

Studiengang: Informatik

Prüfer: Prof. Dr. rer. nat. Dr. h. c.
Kurt Rothermel

Betreuerin: Dr. rer. nat. Cora Burger

begonnen am: 16. Juli 2001

beendet am: 15. Januar 2002

CR-Nummer: H.4.3, H.5.3

Diplomarbeit Nr. 1950

**Eine Architektur zur gemeinsamen
Nutzung von Anwendungen in der
Lehre mit prototypischer Realisierung
von elektronischer Tafel und Lein-
wand**

Peter Oberparleiter

Institut für Parallele und
Verteilte Höchstleistungsrechner
Universität Stuttgart
Breitwiesenstraße 20-22
D-70565 Stuttgart

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Aufgabenstellung	1
1.3	Übersicht	2
1.3.1	Anforderungen	2
1.3.2	Entwurf	2
1.3.3	Umsetzung	2
1.3.4	Elektronische Tafel	2
1.3.5	Bewertung	3
1.3.6	Ausblick	3
2	Anforderungen	5
2.1	Anwendungsszenarien	5
2.2	Gruppenkommunikation	5
2.3	Teilnehmerverwaltung	6
2.4	Anmeldungsunterstützung	6
2.5	Anwendungen	6
2.6	Floor Control	7
2.7	Protokollierung	7
2.8	Dienst-Koordination	7
2.9	Elektronische Tafel und weitere Dienste	7
2.10	Offenheit	8
2.11	Plattformen	8
2.12	Verfügbarkeit	8
2.13	Konsistenz der Schnittstellen	8
2.14	Stabilität	9
2.15	Sicherheit	9
2.16	Firewalltauglichkeit	9

3	Entwurf	11
3.1	Übersicht	11
3.2	Kommunikationssystem	12
3.2.1	Mechanismen und Begriffe	12
3.2.2	Schnittstellen der Sitzungsverwaltung	15
3.2.3	Schnittstellen der Sitzungsteilnehmer	16
3.2.4	Schnittstellen für den Datenaustausch	17
3.3	Sitzungsverwaltung (Session Management)	18
3.4	Rechtevergabe	19
3.5	Protokollierungsdatenbank	21
3.6	Einbindung von Anwendungen	21
3.6.1	Interne Anwendungen	21
3.6.2	Externe Anwendungen	23
3.7	Datenfluss	24
3.7.1	Komplett integrierter Datenfluss	24
3.7.2	Datenfluss ohne Protokollierung	26
3.7.3	Ungeregelter Datenfluss	27
4	Umsetzung	29
4.1	Java	29
4.2	JSDT	29
4.3	Kommunikationssystem	31
4.3.1	Diskussion der Systemstruktur	31
4.3.2	Wichtige Klassen und Schnittstellen	32
4.4	Elektronische Tafel (Whiteboard)	34
4.4.1	Anforderungen	34
4.4.2	Implementierung	35
4.4.3	Diskussion der Systemstruktur	36
4.4.4	Benutzung des Whiteboard-Servers	40
5	Bewertung	51
5.1	Test-Ergebnisse	51

5.1.1	Kommunikationstest	51
5.1.2	Test des Gesamtsystems.	55
5.2	Schwierigkeiten und Lösungen	56
5.2.1	JSDT.	56
6	Zusammenfassung und Ausblick	59
6.1	Zusammenfassung.	59
6.2	Ausblick.	59
6.2.1	Weiterentwicklung.	59
6.2.2	Einbindung neuer Technologien	60
6.2.3	Peer-to-Peer-Implementierung.	60
6.2.4	Sitzungsunterstützung durch Agentensysteme.	61
6.2.5	Spezialanwendungen	61
6.2.6	Desktopsharing.	61
6.2.7	Interaktionsformen und Ergonomie.	62
6.2.8	Generischer Rollenmechanismus.	62
6.3	Abschließende Bewertung.	63
7	Anhang	65
7.1	Kommunikationstest	65
7.1.1	Testumgebung	65
7.1.2	Test-Durchläufe	65
7.1.3	Server-Zeiten	65
7.1.4	LoadTestClient.	66
7.1.5	LoadTestServer	69
8	Literaturverzeichnis	75

1 Einleitung

1.1 Motivation

Die starke Verbreitung elektronischer Kommunikationsmittel in Bereichen des täglichen Lebens hatte bereits in der Vergangenheit Auswirkungen auf deren Einsatz an Hochschulen. Ein Beispiel hierfür ist der Rechnereinsatz zur Unterstützung von Vorlesungen und Präsentationen.

Mit der zunehmenden Verfügbarkeit von tragbaren Computern sowie deren permanenter Vernetzung ergeben sich Perspektiven für die weitere Einbindung dieser Ressourcen in den Lehrablauf. Denkbar ist hier unter anderem der verstärkte Einsatz von kooperativ nutzbaren Anwendungen, z.B. in Form einer *“elektronischen Tafel”*, in Lehrveranstaltungen.

Zu erwartende Ergebnisse einer derartigen Entwicklung sind z.B. die Untersuchung neuer Formen sowie die Verbesserung der Qualität der Interaktion zwischen Dozenten und Zuhörern. Zudem ist eine größere zeitliche und räumliche Unabhängigkeit der Beteiligten möglich.

1.2 Aufgabenstellung

Das Ziel dieser Arbeit ist die Entwicklung einer Architektur zur gemeinsamen Anwendungsnutzung in der Lehre. Diese soll unter anderem die Möglichkeit vorsehen, Interaktionsschritte durch Annotationen und Zeigeoperationen zu ergänzen und für eine spätere Wiedergabe aufzuzeichnen. Das System soll dabei primär für den Einsatz in Vorlesungen und Präsentationen ausgelegt werden.

Eine erste Teilaufgabe besteht aus der Definition und der Implementierung der eigentlichen Systemarchitektur. Hierbei müssen mehrere Sichten beachtet werden: als Grundlage des gesamten Systems dient eine zu entwerfende Kommunikationskomponente. Darauf aufbauend müssen für die Bereitstellung der geforderten Funktionalität verschiedene Einzelkomponenten entworfen bzw. in manchen Fällen, in denen diese Teile bereits in vorhergehenden Arbeiten fertiggestellt wurden, integriert werden. Zudem sind Überlegungen über die Art und Weise zu treffen, auf welche Benutzern der Zugriff auf die Funktionalität der Architektur ermöglicht wird.

In einem zweiten Schritt soll die Funktionsfähigkeit der Implementierung anhand der prototypischen Realisierung einer elektronischen Tafel getestet werden. Die hierbei geforderte *“Tafel-Funktionalität”* dient zudem als Grundlage für die Möglichkeit, die Benutzung beliebiger Anwendungen durch Annotationen und Zeigeoperationen zu unterstützen.

Bei dieser Arbeit soll die Behandlung der Grundlagen zur Modellierung kooperativer Systeme nicht im Vordergrund stehen. Stattdessen soll auf die konkrete Anwendung dieser Modelle eingegangen werden. Sie stellt damit einen Grundstein für das Projekt “SASCIA”¹ dar, das am Institut für Parallele und Verteilte Höchstleistungsrechner an der Universität Stuttgart ins Leben gerufen wurde.

In Zukunft soll so, neben allgemeinen Untersuchungen verschiedener Realisierungsaspekte von Konferenzsystemen, auch der Blick auf Überlegungen über die Interaktion zwischen System, Dozent und Studierenden ermöglicht werden. Von Interesse sind hierbei z.B. Fragen, inwiefern derartige Systeme von Zuhörern eines Vortrags akzeptiert und benutzt werden und ob sich dadurch ein reeller Mehrwert für die entsprechende Lehrveranstaltung ergibt.

1.3 Übersicht

1.3.1 Anforderungen

Die Abgrenzung der geforderten Funktionalität wird in diesem Kapitel anhand der Ergebnisse einer vorhergehenden Arbeit zur Eignung bestehender Systeme durchgeführt (siehe [OBER01]). Dabei fließen zusätzlich allgemeine Überlegungen ein, die sich aus dem primären Anwendungsszenario, dem Einsatz in Vorlesungen, ergeben.

1.3.2 Entwurf

Durch die Abbildung der Anforderungen auf einzelne Komponenten erfolgt eine grobe Strukturierung der Architektur sowie die Definition der notwendigen Begriffe und der entsprechenden Schnittstellen.

1.3.3 Umsetzung

Es folgt eine ausführliche Beschreibung der Implementierung der Kommunikationskomponente, dem eigentlichen Kern der Architektur. Weitere Teile des Systems wurden bereits in eigenständige Arbeiten ausgegliedert und bearbeitet, weshalb diese hier nur zusammenfassend betrachtet werden.

1.3.4 Elektronische Tafel

Die in Kapitel 4 behandelte prototypische Realisierung einer elektronischen Tafel dient dem Test des entworfenen Systems. Sie stellt aber auch einen ersten Schritt zur Abbildung der Systemarchitektur auf eine dem Anwendungsszenario angemessene Benutzungsoberfläche dar.

1. SASCIA steht als Abkürzung für “System Architecture Supporting Cooperative and Interactive Applications”

1.3.5 Bewertung

In einer anschließenden Bewertung werden die Ergebnisse von Tests über die Leistungsfähigkeit der Kommunikationsschnittstelle sowie die Erfahrungen aus dem Entwurf der elektronischen Tafel zusammengefasst. Dabei wird auf aufgetretene Schwierigkeiten und deren Lösung bei der Entwicklung des Systems eingegangen.

1.3.6 Ausblick

Abschließend erfolgt ein Ausblick auf mögliche Erweiterungen der entworfenen Architektur, und auf darauf aufbauende Arbeiten, deren Schwerpunkt auf den im Lehrbetrieb bereits vorhandenen und zukünftig denkbaren Interaktionsformen liegen könnte.

2 Anforderungen

Wie bereits in der Einleitung erwähnt, besteht das Ziel dieser Arbeit in der Bereitstellung eines Systems im Rahmen des Projekts SASCIA, das die Entwicklung kooperativ nutzbarer Anwendungen unterstützt. Zu diesem Zweck müssen Komponenten zur Verfügung gestellt werden, welche die notwendigen Funktionen, wie z.B. Gruppenkommunikation, Sitzungsverwaltung oder dynamische Rechtevergabe, abdecken.

Eine Untersuchung verschiedener existierender Systeme (siehe [OBER01]) führte zu dem Ergebnis, dass sich vorhandene Konferenzsysteme mangels Flexibilität und Erweiterbarkeit nicht uneingeschränkt für diese Aufgabe eignen. Aus diesem Grund fiel die Entscheidung, ein entsprechendes System neu zu entwickeln.

Im folgenden werden die Anforderungen zusammengefasst, die zuvor schon als Maßstab für die untersuchten Systeme angewandt wurden und die nun als Grundlage für den Entwurf der geforderten Architektur dienen.

2.1 Anwendungsszenarien

Die zugrundeliegenden Anwendungsszenarien sind:

1. Vorlesungen und Vorträge
2. Geleitete Übungen
3. Ungeleitete, spontane Übungen, z.B. im Rahmen von Prüfungsvorbereitungen

In den Fällen 1 und 2 ist mit Teilnehmerzahlen von über 20 Teilnehmern zu rechnen. Man kann davon ausgehen, dass sich der Grossteil der Teilnehmer vor Ort befindet - eine Teilnahme per WAN/Internet muss aber ebenfalls unterstützt werden.

2.2 Gruppenkommunikation

Der Austausch von Daten in Gruppen von 20 und mehr Teilnehmern ist die eigentliche Grundvoraussetzung für computergestützte Zusammenarbeit innerhalb der skizzierten Szenarien. Wichtig ist hierbei, dass das verwendete Kommunikationssystem die zuverlässige und geordnete Datenübertragung garantiert, damit Anwendungen auf diesen Voraussetzungen aufbauen können und nicht gezwungen werden, entsprechende Mechanismen selbst bereitzustellen.

Um den Datenverkehr verschiedener Anwendungen voneinander abzugrenzen, ist die Einführung von Kanälen sinnvoll. Es ergibt sich die Notwendigkeit von Mechanismen, um Teilnehmer zu bestimmten Kanälen zuzulassen bzw. abzuweisen und zu entfernen. Dies sollte sowohl vor Beginn als auch während einer Sitzung möglich sein.

2.3 Teilnehmerverwaltung

In physischen Sitzungen treten Teilnehmer meist in unterschiedlichen Rollen auf, im Fall einer Vorlesung sind dies z.B. Dozent und Zuhörer. Entsprechend muss eine Abbildung dieses Rollensystems auch in einem computergestützten System erfolgen. Eine notwendige Voraussetzung hierfür ist die Möglichkeit, einzelne Teilnehmer zu unterscheiden und zu identifizieren. Um Befürchtungen einer automatisierten Überwachung zu entkräften, sollte zudem die anonyme Teilnahme (z.B. durch die Verwendung von Pseudonymen) möglich sein.

Die "Sichtbarkeit" anderer Teilnehmer, bzw. derer Aktionen ist bei geographisch verteilten Personengruppen wichtig, damit das Gefühl der Zusammenarbeit überhaupt entstehen, bzw. verstärkt werden kann. Besonders in weiteren Anwendungsszenarien, wie z.B. bei der Durchführung von Übungen, ist die Bereitstellung entsprechender Informationen über die anwesenden Benutzer deshalb wichtig. Dennoch muss auch hier zugunsten des Schutzes der Privatsphäre der Teilnehmer eine Beschränkung auf die wichtigsten Informationen erfolgen.

2.4 Anmeldungsunterstützung

Eine umfassende Unterstützung für das Auffinden von laufenden Sitzungen und das Anmelden verringert den Mehraufwand, der einem Vorlesungsteilnehmer durch die Benutzung des Systems entsteht. Mit der zunehmenden Verbreitung von Technologien wie z.B. ortsabhängige Dienste und Chipkarten für die Authentifizierung, erschließen sich weitere Möglichkeiten, den Vorgang der Anmeldung zu vereinfachen.

2.5 Anwendungen

Der Grundgedanke dieser Arbeit ist die Förderung computergestützter Zusammenarbeit durch die gemeinsame Nutzung von Anwendungen. Es kann sich hierbei um beliebige Anwendungen handeln, sofern die zu nutzende Funktion durch ein Programm steuerbar ist. Beispiele hierfür sind:

- Steuerung von *PowerPoint*-Präsentationen mit Hilfe der von Microsoft bereitgestellten API
- Rollenbasierte Fernsteuerung von Simulationen
- Durchführung von Teleexperimenten
- Kontrolle einer Modelleisenbahn
- Gemeinsamer Zugriff auf beliebige Anwendungen über *Desktop-Sharing*
- Zusammenarbeit mittels einer elektronischen Tafel

2.6 Floor Control

Als Voraussetzung für eine geregelte Interaktion der Anwender ist ein dynamisches Rechtevergabesystem notwendig. Es dient dabei der zeitweiligen Vergabe von Zugriffs- und Modifikationsrechten für bestimmte Ressourcen. Ein derartiges System wurde bereits in einer vorhergehenden Arbeit [OBER01] entwickelt und muss lediglich integriert werden.

2.7 Protokollierung

Die Aufzeichnung der während der Kooperation erzeugten Daten in der zeitlichen Reihenfolge ihrer Entstehung ermöglicht zu spät kommenden Teilnehmern, bereits abgelaufene Aktivitäten besser nachzuvollziehen. Eine entsprechende Funktion unterstützt zudem die nachträgliche Verarbeitung der produzierten Daten.

2.8 Dienst-Koordination

Der Ablauf einer Vorlesung wird im sozialen Bereich zentral durch den Dozenten koordiniert. Dies wird möglich, da die Zuhörer an einem störungsfreien Veranstaltungsablauf interessiert sind, den nur der Dozent gewährleisten kann. In einer übertragenen Sichtweise sind Zuhörer also Nutzer eines gewissen Dienstes - dem störungsfreien Veranstaltungsablaufs - der durch den Dozenten angeboten wird.

Im Kontext der Systemarchitektur sind sowohl die Teilnahme an einer Sitzung, als auch die Nutzung einer Anwendung als Dienste zu betrachten. Die zentrale Koordination des Zugriffs auf diese Dienste wird nur dadurch möglich, dass auf Anwendungsebene ebenfalls eine Unterteilung in Anbieter und Nutzer erfolgt. Der Anbieterteil übernimmt hierbei die Steuerung der eigentlichen Anwendung, während der Nutzerteil die Kommunikation zwischen entferntem Nutzer und lokaler Anwendung realisiert. Diese Unterteilung muss sich entsprechend auf den Aufbau der Systemkomponenten auswirken.

2.9 Elektronische Tafel und weitere Dienste

Zur Überprüfung der Konzepte dieser Arbeit soll eine elektronische Tafel entwickelt werden, die zwei Hauptfunktionen zur Verfügung stellt:

1. Annotieren von Präsentationsinhalten und beliebigen Anwendungsausgaben mittels Freihandzeichnungen und Texteingaben
2. Zeigen auf Inhalte

Hierbei soll zwischen einer öffentlichen und privaten Sicht unterschieden werden, d.h. für den Zweck von privaten Notizen soll ebenfalls eine nur lokal sichtbare Tafelfläche existieren.

Abgesehen von der “Standard-Komponente” des gemeinsamen Tafelbereichs können weitere Dienste während eines Vortrags von Interesse sein, wie sie auch in [SCHEE01] erwähnt und diskutiert werden. Beispiele hierfür sind:

- Feedback-Systeme, die dem Dozenten durch eine kontinuierlich im Hintergrund laufende Befragung der Teilnehmer einen Überblick z.B. über die Bewertung der Vortragsgeschwindigkeit und -verständlichkeit verschaffen.
- Meldungssysteme, die bei Sitzungen mit entfernten Teilnehmern die Möglichkeit der Meldung per Handzeichen nachbilden.
- Möglichkeiten von automatischen Abstimmungen.
- Systeme zur Durchführung eines Quiz, d.h. einer stoffbezogenen Wissensfrage, z.B. in Form einer *Multiple-Choice* Aufgabe.
- Dienste für den sitzungsenternen Dateitransfer.

Das zu entwickelnde System beinhaltet diese Dienste nicht direkt, sondern stellt vielmehr die Grundlagen für deren Implementierung zur Verfügung.

2.10 Offenheit

Die Architektur muss Anwendungen vollständigen Zugriff auf die implementierte Funktionalität gewähren. Zudem sollte die Möglichkeit von Erweiterungen bereits durch die Art des Entwurfes gegeben sein.

2.11 Plattformen

Im Umfeld des Rechnereinsatzes in der universitären Lehre ist von einer Vielzahl an unterschiedlichen Betriebssystemen, Hardware-Plattformen und Kommunikationsmitteln auszugehen. Bei der Entwicklung des Systems sollte dieser vorhandenen Heterogenität durch eine möglichst weitgehende Plattformunabhängigkeit Rechnung getragen werden.

2.12 Verfügbarkeit

Um eine freie Verfügbarkeit zu gewährleisten, sollte es soweit möglich vermieden werden, kostenpflichtige Teile kommerzieller Hersteller in das System zu integrieren.

2.13 Konsistenz der Schnittstellen

Die Benutzungsschnittstelle des Systems sollte möglichst konsistent gehalten werden, sowohl in Bezug auf die Strukturierung der Schnittstellen für Anwendungsprogrammierer, als auch auf die Benutzungsoberfläche, die dem eigentlichen Benutzer präsentiert wird.

Obwohl die Entwicklung neuer Dienste außerhalb des Systems stattfindet, kann, z.B. durch die Bereitstellung von Standardkomponenten für die Benutzerinteraktion eine gewisse Konsistenz der Oberfläche auch hier vorbereitet werden.

2.14 Stabilität

Damit das System auch nach der anfänglichen Experimentierphase von Benutzern akzeptiert wird, muss es ein solides Maß an Stabilität und Robustheit gegenüber evtl. gewollten Falscheingaben bieten. Dies ist besonders bei einer Neuentwicklung nicht einfach zu erreichen. Eine über den Entwicklungszeitraum hinaus andauernde Optimierung und Fehlerbeseitigung ist jedoch ein Grundstein für das Erreichen dieses Ziels.

2.15 Sicherheit

Je nach Anwendungsszenarien sind sowohl die gesicherte Authentifizierung von Teilnehmern, als auch eine mögliche Verschlüsselung von übertragenen Inhalten, ein wichtiges Argument für die Akzeptanz durch die Benutzer.

2.16 Firewalltauglichkeit

Besonders im Fall von Lehrveranstaltungen ist es sinnvoll, dass der Netzzugang auf die vorlesungsrelevanten Teile beschränkt wird. Hierbei sollte gewährleistet sein, dass das System unabhängig von einer Abschottung des lokalen Netzes durch eine Firewall oder ähnliche Maßnahmen weiterhin funktioniert.

3 Entwurf

Unter Berücksichtigung der zuvor aufgestellten Anforderungen werden in diesem Kapitel die Komponenten der Systemarchitektur identifiziert und abgegrenzt. Außerdem erfolgt eine Strukturierung anhand einer Darstellung der Abhängigkeiten zwischen den einzelnen Komponenten.

3.1 Übersicht

Abbildung 3-1 zeigt eine Übersicht über die in diesem Kapitel beschriebenen Komponenten und skizziert den notwendigen Daten- und Kontrollfluss.

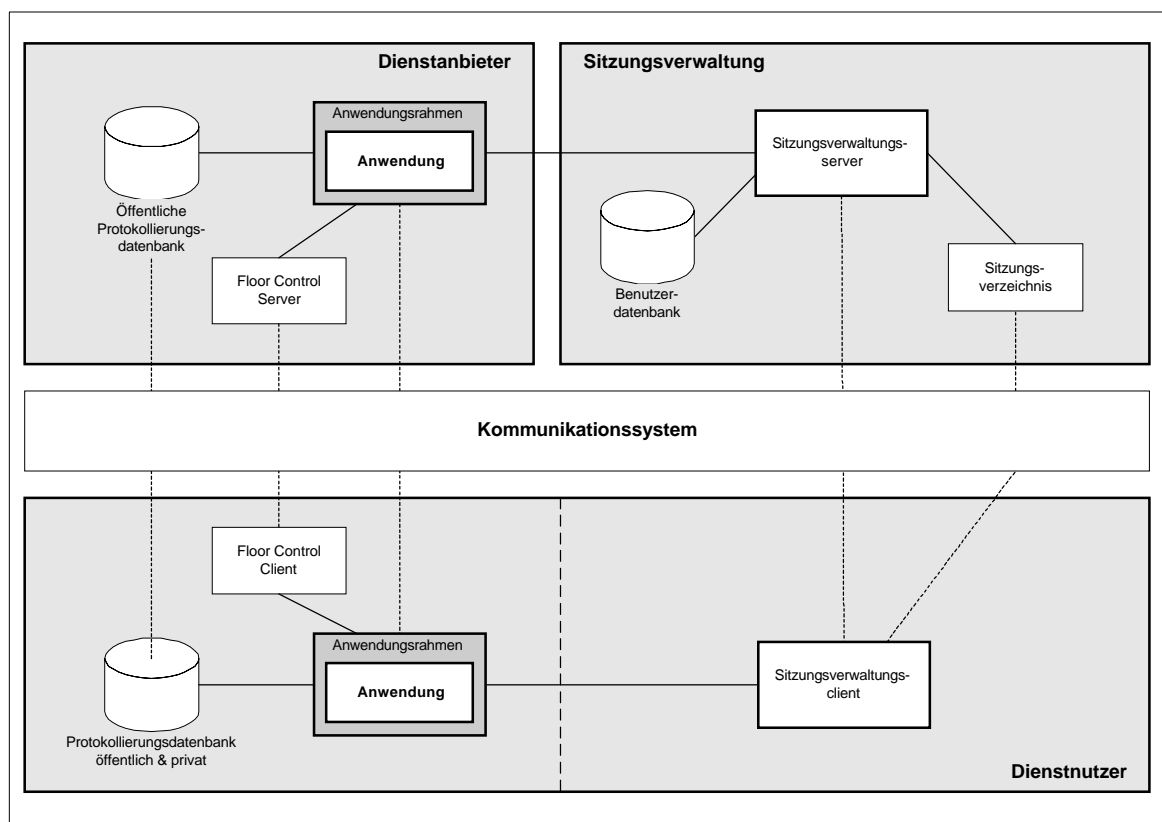


Abbildung 3-1: Übersicht über Komponenten und Abhängigkeiten

Deutlich sichtbar ist die in Abschnitt 2.8 begründete Unterteilung in Komponenten zur Realisierung des Dienstanbieter- und -nutzerprinzips. Die notwendigen administrativen Architekturbestandteile der Sitzungsverwaltung werden ebenfalls gesondert aufgeführt. Alle Bestandteile werden im folgenden näher beschrieben.

3.2 Kommunikationssystem

Grundlage jeglicher Kooperation ist der Austausch von Informationen. Analog dazu muss bei der Entwicklung kooperativer Anwendungen ein Kommunikationssystem gegeben sein, das den Austausch von Anwendungsdaten ermöglicht. Es handelt sich hierbei nicht um ein reines Transportsystem, da neben dem eigentlichen Datenaustausch zusätzliche organisatorische Funktionen für die Teilnehmer- und Sitzungsverwaltung benötigt werden.

3.2.1 Mechanismen und Begriffe

In diesem Abschnitt soll durch die Definition von aufeinander aufbauenden Begriffen und Mechanismen der Umfang und die Funktion des Kommunikationssystems abgegrenzt werden.

Sitzung

Der logische und physische Zusammenhang, in dem Datenaustausch zwischen Teilnehmern stattfinden kann, wird hier als "Sitzung" bezeichnet. Analog zu Lehrveranstaltungen, die zeitlich, räumlich und organisatorisch abgegrenzte Einheiten darstellen, findet auf der Ebene des Kommunikationssystems eine entsprechende Einteilung durch das Konzept der Sitzung statt. Mehrere Sitzungen müssen dabei gleichzeitig und ohne ungewollte gegenseitige Beeinflussung stattfinden können.

Mit einer Sitzung sind verschiedene organisatorische Attribute verbunden. Diese umfassen:

- Adresse der Sitzung
- Liste der zugelassenen Teilnehmer
- Art der Teilnehmerauthentifizierung
- Offenheit für neue Teilnehmer
- ID des Vorsitzenden

Derartige Informationen können sich sowohl vor als auch im Laufe einer Sitzung ändern.

Um Teilnehmern die Einordnung einer Sitzung zu ermöglichen, sind zudem verschiedene Meta-Informationen erforderlich. Beispiele hierfür sind:

- Titel und Inhalt der Veranstaltung
- Name des Vortragenden
- Veranstaltungsort
- Dauer
- Voraussetzungen

Adresse

Damit Benutzer an eine Sitzung teilnehmen können, bedarf es einer innerhalb des Systems eindeutigen Identifizierungsmöglichkeit. Zu diesem Zweck wird jeder aktiven Sitzung eine Adresse zugeordnet, anhand derer Teilnehmer sich mit der Sitzung verbinden können.

Teilnehmer

Für die Teilnahme an einer Sitzung sowie für den Datenaustausch ist die Einführung des "Teilnehmer"-Begriffs notwendig. Er bezeichnet alle an eine Sitzung angemeldeten Benutzer. Sie stellen die Endpunkte der in einer Sitzung stattfindenden Kommunikation dar. Es wird eine innerhalb einer Sitzung eindeutige Identifizierungsmöglichkeit für Teilnehmer benötigt.

Um z.B. Zulassungsbeschränkungen bereits vor Beginn einer Sitzung erlassen zu können, ist es in manchen Fällen notwendig, eine "globale", sitzungsübergreifende Identifizierung von Teilnehmern zu gewährleisten. Eine derartige Identifizierung ermöglicht zudem die Zuordnung von weiteren Daten zu einzelnen Teilnehmern. Denkbar sind hierbei:

- Name des Teilnehmers
- Authentifizierungsmerkmale (siehe folgenden Abschnitt über Authentifizierung)
- Kontaktinformationen (z.B. Telefonnummer, EMail)
- Funktion (z.B. Beruf)
- Priorität innerhalb des Sitzungssystems. Diese Information ist im Zusammenhang der Teilnehmerverwaltung wichtig, um z.B. Professoren allein anhand ihrer Stellung innerhalb des Lehrbetriebs bestimmte Rechte und Rollen zukommen zu lassen.

Rollen

Der in Abschnitt 2.3, "Teilnehmerverwaltung" erwähnte Rollenbegriff erfordert eine Unterscheidung von Teilnehmern bereits auf der Ebene des Kommunikationssystems. So ist die Rolle des Sitzungsverwalters notwendig, da es nicht sinnvoll wäre, das Recht auf bestimmte organisatorische Aktionen, wie z.B. das Beenden einer Sitzung, jedem Teilnehmer zu gewähren.

Die Verwalterrolle wird üblicherweise durch den Dozenten einer Veranstaltung übernommen und muss zu diesem Zweck während des Sitzungsstarts festgelegt werden. Dies geschieht entweder explizit über eine Teilnehmer-ID oder implizit über die Prioritätsverteilung unter den anwesenden Teilnehmern. Denkbar ist zudem auch die Vergabe derartiger Rollen im Verlauf einer Sitzung mittels Abstimmungen, was besonders in Sitzungen unter gleichberechtigten Teilnehmern ohne vorherbestimmte Rollenverteilung sinnvoll sein kann.

Zulassungsverwaltung

Die im vorhergehenden Abschnitt erwähnte Rolle des Sitzungsverwalters erfordert entsprechende Zusatzrechte, die dem Inhaber der Rolle zur Verfügung stehen müssen. Diese Rechte umfassen unter anderem die Möglichkeit, Teilnehmer anhand der globalen ID zu Sitzungen zuzulassen oder auszuschließen, bzw. aus einer laufenden Sitzung zu entfernen.

Authentifizierung

Zum Zweck der Identifizierung von Teilnehmern muss ein Authentifizierungsmechanismus eingeführt werden, der die Zuordnung von sitzungsabhängiger zu globaler Teilnehmer-ID anhand der Überprüfung von Authentifizierungsmerkmalen (wie z.B. Benutzername und Passwort) ermöglicht.

Die Unterscheidung von sitzungsabhängiger und -übergreifender Teilnehmer-ID ist sinnvoll, um die Verwendung von Pseudonymen innerhalb von Sitzungen zu ermöglichen. Entsprechend darf die Zuordnung (und damit die Identifizierung von Teilnehmern) nicht automatisch allen Teilnehmern zugänglich sein.

Betreten und Verlassen von Sitzungen

Will ein Benutzer am Austausch von Daten teilnehmen, so muss sich dieser erst bei der entsprechenden Sitzung anmelden. Dabei sind zwei grundlegende Aktionen erforderlich: das Betreten und Verlassen von Sitzungen. Das Betreten erfolgt anhand der Adresse einer Sitzung. Je nach Sitzungsvorgabe erfolgt dabei die Authentifizierung sowie die Überprüfung der Zulassung des Teilnehmers.

Nachricht

Abgesehen von den eben beschriebenen organisatorischen Aspekten der Kommunikation sind weitere Begriffe für die Handhabung des eigentlichen Datenaustauschs notwendig. Hierzu gehört als zentrales Element die *Nachricht*. Sie umfasst neben dem eigentlichen Dateninhalt auch Metadaten über die Kommunikation selbst, wie z.B. der ID des Absenders.

Kanäle

Zur Unterscheidung verschiedener Datenströme innerhalb einer, bietet sich die Strukturierung der Datenübertragung in Kanäle an. Es handelt sich hierbei um eine rein logische Unterteilung (im Gegensatz zu Sitzungen, deren Abgrenzung auch physischer Natur sein kann), weshalb die Einführung einer entsprechenden Identifikation genügt, die sowohl beim Senden, als auch beim Empfang von Nachrichten den betroffenen Kanal benennen.

Senden und Empfangen von Nachrichten

Die grundlegende Funktion, die durch das Kommunikationssystem bereitgestellt wird, ist die Übertragung von Nachrichten zwischen Teilnehmern einer Sitzung innerhalb eines Kanals. Hierbei sind neben *Unicast*- auch *Broadcast*- und evtl. *Multicastmechanismen* erforderlich, um die Bandbreitennutzung für die Teilnahme von mehr als 20 Teilnehmern zu optimieren.

Die Gegenstelle einer Kommunikation muss analog dazu in der Lage sein, ankommende Nachrichten zu empfangen. Es kann zwischen synchronem (aktiv wartend) und asynchronem (passiv wartend) Empfang unterschieden werden.

Ereignisse

Teilnehmer müssen bei Bedarf über Veränderungen im Zustand des Kommunikationssystems in Kenntnis gesetzt werden. Entsprechende Ereignisse informieren z.B. über das Beenden der Sitzung oder das Beitreten eines neuen Teilnehmers.

3.2.2 Schnittstellen der Sitzungsverwaltung

Die durch das Kommunikationssystem zu implementierende Funktionalität lässt sich in Aktionen unterteilen, die zum einen ausschließlich durch Teilnehmer mit Sitzungsverwaltungsrecht ausgeführt werden dürfen und zum anderen allen Teilnehmern zur Verfügung stehen.

Die an dieser Stelle definierten Mechanismen des Kommunikationssystems ähneln dabei der Funktionalität der in Abschnitt 3.3 behandelten Sitzungsverwaltung. Der Aufgabenbereich dieser übergeordneten Komponente geht jedoch über die hier beschriebene Regelung des blossen physischen Datenaustauschs hinaus und erstreckt sich unter anderem auf die Verwaltung ganzer Anwendungen.

Im folgenden werden die Sitzungsverwaltungsaktionen definiert:

Sitzung Erstellen

Durch das Erstellen einer neuen Sitzung wird eine Adresse festgelegt, anhand derer es Teilnehmern ermöglicht wird, sich mit dieser Sitzung zu verbinden. Startparameter bestimmen hierbei die Werte grundlegender Sitzungseigenschaften. Dies sind die Einstellungen über die Notwendigkeit von Teilnehmerauthentifizierung, die Möglichkeit der Verwendung von Pseudonymen sowie die Festlegung des Sitzungsverwalters. Bei dem letzten Punkt handelt es sich um eine Rollenzuweisung, die prinzipiell auch während der Sitzung abgeändert werden kann.

Sitzung Beenden

Bei Beendigung einer Sitzung werden alle vorhandenen Kommunikationskanäle geschlossen, sowie alle Teilnehmer abgemeldet. Ebenso wird die Adresse der Sitzung ungültig.

Kanal erstellen

Dient dem Erstellen eines neuen Kommunikationskanals mit einem sitzungswweit eindeutigen Namen. Durch das Erstellen eines Kanals wird es Sitzungsteilnehmern ermöglicht, sich mit diesem Kanal zu verbinden, um darauf Daten zu senden und zu empfangen.

Kanal schließen

Schließt einen durch einen Namen identifizierten, bestehenden Kanal. Alle zuvor verbundenen Teilnehmer werden vom betroffenen Kanal getrennt. Eine weitere Kommunikation darüber ist nicht mehr möglich.

Zulassungsbeschränkungen verwalten

Die Beschränkung des Zugangs zu einer Sitzung erfolgt entweder durch Auflisten aller zugelassenen Personen oder in umgekehrter Weise durch die Angabe der nicht zugelassenen Benutzer.

Teilnehmer entfernen

Da die Zulassungsbeschränkung für Sitzungen nur zum Anmeldezeitpunkt wirksam ist, wird zusätzlich die Möglichkeit des aktiven Entfernens von Teilnehmern aus einer Sitzung notwendig.

3.2.3 Schnittstellen der Sitzungsteilnehmer

Die folgenden Aktionen sind auch für Teilnehmer relevant, die nicht die Rolle der Sitzungsverwaltung einnehmen:

Sitzung Beitreten

Durch Angabe der Adresse ist es Benutzern möglich, einer bestehenden Sitzung beizutreten. Als Parameter ist hierbei eventuell die Angabe von Authentifizierungsmerkmalen (z.B. Benutzername und Passwort) notwendig. Das Beitreten zu einer Sitzung ist eine notwendige Vorbedingung für die Teilnahme am Datenaustausch, der nicht direkt in einer Sitzung, sondern erst in Kanälen dieser Sitzung stattfindet.

Sitzung Verlassen

Beendet die Teilnahme an einer Sitzung. Alle offenen Kanäle werden geschlossen, so dass ein weiterer Datenaustausch durch diesen Teilnehmer nicht mehr möglich ist.

Interesse an Ereignissen registrieren lassen

Hierbei wird durch ein entsprechendes System die Möglichkeit geboten, grundlegende Ereignisse über Änderungen des Sitzungszustandes zu empfangen. Dazu zählen Meldungen über:

- Verbindungsfehler
- Sitzungsende (durch Aktion des Sitzungsverwalters)
- Ende der Sitzungsbeteiligung (durch Abmelden oder durch Entferntwerden)
- An- und Abmelden von anderen Teilnehmern

Zusätzlich sind weitere Ereignisarten denkbar, die nicht direkt mit dem Kommunikationssystem zusammenhängen. Dazu gehören z.B.:

- Änderung des Angebots an Diensten und Anwendungen
- Änderung der Sitzungseigenschaften (Meta-Informationen)
- Änderungen bei Rollenverteilungen

3.2.4 Schnittstellen für den Datenaustausch

Der eigentliche Datenaustausch zwischen Teilnehmern erfolgt unter Verwendung der hier beschriebenen Aktionen:

Kanal beitreten

Um am Datenaustausch innerhalb eines Kanals teilzunehmen, muss ein Teilnehmer diesem beitreten. Voraussetzung hierbei ist die Angabe des Kanalnamens.

Kanal verlassen

Durch diese Aktion verlässt ein Teilnehmer einen Kanal. Eine weitere Kommunikation ist erst nach erneutem Beitreten wieder möglich.

Senden

Sendet eine Nachricht innerhalb eines Kanals gezielt an einen oder mehrere Teilnehmer.

Empfangen

Ermöglicht den Empfang von Nachrichten innerhalb eines Kanals. Das eigentliche Empfangen kann sowohl synchron (aktives Warten) als auch asynchron (passives Warten) erfolgen.

Die hier genannten Begriffe, Mechanismen und Schnittstellen des Kommunikationssystems dienen als Grundlage für weitere Komponenten, die im folgenden beschrieben werden.

3.3 Sitzungsverwaltung (Session Management)

Die Sitzungsverwaltungskomponente ermöglicht Benutzern den Zugriff auf grundlegende Funktionen, die für den organisatorischen Ablauf von Sitzungen wichtig sind.

Allgemein

- Abfrage der Sitzungseigenschaften gemäß Abschnitt 3.2.1. Hierzu gehören neben Meta-Informationen auch die Einstellungen der Authentifizierungsart und die Entscheidung darüber, ob neue Teilnehmer zugelassen werden sollen.
- Start von Sitzungen.
- Beenden einer Sitzung.
- Verwalten einer zentralen Liste laufender Sitzungen.
- Verwalten der Zulassungsbeschränkung vor und während einer Sitzung.
- Behandlung von kritischen Fehlern.
- Bereitstellung der Möglichkeit von "Offline-Sitzungen", d.h. dem wiederholten Abspielen von Daten aus bereits beendeten Sitzungen.

Anwendungen

- Starten von Anwendungen
- Beenden von Anwendungen
- Verwalten einer Liste von verfügbaren und bereits gestarteten Anwendungen

Teilnehmer

- Benutzungsschnittstelle für die Anmeldung von Teilnehmern
- Festlegung der Sitzposition einzelner Teilnehmer. Mit dieser Information soll die Zuordnung von Teilnehmer-ID zu anwesenden Personen für den Vortragenden, z.B. im Fall von Wortmeldungen, erleichtert werden.
- Verwalten einer Teilnehmerdatenbank, welche neben personenbezogenen Daten auch die Teilnehmerpriorität für Abstimmungen im Rahmen der Rechtevergabe (siehe Abschnitt 3.4, "Rechtevergabe") beinhaltet

Eine prototypische Realisierung dieser Komponente erfolgte in [SOMM01].

3.4 Rechtevergabe

Ein Rechtevergabesystem (engl. *Floor Control*) dient der zeitweiligen Vergabe von Zugriffs- und Modifikationsrechten für bestimmten Ressourcen. Angelehnt an [HILT] können die Hauptaufgaben eines derartigen Systems wie folgt zusammengefasst werden:

- Verringerung von Redundanzen und Inkonsistenzen und dem daraus resultierenden Nicht-Determinismus im Rahmen von Gruppenarbeit.
- Erreichen einer ausgeglichenen Verteilung von Teilnehmerbeiträgen innerhalb von Sitzungen.
- Regelung der Zusammenarbeit durch für alle Teilnehmer bindende und nachvollziehbare Vorgehensweisen.
- Verstärkung des Gruppenverständnisses und -zusammenhaltes.

Je nach Art und Weise, auf welche die Benutzung einer Ressource an das Rechtevergabesystem gekoppelt ist, unterscheidet man die folgenden Fälle der Vergabepolitik (*Floor Policy*):

- Keine Kontrolle.
- Implizite Kontrolle. Bei der Verwendung der Ressource wird das Recht automatisch beantragt und später wieder selbstständig freigegeben.
- Explizite Kontrolle. Vor der Verwendung muss ein Teilnehmer das Recht beantragen. Nach der Benutzung muss es explizit wieder freigegeben werden.
- Zentrale Kontrolle durch Teilnehmer in der Vorsitzrolle.

Zur Benutzung des Vergabesystems sind verschiedene grundlegende Aktionen erforderlich:

- Beantragung des Rechtes durch einen Teilnehmer. Entsprechende Anträge müssen für andere Teilnehmer sichtbar sein, besonders im Fall von geleiteten Sitzungen.
- Rücknahme eines Rechte-Antrags.
- Rückgabe des Rechtes.
- Entzug des Rechtes.
- Vergabe durch einen Vorsitzenden. Hierbei muss die Vergabe auch dann möglich sein, wenn kein Antrag durch den betroffenen Teilnehmer vorliegt.

Ein System, das den Großteil der hier beschriebenen Funktionalität umfasst wurde in [OBER01] entworfen und implementiert. Eine automatische Zuordnung zwischen Recht und Ressource wird dabei nicht forciert, es handelt sich nur um den reinen Rechtevergabe-Mechanismus. Erst durch die Integration in eine Anwendung wird der eigentliche Sinn der Zugriffsordination erfüllt. Zu diesem Zweck kann eine Anwendung auch gleichzeitig mehrere unterschiedliche *Floor Controls* eröffnen.

Ein einzelnes "Recht" wird durch einen sitzungswelt eindeutigen Namen identifiziert. Teilnehmer haben die Möglichkeit, dieses Recht zu beantragen und abzugeben. Durch die Wahl der Vergabepolitik kann pro *Floor Control* festgelegt werden, ob die Zugriffsordination

zentral durch einen Vorsitzenden moderiert wird, oder ob das Recht automatisch nach dem *first-come-first-served*-Prinzip vergeben wird. Entsprechend ergeben sich in der Implementierung die folgenden Schnittstellen für Teilnehmer:

- Recht beantragen
- Recht abgeben
- Antrag auf Recht zurückziehen
- Recht weitergeben (falls erlaubt)

Für den Inhaber der Vorsitzenden-Rolle stehen diese zusätzlichen Funktionen zur Verfügung:

- Recht vergeben
- Recht entziehen
- Maximale Zahl der gleichzeitig Redeberechtigten (*Floor-Size*) festlegen
- Vergabepolitik festlegen

Da die Rechteverwaltung ohne Einbeziehung des Ressourcenbegriffs, ausgelegt wurde, muss die Zuordnung von Recht zu Ressource explizit durch die Anwendung vorgenommen werden. Die Überprüfung der Berechtigung eines Teilnehmers erfolgt direkt durch den entsprechenden Anwendungsteil. Entsprechend besteht die Möglichkeit, den Rederecht-Zustand von Teilnehmern abzufragen.

Zusätzlich zur eigentlichen Rechteverwaltung, stellt diese Komponente die Möglichkeit von Abstimmungen zur Verfügung. Es erfolgt eine Stimmgewichtung entsprechend der in der Teilnehmerdatenbank (siehe Abschnitt 3.3) gespeicherten Priorität. Teilnehmer können in diesem Zusammenhang die folgenden Aktionen auslösen:

- Abstimmung erstellen (bei entsprechender Berechtigung)
- Abstimmung abrechnen (bei entsprechender Berechtigung)
- Pro- oder Contra-Stimmen abgeben oder sich enthalten

Durch Teilnehmeraktionen verursachte Änderungen im Zustand der *Floor Control* und von Abstimmungen werden asynchron über ein Ereignisverteilungssystem an alle Teilnehmer gemeldet.

Das ursprünglich nur für atomare *Floor Controls* entwickelte System wurde in einer eigenständigen Arbeit (siehe [BAI01]) um das Prinzip der *Sub-Floors* erweitert. Durch diese Erweiterung wird es ermöglicht, ein einzelnes Recht in Teil-Rechte aufzuspalten, die dann nach den bereits beschriebenen Prinzipien verwaltet werden.

3.5 Protokollierungsdatenbank

Für die Speicherung des während einer Interaktion mehrerer Teilnehmer produzierten Datenaufkommens wurde in [SCHMID01] eine entsprechende Anbindung an eine Datenbank entwickelt. Diese implementiert Aspekte einer verteilten Datenbank. Es wird ermöglicht, dass öffentliche, d.h. für alle Teilnehmer gedachte Daten, durch die Speicherung automatisch an alle verteilten Datenbankkopien des Systems verteilt werden.

Der Funktionsumfang der Datenbankanbindung umfasst:

- Verwalten von anwendungsabhängigen Daten unter Berücksichtigung der zeitlichen Reihenfolge der Speicherung.
- Zugriffsmöglichkeit per Datenbankanfrage.
- Einteilung in öffentliche und private Daten mit entsprechender Auswirkung auf die automatische Verteilung.
- Automatisches Kopieren der öffentlichen Daten auf die Teilnehmerrechner.
- Realisierung einer Nachzügler- und Fehlerbehandlung, mit deren Hilfe fehlende Daten im nachhinein automatisch von einer zentralen Datenbank bezogen werden.
- Implementierung anhand eines frei verfügbaren und plattformunabhängigen Datenbanksystems.

3.6 Einbindung von Anwendungen

Die Einbindung von Anwendungen zum Zweck der kooperativen Nutzung erfolgt über einen sogenannten "Anwendungsrahmen". Es müssen dabei zwei Arten von Anwendungen unterschieden werden:

1. **Interne Anwendungen**, die in ihrem Quelltext anpassbar sind und direkt auf die bereitgestellte Architekturfunktionen zugreifen können. Der letztere Punkt beinhaltet dabei, dass die Implementierungssprache der Anwendung und der Architektur kompatibel sein müssen.
2. **Externe Anwendungen**, die nur über die Benutzungsschnittstelle oder über eine begrenzte Programmierschnittstelle steuerbar sind und bei denen Änderungen im Quelltext nicht ohne weiteres möglich sind.

3.6.1 Interne Anwendungen

Damit eine interne Anwendung integriert werden kann, muss sie zum einen selbst spezielle Schnittstellen anbieten, um für die Sitzungsverwaltung handhabbar zu werden. Zum anderen sollten die Funktionen des Systems in einer möglichst kompakten Weise zusammengefasst und gekapselt werden.

Schnittstellen der Systemarchitektur

Diese Schnittstellen ermöglichen Anwendungen, auf Systemfunktionen zuzugreifen. Hierbei muss die Verfügbarkeit der internen Komponentenschnittstellen eingeschränkt werden, um den versehentlichen oder mutwilligen Missbrauch dieser Schnittstellen zu verhindern. Am Beispiel des Kommunikationssystems verdeutlicht bedeutet dies, dass das Erstellen eines neuen Kommunikationskanals in jeder Anwendung möglich sein muss, wohingegen das Beenden der Teilnahme an einer Sitzung nur der Sitzungsverwaltung erlaubt sein darf.

Im folgenden werden die notwendigen Funktionen zusammengefasst, wobei entsprechend Abschnitt 2.8 eine Unterteilung in Anbieter- und Nutzerschnittstellen erfolgt.

Dienstanbieter

- Verwalten von Kommunikationskanälen (siehe Abschnitt 3.2, “Kommunikationssystem”)
- Verwalten von *Floor Control*-Instanzen (siehe Abschnitt 3.4, “Rechtevergabe”)
- Eröffnen und Beenden einer neuen Protokollierungsdatenbank (siehe Abschnitt 3.5, “Protokollierungsdatenbank”)

Dienstnutzer

- Anmelden und Benutzen (z.B. Senden und Empfangen) eines bestehenden Kommunikationskanals
- Anmelden und Benutzen (z.B. Recht beantragen und abgeben) einer bestehenden *Floor Control*
- Anmelden und Benutzen (z.B. Daten speichern und auslesen) einer bestehenden Protokollierungsdatenbank

Schnittstellen der Anwendung

Um für die Architektur handhabbar zu werden, müssen Anwendungen als Minimum 2 Funktionen bereitstellen: Anwendungsstart und-stopp.

Anwendungsstart

Hierdurch wird der Eintrittspunkt für die Ausführung festgelegt. Das System übergibt beim Aufruf dieser Funktion neben anwendungsspezifischen Parametern auch eine Referenz auf die zuvor beschriebene Systemschnittstelle. Der Rückgabewert dieser Funktion teilt der Sitzungsverwaltung mit, auf welche Art die Anwendung beendet wurde. Dies kann zum einen das Resultat einer Benutzerinteraktion sein, es ist aber ebenso das Beenden aufgrund eines kritischen Fehlers möglich, z.B. bei nicht ausreichendem Speicherplatz.

Die Sitzungsverwaltung hat die Möglichkeit, anhand des Rückgabewertes unterschiedlich zu reagieren. Denkbar wäre die Benachrichtigung des Benutzers, damit dieser die Ursache für das Fehlverhalten beseitigen kann. Bei Verwendung des Protokollierungsmechanismus ist zudem nach der Fehlerbehebung ein Wiederanlauf der Anwendung mit den bereits gespeicherten Daten möglich.

Signalisieren des Anwendungsstopps

Diese Funktion wird von der Sitzungsverwaltung dazu verwendet, der Anwendung ein planmäßiges Ende zu ermöglichen indem diese als Reaktion alle belegten Ressourcen freigibt und noch nicht gespeicherte Daten sichert. Sie wird typischerweise bei Beendigung des gesamten Systems aufgerufen. Da eine korrekte Reaktion der Anwendung durch das System nicht gewährleistet werden kann, muss ein entsprechendes Vorgehen durch einen Timeout-Mechanismus mit einem "harten" Anwendungsstopp verbunden werden (denkbare wären hier betriebssystem- und programmiersprachenabhängige Mittel, um Prozesse oder Threads abubrechen).

3.6.2 Externe Anwendungen

Anwendungen, die nicht speziell für die Architektur entwickelt wurden, bzw. bei denen Änderungen des Quelltextes nicht möglich sind, können prinzipiell auf zwei verschiedene Arten in das System integriert werden:

Bereitstellung einer allgemeinen Programmierschnittstelle

Hierbei muss es möglich sein, die Funktion der Anwendung, die kooperativ genutzt werden soll, programmatisch zu steuern. Eine Integration erfolgt in diesem Fall über ein Zwischenprogramm (*Wrapper*), das unter Verwendung der Systemarchitektur allen Teilnehmern die Benutzung der Funktionen der Anwendung ermöglicht und gleichzeitig den Zugriff koordiniert.

Ein Beispiel für die Verwendung derartiger programminterner Schnittstellen ist die in [NGUY00] realisierte Kontrolle von *Microsoft Powerpoint* aus einer Java-Anwendung heraus.

Fernsteuerung der Benutzungsschnittstelle

In Fällen, in denen weder Quelltext noch spezielle Anwendungsschnittstellen vorhanden sind (wie dies beim Großteil der kommerziell verfügbaren Software der Fall ist), ist eine Integration nur über die Fernsteuerung der Benutzungsschnittstelle selbst möglich. Hierbei ist es während der Interaktion notwendig, den Zustand der Benutzungsoberfläche auf andere Rechner zu übertragen. Desweiteren müssen auf entfernten Rechnern getätigte Eingaben der eigentlichen Anwendung zugeführt werden.

Beispiele für diese Vorhergehensweise sind das X Window System, die *Desktop-Sharing*-Möglichkeit bei Microsoft Netmeeting und dem Virtual Network Computing-System (siehe auch [VNC]).

Bei der Integration von externen Anwendungen muss im Einzelfall entschieden werden, wie hoch die Granularität der unterstützten Interaktion sein soll, d.h. ob es sinnvoll ist, jeden einzelnen Schritt (z.B. in Form eines einzelnen Tastendrucks) zu übertragen oder ob die Gesamteingabe genügt.

Als zukünftige Entwicklung ist denkbar, dass die elektronische Tafel, die Bestandteil dieser Arbeit ist, mit einem der bereits bestehenden *Desktop-Sharing* Systeme integriert wird. Hierdurch würde der zuletzt beschriebenen Fall der Fernsteuerung beliebiger Anwendungen abgedeckt und um die Möglichkeit des Zeigens und des Annotierens der Ausgaben erweitert werden.

3.7 Datenfluss

3.7.1 Komplett integrierter Datenfluss

Durch die Benutzung aller bereitgestellten Funktionen ergibt sich innerhalb des Systems ein Datenfluss, wie er in Abbildung 3-2 skizziert wird. Dabei ist diese Struktur keineswegs starr vorgegeben. In den folgenden Abschnitten werden unterschiedliche Kombinationen von Anwendung, *Floor Control* und Datenbank beschrieben.

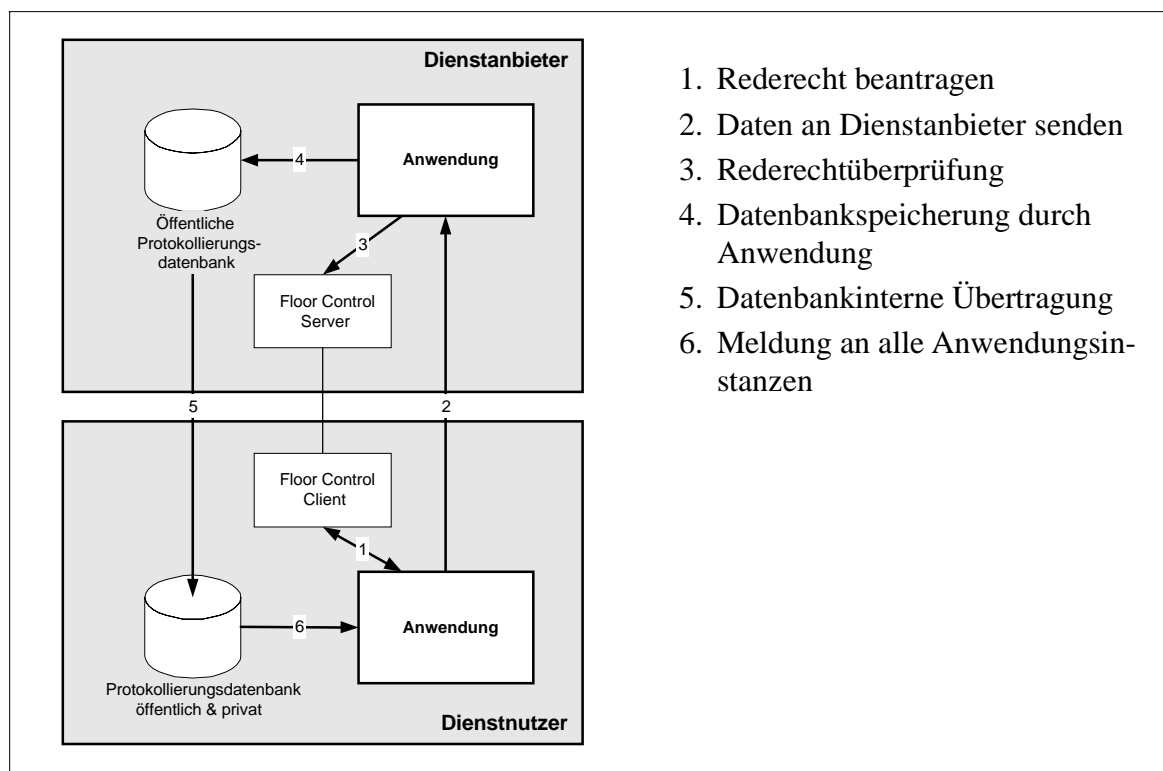


Abbildung 3-2: Komplett integrierter Datenfluss zwischen Dienstanbieter und -nutzer

Aufgrund der Tatsache, dass die Vergabe des Rederechts je nach gewählter Vergabepolitik und -situation (vergleiche Abschnitt 3.4) nicht direkt auf den Antrag folgen muss, besteht der in der Abbildung skizzierte Schritt 1 genaugenommen aus 2 Phasen:

1. Stellen des Antrags
2. Erhalt der Berechtigung mittels asynchronem Ereignis, ausgelöst entweder durch eine Vorsitzendenaktion oder durch den entsprechenden Vergabemechanismus

Die Benutzung der *Floor Control* erfolgt *clientseitig* durch ein eigenes Zugriffsobjekt, das in der Skizze als zusätzliche Komponente repräsentiert wird. Dies dient jedoch lediglich der Kommunikation - der eigentliche Mechanismus wird zentral durch die Server-Komponente realisiert.

Aus der abgebildeten Struktur ergeben sich folgende Vorteile:

- Geregelter Zugriff auf zentrale Ressourcen.
- Speicherung der Kommunikation zur
 - nachträglichen Verarbeitung
 - Unterstützung von zu spät kommenden Teilnehmern durch die automatische Verteilung der bereits ausgetauschten Daten
 - direkten Wiederaufnahme der Teilnahme an einer Sitzung nach einem Fehlerfall

Als nachteilig wirkt sich aus, dass hier eine sternförmige Topologie durch das Vorhandensein eines expliziten Servers erzwungen wird - bei der Verwaltung des Zugriffs auf eine ortsgebundene Ressource (z.B. ein fest-installierter Projektor) wird sich dieses Problem jedoch nicht umgehen lassen. Zudem stellt die Verwendung aller Komponenten einen gewissen organisatorischen Mehraufwand dar.

Da die Zugriffskontrolle separat vom eigentlichen Kommunikationskanal definiert ist und somit das Überprüfen des "Rederechts" und das eigentliche Senden nicht atomar erfolgt, kann es passieren, dass sich die Zugriffssituation zwischen Schritt 1 und 2 ändert. In Protokollen, in denen ein Dienstanutzer auf eine Antwort des Dienstansbieters wartet, kommt es in diesem Fall zu Komplikationen (z.B. in Form von endlosem Warten auf eine nicht stattfindende Antwort des Servers), falls der zwischenzeitliche Verlust des Rederechts nicht beachtet wird.

3.7.2 Datenfluss ohne Protokollierung

Falls eine Protokollierung der bereits ausgetauschten Daten nicht notwendig oder gewünscht ist, kann die Kommunikation auch vollständig über den eigentlichen Kanal erfolgen (Abbildung 3-3).

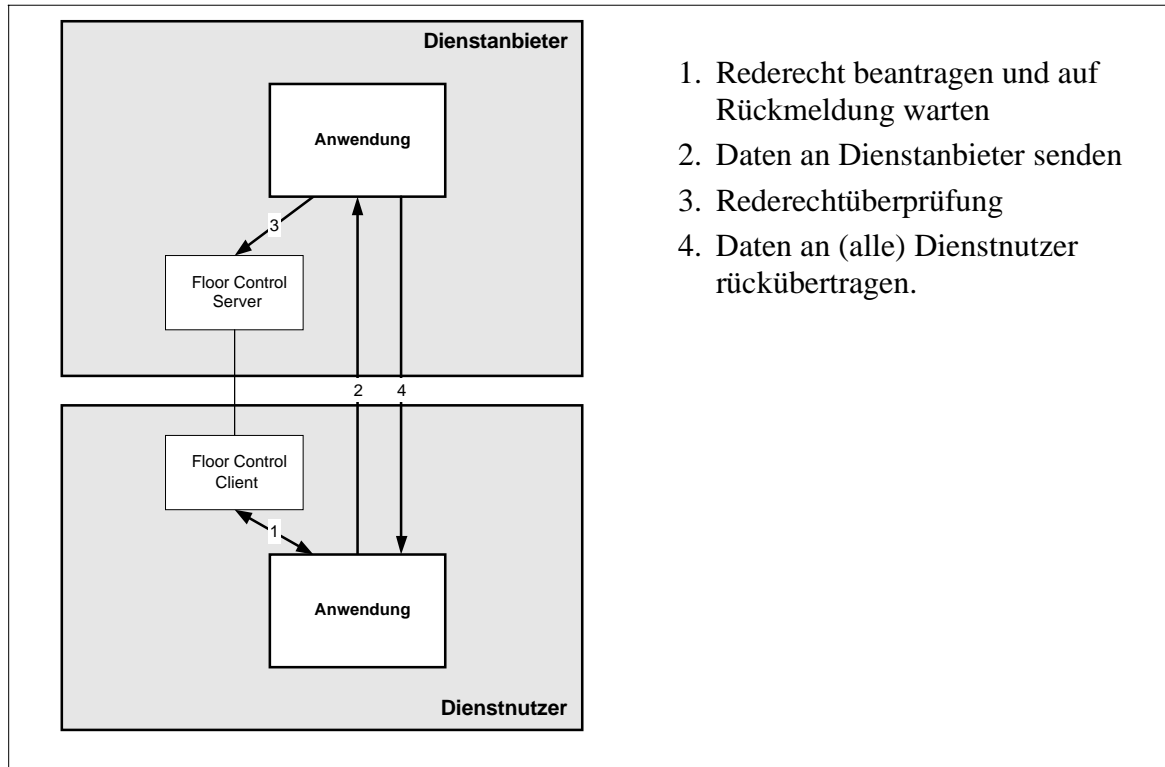


Abbildung 3-3: Datenfluss ohne Protokollierung

Der Mehraufwand für die Speicherung und Verteilung der Daten über den Datenbankkommunikationskanal entfällt hierbei auf Kosten der automatischen Nachzügler- und Fehlerbehandlung, also des wiederholten Sendens fehlender Datensätze.

3.7.3 Ungeregelter Datenfluss

Durch die alleinige Verwendung eines Kommunikationskanals ohne Hinzunahme einer Zugriffsverwaltung oder Protokollierung ist zudem ein unregelmäßiger Datenfluss innerhalb einer Sitzung möglich:

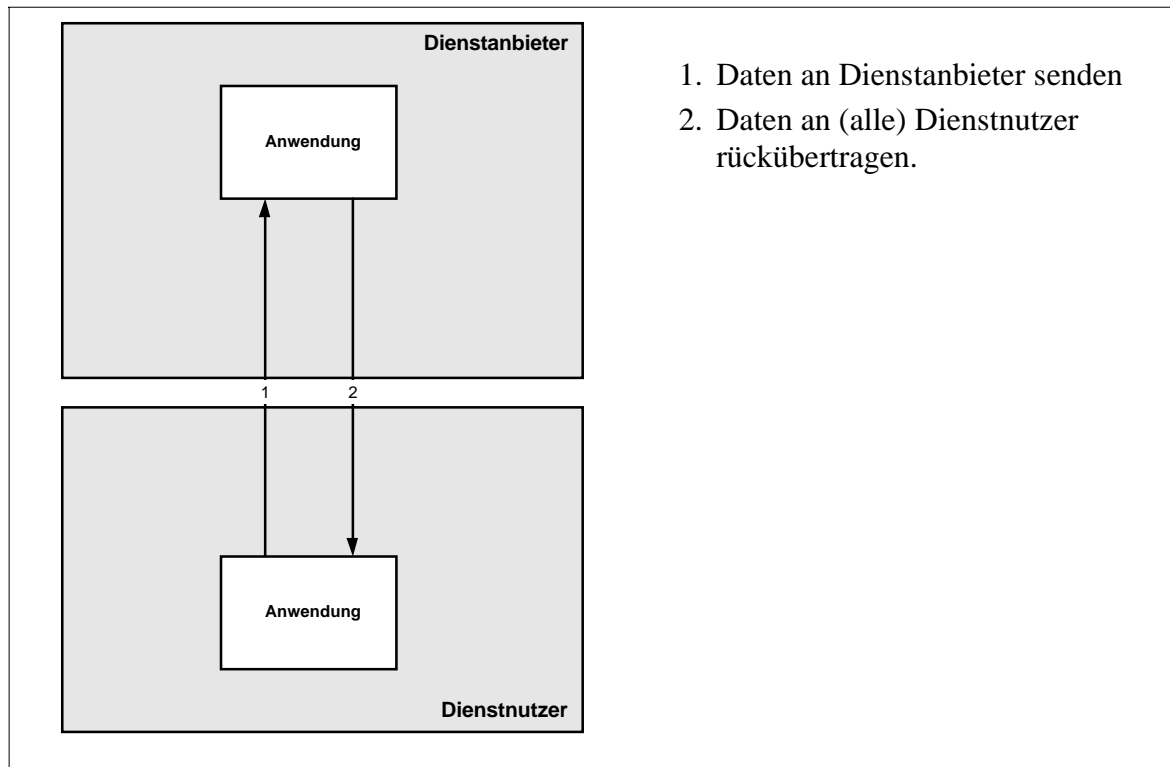


Abbildung 3-4: Ungeregelter Datenfluss

In diesem Fall ist es, aufgrund fehlender zentraler Komponenten, prinzipiell möglich, eine komplett dezentrale Kommunikationsstruktur aufzubauen (z.B. durch eine dynamische Zuordnung der Dienstanwender- und -anbieterrolle). Der ungewollte gleichzeitige Zugriff mehrerer Teilnehmer auf eine Ressource muss in diesem Fall, ebenso wie eine eventuell gewünschte Speicherung der Daten, von der Anwendung selbst geregelt werden.

4 Umsetzung

4.1 Java

Um der Anforderung der Plattformunabhängigkeit bereits auf der Ebene der Programmiersprache zu genügen, empfiehlt sich die Verwendung von *Java* als Implementierungssprache.

Besonders vorteilhaft ist hierbei die freie Verfügbarkeit von umfangreichen Klassenbibliotheken, die Teile der benötigten Funktionalität, beispielsweise in Form von Netzwerk- und Benutzungsoberflächenkomponenten, bereitstellen.

4.2 JSDT

Die vorhergehende Untersuchung existierender Systeme in [OBER01] ergab, dass für Java bereits ein System verfügbar ist, das die Kommunikation von Teilnehmern in Gruppen und Kanälen ermöglicht. Hierbei handelt es sich um das *Java Shared Data Toolkit*, kurz *JSDT*, von *Sun* [JSDTWEB].

JSDT dient dabei als Zwischenschicht zwischen einer Anwendungsprogrammierungsschnittstelle, die Objekte wie Sitzungen, Kanäle und Teilnehmer realisiert und verwaltet, und der darunterliegenden, austauschbaren Transportschicht. Die Übertragung der eigentlichen Daten erfolgt dabei wahlweise unter Benutzung einer der folgenden Kommunikationsmechanismen:

- TCP und UDP
- HTTP-getunnelte Sockets
- Eine Multicast-Protokoll-Implementierung namens *LRMP* (*Lightweight Reliable Multicast Protocol*, siehe [INRIA]).

Die darüberliegende Schnittstelle definiert unter anderem die folgenden Objekte:

Registry

Hierbei handelt es sich um eine zentrale Datenbank, die für die Verwaltung aller JSDT-Objekte notwendig ist.

Client

In JSDT werden *Clients* als Objekte definiert, die als Endpunkte bei einem Mehrpunkt-Datenaustausch dienen. *Clients* werden zudem benötigt, um *Sessions* und *Channels* benutzen zu können.

Session

Sessions dienen der Gruppierung von *Clients* zum Zweck des Datenaustauschs. Eine *Session* wird durch einen *Client* eröffnet und erhält dabei eine eindeutige Adresse. Mittels dieser Adresse können sich weitere *Clients* verbinden. Nur innerhalb einer *Session* können teilnehmende *Clients Channels* eröffnen und diesen beitreten.

Channel

Ein *Channel* dient dem eigentlichen Datenaustausch zwischen *Clients* einer *Session*. Zum Zeitpunkt des Erstellens wird einem *Channel* durch den erstellenden *Client* ein beliebiger, sitzungswelt eindeutiger Name zugeordnet. Dieser Name ermöglicht es anderen *Clients*, dem *Channel* beizutreten. Bei der Adressierung von Nachrichten wird sowohl das Senden an alle, als auch gezielt an einzelne *Clients* unterstützt.

Manageable Objects

Der Begriff der *Manageable Objects* beschreibt die Möglichkeit, vor der Durchführung bestimmter Aktionen auf *Channels* und *Sessions* eine Handshake-Authentifizierung des durchführenden *Clients* vorzunehmen. Die *Channels* und *Sessions* sind in diesem Zusammenhang die *Manageable Objects*, mit denen entsprechende *Manager*-Instanzen assoziiert werden können. Diese Instanzen wiederum überprüfen bei der Durchführung bestimmter Aktionen die Berechtigung der Teilnehmer. Das dabei verwendete Handshake-Verfahren läuft folgendermaßen ab:

1. Die *Manager*-Instanz übergibt eine *Challenge* in Form eines beliebigen Objektes an den entsprechenden *Client* (z.B. eine Aufforderung, sich mit einem Passwort zu authentifizieren).
2. Der *Client* antwortet mit einem Objekt von beliebigem Typ (z.B. dem geforderten Passwort).
3. Die *Manager*-Instanz entscheidet anhand der *Client*-Antwort, ob die Aktion durchgeführt werden darf.

Die Aktionen, die auf diese Weise kontrolliert werden können, umfassen das Erstellen und Entfernen sowie das Beitreten zu *Sessions* und *Channels*.

4.3 Kommunikationssystem

Die hier kurz skizzierte Funktionalität von JSDT erfüllt bereits einen Großteil der in Abschnitt 3.2.1 aufgestellten Forderungen an das Kommunikationssystem. Zwischen den Objekten von JSDT und den Begriffen aus dem Entwurf können die folgenden Beziehungen aufgestellt werden:

1. **Session.** Entspricht einer Sitzung. Jede aktive *Session* in JSDT wird durch eine URL identifiziert. Diese Angabe kann von *Clients* dazu benutzt werden, der *Session* beizutreten, was den Anforderungen an die Sitzungsadresse entspricht.
2. **Client.** *Clients* werden innerhalb einer Sitzung durch einen Namen eindeutig identifiziert und sind notwendig, um an einer Sitzung und an Datenaustausch teilnehmen zu können. Sie erfüllen damit die für den Teilnehmer-Begriff definierten Eigenschaften.
3. **Channel.** Analog zur Kanal-Definition ermöglichen *Channels* die Strukturierung der Kommunikation in logisch abgegrenzte Einheiten, in denen Daten von *Clients* gesendet und synchron oder asynchron empfangen werden können. Die Implementierung garantiert hierbei eine zuverlässige und geordnete Übermittlung der einzelnen Pakete.
4. **Manageable objects.** Die Manager-Schnittstelle der zuvor genannten JSDT-Objekte ermöglicht durch das dynamische Überprüfen der Berechtigung, die Implementierung von Authentifizierung, Zulassungsbeschränkung und benötigten Rollen. Je nach Kontext werden hier die Aktionen "Beitreten zu Sitzungen" und "Entfernen von Objekten" durch verschiedene Mechanismen abgesichert.

4.3.1 Diskussion der Systemstruktur

Bei Verwendung von JSDT für die Implementierung des zuvor entworfenen Kommunikationssystems werden unter anderem die Objekte *Registry*, *Session* und *Manageable Objects* benötigt. Hierbei handelt es sich um zentral verwaltete Komponenten. Es ist dabei nicht möglich, diese Objekte nach dem Start einer Sitzung von einem Teilnehmersystem auf ein anderes zu migrieren, ohne dabei die gesamte Sitzung zu beenden und neu zu starten. Eine dezentrale, *peer-to-peer* basierte Kommunikationstopologie ist aus diesem Grund nicht ohne weiteres umsetzbar.

Ein Lösungsansatz besteht darin, diese JSDT-Komponenten an einem zentralen Ort permanent verfügbar zu machen. Durch die Einführung eines derartigen *single-point-of-failures* ergibt sich jedoch eine Verschlechterung der Gesamtverfügbarkeit. Zudem wird hierdurch die Benutzung des Systems in einer Umgebung ohne vollständiger Netzanbindung (denkbar z.B. beim Betrieb eines *Wireless-Lan* Netzwerkes ohne Basis-Station) verhindert. Aus diesen Gründen empfiehlt es sich, die benötigten Objekte für die Dauer einer Sitzung lokal zu instanzieren und zu verwalten.

4.3.2 Wichtige Klassen und Schnittstellen

Das Vorhandensein von nur zentral verwaltbaren Objekten (*Registry*, *Session*) legt die Implementierung des Systems in einer Client-Server-Architektur nahe. In diesem Fall werden die Teilnehmer für die Dauer einer Sitzung in einen Server und mehrere Clients unterteilt, wobei der Server für die komplette Sitzungsverwaltung zuständig ist.

CommServer

Das *CommServer*-Objekt übernimmt die Verwaltung der JSDT-Komponenten, die für die Implementierung einer Sitzung nach der Definition des Begriffs in Abschnitt 3.2.1 notwendig sind. Hierzu zählen *Registry*, *Session* und *Client*, wobei der letztere in JSDT Voraussetzung für das Erstellen einer Sitzung ist. Ein mit der Session verbundenes *SessionManager*-Objekt regelt bei allen privilegierten Aktionen innerhalb der Sitzung den Zugriff.

Nach der Instanziierung des Objektes existiert noch keine Sitzung, d.h. der Server ist *offline*, bis er explizit durch den Aufruf einer entsprechenden *start*-Methode gestartet wird. Bereits in diesem Zustand kann die Gruppe der zugelassenen Teilnehmer über zwei Listen verwaltet werden. Dies ist zum einen die sogenannte *allowList* (zugelassene Teilnehmer) und zum anderen die *disallowList* (abzuweisende Teilnehmer). Im Standardfall, in dem beide Listen leer sind, werden alle Teilnehmer zugelassen, ansonsten erfolgt eine entsprechende Überprüfung, wobei die *allowList* eine höhere Priorität besitzt, als die *disallowList*. Der folgende Algorithmus findet hierbei Anwendung:

1. Sind beide Listen leer, werden alle Teilnehmer zugelassen
2. Befindet sich ein Teilnehmer in der *allowList*, wird der Teilnehmer zugelassen
3. Befindet sich ein Teilnehmer in der *disallowList* und nicht in der *allowList*, so wird dieser Teilnehmer abgewiesen.

Abgesehen von dieser teilnehmerbezogenen Zulassungsentscheidung, kann über eine Einstellung festgelegt werden, dass die Sitzung allgemein für neue Teilnehmer gesperrt ist. Dies ist z.B. sinnvoll, um bei der Auslastung der verfügbaren Kapazitäten eine Sitzung auf die bereits anwesenden Teilnehmer zu beschränken.

Wurden alle Einstellungen vorgenommen, kann die Sitzung mittels der *start*-Methode erstellt werden. Hierbei wird festgelegt, ob eine Authentifizierung mit der Benutzerdatenbank stattfinden soll, oder ob Teilnehmer direkt zugelassen werden. Diese Einstellung ist auch nach dem Start jederzeit änderbar.

Während des Betriebs ist es möglich, eine Liste der anwesenden Teilnehmer abzufragen sowie einzelne Teilnehmer aus der Sitzung wieder auszuschließen. Obwohl der serverseitig erstellte *Client* alle Teilnehmerfunktionen ausüben kann, wird dieser nicht in der entsprechenden Liste aufgeführt, da der Server innerhalb der Architektur einen Dienst, bzw. eine Ressource repräsentiert und damit nicht der Definition eines aktiven Sitzungsteilnehmers entspricht.

Die hier aufgelistete Funktionalität stellt die Grundlage für die Implementierung der in Abschnitt 3.3 beschriebenen Sitzungsverwaltung dar. Teil dieser Komponente ist ein Dienst, der die Benutzung der nur *serverseitig* verfügbaren Verwaltungsfunktionen durch alle Teilnehmer ermöglicht. Der Zugriff auf diesen Dienst wird dabei durch eine entsprechende *Floor Control*-Instanz geregelt.

CommAddress

Der eigentliche Startvorgang beinhaltet das Erstellen einer neuen *Registry* sowie einer *Session*. Es ergibt sich dabei eine Gesamtadresse, die durch ein `CommAddress`-Objekt gekapselt wird. Diese besteht aus *Registry*- und *Sessionteil* und ermöglicht es Teilnehmern, der Sitzung beizutreten.

CommSessionListener

Informationen über asynchron auftretende Änderungen im Zustand einer Session werden über ein *Listener*-System an den serverseitigen Teil einer Anwendung gemeldet. Das `CommSessionListener`-Objekt definiert die hierzu notwendige Schnittstelle. Es werden zwei Arten von Ereignissen unterschieden:

1. Teilnehmerereignisse. Diese umfassen das Beitreten, das Verlassen sowie das Entfernen von Teilnehmern der Sitzung.
2. Serverereignisse. Sollte der Kommunikationsserver aufgrund eines Fehlers beendet werden, wird dies allen mit diesem Server verbundenen Anwendungen über ein Serverereignis mitgeteilt.

Ähnlich dieser Ereignisverteilung werden ankommende Nachrichten ebenfalls über eine *Listener*-System verteilt. Die Schnittstellen dieses Systems werden im Abschnitt über das `CommChannel`-Objekt behandelt.

CommClient

Analog zur Funktion des `CommServers`, ermöglicht das `CommClient`-Objekt die Teilnahme an einer Sitzung. Um einer Sitzung beizutreten, muss die Sitzungsadresse angegeben werden. Zusätzlich ist der Name, sowie je nach Sitzungsanforderung ein Passwort zur Authentifizierung des Benutzers erforderlich. Das sich daraus ergebende `CommClient`-Objekt unterscheidet sich von einem `CommServer` nur durch die nicht vorhandenen sitzungsverwaltungsspezifischen Funktionen.

CommChannel

Die eigentliche Kommunikation innerhalb einer Sitzung erfolgt über das `CommChannel`-Objekt. Ein entsprechendes Objekt kann serverseitig durch das Erstellen eines neuen Kanals, bzw. durch das Beitreten eines Teilnehmers auf der Clientseite erfolgen. die

Objekte `CommServer` und `CommClient` stellen entsprechende `createChannel`, bzw. `joinChannel`-Methoden zur Verfügung, die lediglich den Namen des Kanals benötigen. Intern erfolgt eine direkte Abbildung auf das JSDT-*Channel*-Objekt.

Ein `CommChannel` stellt hierbei die folgenden Funktionen zur Verfügung:

- Abfrage von kanalbezogenen Informationen (z.B. Namen der anwesenden Teilnehmer).
- Verlassen, bzw. Entfernen des Kanals.
- Senden von Daten an alle oder nur an einzelne Teilnehmer. Die Adressierung erfolgt durch den jeweilige Namen des Teilnehmers.
- Senden von Daten an den Kanalverwalter. Diese Funktion ist notwendig, da dieser nicht in der Liste der Teilnehmer auftaucht und deshalb nicht anders adressiert werden kann.
- Registrieren von Objekten für den Empfang von kanalbezogenen Ereignissen.
- Registrieren von Objekten für den asynchronen Nachrichtempfang.
- Synchrones Empfangen von Nachrichten mit optionaler Angabe der Zeitdauer, die auf eine Nachricht gewartet werden soll.

CommChannelConsumer

Im Fall der asynchronen Datenkommunikation wird ein entsprechend registriertes Objekt ähnlich dem Listenersystem bei Sitzungsereignissen informiert. Hierzu wird die `CommChannelConsumer`-Schnittstelle verwendet, um bei Eintreffen einer neuen Nachricht diese allen registrierten Objekten über eine entsprechende Methode mitzuteilen.

CommChannelListener

Vergleichbar mit dem System der Sitzungsereignisse existiert eine Schnittstelle, um Anwendungen *kanalbezogene* Ereignisse zu melden. Diese Schnittstelle informiert über Änderungen in der Teilnehmergruppe sowie über das (eventuell fehlerbedingte) Beenden der Verbindung zu dem Kanal.

4.4 Elektronische Tafel (Whiteboard)

4.4.1 Anforderungen

Wie in Abschnitt 2.9 gefordert wurde, soll das Whiteboard die Möglichkeit bieten, Annotationen in Form von Freihandzeichnungen und Text auf beliebigen graphischen Inhalten vorzunehmen, sowie auf bestimmte Teile einer Darstellung per ferngesteuertem Zeigersymbol zu verweisen. Hierbei sollen private und öffentliche Annotationen unterschieden werden, d.h. es gibt sowohl eine öffentliche "Tafel-Sicht", auf die alle Teilnehmer einer Sitzung Zugriff haben, als auch eine private, für jeden Teilnehmer exklusiv zugreifbare Sicht.

Das Vorhandensein einer öffentlichen Sicht, erfordert eine entsprechende zentrale Zugriffsregelung. Zu spät kommende Sitzungsteilnehmer sollen automatisch über die bereits erfolgten Annotationen informiert werden. Alle Annotationen sollen zudem in “Seiten” aufgeteilt werden, damit eine Navigation innerhalb der bereits erstellten Inhalte möglich wird.

4.4.2 Implementierung

Damit die gewünschte Funktionalität implementiert werden kann, müssen die zentralen Begriffe auf Systemobjekte abgebildet werden:

Graphischer Inhalt

Ein im Whiteboard zu verwaltender Inhalt muss die folgenden Funktionen anbieten:

- Erstellen eines anfangs leeren, einfarbigen Whiteboard-Inhaltes
- Verwendung der momentanen Bildschirmdarstellung als Inhalt (wird benötigt, um beliebige Ausgaben von Anwendungen annotieren zu können)
- Benutzung von vorbereiteten Inhalten in Form von Grafiken in Standardformaten
- Darstellung des Inhaltes

Diese Funktionalität wird durch das `Background`-Objekt abgedeckt.

Whiteboard-Zustand

Damit mehrere Teilnehmer unabhängig von einander Annotationen auf einem Whiteboard-Inhalt anbringen können, müssen bestimmte Graphikkontext-Informationen für jeden Teilnehmer einzeln verwaltet werden. Hierzu gehören:

- Art des verwendeten Graphikwerkzeugs (Freihandzeichnung, Schwamm, Marker, Text oder Zeigersymbol)
- Momentane Position, Farbe und Größe des Werkzeugs

Diese Informationen werden durch entsprechende Annotations-Operationen verändert. Innerhalb der Whiteboard-Implementierung werden die Daten eines Teilnehmers durch das `Context`-Objekt gekapselt.

Annotation und Zeigeoperationen

Annotationen und Zeigeoperationen (die letzteren stellen einen Spezialfall der Annotationen dar) bestehen aus einer Abfolge von grundlegenden Operationen, die entweder direkt eine graphische Ausgabe erzeugen oder die Kontextinformationen eines Teilnehmers oder den Gesamtzustand des Whiteboards verändern. Die durch das `WBOp`-Objekt repräsentierten Operationen umfassen:

- Einbringen eines neuen Whiteboard-Inhaltes
- Anzeigen eines bestimmten Inhaltes. Die Unterscheidung des Einbringens und Anzeigens von Inhalten soll es ermöglichen, einmal eingebrachte Inhalte (z.B. Präsentationsfolien) wiederzuverwenden.
- Bewegen des Graphikwerkzeuges an eine bestimmte Position
- Zeichnen eines Linienzuges
- Darstellung einer Textzeile
- Einstellen des aktuell benutzten Werkzeugs
- Einstellen der aktuell benutzten Werkzeugfarbe
- Einstellen der aktuell benutzten Werkzeuggröße
- Ausführen der Operation (bewirkt Änderung der Ausgabe oder des Whiteboardzustands)

Seite

Der Begriff einer Whiteboard-Seite wird dadurch realisiert, dass die Abfolge von Whiteboard-Operationen in Untersequenzen eingeteilt wird. Eine derartige Sequenz beginnt mit dem Einbringen des Inhaltes (`Background`) und umfasst alle folgenden Operationen, die sich auf diesen Inhalt beziehen.

Es ergibt sich die Notwendigkeit, die Operationen in einer Form zu speichern, die das Wiederauffinden anhand des Inhaltes, auf dem die Operation stattfindet, ermöglicht. Ein "Blättern" innerhalb mehrerer Whiteboard-Seiten wird also durch das wiederholte Abspielen der entsprechenden Untersequenz von Operationen implementiert. Diese Funktion wird durch das `PageManager`-Objekt realisiert.

4.4.3 Diskussion der Systemstruktur

Für den Fall, dass das Whiteboard mit einem vorhandenen physischen Whiteboard (z.B. bestehend aus einem Projektor und einer berührungssensitiven Oberfläche) integriert werden soll, muss eine Komponente der Anwendung die Ansteuerung dieses physischen Gegenstücks übernehmen. Es bietet sich an, die durch das Kommunikationssystem vorgegebene Unterteilung in Server- und Clientkomponenten derart auf diesen Anwendungsfall abzubilden, dass der Serverteil die Ansteuerung des Projektor-Tafel-Systems übernimmt. Es ist zudem sinnvoll, bei direkten Eingaben über die Tafelfläche auf eine vorherige Berechtigungsprüfung zu verzichten.

Um eine automatische Unterstützung für zu spät kommende Teilnehmer zu gewähren, empfiehlt sich die Verwendung des in Abschnitt 3.7.1 beschriebenen, komplett integrierten Datenflusses. Die *Floor Control*-Komponente übernimmt in diesem Fall die Koordination des Zugriffs auf den öffentlichen Tafelbereich.

Die Kernfunktion der Whiteboard-Anwendung ist dabei die Erstellung und Anzeige von Annotationen als Ergebnis von Benutzerinteraktionen. Diese Funktion muss für drei unterschiedliche Fälle implementiert werden:

1. Für die öffentliche Server-Sicht
2. Für die öffentliche Sicht auf einem Client
3. Für die private Sicht auf einem Client

Öffentliche Server-Sicht

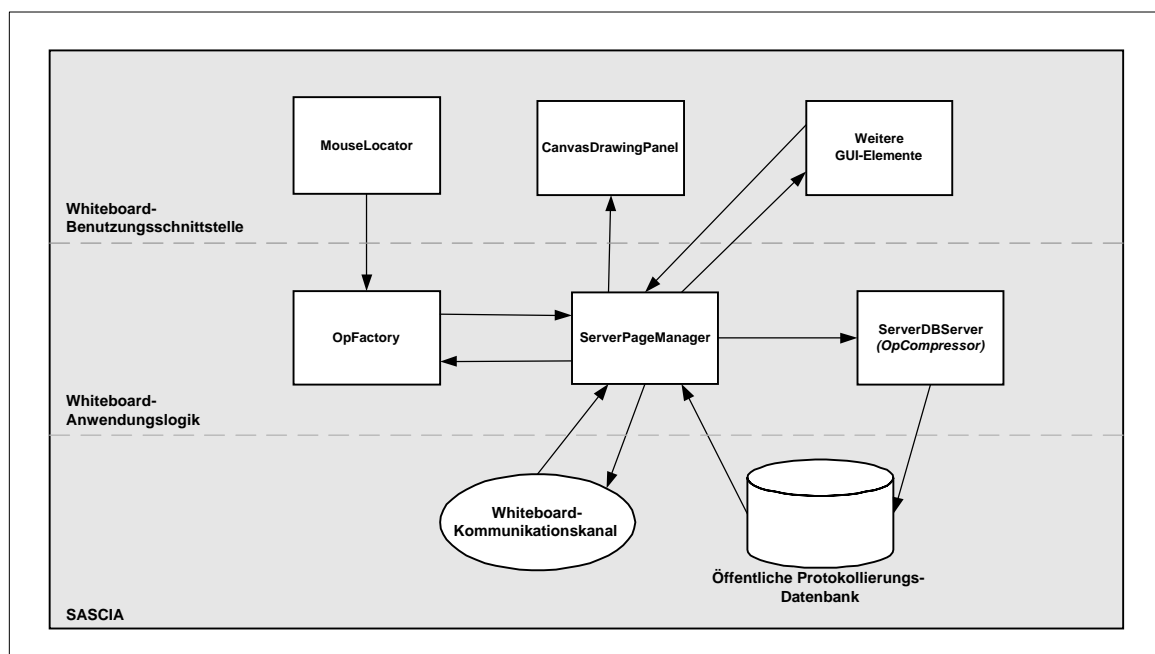


Abbildung 4-1: Datenfluss innerhalb des Whiteboard-Servers

Abbildung 4-1 skizziert den Datenfluss innerhalb des Whiteboard-Servers. Es erfolgt eine Unterscheidung zwischen Daten, die lokal produziert und verarbeitet werden, und solchen, die von Clients erzeugt wurden. Die Datenflussstruktur findet sich sowohl in der Server-Sicht, als auch in der privaten und öffentlichen Client-Sicht wieder.

Die auf dem Server-Whiteboard vorgenommenen Annotationen finden ihren Ursprung in einem `Locator`-Objekt. Diese Art von Objekt abstrahiert die grundlegenden, an Mauseingaben angelehnten Aktionen: *pressed*, *released* und *dragged*. Für jedes einzubindende Eingabegerät mit Zeigercharakteristik muss ein entsprechendes `Locator`-Objekt definiert werden, um die gerätespezifischen Eingabeaktionen in `Locator`-Aktionen umzuwandeln.

Im Rahmen der prototypischen Implementierung existiert bereits ein solches Objekt für die Mauseingabe. Zusätzliche Geräte, wie z.B. die drucksensitive Projektionsfläche eines SMARTboards oder eines Grafiktablets, können auf diese Weise mit geringem Aufwand in das Whiteboard eingebunden werden.

Die *Locator-Events* werden unter Berücksichtigung des momentanen Graphikkontexts des lokalen Teilnehmers (siehe Abschnitt 4.4.2) durch das `OpFactory`-Objekt in Whiteboard-Operationen umgewandelt, die an den `ServerPageManager` weitergeleitet werden.

Der *PageManager* wiederum meldet die produzierten Operationen an das Ausgabeelement `CanvasDrawingPanel`. Benutzeraktionen, die sich auf den Graphikkontext des Teilnehmers auswirken (z.B. Werkzeug- oder Farbwechsel), werden durch weitere GUI-Elemente an den *PageManager* gemeldet, der diese an das `OpFactory`-Objekt weiterleitet, um dessen Kopie des Kontexts konsistent zu halten. Hiermit ist die lokale *Consumer-Producer-Kette* geschlossen.

Die auf der Server-Seite produzierten Annotationen werden über das `ServerDBServer`-Objekt in die öffentliche Datenbank eingespeist und dadurch an alle Clients verteilt. Es erfolgt hierbei keine Rechteüberprüfung, da der Server-Benutzer als Verwalter immer Zugriff auf die Ressource des öffentlichen Whiteboards hat.

Das `ServerDBServer`-Objekt erfüllt dabei zusätzlich die Funktion des `OpCompressors`. Hierbei handelt es sich um eine Warteschlange für auszuliefernde Whiteboard-Operationen. Um die Datenmenge zu minimieren wird versucht, aufeinanderfolgende Einzelzeichenoperationen zu einem Linienzug zusammenfassen. Zu diesem Zweck wird die Auslieferung von Einzelzeichenoperationen für eine gewisse Zeit verzögert und, falls möglich, zusammengefasst. Anwendungstests, die sowohl mit, als auch ohne den Kompressions-Mechanismus durchgeführt wurden, bestätigten einen erheblichen Leistungsgewinn aufgrund dieser Vorgehensweise, während sich die Sichtbarkeit der getätigten Annotationen nur um Sekundenbruchteile verzögerte.

Von Teilnehmern erzeugte Daten werden analog zu den lokalen Operationen ebenfalls an den *PageManager* gemeldet und so, nach erfolgter Zugriffsrechteüberprüfung, dem internen Datenfluss zugeführt. Die Verteilung der Daten erfolgt wie bereits beschrieben über den Weg der öffentlichen Datenbank. Falls ein Teilnehmer jedoch ausgehend von der privaten Whiteboard-Ansicht eine Kopie der Server-Bildschirmausgabe beantragt, so erfolgt die Auslieferung dieser Daten als Ausnahmefall direkt über den Kommunikationskanal.

Öffentliche Client-Sicht

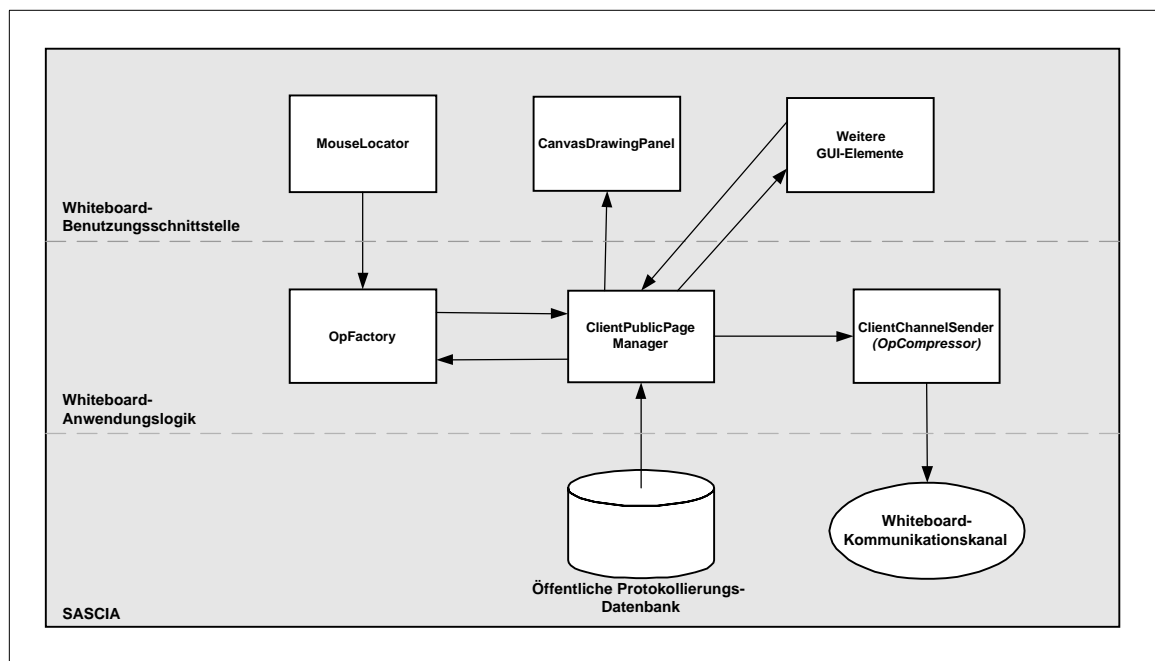


Abbildung 4-2: Datenfluss innerhalb des Whiteboard-Clients, öffentliche Sicht

In der clientseitigen Sicht auf das öffentliche Whiteboard erfolgt der Datenfluss analog zu dem des Servers. Der einzige Unterschied besteht in den Serverschnittstellen:

Lokal erzeugte Daten werden über das `ClientChannelSender`-Objekt unter Verwendung des Kommunikationskanals an den Server gemeldet (siehe Abbildung 4-2). Hierbei erfolgt wiederum eine Kompression der Whiteboard-Operationen. Der Server verteilt die Daten nach erfolgter Zugriffsrechteüberprüfung über die Datenbank an alle Teilnehmer.

Aus Effizienzgründen werden die durch den Teilnehmer selbst erzeugten Operationen direkt angezeigt, ohne auf die Bestätigung durch den Server zu warten. Eventuell sich daraus ergebende Inkonsistenzen zwischen der lokalen und der auf allen anderen Anzeigen erfolgten Whiteboard-Darstellung sind (bei einem ansonsten funktionierenden Gesamtsystem) nur temporär und unkritisch, weshalb sie durch diese Vorgehensweise in Kauf genommen werden können.

Private Client-Sicht

Die in Abbildung 4-3 gezeigte Struktur der privaten Whiteboard-Sicht entspricht größtenteils den bereits beschriebenen Strukturen der öffentlichen Sicht. Die Speicherung der erzeugten Operationen erfolgt hierbei in der privaten Protokollierungsdatenbank. Die Verwendung des Kommunikationskanals ist dabei lediglich notwendig, um bei Bedarf eine Kopie der Ausgaben auf dem Serverbildschirm anzufordern.

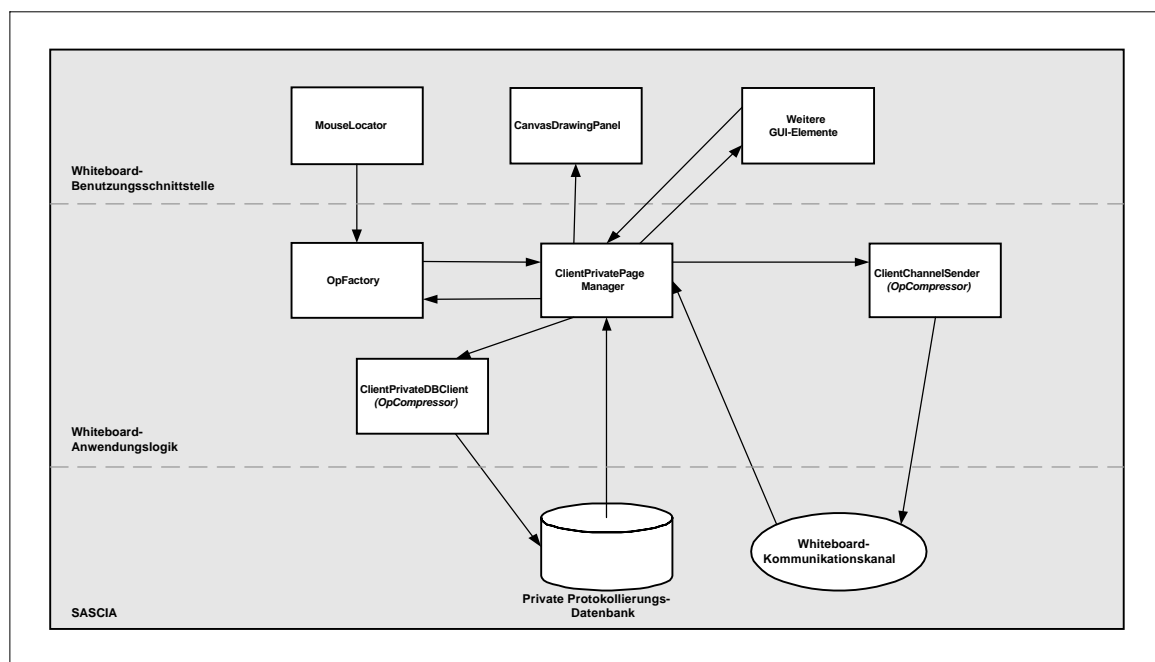


Abbildung 4-3: Datenfluss innerhalb des Whiteboard-Clients, private Sicht

4.4.4 Benutzung des Whiteboard-Servers

Server-GUI

Nach dem Start der Whiteboard-Serveranwendung werden in einem eigenen Fenster die Schaltflächen für die Serversteuerung angezeigt (Abbildung 4-4). Diese ermöglichen den Zugriff auf die 4 Hauptfunktionen des Servers (Symbolleiste von links nach rechts):

- Laden von vorbereiteten Whiteboard-Seiten als Grafiken im JPEG, GIF oder PNG-Format.
- Erstellen von neuen Whiteboard-Seiten, entweder durch Kopieren der Bildschirm- ausgabe einer Anwendung oder durch Einfügen von leeren Seiten.

- Anzeige des Schreibrecht-Dialoges. Stellt ein Teilnehmer einen Antrag auf das “Schreibrecht”, also auf die Benutzung der Ressource “Whiteboard”, so wird dies dem Benutzer des Servers per Färbung des Schreibrecht-Symbols angezeigt. Dies entspricht einer Wortmeldung bei normalen Sitzungen.
- Umschalten auf Whiteboard-Anzeige (siehe Beschreibung im nächsten Abschnitt).

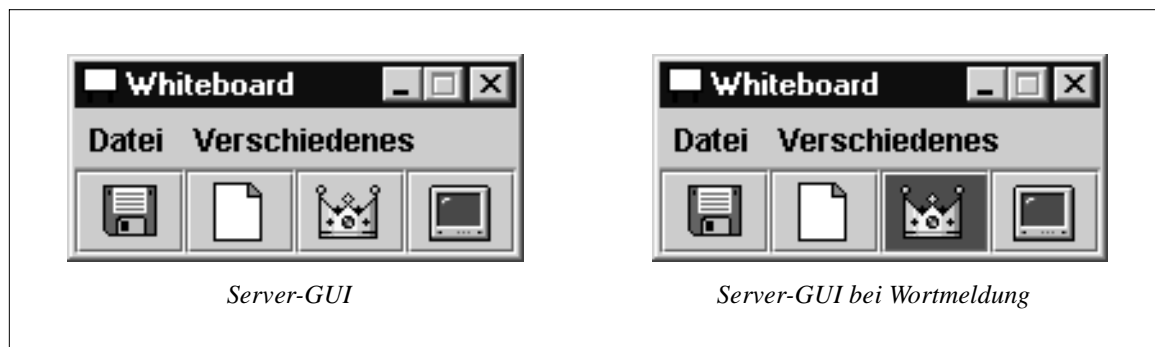


Abbildung 4-4: Whiteboard-Server-GUI

Das oberhalb der Symbolleiste sichtbare Menü beinhaltet die folgenden Unterpunkte:

- Einladen von neuen Inhalten. Dies entspricht der Funktion des ersten Symbols der Leiste.
- Anzeige eines Fensters, in dem interne Systemmeldungen protokolliert werden.
- Beenden des Whiteboards. Dies ist ebenfalls durch das Schließen des Hauptfensters möglich.

Whiteboard-Anzeige

Das Whiteboard selbst wird bei Aktivierung bildschirmfüllend angezeigt. Wurde noch kein Inhalt in das Whiteboard eingefügt, so besteht die Anzeige stattdessen aus einer grauen Fläche. Eine Annotation ist erst nach dem Erstellen, bzw. nach dem Einfügen einer ersten Seite möglich. Innerhalb der Zeichenfläche ist ein Kontext-Menü verfügbar, mittels dessen das zu verwendende Annotations-Werkzeug und die entsprechende Farbe ausgewählt werden können.

Bei Verwendung der Bildschirmkopie-Funktion kann ein “Transparenz-Effekt” erreicht werden, d.h. die Kopie legt sich pixelgenau über den ursprünglichen Bildschirminhalt, so dass der Eindruck entsteht, Annotationen würden direkt auf der Ausgabe einer Anwendung angebracht.

Zu beachten ist hierbei jedoch, dass es sich nur um eine “Quasi-Transparenz” handelt, d.h. Änderungen an der Anwendungsausgabe nach dem Erstellen der Kopie werden nicht in das Whiteboard übertragen.

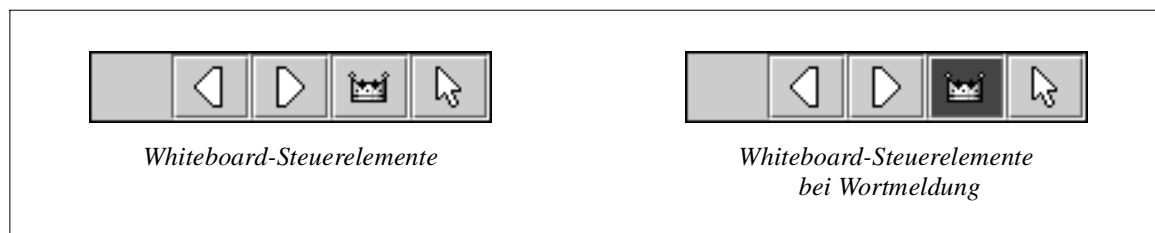


Abbildung 4-5: Steuerelemente im Anzeigemodus

Die Server-Funktionen legen 3 Verwendungsmöglichkeiten für das Whiteboard nahe:

1. Erstellen von Skizzen und Sammeln von Ideen und Stichwörtern
2. Die Annotation von Anwendungen, bzw. deren graphischen Ausgaben
3. Die Durchführung von vorbereiteten Präsentationen mittels des Whiteboards

Im Anzeigemodus ermöglicht ein spezielles Kontrollfenster (siehe Abbildung 4-5) den direkten Zugriff auf die wichtigsten Whiteboard-Funktionen. Die leere Schaltfläche ganz links ist dabei zur Positionierung dieses Fensters innerhalb der Whiteboard-Anzeige gedacht¹. Mit den Pfeilsymbolen wird das Blättern innerhalb der Sammlung von Whiteboard-Seiten ermöglicht. Das folgende Kronensymbol dient der Anzeige des Schreibrecht-Dialogs. Das letzte Symbol in dieser Reihe ermöglicht den Zugriff auf das bereits erwähnte Kontext-Menü für die Wahl des Annotations-Werkzeugs. Diese Funktion ist bei der Verwendung von Eingabegeräten notwendig, die über lediglich eine “Maustaste” verfügen. Ein Beispiel hierfür ist die drucksensitive Projektionstafel des SMARTBoards [SMART].

Schreibrecht-Dialog

Der Zugriff auf das Server-Whiteboard wird über das in Abschnitt 3.4 beschriebene Rechtevergabesystem koordiniert. Die Elemente des Schreibrecht-Dialogs (Abbildung 4-6) ermöglichen Benutzern die Steuerung der Rechtevergabe.

Im Hinblick auf die verschiedenen denkbaren Arten, die momentane Schreibrechts-Situation zu visualisieren, wurde in diesem Dialog ein “Reiter”-System implementiert, mit dessen Hilfe zwischen verschiedenen Ansichten umgeschaltet werden kann. In der Prototyp-Version wurde lediglich die Listenansicht implementiert. Denkbar ist jedoch auch eine “Sitzplan”-Ansicht, in der die Teilnehmer mittels ihrer Position im Vortragsraum identifiziert werden können. Dies würde für Vortragende die Zuordnung von Wortmeldungen im System und der Sitzposition des entsprechenden Teilnehmers im Raum erheblich vereinfachen.

1. Die Verwendung eines Ersatzes für die sonst übliche Titelzeile eines Fensters ist notwendig, da es in Java-Swing Version 1.3 nicht möglich ist, einem Vollbild-Fenster ein normales Fenster (mit normaler Titelzeile) unterzuordnen.

In der Listenansicht gewähren drei Teilnehmerlisten den Überblick über die Schreibrechts-Situation:

1. Liste aller anwesenden Teilnehmer
2. Liste der Teilnehmer, die das Schreibrecht beantragt haben
3. Liste der redeberechtigten Teilnehmer

Die beiden Schaltflächen “Gewähren” und “Entziehen” ermöglichen die Vergabe und den Entzug des Schreibrechts. Soll das Recht gewährt werden, ist der entsprechende Teilnehmer aus einer der links angeordneten Listen auszuwählen und dann die Schaltfläche zu betätigen. Wurde kein Teilnehmer ausgewählt, so wird versucht, allen in der “Meldungsliste” aufgeführten Teilnehmern das Recht zu gewähren. Dies ist nur möglich, wenn entsprechend viele Plätze in der Liste der schreibberechtigten Teilnehmer frei sind.

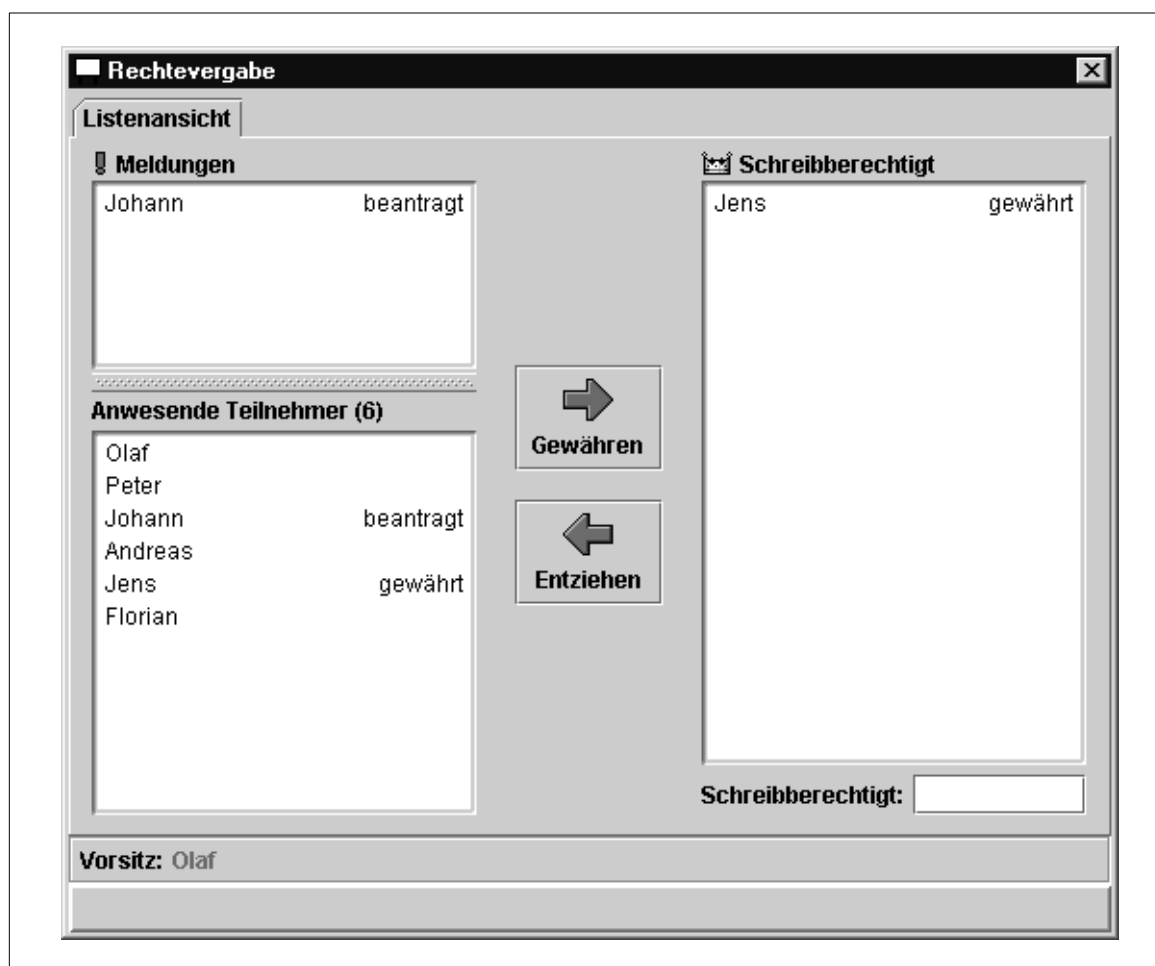


Abbildung 4-6: Schreibrecht-Dialog

Die Zahl der maximal gleichzeitig schreibberechtigten Teilnehmer kann über das Eingabefeld am rechten unteren Rand des Dialoges eingestellt werden.

Ähnlich wie bei der Schreibrechtvergabe verhält es sich auch beim Entzug des Rechtes: wurde kein Teilnehmer bei Betätigung der "Entziehen"-Schaltfläche ausgewählt, so wird versucht, allen momentan schreibberechtigten Teilnehmern das Recht zu entziehen.

Am unteren Rand des Dialogs wird der Name des Vorsitzenden angezeigt. Diese Rolle wird entsprechend der in [OBER01] beschriebenen Mechanismen vergeben. Abgesehen von Benutzern des Whiteboard-Servers hat nur der Vorsitzende die Nutzungs-Berechtigung für den Dialog. Alle anderen Teilnehmer können den Dialog zwar aufrufen, die Kontroll-Schaltflächen sind in diesem Fall jedoch deaktiviert.

Das Ergebnis von Schreibrechts-Aktionen wird in Form von textuellen Meldungen in der untersten Zeile des Dialogs angezeigt. Eine Liste vorhergehender Meldungen ist durch Mausklick auf diese Zeile möglich.

Client-GUI



Abbildung 4-7: Client-GUI

Abbildung 4-7 zeigt die clientseitige Benutzungsschnittstelle. Das Hauptfenster beinhaltet dabei zwei unterschiedliche Whiteboard-Ansichten - öffentlich und privat - die über einen Reiter am oberen Fensterrand ausgewählt werden können.

Die öffentliche Sicht stellt den Zugriff auf das Server-Whiteboard dar, das mit der Anzeige des Servers und aller Teilnehmer synchronisiert wird. Um im öffentlichen Whiteboard Annotationen vornehmen zu können, ist vorher die Beantragung des Schreibrechts notwendig. Die private Sicht ermöglicht das Anbringen eigener Annotationen, die nur auf dem lokalen System sichtbar sind. Abgesehen von der Schaltfläche für die Verwaltung des Rede-rechts unterscheiden sich die Benutzungsschnittstellen hierbei nicht. Im folgenden wird deshalb nur auf die Schnittstelle der öffentlichen Sicht eingegangen.

Whiteboard-Anzeige

Der momentane Inhalt des Whiteboards wird im rechten oberen Teil des Fensters angezeigt. Bei übergrossen Seiten werden Scrollbalken zur Verfügung gestellt, die den Zugriff auf den verdeckten Teil ermöglichen. Innerhalb der Anzeigefläche können mittels des Zeigegeräts (bei gegebener Berechtigung) Annotationen angebracht werden. Die Art der Annotation wird dabei durch das Werkzeug-Schaltenelement festgelegt (siehe Abbildung 4-8). Alternativ sind die wichtigsten Funktionen und Einstellungen auch über ein Kontext-Menü verfügbar, welches durch die Benutzung der zweiten Maustaste sichtbar wird.

Werkzeugwahl

Unterhalb der Whiteboard-Anzeige befinden sich die Schaltflächen für die Wahl des zu verwendenden Annotationswerkzeugs. Bei gegebener Berechtigung kann über die Auswahl eines der fünf Symbole die entsprechende Funktion aktiviert werden:



Abbildung 4-8: Schaltelement für die Wahl des Annotationswerkzeugs

- **Stift.** Hiermit können Freihandzeichnungen auf der Whiteboard-Anzeige angebracht werden. Durch einen Klick mit der zweiten Maustaste auf dieses Symbol kann ein Kontext-Menü aktiviert werden, das die Wahl der Stiftstärke ermöglicht.
- **Schwamm.** Dient dem Löschen von bereits vorgenommenen Annotationen. Hierbei wird die zu löschende Aktion nicht entfernt, sondern durch den ursprünglichen Bildschirminhalt "übermalt". Analog zur Stift-Funktion kann hier mittels des Kontextmenüs die Schwamm-Größe eingestellt werden.
- **Marker.** Hierbei handelt es sich um Freihandzeichnungen mit einem transparenten Stift. Das Kontextmenü ermöglicht ebenfalls die Einstellung der Dicke des Markers.

- Text. Durch Verwendung dieser Funktion kann beliebiger Text über einen Dialog eingegeben und auf der Whiteboard-Anzeige positioniert werden. Die Schriftgröße wird dabei im Kontext-Menü festgelegt.
- Zeiger. Ermöglicht die Darstellung eines positionierbaren Zeigers in Form einer Hand. Dieses Symbol ist dabei nur solange sichtbar, solange die Zeigerfunktion aktiviert ist. Unterhalb des Handsymbols wird der Name des Teilnehmers angezeigt, der den Zeiger fernsteuert.

Farbwahl

Abgesehen von der Schwamm-Funktion besitzt jedes Annotationswerkzeug eine einzeln änderbare Vordergrundfarbe. Diese Farbe kann durch entweder durch die Wahl einer der 6 vorgegebenen Farbeinträge oder durch Klick auf die Vordergrundfarbe selbst mittels eines Farbwahl-Dialogs verändert werden. Die Hintergrundfarbe wird nur bei der Erstellung neuer Seiten verwendet.

Die beiden zusätzlichen Schaltelemente direkt neben der Anzeige der Vorder- und Hintergrundfarbe ermöglichen zum einen den direkten Tausch der beiden Farben, zum anderen kann per Mausklick die Standardkombination Schwarz auf Weiß eingestellt werden.

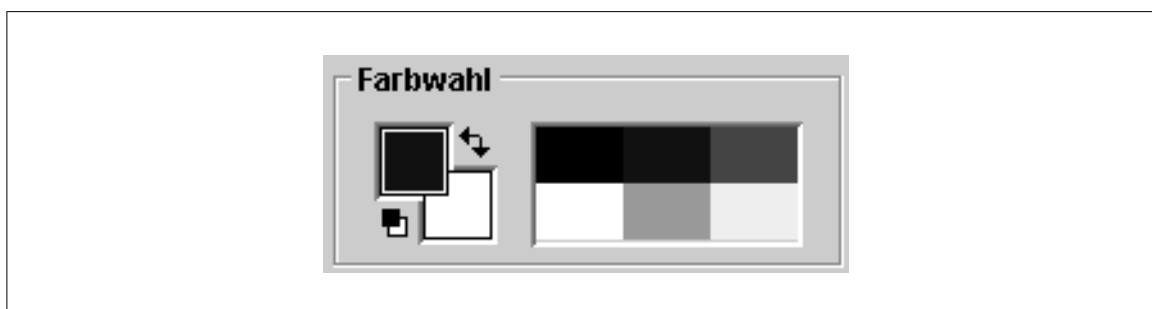


Abbildung 4-9: Schaltelement für die Farbwahl

Seiten erstellen

Es gibt mehrere Möglichkeiten, eine neue Seite für die Anzeige im Whiteboard zu erstellen:



Abbildung 4-10: Schaltelement für das Erstellen neuer Whiteboard-Inhalte

1. Im Hauptmenü des Client-Fensters kann die Funktion "Importieren" gewählt werden, die das Einbinden von Seiten im JPEG, GIF und PNG-Format erlaubt.
2. Über das erste der drei Symbole der "Einfügen"-Schaltfläche in Abbildung 4-10, kann eine zunächst leere Seite mit der aktuellen Hintergrundfarbe eingefügt werden. Derartige Seiten eignen sich z.B. für das Erstellen von Skizzen und Diagrammen. Nach der Wahl des Symbols erscheint ein Dialog, in dem weitere Parameter der neuen Seite abgefragt werden (siehe Abbildung 4-11)

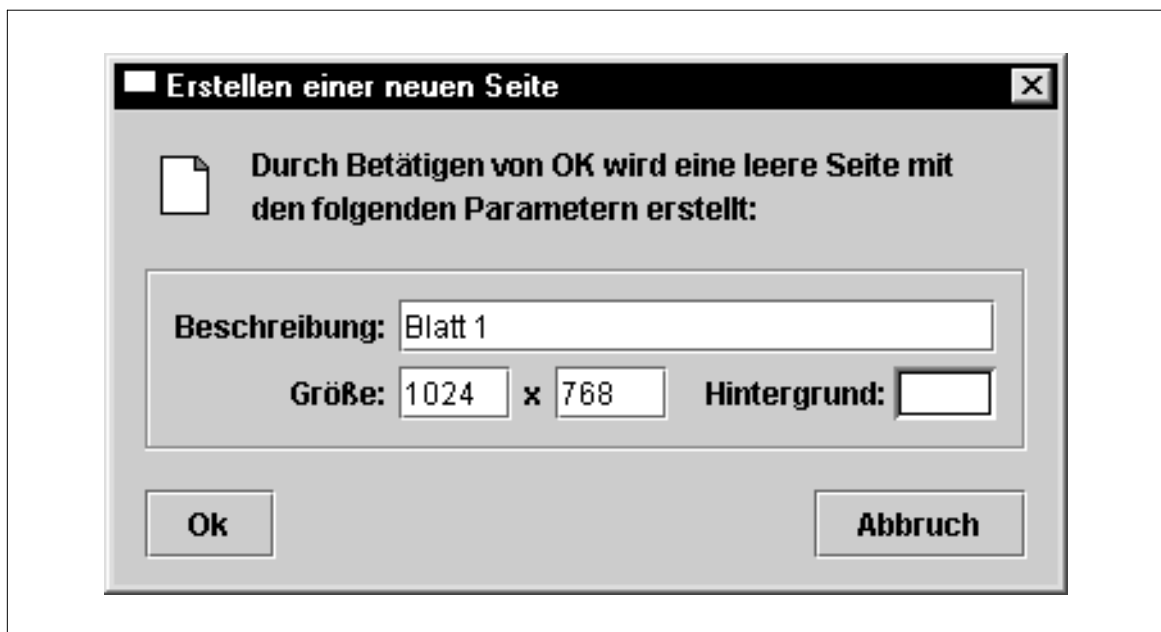


Abbildung 4-11: Dialog für das Erstellen einer neuen Seite

3. Die zweite Schaltfläche ermöglicht das Einfügen einer lokalen Bildschirmkopie. Hierbei wird ebenfalls ein erweiterter Dialog eingeblendet, in dem unter anderem festgelegt wird, ob das Whiteboard-Fenster vor der Aufnahme der Bildschirmkopie ausgeblendet werden soll. Dies ist nützlich, wenn die Ausgabe einer anderen Anwendung in das Whiteboard eingeführt werden soll.
4. Mittels der dritten Schaltfläche kann beantragt werden, dass der momentane Bildschirminhalt des Whiteboard-Servers in das Whiteboard aufgenommen wird. Dies ist ebenfalls in der privaten Sicht möglich.

Navigation

Um innerhalb der Sammlung der bereits erstellten Whiteboard-Inhalte blättern zu können, wird, wie in Abbildung 4-12 ersichtlich, eine entsprechende Liste angezeigt. Durch die Wahl eines Eintrags in dieser Liste wird, die Berechtigung vorausgesetzt, die entsprechende Seite angezeigt. Alternativ kann durch die beiden Schaltflächen “Zurück” und “Weiter” innerhalb der Liste vor- und zurückgeblättert werden.

Im Falle der öffentlichen Sicht sind zudem zwei weitere Elemente vorhanden:

- Tafel einblenden. Diese Funktion dient dazu, das Server-Whiteboard aus- bzw. einzublenden.
- Öffentliches Umblättern. Wird diese Funktion deaktiviert, so ist ein Umblättern in der öffentlichen Sicht auch ohne Schreibberechtigung möglich. Die Änderung der aktuell sichtbaren Seite wirkt sich dabei nur auf das lokale System aus. In der Sei-

tenliste werden die beiden unterschiedlichen aktuellen Positionen einmal durch farbige Hinterlegung des Namens und zum anderen durch die Anzeige eines Tafelsymbols rechts neben dem Namen markiert.



Abbildung 4-12: Navigationsschaltelement

Schreibrechtantrag

Um in der öffentlichen Sicht Annotationen des Whiteboard-Inhalts vornehmen zu können, muss zuerst das entsprechende Recht beantragt werden. Die geschieht über die Schaltelemente “Antrag” und “Rückgabe” (siehe Abbildung 4-13). Wird das Element “Antrag” betätigt, so wird dem System mitgeteilt, dass dieser Teilnehmer das Rederecht beantragt. Je nach Vergabestrategie wird das Recht entweder direkt oder erst später Vergeben. Solange ein Teilnehmer das Schreibrecht beantragt, bleibt das entsprechende Schaltelement aktiviert. Somit ist eine visuelle Rückmeldung über den eigenen Rechtezustand gewährleistet.

Gleichzeitig wird dieser Antrag durch die Färbung des “Rechtevergabe”-Elements auf allen Teilnehmersystem gemeldet. Durch erneute Betätigung des “Antrag”-Schaltelements wird der Antrag zurückgenommen.

Wurde das Recht gewährt, so wird dies durch die Aktivierung des “Rückgabe”-Elements dargestellt. Zudem werden als Konsequenz die in den vorherigen Abschnitten beschriebenen Elemente freigegeben. Wird das Recht nicht mehr benötigt, so kann es durch die Betätigung dieser Schaltfläche wieder zurückgegeben werden.



Abbildung 4-13: Schaltelemente für die Beantragung des Schreibrechts

Abgesehen von diesen grundlegenden Aktionen, werden im oberen Teil der Name des Vorsitzenden, die Zahl der gleichzeitig Schreibberechtigten sowie die Zahl der Teilnehmer insgesamt angezeigt. Der vorsitzende Teilnehmer kann zudem die Zahl der Schreibberechtigten ändern. Über die Schaltfläche “Rechtevergabe” kann der Dialog “Schreibrecht” eingeblendet werden, der eine genauere Aufschlüsselung der Teilnehmersituation ermöglicht.

5 Bewertung

5.1 Test-Ergebnisse

5.1.1 Kommunikationstest

Um die Funktionstüchtigkeit des Kommunikationssystems zu überprüfen, wurde eine Testreihe durchgeführt, bei der ein Server-Programm Datenpakete mit zufälliger Größe erzeugt und an alle Clients versendet. Diese wiederum bestätigen den Empfang durch Senden der Größe des jeweiligen Pakets. Als Ergebnis wurden bei jedem Durchlauf die folgenden Daten protokolliert:

- Paketgröße
- Ausführdauer der Sendefunktion
- Dauer bis zum Eintreffen der letzten Antwortnachricht

Es fanden Variationen der Zahl der verbundenen Clients statt (1 und 3 Clients). Zudem wurde sowohl die Socket-basierte, als auch die Multicast-Variante der von JSDT verwendeten Kommunikationsschicht getestet.

Umgebung

Als Hardwareumgebung wurden IBM-kompatible Workstations mit Pentium II-450 MHz Prozessor unter Windows NT 4.0 verwendet. Die Tests liefen innerhalb eines 100MBit Ethernet-Netzes ab. Auf Java und auf die Projekt-Dateien wurde per Netzlaufwerk zugegriffen (siehe Abschnitt 7.1.1, "Testumgebung").

Ergebnis des TCP-Unicast-basierten Tests

Es fanden zwei unterschiedliche Testreihen statt. Diese unterschieden sich in der Zahl der beteiligten Clients. Jede Testreihe bestand dabei aus 3 Durchläufen, in denen jeweils 100 Pakete gesendet und von allen Clients bestätigt wurden.

Das in den Diagrammen Abbildung 5-1 bis Abbildung 5-6 sichtbare Antwortzeit-Verhalten bestätigt zum einen die Funktionsfähigkeit des Kommunikationssystems in kleinen Gruppen. Es wird zudem sichtbar, dass die Zeit bis zum Eintreffen der Client-Antwort tendenziell linear von der Größe der übertragenen Pakete abhängt. Abweichungen von dieser Tendenz lassen sich zum Beispiel durch den Nicht-Determinismus erklären, mit dem die Speicherverwaltung der benutzten Java-Implementierung die *Garbage-Collection* durchführt.

Ergebnisse mit einem Client

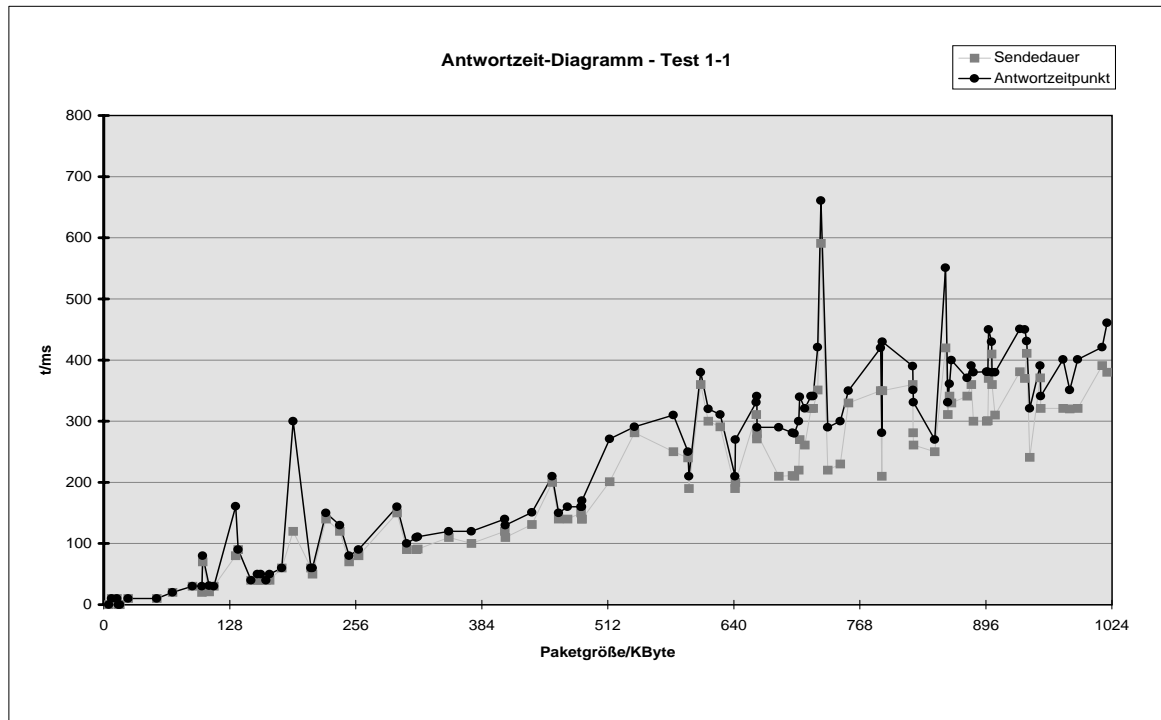


Abbildung 5-1: Antwortzeit-Diagramm: 1 Client, 1. Durchlauf

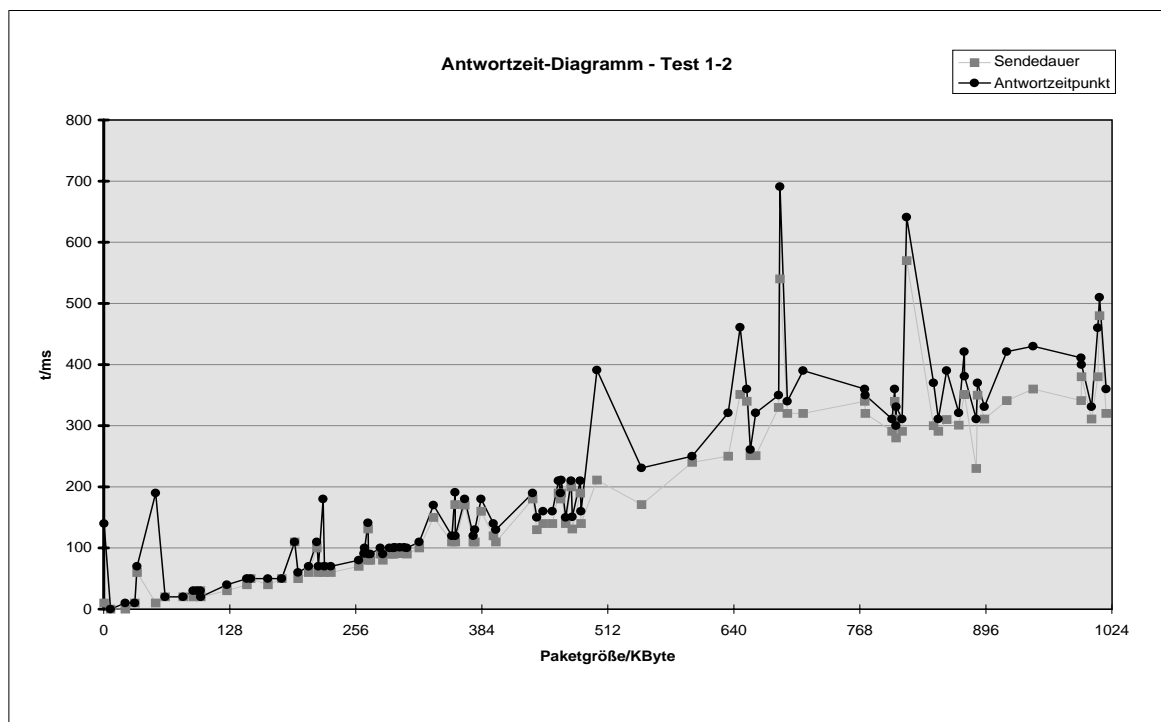


Abbildung 5-2: Antwortzeit-Diagramm: 1 Client, 2. Durchlauf

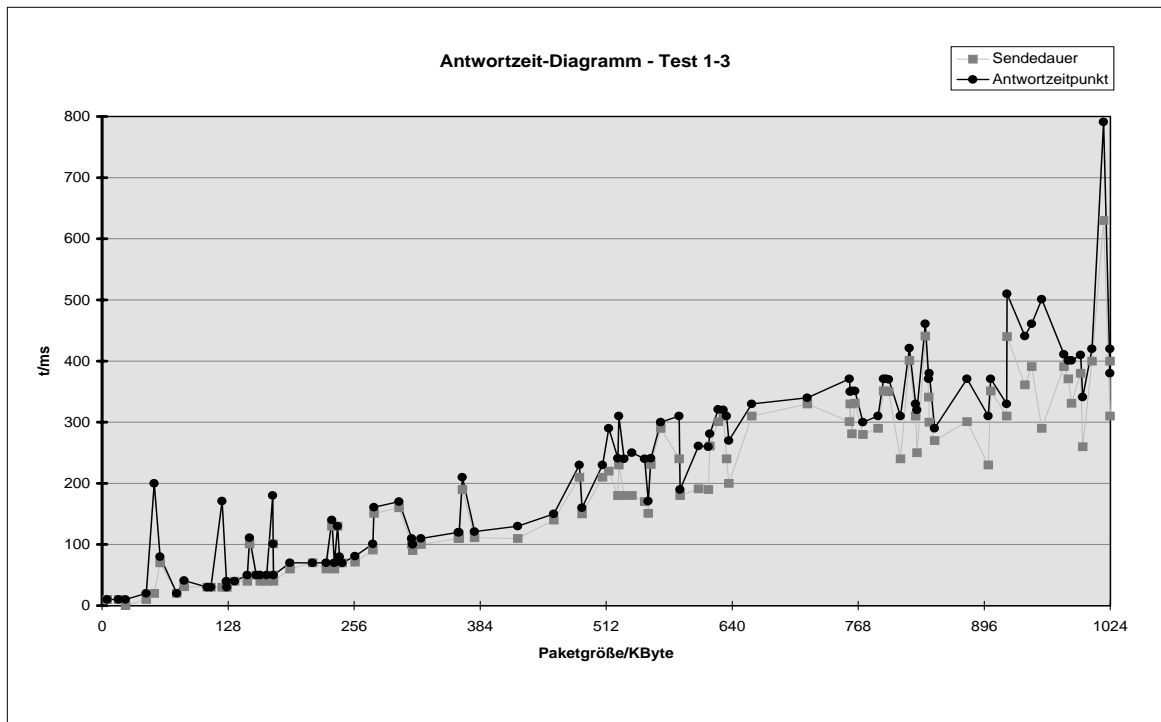


Abbildung 5-3: Antwortzeit-Diagramm: 1 Client, 3. Durchlauf

Ergebnis mit 3 Clients

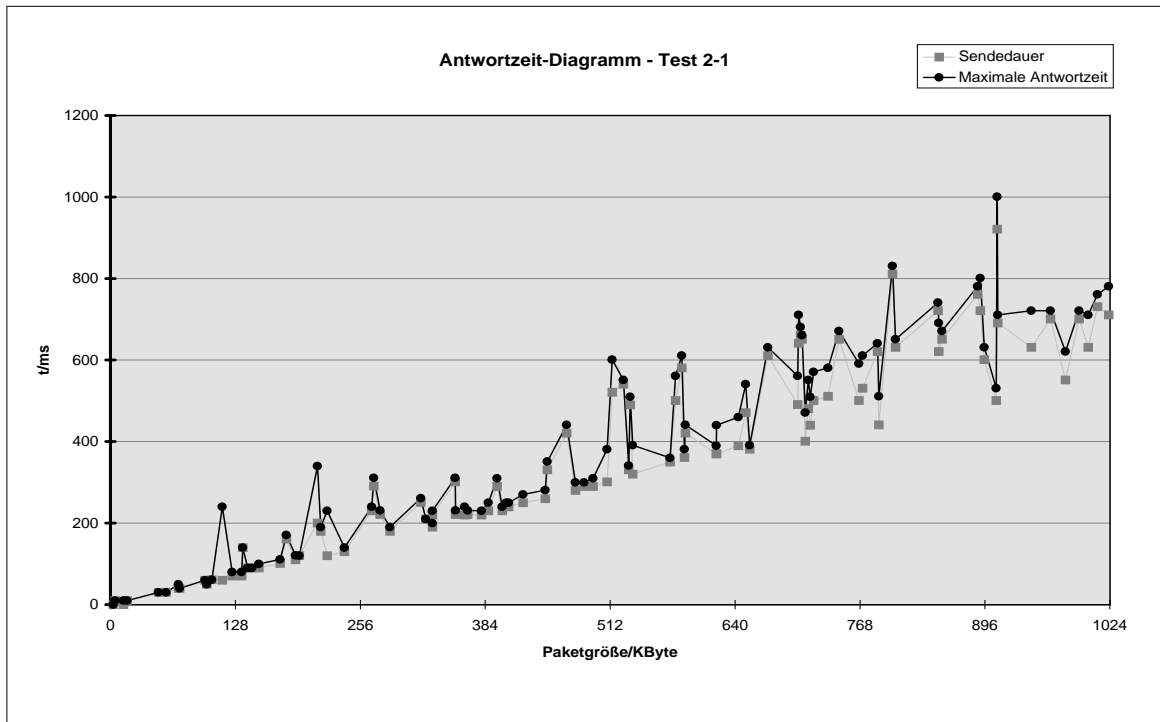


Abbildung 5-4: Antwortzeit-Diagramm: 3 Clients, 1. Durchlauf

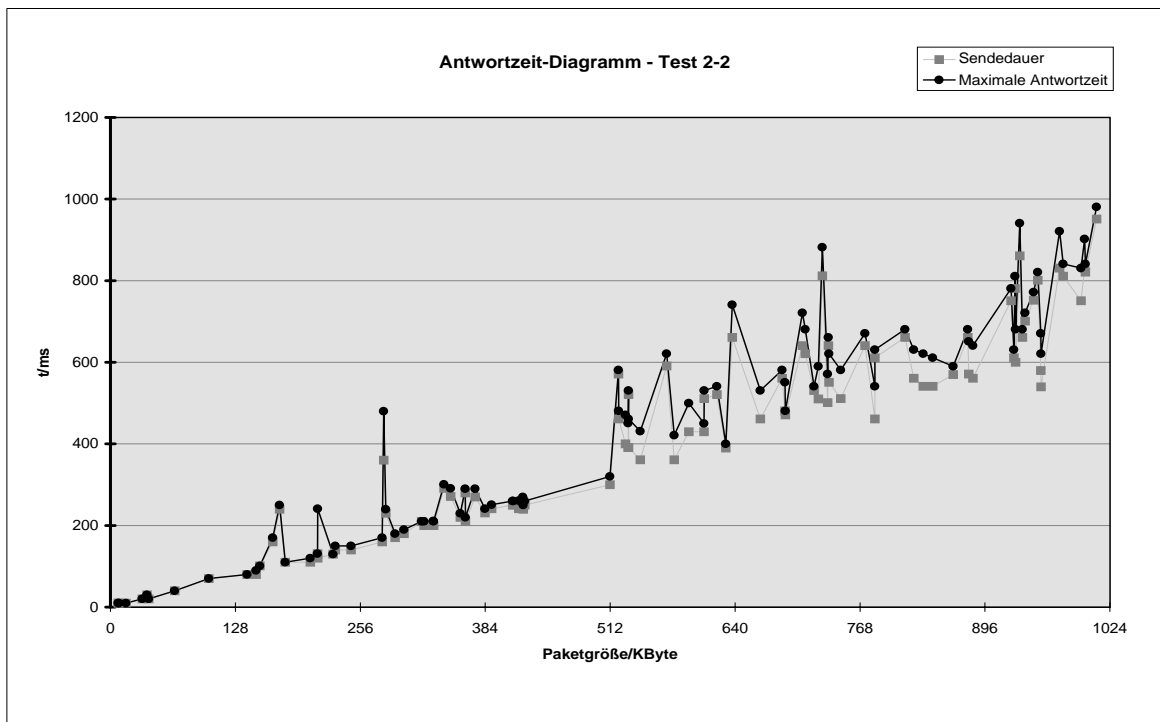


Abbildung 5-5: Antwortzeit-Diagramm: 3 Clients, 2. Durchlauf

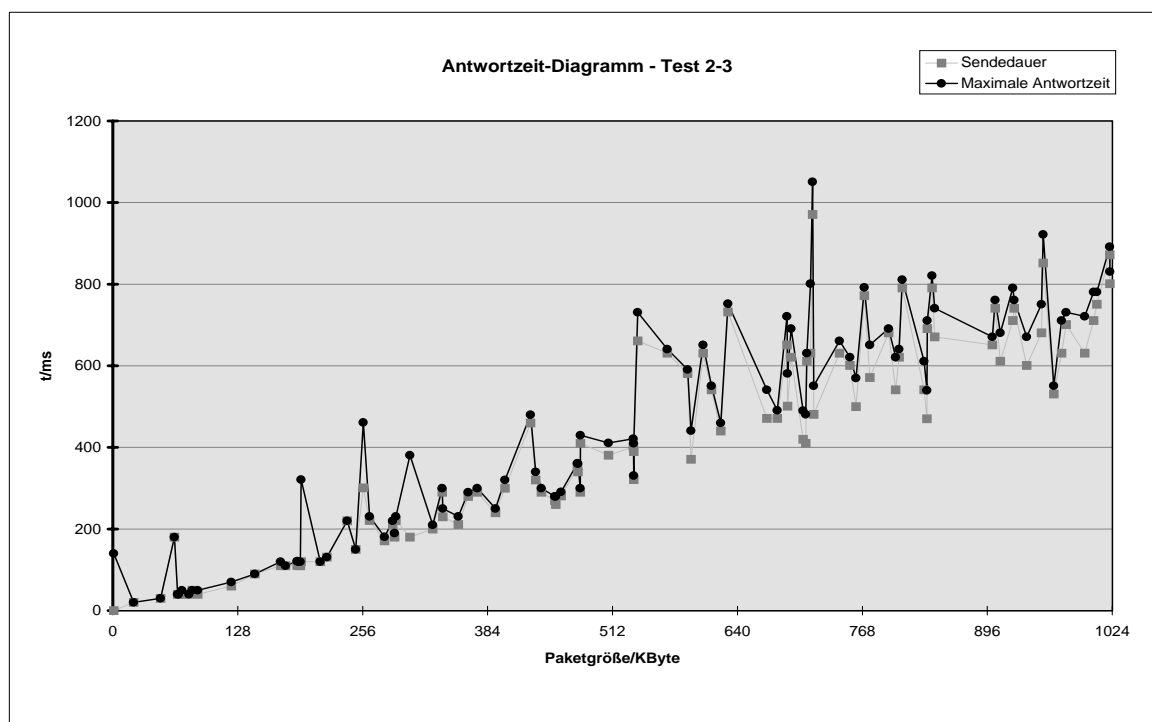


Abbildung 5-6: Antwortzeit-Diagramm, 3 Clients, 3. Durchlauf

Ergebnis des LRMP-Multicast-basierten Tests

Bei Verwendung der auf dem *Light-Weight Reliable Multicast Protocol (LRMP)* basierenden JSDT-Implementierung kam es zu Einbrüchen der Datenübertragungsrates bei Paketen, von mehr als 8 KB Größe. Die resultierenden Übertragungsraten von teilweise weniger als 50 KB/s verhinderten die Durchführung der geplanten Testreihen in akzeptabler Zeit. Das Multicast-Kriterium aus der Liste der Anforderungen an das Kommunikationssystem konnte mit der vorliegenden Implementierung nicht erfüllt werden.

5.1.2 Test des Gesamtsystems

Zur Überprüfung der Funktionsfähigkeit der zu einem Gesamtsystem integrierten Einzelkomponenten sowie der Benutzbarkeit der definierten Schnittstellen wurde das in Abschnitt 4.4 beschriebene Whiteboard unter Verwendung dieses Systems implementiert.

Die dabei entstandene Anwendung erwies sich als funktionstüchtig. Alle geforderten Whiteboardeigenschaften konnten anhand der bereitgestellten Schnittstellen realisiert werden. Das Whiteboard selbst zeigte bei Anwendungstests die gewünschte Funktionalität, so dass ein Rückschluss auf die prinzipielle Funktionstüchtigkeit des Gesamtsystems möglich ist.

Das entwickelte System wurde zudem bereits in einer anderen Arbeit [BAI01] erfolgreich dazu benutzt, eine Komponente zur Visualisierung von Kommunikationsprotokollen für die gemeinsame Nutzung durch mehrere Teilnehmer anzupassen.

5.2 Schwierigkeiten und Lösungen

5.2.1 JSDT

Die Verwendung des *Java Shared Data Toolkits* verkürzte die Entwicklungszeit des Kommunikationssystems erheblich. Wie in Abschnitt 5.1 gezeigt wurde, erfüllt zumindest die JSDT-Implementierung auf TCP-Socket-Basis die zuvor definierten Anforderungen. Dennoch zeigten sich im Zuge der Implementierung des Gesamtsystems und des Whiteboards verschiedene Fehler, bzw. undokumentierte oder unerwartete Verhaltensweisen von JSDT, die im folgenden beschrieben werden.

Serialisierbarkeitsprobleme

Die JSDT-Nachrichtenklasse `Data` erlaubt das Erstellen von Nachrichten mit beliebigen Inhaltstypen. Schlägt das Versenden dieser Nachrichten aufgrund eines nicht serialisierbaren Inhaltsobjektes fehl, so wird dieser Fehler nicht gemeldet. Die aufgerufene Sendemethode führt die geforderte Funktion also *nicht* aus, ohne dass eine Anwendung dieses Fehlverhalten erkennen könnte. Das Kriterium des zuverlässigen Datentransports ist also aus Sicht von JSDT-Anwendungen aufgrund ungünstig gewählter Schnittstellen nicht erfüllt.

Die Implementierung des Kommunikationssystems umgeht diese Schwierigkeit, indem in der eigenen Schnittstellendefinition nur serialisierbare Objekte versendet werden können. Eine fehlerhafte Verwendung der Sendemethode kann so bereits während des Kompilervorgangs der Anwendung erkannt werden.

Authentifizierungsprobleme

Bei Verwendung der von JSDT angebotenen Authentifizierungsschnittstelle kommt es in bestimmten Fällen zu undokumentiertem Verhalten. So wird scheinbar ein Client, der sich im gleichen Prozess wie der Server befindet, immer zugelassen, auch wenn die eigentliche Authentifizierung fehlschlagen müsste. Zudem muss die Reihenfolge der Funktionsaufrufe während einer Authentifizierungsaktion strikt eingehalten werden.

Wird zum Beispiel auf einen Antrag eines Clients auf eine privilegierte Aktion nicht die in der JSDT-Dokumentation beschriebene `authenticate`-Methode des Clients aufgerufen, kommt es zu einer nicht-nachvollziehbaren Beeinträchtigung nachfolgender Authentifizierungsaktionen. Wird zudem ein nicht-serialisierbares *Challenge*-Objekt verwendet (siehe dazu Beschreibung des Verfahrens in Abschnitt 4.2), schlägt die Authentifizierung ohne Angabe der Gründe fehl.

Alle für die Implementierung des Kommunikationssystems relevanten Fälle dieses Verhaltens wurden durch entsprechende *Workarounds* kompensiert.

Registry-Problem

Bei Verwendung der TCP-Socket-Implementierung muss für den Start einer JSDT-*Registry* die IP-Adresse des Servers per DNS in einen Hostnamen auflösbar sein. Ist dies nicht der Fall, z.B. weil DNS nicht verfügbar oder nicht richtig konfiguriert wurde, so kann keine *Registry* und damit kein Server gestartet werden. JSDT stellt außerdem Schnittstellen zur Verfügung, um eine *Registry*, die von einem anderen Prozess gestartet wurde, ohne Authentifizierung zu beenden. Dies kann zu Problemen führen, wenn sowohl der Server, als auch die Clients einer Sitzung auf dem gleichen Rechner ausgeführt werden. In diesem Fall könnte ein böswilliger Benutzer die Sitzung ohne Berechtigung beenden.

Sendeproblem

Werden über die von JSDT bereitgestellten Channels Datenpakete von mehr als 1 MB Größe übertragen, kann es passieren, dass die serverseitige Sendeoperation fehlschlägt, wenn nur ein Client innerhalb der adressierten Gruppe nicht zum Empfang bereit ist. Dieses Problem tritt bei der Verwendung der `receive`-Methode auf, wenn Daten von einem Kanal synchron empfangen werden sollen.

Bei der Registrierung von asynchronen `ChannelConsumer`-Objekten hingegen konnte dieses Fehlverhalten nicht beobachtet werden. Die Implementierung des Kommunikationssystems umgeht dieses Problem, indem der synchrone Empfang von Nachrichten durch einen eigenen, per asynchronem `ChannelConsumer` gefüllten Puffer simuliert wird.

Reihenfolge-Problem bei Ereignissen

Die von JSDT angebotene Schnittstelle für den Empfang von kanalbezogenen Ereignissen funktioniert in Bezug auf die Reihenfolge der Ereignisse nicht zuverlässig. So kann es passieren, dass die Nachricht eines Teilnehmers empfangen wird, bevor die Meldung über das Beitreten des entsprechenden Teilnehmers zu diesem Kanal ausgeliefert wurde. Die Aktualität der Informationen über die Namen aller mit einem Kanal verbundenen Teilnehmer kann aus diesem Grund nicht gewährleistet werden.

Multicast-Problem

Die Ergebnisse der in Abschnitt 5.1.1 dokumentierten Tests der Multicast-Implementierung von JSDT führen zu der Folgerung, dass die Datenübertragungsraten bei Paketen von mehr als 8 KB Größe auf ein für Anwendungen nicht akzeptables Maß einbricht. Die Unterstützung von Multicast-Kommunikation, durch die das Datenaufkommen besonders bei Sitzungen mit über 20 Teilnehmern minimiert werden sollte, kann also mittels der getesteten Implementierung nicht realisiert werden.

Zusammenfassung

Die aufgetretenen Fehler konnten zumindest unter Verwendung der TCP-Socket-basierten JSDT-Implementierung ausreichend kompensiert werden. Als besonders negativ fällt jedoch ins Gewicht, dass die Weiterentwicklung von JSDT nicht gesichert ist. Die letzte

offizielle Version von JSDT ist auf Oktober 1999 datiert. Eine im entsprechenden Entwicklerforum getroffene Aussage über die Freigabe von JSDT als Open-Source-Projekt hat sich zum Zeitpunkt dieser Ausarbeitung noch nicht erfüllt. Die Zukunft des Toolkits ist demnach nicht gesichert.

Da die Schnittstellen des in dieser Arbeit implementierten Kommunikationssystems jedoch von JSDT abstrahiert wurden, ist es bei Bedarf möglich, JSDT komplett durch ein anderes System zu ersetzen. Tatsächlich erscheint diese Entwicklung im Zusammenhang mit dem bereits in Vorbereitung befindlichen Entwurf einer *Peer-to-Peer*-basierten Kommunikationskomponente (siehe Abschnitt 6.2) als in naher Zukunft absehbar.

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

Aufgabe dieser Arbeit war der Entwurf einer Systemarchitektur, die bei der Entwicklung kooperativ nutzbarer Anwendungen eingesetzt werden kann. Auf den Entwurf sollten dabei Implementierung sowie Test anhand der prototypischen Realisierung einer elektronischen Tafel erfolgen. Als Grundlage dienten verschiedene vorhergehende Arbeiten zu diesem Thema, deren Ergebnisse in die Architektur eingebunden werden sollten.

Im Rahmen der Anforderungsanalyse wurde die Verwendung in Vorträgen und Übungen als Hauptszenario für die Architektur identifiziert. Unter Einbeziehung der Ergebnisse aus [OBER01] folgte die Erörterung der sich daraus ergebenden Einzelanforderungen.

Anhand der Ergebnisse der Anforderungsanalyse erfolgte der Entwurf der Architektur. Dabei wurden die wesentlichen Komponenten wie folgt identifiziert und abgegrenzt: Kommunikationssystem, Teilnehmer- und Sitzungsverwaltung, Rechtevergabe, Protokollierungsdatenbank und Anwendungsrahmen für die Einbindung von Anwendungen.

Als weiteres Ergebnis der Anforderungsanalyse wurde der Entwurf in der Programmiersprache JAVA umgesetzt. Die Kommunikationskomponente wurde durch Kapselung des Programmpakets JSDT erreicht. Hierbei ergab sich eine Client-Server-Architektur. Die weiteren, bereits erstellten Komponenten wurden in das Gesamtsystem integriert.

Die Funktion des Kommunikationssystems wurde anhand einer Reihe von Tests überprüft. In der TCP-Socket-basierten Implementierung ergab sich eine zufriedenstellende Leistung des Systems. Bei Versuchen mit der Multicast-Implementierung stellte sich jedoch nur ein mangelhaftes Ergebnis ein. Es folgte ein erster positiver Test der Funktionsfähigkeit des Gesamtsystems anhand der prototypischen Implementierung einer elektronischen Tafel.

In einem abschliessenden Abschnitt wurden die Probleme bei der Implementierung des Kommunikationssystem unter Verwendung von JSDT zusammengefasst. Es wurden Lösungen dokumentiert und auf eine weitere Arbeit verwiesen, welche die Weiterentwicklung des Kommunikationssystem zum Ziel hat.

6.2 Ausblick

6.2.1 Weiterentwicklung

Die Implementierung des Gesamtsystems stellt lediglich eine erste, prototypische Version dar. Es bedarf noch einiger Verbesserung, um die in der Anforderungsanalyse geforderte Stabilität gewährleisten zu können und um eine optimale Ausnutzung der vorhandenen Res-

sources zu ermöglichen. Zudem ist die Funktionalität der Sitzungsanmeldung und -verwaltung noch nicht voll in das System integriert, bzw. hierbei traten noch nicht behobene Probleme auf.

Besonders das Whiteboard bedarf noch einiger Überarbeitung, bevor es ernsthaft in Vorträge und Vorlesungen einbezogen werden kann. Hierbei ist eine Verringerung des Speicherverbrauchs, sowie die Einbindung von Unterstützungen für weitere, für Vorlesungsfolien gebräuchliche Grafikformate wie Postscript oder PDF und die Integration mit Präsentationsprogrammen wie PowerPoint anzustreben. Als funktionelle Erweiterung bietet sich die Darstellung der für die Zugriffsrechtevergabe notwendigen Teilnehmerliste in Form eines Sitzplans dar, die eine Vereinfachung der Zuordnung von systeminterner Wortmeldung und Sitzposition des entsprechenden Teilnehmers ermöglichen würde.

Zum Zeitpunkt dieser Ausarbeitung wird bereits an der Erweiterung der Sitzungsverwaltungsfunktionalität gearbeitet. Man kann davon ausgehen, dass die Weiterentwicklung der Systemarchitektur für absehbare Zeit gewährleistet ist.

Es besteht ebenfalls ein Bedarf an der Einbindung weiterer Anwendungen. Die prinzipiell vorstellbare Integration von Werkzeugen zur Erzeugung und Verwaltung von Audio- und Videoströmen würde das Spektrum der möglichen Kooperation innerhalb des Systems erheblich vergrößern.

6.2.2 Einbindung neuer Technologien

Es ist denkbar, dass sich die Benutzung des in dieser Arbeit entworfenen Konferenzsystems durch die Einbindung verschiedener neuer Technologien weiter vereinfachen lässt. So ist z.B. eine Authentifizierung von Teilnehmern anhand von auf SmartCards implementierten asymmetrischen Verschlüsselungsverfahren denkbar. Desweiteren wäre die automatische Bestimmung der Sitzposition eines Teilnehmers für die Vergabe des Zugriffsrecht eine wertvolle Zusatzinformation.

6.2.3 Peer-to-Peer-Implementierung

Im Rahmen einer weiteren Arbeit [NGUY02] wird untersucht, inwiefern die Systemarchitektur anstatt durch einen zentralen Server auch durch Kombinationen von gleichberechtigten Komponenten umsetzbar ist. Eine entsprechende Implementierung soll soweit möglich auf der Vorlage der bereits erstellten Schnittstellen erfolgen, so dass die entworfenen Komponenten ohne größere Anpassungen direkt genutzt werden können.

Bei einer derartigen Vorgehensweise würde die zugrundeliegende Kommunikation höchstwahrscheinlich komplett per Multicast erfolgen, wobei sich beliebige Topologien ergeben könnten. Die Identifizierung von Teilnehmern und Kommunikationskanälen ist in diesem Fall z.B. durch den Einsatz von asymmetrischen Verschlüsselungsverfahren denkbar, was sich als Nebeneffekt positiv auf die Sicherheit der übertragenen Daten auswirken würde. Die bereits in dieser Arbeit identifizierte Dienstanbieter- und -nutzerrolle könnte sich so ohne eine starre Bindung an Client-Server-Strukturen dynamisch realisieren lassen.

Im Gegensatz zum zentralistischen Ansatz dieser Arbeit, deren Schwerpunkt auf der Verwendung in Vortragsszenarien lag, steht bei einem *Peer-to-Peer*-Entwurf die Kooperation in kleinen, spontan entstehenden Gruppen im Vordergrund. Es ist zu erwarten, dass sich die Ergebnisse dieser Herangehensweise ohne größere Probleme auch auf den Vortragsfall anwenden lassen, um auch hier eine bessere Flexibilität und eine gestiegene Ausfallsicherheit durch die Verwendung dezentraler Komponenten erreichen zu können.

6.2.4 Sitzungsunterstützung durch Agentensysteme

Eine andere Arbeit innerhalb des Projekts SASCIA befasst sich mit der Entwicklung eines persönlichen Beobachtungsagenten für laufende Sitzungen (siehe [HAAF02]). Dieser soll in Vertretung eines Teilnehmers an Sitzungen teilnehmen und anhand des auftretenden Datenaustauschs regelbasiert entscheiden, ob es sich um ein relevantes Thema handelt. Sind entsprechende Kriterien erfüllt, soll der Teilnehmer über die Sitzung benachrichtigt werden.

Ein derartiges System kann sehr hilfreich bei der Auswahl zu besuchender Sitzungen sein, besonders, wenn, wie es gelegentlich vorkommt, derartige Veranstaltungen in konkurrierender Weise an gleichen Terminen stattfinden. Durch Mechanismen, die den Zustand einer Sitzung (z.B. Fortschritt bei einem vorgegebenen Arbeitsablauf) erkennen und repräsentieren können, ist zudem eine Unterstützung des Splittens und Wiederausammenführens von Übungsgruppen zur besseren Zeitausnutzung denkbar.

6.2.5 Spezialanwendungen

Wie bereits durch das kooperativ nutzbare Whiteboard sowie die in [BAI01] realisierte gemeinsame Nutzung eines zuvor nur für den Einzelnutzerbetrieb ausgelegten Programms zur Visualisierung von Kommunikationsprotokollen gezeigt wurde, ist das Erstellen von spezialisierten Anwendung dank der entworfenen Systemarchitektur in überschaubarer Zeit möglich.

Durch die Verwendung derartiger Programme ergeben sich interessante Perspektiven für die verstärkte Einbeziehung von Vortragsteilnehmern in den Lehrprozess. Zudem können derartige Programme als sinnvolle Mittel der Unterstützung von gemeinsamen Übungen eingesetzt werden, auch in Fällen, in denen sich die Teilnehmer von Übungsgruppen nicht alle an einem Ort befinden.

6.2.6 Desktopsharing

Für die Demonstration von Funktionsweisen bereits existierender Anwendungen, die zudem nicht erweiterbar sind und auch sonst keine Möglichkeit der Fernsteuerung bieten, wäre die Integration eines *Desktopsharing*-Systems eine ideale Erweiterung. In diesem Fall wäre es möglich, die komplette Interaktion eines Teilnehmers mit einer lokal verfügbaren Anwendung Tastendruck für Tastendruck durch Kopieren der graphischen Ausgabe auf den Anzeigen der restlichen Teilnehmer zu verfolgen und auch im Nachhinein wiederholt abzu-

spielen. Denkbar ist zudem die geregelte Vergabe der Kontrolle über das entsprechende Programm an weitere Teilnehmer, so dass ein Ergebnis durch die Zusammenarbeit mehrerer Teilnehmer mit einer nur auf einem Rechner verfügbaren Anwendung entstehen kann.

Für die Integration in die entworfene Systemarchitektur wären in diesem Zusammenhang *Desktopsharing*-Systeme geeignet, deren Quelltext frei verfügbar und idealerweise in der Implementierungssprache Java gehalten ist. Diese Kriterien werden z.B. durch das *Virtual Network Computing*-System der AT&T-Laboratories Cambridge erfüllt (siehe [VNC]).

6.2.7 Interaktionsformen und Ergonomie

Abgesehen vom direkten, anwendungsbezogenen Nutzen, den die Einführung von computerbasierten Anwendungen innerhalb von Vorlesungen darstellt, wären interaktionsbezogene Untersuchungen in diesem Umfeld nützlich. Denkbare Fragestellungen wären hierbei:

- Wie muss ein Computersystem gestaltet und in eine bestehende Kooperationsform (in diesem Fall die Form des Vortrags oder der geleiteten und ungeleiteten Übung) integriert werden, so dass das Entstehen einer Hemmschwelle vor dessen Benutzung vermieden wird?
- Inwiefern lässt sich die real stattfindende Interaktion (z.B. eine Wortmeldung durch Heben einer Hand) automatisch auf Systemaktionen (z.B. Beantragen des Zugriffsrechts) abbilden und in welchen Fällen ist dies sinnvoll?
- Wie wirkt sich die Verwendung derartiger Systeme langfristig auf die stattfindende Interaktion aus?

6.2.8 Generischer Rollenmechanismus

Im Laufe der Entwicklung der Systemarchitektur hat sich gezeigt, dass auf allen Ebenen eine dynamische Zuordnung von Rollen zu bestimmten Teilnehmern notwendig ist, bzw. dass sich verschiedene Mechanismen auf ein Rollenverteilungssystem reduzieren lassen. Beispiele hierfür sind:

- Client- und Serverrolle bei den folgenden Komponenten: Kommunikationssystem, *Floor Control*, Protokoll Datenbank, *Session-Management*.
- Dienstanbieter- und -nutzerrolle bei Anwendungen.
- Rechteinhaber- und Vorsitzendenrolle im Rahmen der *Floor Control*.
- Kanalverwalter- und -nutzerrolle im Kommunikationssystem.

Durch die Einführung eines generischen Rollenmechanismus könnte der Großteil der mehrfach und in dieser Arbeit nur selten dynamisch implementierten Zuweisungen von Rolle zu Teilnehmer auf ein einheitliches System zusammengefasst werden. Ein derartiges System müsste unter anderem die folgenden Aktionen unterstützen:

- Verwalten einer Liste von Rolleninhabern.
- Methoden zum Hinzufügen und Entfernen von Rolleninhabern.

- Verbreiten von Mitteilungen über Änderungen in der Rollenverteilung durch ein entsprechendes Ereignissystem.
- Prioritäten innerhalb der Gruppe der Rolleninhaber, die verhindern, dass Teilnehmer bestimmter Prioritätsstufen andere Teilnehmer einer höheren Stufe aus der Liste der Rolleninhaber ausschließen können. Denkbar wären hier z.B. die Stufen:
 - “zugewiesener Rolleninhaber” mit höchster Priorität. Sinnvoll wenn Teilnehmern eine Rolle bereits vor Beginn einer Sitzung zugewiesen werden soll.
 - “ernannter Rolleninhaber”. Diese Stufe könnte alle Teilnehmer umfassen, die durch “zugewiesene” oder andere Rolleninhaber der selben Stufe ernannt wurden.
 - “gewählte Rolleninhaber”. Per Mehrheitsentscheid aller Teilnehmer gewählter Inhaber einer Rolle.
- Weitergabe von Rollen durch deren Inhaber.
- Verwaltung von Einstellungen, z.B. ob Rolleninhaber gewählt werden dürfen, was für eine Mehrheit dafür notwendig ist, wieviele Rolleninhaber es geben darf, ob die Rollenzuweisung nach einer bestimmten Zeit automatisch aufgehoben wird oder ob die Rollenvergabe nach einem vorgegebenen Schema ablaufen soll (z.B. *round robin*).

6.3 Abschließende Bewertung

Beginnend mit der ursprünglichen Forderung, den Zugang zu einer zentralen Ressource (einem physischen Whiteboard) innerhalb einer Vorlesung allen Teilnehmern koordiniert zugänglich zu machen, wurde durch die stetige Erweiterung und Systematisierung der Anforderung im Jahr 2000 das Projekt SASCIA an der Fakultät Informatik der Universität Stuttgart ins Leben gerufen. Die in der Folge entwickelten Teilkomponenten wurden in dieser Arbeit erstmalig integriert und - in Hinblick auf die grundlegende Funktionalität - erfolgreich getestet.

Obwohl es bereits verschiedene Projekte mit dem Ziel der Entwicklung eines Rahmensystems zur Unterstützung von Gruppenarbeit gibt, die einen ähnlichen Funktionsumfang bereitstellen (siehe [MACS99]), ergeben sich aus den unterschiedlich gewählten Vorgehensweisen, Schwerpunkten und Systemabgrenzungen neue Erkenntnisse. Der verstärkte Einsatz derartiger System ermöglicht zudem weitere Untersuchungen über die Auswirkungen der Benutzung von computerunterstützten Systemen im aktiven Lehrbetrieb.

7 Anhang

7.1 Kommunikationstest

7.1.1 Testumgebung

- IBM-kompatible PCs
- Microsoft Windows NT 4.0
- 450 MHz Pentium II Prozessor
- 128 MB Hauptspeicher
- 100 MBit Ethernet-Netz (gleiches Subnetz, Ping-Antwortzeit < 10ms)
- Java Version 1.3
- Zugriff auf Java-Installation und Projekt-Dateien per Netzlaufwerk

7.1.2 Test-Durchläufe

- 1 Client, 100 Pakete bis 1 MB Größe, 1. Durchlauf (Abbildung 5-1)
- 1 Client, 100 Pakete bis 1 MB Größe, 2. Durchlauf (Abbildung 5-2)
- 1 Client, 100 Pakete bis 1 MB Größe, 3. Durchlauf (Abbildung 5-3)
- 3 Clients, 100 Pakete bis 1 MB Größe, 1. Durchlauf (Abbildung 5-4)
- 3 Clients, 100 Pakete bis 1 MB Größe, 2. Durchlauf (Abbildung 5-5)
- 3 Clients, 100 Pakete bis 1 MB Größe, 3. Durchlauf (Abbildung 5-6)

Die Test-Programme berechneten zudem, dass die Zeitmessung auf der Serverseite mit einer Auflösung von 10 ms erfolgten.

7.1.3 Server-Zeiten

Bei allen Durchläufen wurde die Zeit gemessen, die notwendig war, um einen Server in den Zustand zu bringen, Clients bedienen zu können.

- Minimale Dauer: 5,0 s
- Maximale Dauer: 6,8 s
- Durchschnittliche Dauer: 5,7 s

7.1.4 LoadTestClient

```
import sascia.*;

/** This is the client side of a performance test for
 * the sascia Comm-system. It connects to a specified
 * server and replies to byte-array packets with an
 * Integer containing the size of the most recently
 * received packet.
 *
 * Usage: Usage: LoadTestClient <registryhost> <registryport>
 *
 * <sessionurl>
 *
 * @author Peter Oberparleiter
 */

public class LoadTestClient implements CommChannelConsumer
{
    private CommAddress serverAddress;
    private CommClient myClient;
    private CommChannel myChannel;

    private MiscSemaphore running = new MiscSemaphore();

    public static void main (String args[])
    {
        LoadTestClient testClient = new LoadTestClient();
        testClient.start(args);
    }

    /** The actual server program. */
    public void start(String args[])
    {
        String myName;

        // Get the address from command line options
        if (args.length != 3)
        {
            System.out.println(
                "Usage: LoadTestClient <registryhost> "+
                "<registryport> <sessionurl>");
            System.exit(1);
        }
    }
}
```

Abbildung 7-1: Quelltext des Kommunikationstest-Clients

```
try
{
    serverAddress = new CommAddress(args[0], args[1],
                                    args[2]);

    System.out.println("Opening connection to server at");
    System.out.println(serverAddress.toString());

    // Try to establish a connection to the server
    for (int i=0; ; )
    {
        try
        {
            // Try to contact the server
            myName = Integer.toString(i);
            myClient = new CommClient(myName, "", serverAddress);
            break;
        }
        catch (CommNameInUseException e)
        {
            // A client with that number is already connected
            // Try the next number
            i++;
        }
    }

    // Join the test channel
    myChannel = myClient.joinChannel(
                LoadTestServer.channelName);

    myChannel.addCommChannelConsumer(this);

    System.out.println("Client #" + myName + " ready.");

    running.P();

    System.out.println("Closing down...");

    // Close the connection
    myChannel.removeCommChannelConsumer(this);
    myChannel.leave();
    myClient.disconnect();
    myClient = null;
}
```

Abbildung 7-1: Quelltext des Kommunikationstest-Clients

```

catch (NumberFormatException e)
{
    System.out.println("Invalid registry port specified.");
    System.exit(1);
}
catch (SASCIAException e)
{
    // Something went wrong
    System.out.print("SASCIA-Exception: ");
    System.out.println(e.toString());
    System.exit(1);
}
catch (Exception e)
{
    // Something went wrong
    System.out.print("Exception: ");
    System.out.println(e.toString());
    System.exit(1);
}

System.out.println("Done.");
System.exit(0);
}

/** This method processes client replies. */
public void notifyIncomingMessage(CommMessage serverMessage)
{
    if (!serverMessage.isOwnerMessage()) return;

    try
    {
        byte[] serverMessageContent =
            (byte[]) serverMessage.getAsObject();
        String reply;
        // Reply by sending the packet length
        reply = Integer.toString(serverMessageContent.length);

        myChannel.sendToOthers(reply);
        if (reply.equals("0")) running.V();
    }
    catch (Exception e)
    {
        // Something went wrong
        System.out.print("Exception: ");
        System.out.println(e.toString());
    }
}

```

Abbildung 7-1: Quelltext des Kommunikationstest-Clients

```
        System.exit(1);
    }
}
}
```

Abbildung 7-1: Quelltext des Kommunikationstest-Clients

7.1.5 LoadTestServer

```
import sascia.*;
import java.util.Random;

/** This is the server side of a performance test for
 * the sascia Comm-system. It opens a server and waits for
 * clients to connect until the user begins the test by
 * pressing return. It then starts sending packets of random
 * size to all clients and waits for them to respond
 * with the packet-size.
 *
 * Usage: LoadTestServer <num packets> <max packet size>
 *
 * @author Peter Oberparleiter
 */

public class LoadTestServer implements CommChannelConsumer
{
    public static final String channelName = "testchannel";

    private int numPackets;
    private int maxPacketSize;
    private int numClients;

    private byte[] packet;

    private long startTime;
    private long sendTime;
    private long sendPeriod;

    private MiscSemaphore semaphore;

    public static void main (String args[])
    {
        LoadTestServer testServer = new LoadTestServer();
        testServer.start(args);
    }
}
```

Abbildung 7-2: Quelltext des Kommunikationstest-Servers

```
public void start(String args[])
{
    semaphore = new MiscSemaphore();

    if (args.length != 2)
    {
        System.out.println("Usage: LoadTestServer <numpackets> "+
            "<max packetsize>");
        System.exit(1);
    }

    try
    {
        numPackets = Integer.parseInt(args[0]);
        maxPacketSize = Integer.parseInt(args[1]);
    }
    catch (NumberFormatException e)
    {
        System.out.println("Invalid parameters - need numbers.");
        System.exit(1);
    }

    System.out.println("Average timer resolution is: "+
        getResolution()+" ms.");

    try
    {
        Random random = new Random(getTimer());

        startTimer();

        CommServer myServer = new CommServer();
        CommChannel myChannel;

        // Start server
        myServer.start("socket", false);
        myChannel = myServer.createChannel(channelName);
        myChannel.addCommChannelConsumer(this);

        System.out.println("Server and channel online after "+
            getTimer()+" ms.");

        System.out.println("Server may be reached at");
        System.out.println(myServer.getAddress().toString());
    }
}
```

Abbildung 7-2: Quelltext des Kommunikationstest-Servers

```
System.out.println();
System.out.println("Press [return] to start the test.");

// Wait for clients to join
waitForReturn();

myServer.setOpen(false);
numClients = myServer.getParticipantCount();

System.out.println("Starting test with "+
    numClients+" Clients, "+
    numPackets+" packets sized randomly from 1 to "+
    maxPacketSize+" bytes.");

// Perform test task
for (int i=0; i<=numPackets; i++)
{
    if (i!=numPackets)
    {
        // Generate a packet of random size
        packet = new byte[random.nextInt(maxPacketSize)+1];
    }
    else
    {
        // Generate a packet of size 0 to indicate end of test
        packet = new byte[0];
    }

    // Send packet
    sendTime = getTimer();
    myChannel.sendToOthers(packet);
    sendPeriod = getTimer()-sendTime;
    System.out.println("SEND      : ("+packet.length+
        " Bytes/ send done after "+sendPeriod+
        " ms)");

    semaphore.P(numClients);
}

// Wait for clients to disconnect
while (myChannel.getParticipantCount()>0)
{
    Thread.sleep(500);
}
```

Abbildung 7-2: Quelltext des Kommunikationstest-Servers

```
        // Shut server down
        myChannel.removeCommChannelConsumer(this);
        myChannel.leave();
        myServer.stop();
        myServer = null;
    }
    catch (SASCIAException e)
    {
        System.out.println("SASCIAException: "+e.toString());
        waitForReturn();
        System.exit(1);
    }
    catch (Exception e)
    {
        System.out.println("Exception: "+e.toString());
        waitForReturn();
        System.exit(1);
    }

    System.out.println("Done.");

    // Has to be done because JSDT-Threads won't stop
    System.exit(0);
}

public void notifyIncomingMessage(CommMessage replyMessage)
{
    int replyLength =
Integer.parseInt(replyMessage.getAsString());
    int clientNum = Integer.parseInt(replyMessage.getSender());

    if (replyLength != packet.length)
    {
        System.out.println(
            "INVALID : #"+clientNum+
            " "+replyLength+"!="+packet.length+
            " ("+(getTimer()-sendTime)+" ms after send start)");
    }
    else
    {
        System.out.println(
            "RESPONSE: #"+clientNum+
            " ("+(getTimer()-sendTime)+" ms after send start)");
    }
}
```

Abbildung 7-2: Quelltext des Kommunikationstest-Servers

```
        semaphore.V();
    }

    public static void waitForReturn()
    {
        try
        {
            System.in.read();
            System.in.skip(System.in.available());
        }
        catch (java.io.IOException e) {}
    }

    public synchronized void startTimer()
    {
        startTime = System.currentTimeMillis();
    }

    public synchronized long getTimer()
    {
        return System.currentTimeMillis()-startTime;
    }

    public static long getResolution()
    {
        long testStartTime=0;
        long testMidTime=0;
        long testEndTime=0;
        long NUM_MEASURES = 10;

        testStartTime = System.currentTimeMillis();

        for (int i=0; i<NUM_MEASURES; i++)
        {
            testMidTime = System.currentTimeMillis();

            while ( (testEndTime = System.currentTimeMillis()) ==
                    testMidTime);
        }

        return (testEndTime-testStartTime)/NUM_MEASURES;
    }
}
```

Abbildung 7-2: Quelltext des Kommunikationstest-Servers



Abbildung 7-2: Quelltext des Kommunikationstest-Servers

8 Literaturverzeichnis

- BAI01** Xue Bai
Gemeinsames Lernen von Kommunikationsprotokollen durch elektronisch unterstütztes Rollenspiel
Diplomarbeit, Oktober 2001, IPVR, Universität Stuttgart
- BURG97** Cora Burger
Groupware - Kooperationsunterstützung für verteilte Anwendungen
ISBN-3-920993-60-8, dpunkt-Verlag
- HAAF02** Marco Haaf
Integration eines persönlichen Agenten zur Beobachtung laufender SASCIA-Sitzungen und regelabhängige Benutzerbenachrichtigung
Diplomarbeit, Mai 2002, IPVR, Universität Stuttgart
- HILT** Volker Hilt, Werner Geyer
A Model for Collaborative Services in Distributed Learning Environments
Extended version of a paper accepted at IDMS'97
- INRIA** Institut National de Recherche en Informatique et en Automatique
Light-weight Reliable Multicast protocol
<http://webcanal.inria.fr/lrmp/index.html>
- JSDTFORUM** Diskussionsforum für JSDT Benutzer
Archives of JSDT-INTEREST@JAVA.SUN.COM
<http://archives.java.sun.com/archives/jsdt-interest.html>
- JSDTWEB** Sun Microsystems, Inc.
Java (TM) Shared Data Toolkit Home Page
<http://java.sun.com/products/java-media/jsdt/>
- MACS99** O. Brand, W. Mahalek, D. Sturzebecher, M. Zitterbart
MACS - A Modular Collaboration Environment
Proceedings of the IASTED International Conference '99
- NGUY00** Nguyen-Salamanis, Khan-Loan
Adhoc-Verwendung einer elektronischen Tafel unter Verwendung der Jini-Technologie
Universität Stuttgart, Fakultät Informatik, Studienarbeit Nr. 1778
ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/STUD-1778/
- NGUY02** Khan-Loan Nguyen-Salamanis
Erstellung einer Peer-to-Peer-Architektur für die gemeinsame Nutzung von Anwendungen
Diplomarbeit, Mai 2002, IPVR, Universität Stuttgart
- OBER01** Peter Oberparleiter
Vergabe von Zugangsberechtigungen für die gemeinsame Nutzung von Smartboards in der Vorlesung
Studienarbeit Nr. 1807, IPVR, Universität Stuttgart
- SCHEE01** Nicolai Scheele
Enabling Interactive Education with Handheld Devices
Diplomarbeit, Lehrstuhl für Praktische Informatik IV, Universität Mannheim

- SCHMID01** Andreas Schmid
Prototypische Erstellung einer Protokollierung für den Einsatz in Lehrveranstaltungen
Studienarbeit Nr. 1816, IPVR, Universität Stuttgart
- SMART** SMART Technologies Inc.
Webseite der Firma "SMART Technologies Inc."
<http://www.smarttech.com>
- SOMM01** Marcus Sommer
Prototypische Erstellung einer Sitzungsverwaltung für den Einsatz in Lehrveranstaltungen
Studienarbeit Nr. 1817, IPVR, Universität Stuttgart
- VNC** AT&T Laboratories Cambridge
Virtual Network Computing
<http://www.uk.research.att.com/vnc/>