, wird diese Nachricht an alle Peers über die JXTA-Multicast-Kommunikation übertragen. Wenn eine Nachricht größer als 10 KByte ist, wird diese Nachricht über die JXTA-Unicast-Kommunikation an jeden Peer gesendet.

Die Behebung des Multicast-Problems in JXTA ist in Zukunft nicht gesichert. Bei Bedarf ist es möglich, JXTA komplett durch ein anderes System zu ersetzen, da die Schnittstellen des in dieser Arbeit implementierten Kommunikationssystems von JXTA abstrahiert wurden.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

Aufgabe dieser Diplomarbeit war der Entwurf eines dezentralen Peer-to-Peer-Systems, das der gemeinsame Nutzung von Anwendungen, z.B. Whiteboard, Textbearbeitung, Folien-Präsentation usw., dient. Das System muss ohne Server funktionieren. Die Zusammenarbeit bei ad hoc Treffen, Vorlesungen oder Seminaren, werden als Hauptszenarien für die Architektur identifiziert. Aus diesen Szenarien und den Anforderungen an die Aufgabenerstellung folgen die allgemeinen und speziellen Anforderungen für das System.

Um die Entwicklungszeit der Software zu verkürzen, wurden anhand der Anforderungen die Eignung der existierenden Peer-to-Peer-Systeme PROEM und JXTA sowie die Wiederverwendung von SASCIA-Komponenten untersucht. Die Untersuchung lieferte die Ergebnisse: JXTA kann für die Kommunikation des Systems eingesetzt werden; die Aufzeichnungsdatenbank von SASCIA kann für das Framework übernommen werden.

Anhand dieser Ergebnisse und des Ergebnisses der Untersuchung von Replikationsfaktoren der Anwendungschichten erfolgte der Entwurf der Architektur. Die Hauptkomponenten des Peer-to-Peer-Frameworks sind: das Kommunikationssystem, die Sitzungs- und Teilnehmerverwaltung, die Floor Control und die Aufzeichnungsdatenbank. Das JXTA-basierte Kommunikationssystem dient der Nachrichtenübertragung an alle Peers. Die Nachrichten können z.B. Informationen über Sitzungs- und Teilnehmerverwaltung, über Floor Control oder über die Änderungen des Applikation Sharing enthalten. Die Sitzungs- und Teilnehmerverwaltung dient dem Erzeugen von Peergruppen, der Anmeldung und der Abmeldung der Teilnehmer einer Peergruppe. Die Floor Control dient der Rechtevergabe, wer wann welche Aktion der Anwendung machen darf. Die Aufzeichnungsdatenbank dient der Speicherung und der Wiedergabe von Aufzeichnungen einer Sitzung.

Das Kommunikationssystem, die Floor Control und die Sitzungs- und Teilnehmerverwaltung sind in der Programmiersprache JAVA implementiert. Ein Interface für die Integration der Aufzeichnungsdatenbank ist in das Framework enthalten. Die Anbindung ist im Moment noch nicht implementiert.

Um die Funktionstüchtigkeit des Kommunikationssystems zu überprüfen, wurde eine Reihe von Tests durchgeführt. Mit dem JXTA-Unicast-Pipe ergab sich eine akzeptable Leistung des Systems. Mit dem JXTA-Multicast-Pipe gab es ein Problem des Datenpaketverlusts, aufgrund eines Fehlers im verwendeten Java-Multicast. Dieses Problem wurde umgegangen, indem die Datenpakete der Anwendung überprüft werden, bevor sie an alle Peers gesendet werden. Sind die Datenpakete kleiner als 10 KByte, werden sie über den Multicast-Kanal übertragen. Sind die Datenpakete größer als 10 KByte, werden sie über die Unicast-Kanäle übertragen.

Anhand einer einfachen Beispielanwendung wurden die Funktionsfähigkeit des Frameworks (Floor Control und Sitzungs- und Teilnehmerverwaltung) nachgewiesen.

8.2 Ausblick

Es bedarf noch der Verbesserung des Kommunikationssystems, um das System in der Praxis einsetzen zu können. Das JXTA-basierte Kommunikatonssystem funktioniert nicht immer zuverlässig. Es muss durch ein anderes zuverlässiges Kommunikationssystem ersetzt werden oder es muss an ein entsprechendes Protokoll ergänzt werden.

Die Aufzeichnungsdatenbank von SASCIA muss noch an die Peer-to-Peer Umgebung angepasst werden. Eine mögliche Anpassung ist im Kapitel 5 beschrieben. Das Interface für die Integration der Aufzeichnungsdatenbank ist im Framework bereitgestellt.

Bis jetzt enthält das P2P-System keine Authentifikation von Benutzern. Deshalb ist es nicht möglich, die Teilnahme an einer Peergruppe einzuschränken. Der Benutzername und das Kennwort, die der Benutzer bei der JXTA-Konfiguration eingibt, sind nur für die JXTA-Verwaltung. Für die Einschränkung der Teilnahme an einer Peergruppe wird eine Authentifikation benötigt.

Die Lizenzvergabe und das Copyright sind Probleme bei der Nutzung des P2P-Systems, wenn der Code der Anwendung und die Daten für die Zusammenarbeit an alle Peers übertragen werden. Z.B. funktioniert die Lizenzvergabe entweder durch Bezahlung beim Kauf oder durch einen Lizenz-Server. Für das P2P-Framework können diese Methoden nicht eingesetzt werden. Eine Lösung für das Lizenz- und Copyright-Problem muss entwickelt werden.

Bisher ist die periodische Wiederholung der Veröffenlichung von Peergruppe und Propagate Pipe Advertisements bei jedem Peer realisiert, um das Auffinden einer Peergruppe für die späteten Teilnehmer zu gewähleisten. Dies sollte optimiert werden, indem nur ein Peer die Wieder-

holung macht, z.B. der erste Peer in der Mitgliedliste. Falls der Peer ausfällt, übernimmt der nächste in der Mitgliedliste diese Aufgabe. Die Voraussetzung für die Optimierung ist eine zuverlässige Kommunikation und die Konsistenz der Mitgliedlisten.

Beim Anfordern einer Anwendung wird der Code der Anwendung jedes Mal an den angefragten Teilnehmer übertragen. Durch den Vergleich zwischen den vorhandenen Anwendungen und der angefragten Anwendung kann die Übertragung der Datei gespart werden. Der Name der Anwendung wird beim Speichern vom Teilnehmer eingegeben. Wenn der Vergleich auf dem Namen der angebotenen Anwendung und z.B. anderen Informationen wie CheckSum und der Länge der Zipdatei basiert, entsteht das Problem einer Anwendung mit verschiedenen Namen oder gleicher Namen für verschiedene Anwendungen. Eine für die Benutzer durchschaubare Lösung für dieses Problem wird benötigt.

Für das Peer-to-Peer-Computing ist die Schnittstelle der IO Schicht für die entfernte Datenbank nicht notwendig. Deshalb wurde diese Schnittstelle im Rahmen dieser Arbeit nicht implementiert. Es sollte untersucht werden, ob ein einfaches Kommunikationprotokoll oder ein vollständiges Transaktionssystem benötigt wird.

Bisher ist die gleiche Reihenfolge der Operationen bei jedem Peer durch die Floor Control gesichert. Die Floor Control garantiert aber nicht, dass die Änderungen der Anwendung an jeden Peer ankommen. Eine zuverlässige Kommunikation wurde für die Floor Control und die Teilnehmerverwaltung vorausgesetzt. Deshalb wurde dieses Problem in der Arbeit nicht behandelt. Bei einer unzuverlässigen Kommunikation sollte ein Synchronisation-Protokoll für die Behandlung des Ereignisverlusts entwickelt werden. Ein Vorschlag wird im Anhang beschrieben. Für diesen Vorschlag wird die Konsistenz der Mitgliedlisten benötigt. Und für die Konsistenz der Mitgliedlisten sollte der Peer-Ring (siehe Anhang) verwendet werden.

Es ist denkbar, dass einige weitere Funktionen für das P2P-Framework entwickelt werden. Z.B. der Ausschluss von Benutzern. Dafür wird die Authentifikation von Benutzern benötigt.

9Anhang

9.1 Projektplan

Titel: Erstellung einer Peer-to-Peer-Architektur für die gemeinsame Nutzung von Anwendungen

Autor: Nguyen-Salamanis, Khanh-Loan

Datum: 01.12.2001

9.1.1 Einleitung

Der Projektplan dient der/dem Bearbeiter(in) und der/dem Betreuer(in) als Grundlage für die Projektüberwachung und -steuerung. Er legt den Projektablauf aber nicht ein für allemal fest, sondern soll im Laufe der Arbeit ergänzt und angepaßt werden. So ist es z.B. möglich, dass anfänglich spezifizierte Arbeitspakete weiter verfeinert oder Arbeitspakete zugunsten anderer aufgegeben werden.

Der Projektplan kann in Absprache mit der/dem Betreuer(in) in beiderseitigem Einverständnis in den folgenden Fällen angepaßt werden:

- bei Verzug,
- bei vorzeitigem Abschluß eines Arbeitspakets oder
- bei Gewinn neuer Erkenntnisse.

Der aktualisierte Projektplan soll in den Anhang der Ausarbeitung einfließen.

Projektbeschreibung: Bei diesem Projekt handelt es sich um eine Diplomarbeit an der Abteilung Verteilte Systeme des Instituts für Parallele und Verteilte Höchstleistungsrechner der Universität Stuttgart.

Bei dieser Diplomarbeit soll eine Peer-to-Peer- Architektur für ein Application-Sharing-System entworfen und implementiert werden. Dieses System berücksichtigt nach Möglichkeit die im Projekt SASCIA definierten Schnittstellen, um Kompatibilität zu gewährleisten. Der Prototyp ist anhand ausgewählter Szenarien und Messungen zu bewerten.

Der Arbeitungszeitraum erstreckt sich über 6 Monate, beginnend am 01.12.2001 bis am 31.5.2002.

Die Diplomarbeit wird betreut von Dr. rer. nat. Cora Burger.

In den nachfolgenden Abschnitten werden zunächst die Arbeitspakete identifiziert und beschrieben, dann wird ein Zeitplan zu deren Durchfürung festgelegt. Schließlich erfolgt die Definition der Meinlensteine und der dafür zu erstellenden Dokumente.

9.1.2 Beschreibung der Arbeitspakete

Dieser Abschnitt umfaßt die Definition der wesentlichen Arbeitspakete, die Abhängigkeiten zwischen einzelnen Paketen und die Identifikation von kritischen Punkten, die die Duchführung des Projektes gefährden könnten.

Definition der Arbeitspakete

- **AP1, Projektplan**: Erstellung eines initialen Projektplanes
  Status: abgeschlossen
- **AP2, Einarbeitung**: Das SASCIA System, Konzepte des Peer-to-Peer Computing, das Jxta System, das Proem System und JSDT
  Status:-
- **AP3, Anforderungsanalyse**: Spezifizieren der für die Erfüllung der geforderten Funktionalität notwendigen Anforderungen an ein Application-Sharing-System
  Status:-
- **AP4, Recherche**: Bewertung vorhandener Peer-to-Peer Systeme anhand des ermittelte Anforderungsprofils, anschließende Betrachtung der Designalternativen in Application Sharing-System
  Status:-
- **AP5, Systementwurf**: Entwurf einer Peer-to-Peer-Architektur für ein Application-Sharing-System
  Status:-
- **AP6, Schnittstellendefinition**: Definition der Schnittstellen zwischen den Komponenten des Systems
  Status:-

- **AP7, Klassenstrukturen**: Feiner, detailierter Entwurf des Systems
  Status:-
- **AP8, Testszenarien**: Entwurf einer einfachen P2P-Applikation zum Testen
  Status:-
- **AP9, Implementierung:** Prototypische Implementierung des entworfenen Systems
  Status:-
- **AP10, Test**: Untersuchung der Implementierung anhand der Testszenarien
  Status:-
- **AP11, Testergebnisanalyse**: Analyse und Bewertung der Testergebnisse
  Status:-
- **AP12, Ausarbeitung**: Dokumentation von Umfeld, Anforderung, Entwurf, Implementierung und Bewertung anhand der Testszenarien
  Status:-
- **AP13, Vortrag**: Präsentation der Ergebnisse im Rahmen des VS-Kolloquium
  Status:-

Abhängigkeiten der Arbeitspakete

Zwischen den Arbeitspaketen der Diplomarbeit bestehen die folgenden Abhängigkeiten:

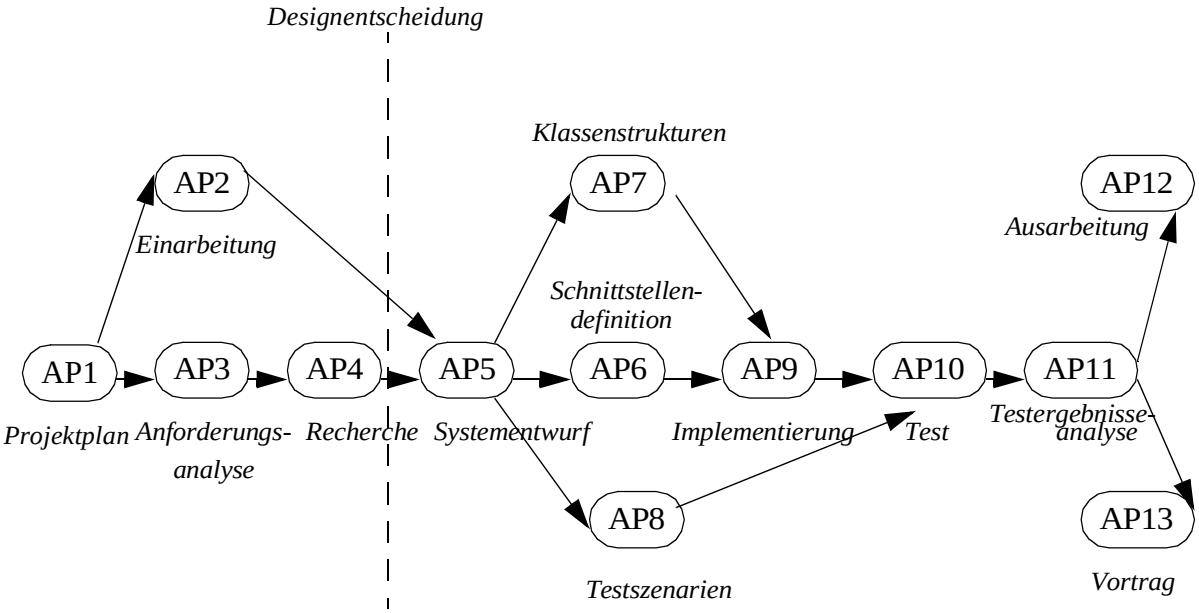

Abbildung 9.1: Abhängigkeiten der Arbeitspakete

9.1.3 Zeitplan

In diesem Abschnitt erfolgt die zeitliche Planung der Arbeitspakete und Meilensteine. Der Bearbeitungszeitraum ist 6 Monate oder 26 Wochen. Wenn sich der Zeitraum mehrerer Arbeitspakete überlappt, werden diese parallel bearbeitet. Am Ende des Bearbeitungszeitraum ist ein Puffer eingeplant worden, der für Verzögerungen oder andere unvorhergesehene Tätigkeiten genutzt werden soll.

geplanter Zeitraum	Aufwand	Arbeitspakete und Meinlensteine	tatsächlicher Zeitraum
1. Woche	1 Tag	AP1 Projektplan	
2.-6. Woche	2 PW	AP2 Einarbeitung	
2. Woche	1 PW	AP3 Anforderungsanalyse	
3.-6.Woche	2 PW	AP4 Recherche	
1. Meilenstein: Designentscheidung			
7. Woche	2 PW	AP5 Systementwurf	
8. Woche	1 PW	AP6 Schnittstellendefinition	
9. Woche	1 PW	AP7 Klassenstrukturen	
10. Woche	1 PW	AP8 Testszenarien	
11.-16. Woche	2 PW	AP9 Implementierung	
2. Meilenstein: Prototyp			
17. Woche	1 PW	AP10 Test	
17.Woche	1 PW	AP11 Testergebnis-Analyse und Bewertung	
18.-24. Woche	2 PW	AP12 Ausarbeitung	
24. Woche	1 PW	AP13 Vortrag	
25. Woche	1 PW	Revision der Dokumente (Betreuerin)	
26. Woche	1 PW	Puffer	

(PW = Personenwoche)

9.1.4 Dokumente und Meilensteine

Als Ergebnis der jeweiligen Arbeitspakete sollen zu den Meilensteinen folgende Dokumente entstehen, die nach diesen einer Versionskontrolle unterliegen und nur in Absprache mit der/

dem Betreuer(in) geändert werden sollen.

- **Anforderungsanalyse**: Beschreibt die Anforderungen, die zur Erfüllung der Funktionalität an ein Application-Sharing-System gestellt werden
- **Recherche-Ergebnis**: Enthält das Resultat der Bewertung vorhandener Peer-to-Peer Systeme anhand des ermittelten Anforderungsprofils sowie eine Designentscheidung für den Entwurf der P2P-Architektur
- **Entwurf**: Zu diesem Dokument gehören die Ergebnisse des System- und des Schnittstellenentwurfs sowie der Klassenstrukturen
- **Prototyp**: Kurze Dokumentation der prototypischen Realisierung
- **Ausarbeitung**: Dokumentation von Anforderungen, Recherche, Entwurf, Implementierung und Bewertung

9.2 Peer-Ring

Bei einer unzuverlässige Kommunikation sollte Peer-Ring verwendet werden, um den Ausfall eines Peers zu erkennen bzw. die Konsistenz der Mitgliedlisten zu erhalten. Um die Belastung des P2P-Netzwerks durch das ständige Senden von Kontrollnachrichten per Multicast an alle Peers zu vermeiden (siehe 5.2.3.3), wird ein **Peer-Ring** für das Senden der Kontrollnachricht verwendet.

Das Prinzip des Sendens des Peer-Rings ist eine Point-to-Point-Verbindung. Aus der sortierten Mitgliedliste, die nach der PeerID sortiert wird, entsteht der Peer-Ring. D.h. ein Peer mit der kleineren PeerID sendet die Kontrollnachricht an den benachbarten Peer mit der größeren PeerID. Wenn der Peer der letzte in der sortierten Mitgliedliste ist, sendet er die Kontrollnachricht an den ersten Peer (siehe Abb.9.2).

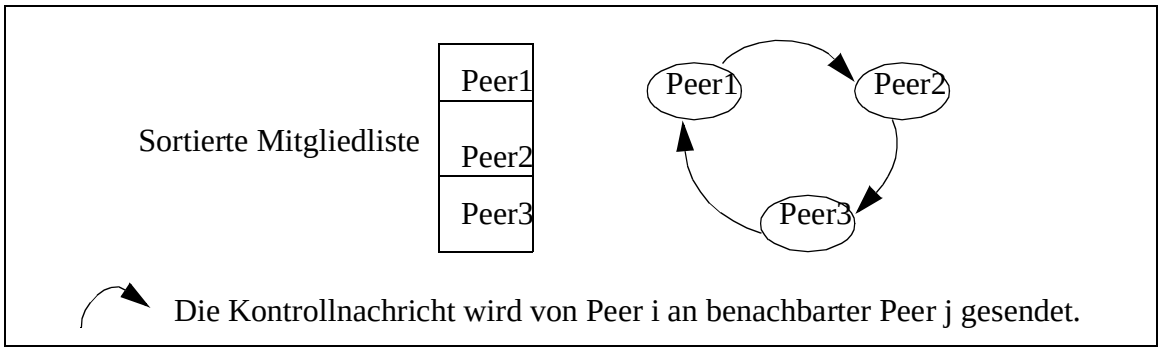

Abbildung 9.2: Das Erzeugen eines Peer-Rings

Die Kontrollnachricht enthält die **Kontrollliste** mit den PeerIDs der Mitglieder der Peergruppe. Wenn ein Peer die Kontrollliste empfängt, vergleicht er sie mit seiner lokalen Mitgliedliste. Wenn die beiden Listen identisch sind, sendet er die Kontrollliste weiter an den benachbarten Peer. Erhält ein Peer ein zweitesmal die Kontrollliste, bedeutet dies, dass die Kontrollliste eine Runde des Peer-Rings durchgelaufen ist. Wenn die Kontrollliste keine Änderungen enthält, sind die Mitgliedlisten konsistent.

Die Kontrollliste enthält eine **Historie** der Mitglieder.

Ein Peer wird als entweder "+" oder "normal" oder "-" für die Kontrollliste und für die Mitgliedlisten bewertet. "+" bedeutet neues Mitglied, "-" bedeutet Ausfall oder Abmeldung, "normal" bedeutet aktuelles Mitglied. Ein Peer mit entweder "+" oder "-" Zustand wird für die Voting nicht gezählt.

Empfängt ein Peer die Anmeldungsnachricht eines Peers , trägt er den neuen Peer mit "+" in seine lokale Mitgliedliste ein. Wenn der Peer gerade die Kontrollliste hat, trägt er den neuen Peer mit "+" in die Kontrollliste ein.

Empfängt ein Peer die Abmeldungsnachricht eines Peers, trägt er den abgemeldeten Peer mit "-" in die lokale Mitgliedliste ein. Wenn der Peer gerade die Kontrollliste hat, trägt er den abgemeldeten Peer mit "-" in die Kontrollliste ein.

Der erste Peer in der Mitgliedliste ist für die Änderung vom "+" auf "normal" Zustand eines Peers in der Kontrollliste zuständig. D.h. wenn die Kontrollliste mit einem Peer mit "+" eine Runde bei dem ersten Peer vorbei ist, setzt er "+" auf "normal". Falls der ersten Peer ausfällt, übernimmt der nächste in der Mitgliedliste.

Empfängt ein Peer nach einer gewissen Zeit keine Kontrollnachrichtbestätigung von seinem benachbarten Peer, weiss er dass der Peer ausgefallen ist. Erfährt ein Peer den Ausfall seines benachbarten Peers, setzt er den Zustand des ausgefallen Peers auf "-" in seine lokale Mitgliedliste. Falls er die Kontrollliste empfängt, setzt er den Zustand des ausgefallen Peers auf "-" in die Kontrollliste. Nur der Peer, der Nachbar des ausgefallenen Peer ist, darf den ausgefallenen Peer aus der Kontrollliste nach einer Runde entfernen. Falls der Peer ausfällt, übernimmt der nächste benachbarte Peer diese Aufgabe.

Ein Peer kann die lokale Mitgliedliste durch die Kontrollliste und umgekehrt die Kontrollliste durch seine lokale Mitgliedliste aktuallisieren. Seine Aktionen für die beiden Listen jenach Fälle werden in Tabelle 9.1 gezeigt. Ein Peer wird mit entweder "nicht da", oder "+" oder "-" oder "normal" Zustand betrachtet.

Nachdem Aktuallisierung der Kontrollliste und der lokalen Mitgliedlisten sendet er die Kontrollnachricht an den nächsten funktionsfähigen Peer in dem Peer-Ring. Dies bedeutet, wenn ein Peer dazu kommt oder ausfällt, ändern sich die Listen und sind zwischenzeitlich nicht konsistent. Aber nach ein oder zwei Durchläufen des Peer-Rings sind die Mitgliedlisten wieder konsistent und der Peer-Ring ist bei jedem Peer gleich.

Zustand eines Peers in Mitgliedliste	Zustand eins Peers in Kontrollliste	Aktion für Mitgliedliste	Aktion für Kontrollliste
nicht da	nicht da	keine	keine
nicht da	+	+ einfügen	keine
nicht da	-	keine	keine
nicht da	normal	normal einfügen	keine
+	nicht da	keine	+ einfügen
+	+	keine	keine
+	-	den Peer entfernen	keine
+	normal	auf normal setzen	keine
normal	nicht da	keine	normal einfügen
normal	+	auf + setzen	keine
normal	-	den Peer entfernen	keine
normal	normal	keine	keine
-	nicht da	den Peer entfernen	keine
-	+	auf + setzen	keine
-	-	den Peer entfernen	keine
-	normal	keine	auf - setzen

Tabelle 9.1: Aktionen eines Peers für die Aktuallisierung der Mitglied- und Kontrollliste

Empfängt der erste Peer in der Mitgliedliste nach einer gewissen Zeit die Kontrollliste nicht, geht er davon aus, dass die Kontrollliste verlorengeht. Er erzeugt eine andere Kontrollliste und sendet sie an den benachbarten Peer und der Peer-Ring funktioniert weiter. Falls die Mitgliedlisten im Moment noch inkonsistent sind, werden eventuell mehrere Kontrolllisten von mehreren ersten Peers erzeugt. Wenn dies der Fall ist, empfängt ein Peer die Kontrollliste, überprüft, ob dies seine erzeugten Kontrollliste ist. Wenn dies nicht seine Kontrollliste ist, aktuallisiert er die beiden Listen wie oben beschrieben, dann überprüft er noch mal nach der Aktuallisierung, ob er immer noch der erste in der Mitgliedliste ist. Wenn er immer noch der erste in der Mitgliedliste ist, wirft er die nicht richtige Kontrollliste weg. Wenn er nicht mehr der erste in der Mitgliedliste ist, sendet er die Kontrollliste an den benachbarten Peer weiter.

Es kann z.B. wegen Netzwerkstörungen eine irrtumliche Meldung der Abwesenheit eines Peers geben. D.h. ein Peer bekommt keine Kontrollnachrichtbestätigung des benachbarten Peers, er geht davon aus, dass der Peer nicht verfügbar ist und sendet eine Abwesenheit-Kontrollnachricht an alle Peers. Aber der benachbarte Peer ist immer noch da. In diesem Fall kann der getroffene Peer wieder eine Anmeldungsnachricht an alle Peers senden, wie er sie schon beim Beitreten zur Peergruppe gesendet hat. (Die Verbindung zur Peergruppe auf seiner Seite ist immer noch da. Er bricht die Kommunikation erst ab, wenn er die Peergruppe wirklich verlassen will. In diesem Fall sendet er die Abmeldungsnachricht an alle Peers wie oben erwähnt). Der Vorgang des Empfangens einer Anmeldungsnachricht bei jedem Peer findet wie im Kapitel 5.2.3.3 erwähnt statt. Er ist wieder ein Mitglied der Peergruppe.

9.3 Synchronisation-Protokoll

Die Aufgabe des Synchronisation Protokolls ist das Sicherstellen, dass alle Peers ein Ereignis erhalten. Ereignisse sind z.B. das Ändern der Daten der Anwendung, die für die Zusammenarbeit ist. Das Synchronisation Protokoll basiert auf dem 3-way-handshake- Protokoll (siehe Abb 9.3).

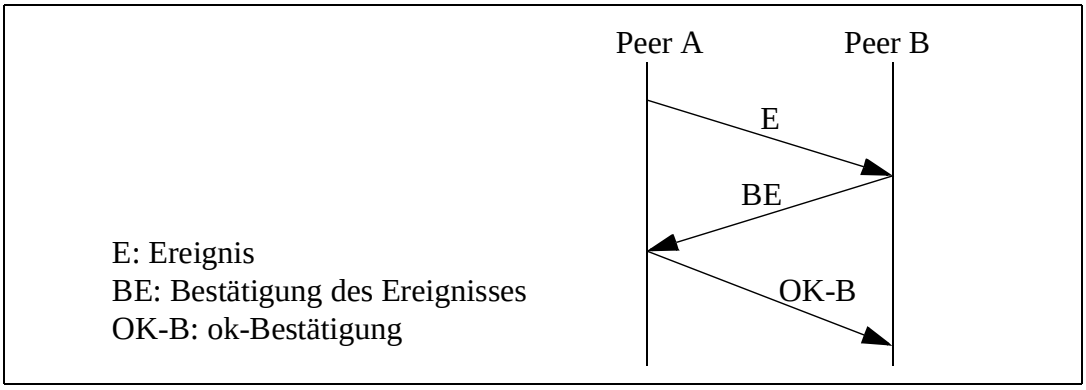

Abbildung 9.3: Das 3-way-handshake-Protokoll

- Peer A sendet ein Ereignis an Peer B.
- Peer B empfängt das Ereignis und sendet eine Bestätigung an Peer A.
- Peer A empfängt die Bestätigung von Peer B und sendet eine OK-Bestätigung an Peer B, damit Peer B das Ereignis verarbeiten kann/darf.

Für dieses Protokoll wird die Konsistenz der Mitgliedliste benötigt. Die Anwesenheit und die Abwesenheit der Peers in der Peergruppe müssen für die Bestätigung korrekt erkannt werden.

Sendet ein Peer ein Ereignis an alle Peers und empfängt er die Bestätigung von allen Peers, dann sendet er die OK-Bestätigung an alle Peers.

Empfängt der Ereignissender nach timeout keine Bestätigung eines Peers, überprüft er durch die Mitgliedliste zuerst, ob der Peer noch da ist. Wenn ja, sendet er das Ereignis noch mal an den Peer. Dieser Vorgang wird solange wiederholt, bis der Ereignissender die Bestätigung des Peers erhält. Wenn nein, ignoriert er die Bestätigung des abwesenden Peers.

Empfängt ein Peer A nach timeout keine OK-Bestätigung des Ereignissenders, überprüft Peer A durch die Mitgliedliste zuerst, ob der Ereignissender noch da ist. Ist der Ereignissender noch da, sendet der Peer A seine Bestätigung an den Ereignissender noch mal. Dieser Vorgang wird solange wiederholt, bis der Peer A die OK-Bestätigung des Ereignissenders erhält. Ist der Er-

eignissender nicht mehr da, muss Peer A die anderen Peers fragen, ob ein Peer die OK-Bestätigung des Ereignissenders erhalten hat. Wenn ja, sendet z.B. ein Peer B die Antwort an den Peer A, damit Peer A das Ereignis verarbeiten kann. Empfängt der Peer A nach timeout keine Antwort, ignoriert er das Ereignis.

9.4 Literaturverzeichnis

[**BAI 01**] Xue Bai
Gemeinsames Lernen von Kommunikationsprotokollen durch elektronisch unterstütztes Rollenspiel
Universität Stuttgart, Fakultät Informatik, Diplomarbeit Nr. 1930, 2001
ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/DIP-1930/

[**BGHK 00**] G. A. Bolcer, M. Gorlick, A. S. Hitomi, Peter Kammer, et al
Peer-to-Peer-Architectures and the Magi (TM) Open-Source Infrastructure 2000
Endeavors Technology, Inc.
http://www.endtech.com/

[**BURGER 01**] Burger, Cora et al..
Application sharing in teaching context with wireless networks.
Technical Report 2001/07, Faculty of Computer Science, University of Stuttgart, December 2001

[**BURGER 97**] Burger, Cora
Groupware
dpunkt-Verlag ISBN 3-920993-60-8

[**COMNEWS**] Computer-News
http://www.zvw.de/aktuell/comp/2000/29/comp16.html

[**GROOVE**] Groove Networks
http://groove.net/

[**INSTANTP2P**] InstantP2P
http://instantp2p.jxta.org/

[**INTEL**] Intel
http://www.intel.at/deutsch/eBusiness/products/peertopeer/

[**JINI**] Jini
http://www.sun.com/jini/

[**JSDT**] Java (TM) Share Data Toolkit Home Page (JSDT)
http://java.sun.com/products/java-media/jsdt/

[**JSPACES**] JavaSpaces (TM)
http://java.sun.com/products/javaspaces/

[**JUGEIDE**] JXTA User Guide
http://instantp2p.jxta.org/Userguide.html

[**JXTA**] Project JXTA
Technical Specification 1.0.
http://www.jxta.org/project/www/white_papers.html

[**JXTASHELL**] JXTA Shell
http://shell.jxta.org

[**KRISHNAN 01**] Krishnan, N..
The Jxta Solution to P2P
Javaworld, October 2001
http://www.javaworld.com/javaworld/jw-10-2001/jw-1019-jxta.html

[KSPTFS 01] Kortuem, G., Schneider, J., Preuitt, D., Thompson, C., Fickas, S., Segall, Z.
When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad hoc Networks
University of Oregon, Aug. 2001
http://www.cs.uoregon.edu/research/wearables/papers.html

[**NAPSTER**] Napster
http://www.napster.com

[**NGUYEN 00**] Nguyen-Salamanis, Khanh-Loan
Adhoc-Verwendung einer elektronischen Tafel unter Verwendung der Jini-Technologie
Universität Stuttgart, Fakultät Informatik, Studienarbeit Nr. 1778, 2000
ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/STUD-1778/

[**OBER 01**] Peter Oberparleiter
Vergabe von Zugangsberechtigungen (floor control) für die gemeinsame Nutzung von SMARTboards in der Vorlesung
Universität Stuttgart, Fakultät Informatik, Studienarbeit Nr. 1807, 2001
ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/STUD-1807/

[**OBER 02**] Peter Oberparleiter
Eine Architektur zur gemeinsamen Nutzung von Anwendungen in der Lehre mit prototypische Realisierung von Tafel und Leinwand.
Universität Stuttgart, Fakultät Informatik, Diplomarbeit Nr. 1950, 2002
ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/DIP-1950/

[**P2PWG**] Peer-to-Peer Working Group
http://www.peer-to-peerwg.org/

[**SASCIA**] Project SASCIA (**S**ystem **A**rchitecture **S**upporting **C**ooperative **I**nteractive **A**pplications)
http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/Festival/sascia.html

[**SCHMID 01**] Andreas Schmid
Prototypische Erstellung einer Protokollierung für den Einsatz in Lehrveranstaltungen
Universität Stuttgart, Fakultät Informatik, Studienarbeit Nr. 1816, 2001
ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/STUD-1816/

[**SOMMER 01**] Marcus Sommer
Prototypische Erstellung einer Sitzungsverwaltung für den Einsatz in Lehrveranstaltungen
Universität Stuttgart, Fakultät Informatik, Studienarbeit Nr. 1817, 2001
ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/STUD-1817/

9.5 API

In diesem Abschnitt werden einige Klassen vorgestellt:

Class PropagatePipeComm

This class is responsible for the creating input and output pipe for the propagate pipe.

Author: Nguyen-Salamanis, Khanh-Loan

Field Summary

java.util. *Vector messagesOnPPVector*
Vector for the received messages on propagate pipe

Constructor Summary

PropagatePipeComm(net.jxta.peergroup.PeerGroup group, net.jxta.protocol.PipeAdvertisement proPipeAdv)

Method Summary

void **createIOPipe**(net.jxta.protocol.PipeAdvertisement pipeAdv)
Load the pipe advertisement generated by PipeAdvPopulator.
void **pipeMsgEvent**(net.jxta.pipe.PipeMsgEvent event)
Handles an incoming message.
void **sendByteServiceMessage**(byte[] messageObject)
Send message to the remote peer using output pipe
void **sendByteServiceMessage**(java.lang.String name, byte[] messageObject)
Send message to the remote peer using output pipe
void **sendStringServiceMessage**(java.lang.String name, java.lang.String valueStr)
Send a message to the remote peer in the peer group using the output pipe.

Class SecureUnicastPipeComm

This class is responsible for the creating the input pipe to listening the messages

Author: Nguyen-Salamanis, Khanh-Loan

Field Summary

java.util.*Vector messagesOnUPVector*

Vector for storing the receiving messages on the unicast pipe

Constructor Summary

SecureUnicastPipeComm(net.jxta.peergroup.PeerGroup group, net.jxta.protocol.PipeAdvertisement pipeAdv)

Method Summary

void **pipeMsgEvent**(net.jxta.pipe.PipeMsgEvent event)

Handles an incoming message.

Class SendServiceOnSecureUnicastPipe

This class is responsible for the creating of the outpup pipe which to be used to send the messages to the other peer. It offers the methods for sending of messages

Author: Nguyen-Salamanis, Khanh-Loan

Field Summary

net.jxta.pipe.*OutputPipe outputPipe*

The pipe to be used to send the message to the remote peer.

Constructor Summary

SendServiceOnSecureUnicastPipe(java.lang.String peerID, net.jxta.peergroup.PeerGroup group, CreatedAndJoinedPeerGroupList cajpgList)

Method Summary

void **createOutputSUPipe**(java.lang.String peerIDStrForSUP)

This method creates a output pipe for sending

void **outputPipeEvent**(net.jxta.pipe.OutputPipeEvent event)

The OutputPipeListener event that is triggered when an OutputPipe is resolved by the call to PipeService.createOutputPipe.

void **sendByteServiceMessage**(java.lang.String name, byte[] array)

Send message to the remote peer using output pipe

void **sendStringServiceMessage**(java.lang.String name, java.lang.String valueStr)

Send a message to the remote peer in the peer group using the output pipe.

Class SynchronSend

This class synchronizes the sending of messages

Author: Nguyen-Salamanis, Khanh-Loan

Method Summary

static SynchronSend **getSynchronSendObject()**

This method creates an instance of SynchronSend

void **sendByteOnPropagate**(byte[] message, PropagatePipeComm ppc)

This method sends the byte array messages on Multicast

void **sendByteOnPropagate**(java.lang.String name, byte[] message, PropagatePipeComm ppc)

This method sends the byte array messages on Multicast

void **sendByteOnUnicast**(java.lang.String name, byte[] message, SendServiceOnSecureUnicastPipe sendServiceOnUP)

This method sends the byte array messages on Unicast

void **sendStringOnPropagate**(java.lang.String name, java.lang.String message, PropagatePipeComm ppc)

This method sends the string messages on Multicast

void **sendStringOnUnicast**(java.lang.String name, java.lang.String message, SendServiceOnSecureUnicastPipe sendServiceOnUP)

This method sends the string messages on Unicast

Class AnalyseMessageOnPropagatePipe

This class analyses the receiving messages on the propagate pipe and handles them.
Author: Nguyen-Salamanis, Khanh-Loan

Field Summary

java.util.Hashtable *appHashtable*
Hash table for storing informations of offered application

java.util.Hashtable *appSharingHashtable*
Hash table for storing informations of offered application for sharing

java.util.Hashtable *appSRegisterHashtable*
Hash table for registration of the instance of StartApplicationInterface

boolean *codeReceive*
boolean for marking whether the code of the application is received

java.util.Hashtable *dataHashtable*
Hash table for storing informations of offered data

java.util.Hashtable *fcNoModeratorUIRegister*
Hash table for registration of FloorControlUIByPeers

java.util.Hashtable *fcWithModeratorUIRegister*
Hash table for registration of FloorControlWithModeratorUI

java.util.Hashtable *fcWithModeUIByPeersRegister*
Hash table for registration of FCWithModeratorUIByPeers

java.util.TreeSet *memberTreeSet*
Treeset for the member messages

java.util.Vector *messagesOnPPVector*
Vector for the received messages on propagate pipe

java.util.Hashtable *pathHashtable*
Hashtable for storing path of the code of data or application

java.util.Hashtable *peerPresentHashtable*
Hash table for storing instances of StorePeerPresentInfo

java.util.Vector requiredServiceVector
Vector for storing of the required data and applications

java.util.Hashtable *sendServiceOnSUPHashtable*
Hashtable for storing instances of SendServiceOnSecureUnicastPipe

Constructor Summary

AnalyseMessageOnPropagatePipe(PropagatePipeComm ppc, CreatedAndJoinedPeerGroupList cajpgList, net.jxta.peergroup.PeerGroup group)

Method Summary

void **getActualDataAndSendIt**(StoreAppSharingInfo storeAppSharingInfo, SendServiceOnSecureUnicastPipe sendServiceOnSUP)

This method converts actual data of application into a byte array and sends it to the peer- which has required the application to be shared

void **registerAppSharing**(net.jxta.codat.CodatID appSID, StartApplicationInterface startAppSharingIn)

This method registers the instance of StartApplicationInterface in a hash table with the key is the application sharing id

void **registerFCNoModeratorUI**(net.jxta.codat.CodatID appSID, FloorControlUIByPeers fcUIByPeers)

This method registers the instance of FloorControlUIByPeers in a hash table with the key is the application sharing id

void **registerFCWithModeratorUI**(net.jxta.codat.CodatID appSID, FloorControlWithModeratorUI fcWithModeratorUI)

This method registers the instance of FloorControlWithModeratorUI in a hash table with the key is the application sharing id

void **registerFCWithModeUIByPeers**(net.jxta.codat.CodatID appSID, FCWithModeratorUIByPeers fcWithModeratorUIByPeers)

This method registers the instance of FCWithModeratorUIByPeers in a hash table with the key is the application sharing id

void **run()**

This method analyses the receiving messages

Class AnalyseMessageOnUnicastPipe

This class analyses the receiving messages on the unicast pipe and handles them
Author: Nguyen-Salamanis, Khanh-Loan

Field Summary

ApplicationSharingInterface *appSharingInterface*

An instance of ApplicationSharingInterface

boolean *receivePGInfo*

boolean for marking whether the peer group informations are received

boolean *waitMessage*

boolean for marking whether the wait message of the moderator is received

Constructor Summary

AnalyseMessageOnUnicastPipe(CreatedAndJoinedPeerGroupList cajpgList, net.jxta.peergroup.PeerGroup group, SecureUnicastPipeComm supc)

Method Summary

void **run()**

This method handles the receiving messages on the unicast

Class StartApplicationInterface

This interface is implemented by the application

Author: Nguyen-Salamanis, Khanh-Loan

Constructor Summary

StartApplicationInterface()

Method Summary

abstract void **getActualData**(byte[] actualData)

The application gets the actual data.

abstract **void getActualDataOptions**(boolean dataScrollState, boolean dataChangeState)

The application gets the actual data options.

abstract java.lang.Object **giveActualData()**

The application gives its actual data.

abstract void **receiveMessageFromOtherPeers**(byte[] message)

This method delivers the receiving message.

abstract void **setFloor**(boolean floorForYou)

The application gets the floor

abstract void **starter**(ApplicationSharingInterface appSharingInterface, StoreAppSharingInfo storeAppSharingInfo)

This method is for application sharing.

Class ApplicationSharingInterface

This interface is implemented by the p2p-Framework

Author: Nguyen-Salamanis, Khanh-Loan

Constructor Summary

ApplicationSharingInterface(PropagatePipeComm pPipeComm, net.jxta.codat.CodatID asID, CreatedAndJoinedPeerGroupList cajpgList)

Method Summary

void **sendMessageToPeers**(byte[] message)

This method send message from the application to all peers in the peer group