Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Horst Mund

Berechtigungsstrukturen in kollaborativen Umgebungen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Horst Mund

Berechtigungsstrukturen in kollaborativen Umgebungen

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Angewandte Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Kai von Luck
Zweitgutachter : Prof. Dr. Jörg Raasch

Abgegeben am 10. Juli 2006

Horst Mund

Thema der Diplomarbeit

Berechtigungsstrukturen für kollaborative Umgebungen

Stichworte

Verteilte Systeme, CSCW, Autorisation, Authentifikation, RBAC, Persistenz, Konferenzraum

Kurzzusammenfassung

Arbeitstreffen haben einen festen Platz in der Geschäftswelt. Die Teilnehmer dieser Treffen benötigen technische Werkzeuge, um effizient zusammenzuarbeiten und zu kommunizieren. Die Sicherheit und Persistierung von Objekten stellt einen wichtigen Gesichtspunkt bei der Konstruktion von diesen Werkzeugen dar. In dieser Arbeit wird eine Architektur entworfen, die ein besonderes Augenmerk auf die Nutzung von Sicherheits- und Persistenzdiensten hat. Einige Komponenten dieser Architektur werden realisiert, um die Praxistauglichkeit zu zeigen.

Horst Mund

Title of the paper

Authorization and Authentication structures in collaborative environments

Keywords

Authorization, Authentication, Persistence, CSCW, Distributed Systems, RBAC

Abstract

The thesis covers problems of computer support for business meetings. In constructing such computer support special attention lies on security and persistence. A design for such environment has given; relevant parts are implemented (proof of concept).

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Das Zusammenkommen zu einem Meeting ist heute fester Bestandteil der Geschäftswelt. Dieser Personenkreis findet sich physikalisch in einem festen Raum zusammen. Die Kommunikation findet durch den Austausch von Worten in gesprochener und geschriebener Form, Bildern und Videos statt. Die Zusammenarbeit erfolgt zur Durchführung gemeinsamer Arbeiten, beispielsweise das Erstellen einer Präsentation. Die Form der Kommunikation ist synchron, d.h. die Teilnehmer erhalten die Information im gleichen Moment, in dem sie vom Kommunikationspartner erzeugt wird.

Die technische Unterstützung der Teilnehmer ist durch eine Vielzahl an Hilfsmitteln in der Vergangenheit stark angestiegen, Beispiele dafür sind Projektoren, Notebooks und PDA´s. Die Teilnehmer haben gemeinsame Ziele, den Austausch von Dokumenten und das Visualisieren, sowie die Entwicklung von Lösungen für Probleme. Der Datenaustausch zwischen den Teilnehmern ist nur durch weitreichende Einstellungen auf den jeweiligen Endgeräten vorzunehmen, damit nutzen die Teilnehmer die digitalen Endgeräte während des Meetings nur für eigene persönliche Notizen. Beamer oder Videoprojektoren erleichtern die Arbeit der Gruppe, da die Präsentationen nicht vorher ausgedruckt werden müssen, sondern jetzt direkt dargestellt werden können. Leider ist der Beamer nur von einer Person und einem Endgerät zur Zeit nutzbar.

Die nichttechnischen Hilfsmittel sind Whiteboards und Flipcharts, die sich dazu eignen komplexe Sachverhalte visuell darzustellen. Sie erlauben das gemeinsame Erstellen von Inhalten in kleinen Arbeitsgruppen, da hier die Teilnehmer gleichzeitig schreiben können. Damit ist interaktives und direktes Zusammenarbeiten möglich, allerdings tritt die Schwierigkeit auf diese Skizzen automatisiert in ein digitales Format zu überführen.

Es bleibt festzustellen, dass die digitalen Endgeräte zwar eine Erleichterung der persönlichen Arbeit und deren Präsentation erleichtern, die direkte Zusammenarbeit fördern sie aber nicht. Dafür werden immer noch die klassischen Methoden bevorzugt.

Diverse Dokumenttypen können im Raum auftreten, diese können in verschiedenen Formaten Informationen beinhalten. Dazu gehören typischerweise Binär- und Ascii-Formate, im

Raum also z. B. Quellcode, Projektpläne, Bilder. Die vorhandenen Objekte im Raum werden gemeinsam manipuliert oder neu erschaffen.

In diesen kollaborativen Umgebungen ist das flexible Miteinanderarbeiten gewünscht, aber dabei sollte der Aspekt der Sicherheit genauer betrachtet werden. Es ist wichtig, die Sicherheitsbelange der Benutzer zu erkennen und zu gewährleisten. Dies kann mit einer geeigneten Sicherheitsstrategie umgesetzt werden.

Die Anforderungen für ein Zugriffsmodell in kollaborativen Umgebungen sind bekannt und wurden in der Theorie auch mehrfach dokumentiert. Allerdings fanden in der Vergangenheit kaum Implementierungen statt. Man kann zwei Gründe für die fehlende Umsetzung in die Praxis spezifizieren. Zum Einen sind diese Modelle sehr komplex, sie sind also nicht sehr einfach zu implementieren und bieten dem Benutzer keine intuitive Schnittstelle. Es fällt den Benutzern sehr schwer das Berechtigungsmodell zu verstehen. Ein zweiter Grund ist das, alle Prototypen, die für kollaborative Umgebungen entwickelt wurden, die Umsetzung einer Zugriffkontrolle eine sehr geringe Priorität genießt. Ein Prototyp kann für das erste Testen der Funktionalität sehr gut ohne Zugriffskontrolle auskommen, wobei die Zugriffskontrolle ohne laufenden Prototypen wenig Sinn macht. Daher kommt die Eigenart die Zugriffkontrolle als „nice-to-have“ einzustufen. Da die meisten kollaborativen Umgebungen nicht über den Status des Prototypen hinauskommen, ist daher die Entwicklung in diesem Bereich noch nicht sehr weit fortgeschritten.

1.2 Zielsetzung

Im Rahmen dieser Diplomarbeit wird ein System implementiert, dass eine effiziente Zusammenarbeit in einem oder mehreren Konferenzräumen ermöglicht. Dieser interaktive Konferenzraum besteht aus diversen digitalen Endgeräten z. B. Notebooks, Beamer, PDA´s. Weiterhin soll es elektronische Whiteboards mit Touchscreenfunktionalität geben, an denen direkt kollaborativ gearbeitet werden kann, siehe dazu Abbildung 1.1.

Dieser interaktive Konferenzraum sollte die folgenden Eigenschaften erfüllen:

- Persistenz
- Verknüpfbarkeit aller Objekte
- Erschaffen neuer Informationen
- Strukturierungsfunktion
- Universelle Zugreifbarkeit
- Zugriffsschutz

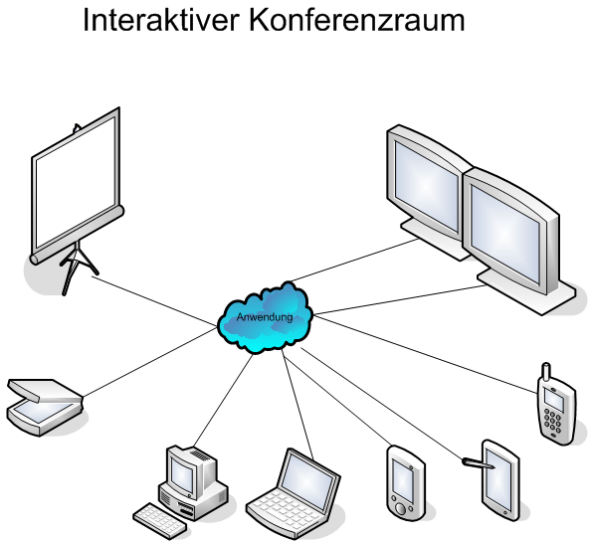

Abbildung 1.1: Digitaler Konferenzraum

- Gegenseitige Wahrnehmung
- Kooperatives Arbeiten mit Hilfe der Raumfunktionen

Ich werde mich speziell mit der Entwicklung von Persistenzdiensten im interaktiven Konferenzraum beschäftigen. Dabei werden Rechtestrukturen und die notwendigen Komponenten zur Umsetzung eines Sicherheitsmodells spezifiziert. Im Laufe der Analyse wird ein geeignetes Zugriffsmodell ausgewählt und mögliche Rollen festgelegt. Weiter werde ich einen generellen Blick auf eine kollaborative Umgebung werfen und Probleme aufdecken, die einer gesonderten Betrachtung bedürfen.

1.3 Gliederung der Arbeit

Im Kapitel 2 wird das zugrunde liegende Szenario spezifiziert und die sich daraus ergebenen fachlichen Anforderungen verdeutlicht und ein Blick auf vorhandene kollaborative Umgebungen am Markt geworfen. Diese werden auf den Hinblick hin der Umsetzung und Implementierung einer Zugriffkontrolle untersucht. Ferner wird auf die Umsetzung von Persistenzdiensten ein besonderes Augenmerk gelegt. Das Kapitel 3 findet eine Analyse der Anwendungsfälle statt und basierend darauf werden die technischen Anforderungen an das System spezifiziert. Es wird der Fokus auf Sicherheitsanforderungen an kollaborative Umgebungen gesetzt und die Persistenz wird genauer analysiert. Die Analyse der Persistenzdienste werden genauso spezifiziert, wie die Anforderungen aus Sicht der Informatik. Die gängigsten Zugriffmodelle werden verglichen und analysiert, der Aspekt der Zuordnung von Inhalten zu den

Benutzern wird auch betrachtet. Die hier heraus gehenden Technologien werden dann im Design verwendet. Im Weiteren werden die Systemkomponenten weiter granuliert und in den modularen Aufbau des Systems integriert. Im Kapitel 4 werden die im Kapitel 2 identifizierten Anwendungsfälle weiter verfeinert, sodass sich einige Systemkomponenten daraus herauskristallisieren. Anhand dieser wird eine Systemarchitektur entworfen und die Komponenten festgelegt. Die Funktionalität der Komponenten wird entworfen und dabei auch die Schnittstellen ermittelt. Es wird grundlegendes zum Objektmodell erläutert, sowie das Zugriffsrechtemodell genauer spezifiziert. Anhand der beiden Subsysteme, Persistenz und Zugriffsrechte, werden die einzelnen Punkte noch einmal genau untersucht und Möglichkeiten zur Realisierung aufgezeigt. Am Ende dieses Kapitels wird evaluiert, inwieweit die Realisierung den Anforderungen entsprach. Im letzten Kapitel 5 werden die im Rahmen dieser Arbeit gesammelten Erfahrungen, Erkenntnisse und Zukunftsvisionen besprochen. Dabei wird zuerst die gesamte Arbeit zusammengefasst und die erreichten Ziele dokumentiert. Zum Schluss wird ein Ausblick gegeben wie diese Arbeit weiter fortgesetzt werden kann.

2 Szenario

Das Anwendungsgebiet für einen interaktiven Konferenzraum liegt in der Unterstützung von Arbeitsgruppen, die gemeinsam ein Ziel verfolgen. Dabei geht es in erster Linie um die gemeinsame Entwicklung einer Lösung für ein Problem. Diese Arbeitsgruppen können aus sehr verschiedenen Arbeitsbereichen kommen, z. B. Informatik, Wirtschaftswissenschaft, etc. Ich werde meine Arbeit auf das Szenario der Softwareentwicklung fokussieren. Im Einzelnen bedeutet dies, welche Situationen ergeben sich bei der kompletten Entwicklung eines Softwareprojektes und welche Auswirkungen hat dies auf die Nutzung des interaktiven Konferenzraumes. Dabei wird nicht nur die Arbeit der Softwareentwickler alleine betrachtet, sondern auch die Zusammenarbeit mit dem Kunden. Ein Softwareentwicklungsprojekt zeichnete sich früher durch fest vorgegebene Teilschritte aus, so z. B. im Wasserfallmodell. Heutzutage kann jeder einzelne Projektschritt wieder revidiert werden, und dadurch Auswirkungen auf das gesamte Softwareprojekt haben. Dadurch findet eine kontinuierliche Veränderung im Projektprozess statt, und die Projektgruppe findet sich für die unterschiedlichsten Anlässe im Raum zusammen. Einige Anlässe für ein Meeting zähle ich im Folgenden auf und werde die Phase des *Allgemeinen Brainstormings* genauer betrachten. Genaueres zu den folgenden Situationen ist in der Diplomarbeit von Carola Neumann, [Neumann (2006)], nachzulesen.

- Angebotserstellung / Grobanalyse
- Feinanalyse
- Entwurf / Planung
- Implementierung
- Auslieferung
- Nachbesprechung
- Krisengespräche

Ein typischer Anlass für ein Meeting wäre ein Brainstorming, bei dem der Entwickler, Architekt und Projektleiter anwesend sind. Dort bespricht man das kommende Projekt und sammelt dabei erstmal Ideen. Jeder Teilnehmer hat schon Dokumente vorbereitet oder man greift auf die Projekthistorie zurück. Auf dem Whiteboard wird der erste Architekturvorschlag

erstellt und ein Teilnehmer zeigt über sein Notebook am Beamer seine Präsentation. Genauso wäre es möglich, dass eine Gruppe einen Overheadprojektor oder einen Flipchart für Notizen nutzt.

2.1 Akteure

In einem Softwareentwicklungsprojekt lassen sich diverse Akteure und ihre Tätigkeiten ausmachen. Im Folgenden soll nun kurz skizziert werden, welche Akteure grundsätzlich beteiligt sind. Einen genaueren Einblick liefert die Diplomarbeit von Carola Neumann [Neumann (2006)].

2.1.1 Manager

Er kümmert sich um die Erstellung des Angebotes und stellt sicher, dass der Aufwand zu den Kosten in einem vernünftigen Verhältnis zueinander steht. Er kann Vertragsdokumente mit einer digitalen Unterschrift unterzeichnen.

2.1.2 Projektleiter

Er kümmert sich um die Ressourcen- und Zeitplanung. Er behält im Hintergrund den Überblick und stellt sicher, dass die Termine eingehalten werden. Er kann z. B. Schnittstellendefinitionen mit einer digitalen Unterschrift unterzeichnen.

2.1.3 Entwickler

Diese Person stellt sicher, dass sein Softwaremodul pünktlich fertig gestellt wird. Er steht auch für Architekturdiskussionen zur Verfügung.

2.1.4 Architekt

Der Architekt erstellt die Architektur des Systems und entwickelt aber auch Softwarekomponenten. Er kann typischerweise auf einen großen Erfahrungsschatz zurückgreifen.

2.1.5 Manager (Kunde)

Er steht in Verhandlungen mit der Softwarefirma, kümmert sich darum, dass das Pflichtenheft erfolgreich umgesetzt wird. Er hat die Möglichkeit per digitaler Unterschrift Vertragsdokumente oder Releases zu signieren und zu unterschreiben.

2.1.6 Endnutzer (Kunde)

Er steht den Entwicklern für Nachfragen bezüglich der Oberfläche und Funktionalität zur Verfügung und wird zum Testen des Prototypen herangezogen.

2.2 Fachliche Anforderungen

Hier werden die fachlichen Anforderungen für den interaktiven Konferenzraum beschrieben, die sich aus dem Szenario des Softwareentwicklungsprojektes ergeben.

2.2.1 Digitales Unterschreiben von Dokumenten

In einem Softwareentwicklungsprojekt werden Dokumente vom Erzeuger oder Vorgesetzten unterzeichnet um die Authentizität des Dokumentes zu beurkunden. Das können z. B. Angebote, Projektdefinitionen oder auch Codestücke sein. Unterzeichnete Dokumente haben eine lange Tradition im Rechtssystem, so dass sie zumindest als Augenscheinbeweis und möglicherweise auch als Urkunde genutzt werden können. Wird auf eine geeignete Behandlung solcher sensibler Dokumente verzichtet, dann kann es zu Problemsituationen kommen, die ich genauer behandeln möchte:

- der Empfang von Informationen könnte bestritten werden, daher sind Empfangsbestätigungen notwendig.
- der Empfänger könnte die Verbindlichkeit oder die Authentizität der Information bezweifeln und sie daher ignorieren.
- der Inhalt der Information könnte bestritten werden, da eine Fälschung des Dokumenteninhalts während der Übertragung nicht endgültig auszuschließen wäre.

Es geht ja aber vor allem darum, bei Nichterfüllen oder bei Fehlern die Verantwortung von Personen richtig zuordnen zu können und durch Nutzung von digitalen Unterschriften Fehler überhaupt nicht zuzulassen. Also brauchen wir Maßnahmen, welche die Authentizität von elektronisch übermittelten Dokumenten sicherstellen. Die rechtliche Seite wird durch das Gesetz zur digitalen Signatur (SigG) gesichert, welches in das Gesetz zur Regelung der Rahmenbedingungen für Informations- und Kommunikationsdienste (IuKDG) eingebunden ist. Diese sieht unter [Bundesrepublik Deutschland (2006)] diverse Vorschriften vor, die es zu erfüllen gilt. Da wäre z. B. Paragraph 14 Technische Komponenten, Abschnitt 1.

> (1) Für die Erzeugung und Speicherung von Signaturschlüsseln sowie die Erzeugung und Prüfung digitaler Signaturen sind technische Komponenten mit Sicherheitsvorkehrungen erforderlich, die Fälschungen digitaler Signaturen und Verfälschungen signierter Daten zuverlässig erkennbar machen und gegen unberechtigte Nutzung privater Signaturschlüssel schützen.

Damit müssen die gesetzlichen Anforderungen in diesem Bereich bei der Entwicklung der Architektur beachtet werden.

2.2.2 Transparenz

Der Begriff der Transparenz bezeichnet allgemein das Verbergen bestimmter Einzelheiten oder Gegebenheiten vor den Benutzern. Der Zugriff auf verteilte Dateien und Geräte wird vereinfacht und die Hard- und Software kann ökonomischer genutzt werden. Das fördert die Nutzung des Raumes durch technische Laien, es muss eine hohe Abstraktionsebene vorhanden sein. Die Technik darf für den Benutzer also nicht im Vordergrund stehen. Für den interaktiven Konferenzraum sind folgende Arten von Transparenz wünschenswert:

- **Zugriffstransparenz**, Der Benutzer greift auf den Datenbestand zu, als ob sich dieser lokal auf seinem Rechner befinden würde. Auf die jeweilige Ressource kann in derselben Art und Weise zugegriffen werden, unabhängig davon, ob sie lokal oder entfernt abgespeichert ist.
- **Nebenläufigkeitstransparenz**, Diese bezeichnet die Kontrolle der Synchronisation nebenläufiger oder konkurrierender Zugriffe auf den Datenbestand.
- **Skalierungstransparenz**, Eine Erweiterung des Systems um einen zusätzlichen Teilnehmer kann ohne Modifikation der Systemstruktur oder der Anwendungsalgorithmen durchgeführt werden. Die inkrementelle Erweiterung stellt ein grundlegendes Merkmal verteilter Systeme dar.

- **Replikationstransparenz**, beim Zugriff auf Objekte ist keine Kenntnis über eventuell existierende Replikate notwendig. Ressourcen werden im verteilten System vor allem aus Gründen der Verfügbarkeit vervielfältigt.
- **Fehlertransparenz**, zwei grundsätzliche Fehlertypen sind möglich: Ausfälle der Kommunikationsverbindungen oder Rechnerausfälle. Diese werden vor den Benutzern weitgehend verborgen gehalten. Wenn beispielsweise Daten repliziert im System existieren, wird beim Ausfall eines Dateiservers automatisch auf die Dateien eines anderen Dateiservers zugegriffen.

2.2.3 Multiuserumgebung

Generell kann man sagen, dass man hier eine Multiuserumgebung vorfindet. Es soll möglich sein, das ein Benutzer an mehreren Geräten arbeitet, sowie mehrere Benutzer an einem Gerät. Es wäre vorstellbar, das bei einer Präsentation auf dem PDA die Foliennotizen erscheinen und auf dem großen Wanddisplay die eigentliche Präsentation.

2.2.4 Versionskontrolle

Es soll möglich sein, auf mehrere Versionen eines Dokuments zugreifen zu können, diese sollen dem Benutzer einfach zugänglich sein. Basierend auf diesen Versionen soll es auch möglich sein zu alten Versionsständen zurückkehren zu können.

2.2.5 Zugangskontrolle

Der Konferenzraum soll hier als eine Einheit betrachtet werden, beim Betreten des Raumes ist eine Anmeldung erforderlich. Dazu gehört auch das Anmelden an im Raum genutzten Endgeräten. Dokumente dürfen nur von dazu berechtigten Benutzern eingesehen und verändert werden, dazu muss das System die Authentifikation und Autorisation von Benutzern unterstützen.

2.2.6 Rollen

Ich beschränke mich in dieser Diplomarbeit auf das Szenario der Softwareentwicklung, aber man kann losgelöst von diesem Themenbereich Benutzerrollen identifizieren, die in diesem Konferenzraum Tätigkeiten ausüben können. Es muss möglich sein, zur Vereinfachung der

Administration den Benutzern Rollen zuordnen zu können, die vorgestellten Akteure in unserem Szenario müssen damit abbildbar sein. Allerdings sollten die definierten Rollen auch für weitere Szenarien anwendbar sein, um eine größtmögliche Flexibilität zu erreichen.

2.2.7 Datenablage

Die Daten sollen für den Benutzer transparent gespeichert werden, das heißt er muss nicht den Vorgang des Speicherns explizit auslösen sondern das System handelt eigenständig. Das System legt die Daten unter Berücksichtigung von Metadaten ab und garantiert damit auch das Wiederfinden. Die Metadaten können z. B. den Benutzer, das Projekt, das Format des Dokumentes beinhalten. Der Benutzer muss nicht die genauen Pfade der Dokumente kennen, sondern die Daten werden zum jeweiligen Arbeitszweck bereitgestellt oder können über Eingabe von Metadaten gesucht werden.

2.2.8 Gemeinsames Arbeiten

Alle Nutzer im Raum sollen auf dieselben Dokumente gleichzeitig zugreifen können, die Änderungen jedes Teilnehmers sollen auch sofort für alle sichtbar werden. Irrelevant ist an welchen Geräten die Benutzer gemeinsam arbeiten, jedes Gerät im Raum kann genutzt werden.

2.2.9 Autonomes Arbeiten

Wenn die zentrale Datenablage nicht erreichbar ist, dann wird eine Kopie des Dokumentes auf dem Endgerät abgelegt und kann damit auch außerhalb des Raumes weiter bearbeitet werden. Später wird dann dieses Dokument mit dem Zentraldokument automatisch synchronisiert. Das kann nötig sein, um Teilgruppen Teile des Dokumentes zur Bearbeitung zu überlassen.

2.2.10 Anpassung an Endgerät

Je nach verwendetem Endgerät im Raum muss die Möglichkeit zur Darstellung oder Bearbeitung den Gegebenheiten des Endgerätes angepasst werden. Dazu gehört z. B. die Rücksichtnahme auf Speicher und Prozessorleistung der Endgeräte.

2.2.11 Gruppenbewusstsein

Nach [Schlichter (2002)] ist eine wichtige Anforderung an den interaktiven Konferenzraum das Gruppenbewusstsein, auch *group awareness* genannt. Vereinfacht gesagt bedeutet dies, dass für jeden Benutzer des Raumes die Aktivitäten oder die Anwesenheit anderer Benutzer im Raum augenscheinlich oder transparent werden. Anwendungen, die als echte CSCW-Anwendungen[1] konzipiert sind, besitzen die Eigenschaft *collaboration awareness*. Systeme, die die gemeinsame Nutzung von Einzelbenutzer-Applikationen (Word, Excel) erlauben, heißen *collaboration transparent*. Diese Eigenschaften besitzen eine sehr hohe Relevanz für die Zusammenarbeit im interaktiven Konferenzraum. Maßnahmen zur Stärkung des Gruppenbewusstseins fördern in einer sehr positiven Weise den Gruppenprozess und führen zu einer Steigerung der Effektivität im Team. Die wesentlichen Vorteile sind:

- **Verbesserung der Kommunikation**, die Kommunikation der Teammitglieder wird deutlich gefördert und verbessert, da jedes Teammitglied über die Arbeitsvorgänge und Belastung der anderen Mitglieder informiert ist.
- **Bewusste Entscheidungen**, jedes Teammitglied ist über den derzeitigen Stand zur Erreichung der gemeinsamen Ziele informiert, auftretende Schwierigkeiten können von allen erkannt werden. Dieses Wissen um den Stand des gemeinsamen Projektes führt zu einer bewussteren Entscheidung des Teammitgliedes der nächsten Aktivität im Sinne des gemeinsam zu erreichenden Projektziels.

Allerdings ist ein richtiges Maß zur Förderung des Gruppenbewusstseins nötig, damit es nicht zu Störungen im gemeinsamen Arbeiten kommt. Hier sind auch Aspekte des Datenschutzes und die Wahrung der Privatsphäre zu beachten.

2.2.12 Persistenz

Ein kollaborativer Vorgang im Konferenzraum kann synchron oder asynchron zwischen den Teilnehmer ablaufen. Oft schaffen die Teilnehmer nicht alle ihre Ziele in einer Sitzung zu erreichen, daher müssen sie später an derselben Stelle weitermachen können. Daher sollten die erstellten Dokumente automatisch gespeichert und für die nächste Sitzung automatisch zur Verfügung gestellt werden können. Es muss einen automatischen oder manuellen Prozess geben, der es einem Benutzer erlaubt, seine Daten auf seinem Endgerät autonom weiter zu bearbeiten. Je nach Berechtigung werden die Daten vom zentralen Server auf sein Endgerät kopiert. Dies sollte auch möglich sein, wenn Dokumente auf dem Endgerät von außen mit in den Raum gebracht werden.

[1] Computer Supported Cooperative Work,[Schlichter (2002)]

2.2.13 Import und Export von Daten

Die Benutzer sollen in der Lage sein, ihre Daten so automatisiert wie möglich in das System zu importieren. Das sollte, wenn möglich, automatisiert und transparent für den Benutzer stattfinden. Genauso soll der Benutzer die Möglichkeit haben, außerhalb des Raumes Daten weiter bearbeiten zu können. Es muss also ein Datenim- und export stattfinden können.

2.3 Kollaborative Umgebungen

Es gibt bereits einige Produkte, die versuchen den Rechner als Unterstützung für Gruppenarbeit zu etablieren. Hier gibt es im Wesentlichen drei Ansätze:

- **Hardwarelösungen**, sie betrachten meist nur einen Aspekt der Gruppenarbeit und bieten dementsprechend nur eine Lösung für ein Teilaspekt an.
- **Groupwarelösungen**, hier sind komplett entwickelte Groupwarelösungen zu finden, die bereits einen großen Teil der Anforderungen an einen interaktiven Konferenzraum erfüllen. Sie sind aber als ein in sich geschlossenes System konzipiert worden.
- **Ubiquitous Computing Umgebungen**, diese Produkte wurden für die Verwendung in verteilten Umgebungen entworfen.

In diesem Kapitel werden verschiedene Lösungen zur computergestützten Gruppenarbeit vorgestellt und ihre Merkmale mit den erarbeiteten Anforderungen verglichen. Ich werde die Analyse des möglichen Systems auf die Persistenz und Benutzerverwaltung konzentrieren, um so mögliche Komponenten zur Nutzung in meinem Entwurf zu identifizieren.

2.3.1 Hardwarelösungen

Smartboard

Hierbei handelt es sich um ein interaktives Whiteboard [Smarttech (2005)], dass an einen Computer angeschlossen wird. Man benötigt ferner einen Projektor, um das Bild auf das Smartboard zu projizieren. Dann ist es möglich das Smartboard wie einen Computer zu benutzen, man kann dann mit den Finger die Umgebung steuern, siehe dazu Abbildung 2.1. Dank spezieller Stifte kann man auf dem Smartboard schreiben und dank sofortiger Digitalisierung ist es möglich die erstellten Dokumente zu speichern und mitzunehmen.

Abbildung 2.1: Smartboard

Blueboard

Das Konzept hinter dem Blueboard [Russell und Gossweiler (2001)] ist für schnelles Arbeiten und für kleine Benutzergruppen gedacht. Es handelt sich um große TFT-Monitore, die mit Touchscreenfunktionalität ausgestattet sind. Sie sollen an zentralen Orten den Benutzern eine schnelle Möglichkeit zur Kontaktaufnahme und zum Datenaustausch bieten. Sobald ein Benutzer angemeldet ist, wird sein so genanntes „P-Con“ auf dem Schirm dargestellt, siehe dazu Abbildung 2.2. Durch „Drag and Drop“ ist nun möglich Dateien auszutauschen. Durch Ablegen der Datei auf seinem Icon kann er Dateien kopieren und durch Ablegen von Dateien auf dem öffentlichen Desktop werden sie der Allgemeinheit zur Verfügung gestellt.

Fazit

Beim Smartboard wird die Persistenz manuell durch Speichern der erstellten Inhalte umgesetzt. Der Zugriff auf die Daten erfolgt lokal und von nur einem Benutzer. Damit ist die Anforderung des Mehrbenutzerbetriebes und der ortstransparente Zugriff auf Daten nicht erfüllt. Die Speicherung und das Widerauffinden der Daten erfolgt rein über den Dateinamen, damit ist kein Anhängen von Metadaten möglich. Eine Benutzerverwaltung existiert abhängig vom eingesetzten Betriebssystem. Das Blueboard bietet eine darunterliegende Datenbank und eine einfache Benutzerverwaltung. Die Identifikation der Benutzer geschieht über Token

Abbildung 2.2: Blueboard

in unterschiedlicher Form. Es handelt sich hier um ein abgeschlossenes System, dass die verteilte Nutzung von Daten nicht zulässt. Generell lässt sich sagen, dass das Smartboard und Blueboard für sich autonome Systeme sind, die sich nicht ohne weiteres in eine verteilte Umgebung integrieren lassen.

2.3.2 Groupwarelösungen

BSCW

Das BSCW Shared Workspace System [Fraunhofer (2006)] ist ein Werkzeug für effiziente Zusammenarbeit im Internet und Intranet. Es wird vom Fraunhofer-Institut FIT und dem OrbiTeam Software GmbH entwickelt. Das Programm unterstützt asynchrone und synchrone Zusammenarbeit über das Internet oder im lokalen Netz. Die synchronen Funktionen beschränken sich hierbei auf ein (optionales) Messenger-ähnliches Java-Applet, mit dem sich in Echtzeit die Arbeit der Kollegen verfolgen und kommentieren lässt, auf eine Schnittstelle für Video-Konferenz-Programme sowie auf den integrierten Gruppenkalender, über den Treffen geplant werden können. Der Schwerpunkt lässt sich auf dem asynchronen Teil ausmachen, nämlich den gemeinsamen Arbeitsbereichen mit Dokumenten-Verwaltung. In den Arbeitsbereichen kann der Benutzer eigene Ordner anlegen, dort Dokumente, Kalender, Links usw. einstellen, Bearbeiten, Versionisieren und insbesondere mit anderen Benutzern teilen. Für die Verwaltung der Rechte der einzelnen Benutzer wurde für BSCW ein rollenbasiertes System eingesetzt. Der Benutzer hat auf seine eigenen Dokumente alle nötigen Zugriffsrechte. Will er Ordner mit anderen teilen, ordnet er diesen bei der Einladung einer bestimmten Rolle zu. Diese Rolle bestimmt, welche Rechte der eingeladene Benutzer in diesem Ordner hat. Im Hinblick auf die eingesetzten Technologien ist leider wenig dokumentiert, auf Serverseite kommen die Skriptsprache Python, ein entsprechender Web-Browser sowie eine nicht näher spezifizierte Datenbank zum Einsatz. Auf Client-Seite wird starker Gebrauch von JavaScript

gemacht, dazu kommen optional Java-Applets (in Form von JBrowser, einem Java-Pendant zur HTML-Ansicht, und JMonitor, dem Anfangs genannten Monitoring- und Kommunikationsapplet).

Intraline

Der Hersteller bezeichnet sein Produkt Intraline [Intraline (2006)] als Java-basierte Standardsoftware zur Generierung von Intranets. Das Produkt ist einer der Vorreiter im Bereich der Verschmelzung von Intranet- und Groupware-Technologien.

Die Groupware-Komponente der Intraline bietet neben der E-Mail-Verwaltung sowie dem obligatorischen Termin-, Aufgaben- und Projekt-Management außerdem ein firmeninternes Nachrichten-System, eine ausgefeilte Dokumentenverwaltung, ein Portal und einen Notizblock. Die Software ist funktional in Module aufgeteilt, deren Zugriffsrechte einzeln auf Mitarbeiter, auf Gruppen oder auch projektbezogen steuerbar sind.

Fazit

Diese Systeme bieten schon eine sehr hohe Erfüllung der Anforderungen, die unserem interaktiven Konferenzraum entsprechen. Das BSCW-System bietet bereits ein ausgefeilte Benutzerstruktur und eine Speicherung der Daten durch Metadaten. Leider ist dieses System nur für eine browsergesteuerte Zusammenarbeit gedacht, bei dem jeder Benutzer auf seiner lokalen Kopie arbeitet, und nicht wirklich gemeinsam an einem Objekt. Durch die fehlende Modularität und die nötige Dokumentation bietet sich die Nutzung einzelner Module nicht an. Auch Intraline bietet nicht den verteilten und dynamischen Ansatz den wir mit dem interaktiven Konferenzraum verfolgen.

2.3.3 Ubiquitous Computing Umgebungen

Interactive Workspaces

Dieses Projekt [Stanford University (2005)] wird an der Universität Stanford von Terry Winograd und anderen vorangetrieben. Es handelt sich um das Schaffen einer Plattform zur Unterstützung menschlicher Interaktion unter Benutzung von großen Displays. Daraus entstanden mehrere Unterprojekte, z. B. IRoom und Teamspace.

IRoom

Der IRoom (Interactive Room) besteht aus vier elektronischen Whiteboards und einem Display in einem Konferenztisch eingelassen, alles Touchscreens. Sonst befinden sich noch weitere mobile Geräte in diesem Raum zum Teil mit WLAN-Funktionalität. Man findet dort ein komplexes System von heterogenen Geräten und sich dynamisch ändernden Zuständen vor, das den Benutzer bei seiner täglichen Arbeit unterstützen soll. Man dachte hierbei an drei typische Arbeitsabläufe im IRoom:

- Datenaustausch über Geräte- und Benutzergrenzen hinweg
- Gemeinsames Nutzen und Kontrollieren von Geräten im Raum
- Koordination von Anwendungen

Für diese Zwecke hat man IRos entwickelt.

IRos

IRos (Interactive Room Operating System) [Stanford (2006)] bezeichnet eine Systeminfrastruktur oder auch Middleware, die alle Geräte und Komponenten vereint und für deren Kommunikation sorgt, siehe Abbildung 2.3. Es besteht aus drei Teilen:

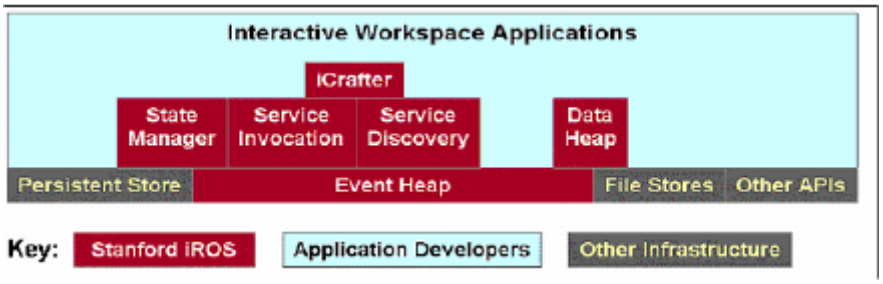

Abbildung 2.3: IRos Infrastruktur

- **Event Heap**, stellt eine generelle Schnittstelle für Anwendungen bereit, um sie soweit wie möglich zu entkoppeln. Durch sein zentrales Eventhandling stellt der Event Heap eine wichtige Komponente im IRos dar und bietet diverse Schnittstellen zur Erstellung von Clientapplikationen.
- **Data Heap**, stellt einen Datenspeicher für die Objekte dar. Jedes Objekt wird anhand von Metadaten und Formatangaben abgelegt. Durch das Nutzen von Metadaten zur Objektklassifikation ist es für die Anwendungen irrelevant, was für eine physikalische Datenstruktur dort hinter steht.

- **ICrafter**, basiert auf dem Event Heap und stellt Services zur Verfügung. Dabei stellt er für das jeweilige Device des Users automatisch das beste Interface zur Verfügung.

Eine interessante im IRoom laufende Anwendung ist der **Smart Presenter**. Hierbei werden auf mehreren Displays mit Hilfe von Webseiten und Powerpointdateien Präsentation zusammengestellt und gezeigt.

teamspace Hierbei handelt sich um ein auf IROS basierendes Projekt, das sich darauf konzentriert die Arbeit einer Gruppe im Konferenzraum zu erleichtern. Es sind auch hier mehrere große Displays im Raum verteilt. Es ist möglich über eine Clientanwendung diverse Services zu starten und damit die Funktionalität im Raum zu nutzen. Damit lassen sich z. B. von einem Punkt aus mehrere Displays steuern und Nachrichten schicken. Es ist genauso möglich Dateien und Urls an alle Teilnehmer im Raum zu senden. Die gemeinsame Nutzung der Geräte steht hier im Vordergrund, Benutzerauthentifizierung und die Datenablage wird eher stiefmütterlich behandelt. Der Data Heap vom IROS wird beim Teamspace-Projekt nicht genutzt.

Abbildung 2.4: Teamspace

Fazit

Die hier vorgestellten Systeme haben die Unterstützung von Menschen bei Meetings und der Nutzung entsprechender Technik im Auge. Es werden teilweise auch Infrastrukturdienste zur Verfügung gestellt, hier ist als Beispiel IRos zu nennen, aber sie bieten keine nutzbaren Persistenzdienste an. Roomware bietet eine Benutzerauthentifikation an, aber alle Benutzer

haben die gleichen Rechte auf diesen Systemen. Daher erfüllen diese Systeme nicht die gestellten Anforderungen.

2.4 Zusammenfassung

Alle vorgestellten Techniken erfüllen nur einen Teil der gestellten Anforderungen. Man hat zum einen durch die Groupwaresysteme eine fortgeschrittene Umsetzung von Autorisation- und Authentifikationsdiensten. Aber es mangelt an einer Modularisierung der Anwendung und an einem verteilten Einsatz. Eine große Abdeckung besteht naturgemäß bei den Ubiquitous Computing-Umgebungen, die ein ähnliches Szenario haben. Es ist möglich einige Dienste in Teilbereiche zu integrieren, aber Persistenz und Sicherheitsbelange wurden komplett ausgeklammert.

Das hier geschilderte Szenario soll nun die Grundlage für diese Arbeit sein. Bestehende Systeme bieten keine Lösung für unsere Anforderungen an, daher werde ich in den nächsten Kapiteln untersuchen, inwieweit sich die Anforderungen umsetzen lassen.

3 Analyse

Beim Entwurf dieses Systems werden einige Annahmen getroffen, die Auswirkungen auf die Anforderungen und das spätere Design haben werden. In meinem Entwurf gehe ich von einer gutmütigen Umgebung aus. Die beteiligten Softwarekomponenten halten sich an die vereinbarten Protokolle, und es findet keine Verschlüsselung der Übertragung statt. Dies lässt sich aber durch die Modularität und Flexibilität meines Entwurfes jederzeit nachrüsten. Es wird davon ausgegangen, dass keine fremden Hardwaregeräte in den Raum integriert werden können. Ich gehe für meinen Entwurf von einer gegebenen zentralen Kommunikationskomponente aus, über die Nachrichten ausgetauscht und Daten verteilt werden können.

3.1 Ermittlung der Anwendungsfälle

Da in dieser Arbeit die Persistenz und das Berechtigungsmodell im Mittelpunkt steht, werden diese Anwendungsfälle nun näher betrachtet. Dazu wurden Anwendungsfalldiagramme nach der UML (Unified Modelling Language)[1]

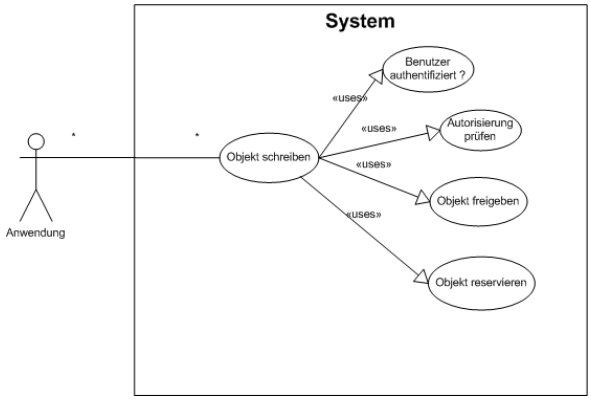

Abbildung 3.1: Usecase Objekt speichern

[1] Die Unified Modelling Language ist eine Spezifikation der OMG (Object Management Group) zur Visualisierung, Konstruktion und Dokumentation von Modellen für Softwaresysteme und Geschäftsmodelle [habil. Reiner R. Dumke (2006)].

Der Usecase *Objekt speichern*, siehe Abbildung 3.1, stellt einen zentralen Punkt für die Entwicklung der Persistenzschicht dar. Eine Anwendung möchte den Dienst zur Objektspeicherung nutzen. Dazu wird im ersten Schritt geprüft, ob der Benutzer authentifiziert ist, und im zweiten Schritt ob die Erlaubnis für diese Operation besteht. Wenn eine dieser Überprüfung fehlschlägt, dann wird die Aktion abgebrochen. Es ist nötig, das Objekt, falls es noch nicht existieren sollte, zu sperren, um einen gleichzeitigen Zugriff auf dieses Objekt zu unterbinden. Wenn der Schreibvorgang erfolgt ist, dann wird die Reservierung des Objektes wieder freigegeben.

In Abbildung 3.2 wird der Usecase *Objekt lesen* dargestellt. Fordert eine Anwendung ein Objekt an, dann wird geprüft, ob der Benutzer authentifiziert ist. Sollte dies der Fall sein, so wird die Autorisierung der geplanten Operation verifiziert. Schlägt einer dieser beiden Operationen fehl, dann wird die Aktion abgebrochen und eine Fehlermeldung ausgegeben. Wichtig ist auch die Reservierung des Objektes, damit niemand eine veraltete Version lesen kann. Wenn die Leseaktion abgeschlossen ist, dann wird die Reservierung wieder aufgehoben.

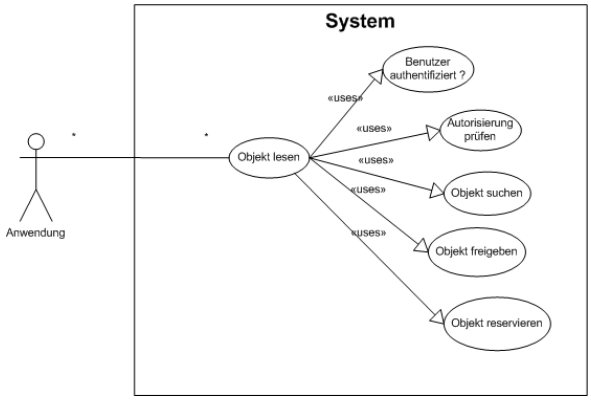

Abbildung 3.2: Usecase Objekt Lesen

3.2 Technische Anforderungen

3.2.1 Anforderungen an die Sicherheit in kollaborative Umgebungen

Die Sicherheit im interaktiven Konferenzraum hat eine große Bedeutung, es gilt dabei für jede Operation die Sicherheitsrichtlinien anzuwenden. Allerdings dürfen die Benutzer im gemeinsamen Erarbeiten von Informationen nicht zu stark beeinträchtigt werden. Man muss also einen guten Mittelweg zwischen kollaborativen Arbeiten und dem Umsetzen der Sicherheitsrichtlinien finden.

Orientierung am Dokumentenmodell

Rechte und Rollen sollen sich abhängig vom jeweiligen Objekt definieren lassen. Für eine Applikation sollen andere Zugriffsrechte gelten als für Dokumente.

Durchsetzung der Sicherheitsrichtlinien über die gesamte Lebensdauer des Objektes

Während der gesamten Lebensdauer des Objektes muss gewährleistet sein, dass immer und zu jeder Zeit ein Objektzugriff nicht ohne Berücksichtigung der Sicherheitsrichtlinien vorgenommen wird. Wirklich jeder Zugriff auf ein Objekt sollte geprüft werden.

Konsistenter Zustand an jedem Speicherort

Die Berechtigungen für ein Objekt sollen an jedem Speicherort immer konsistent und wirksam sein.

Feinkörniges Berechtigungskonzept

Das Berechtigungskonzept sollte die Möglichkeit bieten eine sehr detaillierte und feingranulare Berechtigungszuweisung vornehmen zu können. Berechtigungen sollten deshalb sehr differenziert und granular an einzelne Benutzer und Gruppen von Benutzern vergeben werden können.

Einfaches und flexibles Zuweisen von Rechten

Die Zugriffsrechte sollen einfach und flexibel zuweisbar sein, das hat zur Folge, das das Sicherheitsmodell zwar alle Fälle berücksichtigt, aber trotzdem leicht zu verstehen ist. Die Flexibilität eines Systems zeichnet sich auch durch das flexible Zuweisen von Zugriffsrechten zur Laufzeit aus.

Benutzerbindung

Jede Operation auf oder mit Objekten muss einem Benutzer zugeordnet werden, damit eine Sicherheitsprüfung auch stattfinden kann.

Vererbung

Die Vererbung von Rechten sollte möglich sein, damit nicht für jedes einzelne Objekt die Zugriffsrechte manuell festgelegt werden müssen. [Shen und Dewan (1992)] sehen in der Vererbung einen zentralen Punkt in der Zugriffskontrolle in kollaborativen Umgebungen.

Negative Rechte

Nach [Shen und Dewan (1992)] stellen negative Rechte ein flexibles Mittel dar. Es muss möglich sein einen Zugriff oder eine Operation explizit zu verbieten, was die Flexibilität enorm steigert, weil so ein einfacheres Vergeben von Benutzerrechten möglich ist. Bei einem Verzeichnis, das 100 Dateien enthält, von denen ein Benutzer 99 lesen darf, ist es einfacher ein negatives Recht für jene Datei darin zu erteilen, die der Benutzer nicht lesen darf. Wenn ein Benutzer ein Leserecht für eine Datei hat, und durch eine andere Gruppenzugehörigkeit ein negatives Leserecht innehat, dann muss das restriktivere Recht die höchste Priorität haben, um keinerlei Umgehung der Sicherheitsrichtlinien zu begünstigen. Hier hat also das Verweigern des Lesezugriffs die höhere Priorität.

Berechtigungen sind kumulativ

Dies bedeutet, dass die Rechte sich addieren. Hat ein Benutzer aus der Mitgliedschaft in verschiedenen Gruppen das Recht *Lesen* und das Recht *Ändern*, dann hat er das effektive Recht *Lesen und Ändern*.

Delegation

Delegation beschreibt den Transfer oder die Vererbung von Rechten. Es soll möglich sein, dass ein Benutzer auf einfache Weise Berechtigungen an einen anderen Benutzer oder Gruppe weitergibt. Dabei ist es nötig, dass das System einen Kontrollmechanismen einführt, welcher sicherstellt, dass nur Delegationen akzeptiert werden, die keine Bedingungen verletzen.

3.2.2 Allgemeine Sicherheitsanforderungen

Es sind einige wichtige Punkte bei der Definition von Sicherheitsanforderungen zu beachten, diese werde ich im Folgenden kurz skizzieren.

Transparenz

Die sicherheitsrelevanten Mechanismen müssen für den Benutzer weitestgehend unsichtbar und automatisch realisiert sein. Dies gilt auch für die Anwendungen die keine Kenntnis der Sicherheitsarchitektur haben müssen. Damit lassen sich auch Anwendungen nutzen, die keine eigenen sicherheitsrelevanten Funktionen implementiert haben.

Autorisierung

Hauptaufgabe einer Autorisierungskomponente ist es, zu prüfen, ob ein Benutzer, der dem System durch Authentisierung bekannt ist, eine bestimmte Aktion ausführen darf oder nicht. Hierzu benötigt die Komponente entsprechendes Wissen, nämlich die Menge aller potenziell zu prüfenden Berechtigungen sowie eine Zuordnung zwischen Benutzern und Berechtigungen. Diese Daten müssen administriert werden, wofür die Komponente entsprechende Funktionen zur Verfügung stellen muss.

Kommunikationskonzepte Zwischen Dienstanbieter, Dienstnutzer und der Autorisierungskomponente wird während der Autorisierung kommuniziert, um die Berechtigung des Benutzers für den Dienstzugriff sicherzustellen. Laut [Vollbrecht und Calhoun (2000)] werden dabei drei Strategien unterschieden:

- **The Agent Sequence**, siehe dazu Abbildung 3.3, bei dieser Strategie handelt der Autorisierungsservice als Agent im Auftrag des Dienstnutzers. Ist die Berechtigung des Benutzers sichergestellt, wird der angeforderte Dienst durch den Autorisierungsservice bereitgestellt.

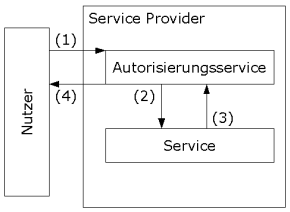

Abbildung 3.3: Agent Sequence

- **The Pull Sequence**, siehe dazu Abbildung 3.4, wenn ein Benutzer einen Dienst anfordert, dann kommuniziert der Dienst mit dem Autorisierungsservice um die Berechtigung des Benutzers festzustellen. Dies läuft völlig transparent für den Dienstnutzer ab.

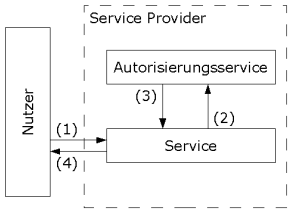

Abbildung 3.4: Pull Sequence

- **The Push Sequence**, siehe dazu Abbildung 3.5, möchte ein Benutzer einen Dienst nutzen, dann muss sich der Benutzer bereits im Vorfeld beim Autorisierungsservice einen Nachweis seiner Berechtigung anfordern. Mittels dieses Nachweises kann der Dienst die Berechtigung des Benutzers überprüfen.

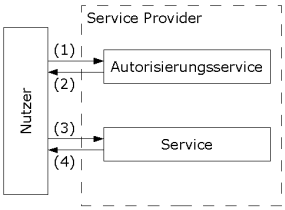

Abbildung 3.5: Push Sequence

Vertraulichkeit

Nach [Eckert (2006)] muss sichergestellt werden, das durch Sicherheitsmechanismen personen- und unternehmensbezogene Daten nur den Personen zugänglich gemacht werden, die die Berechtigung dafür haben. Unberechtigter Zugriff muss unterbunden werden. Die Vertraulichkeit hat in einem Konferenzraum eine zentrale Bedeutung, da die auf Zusammenarbeit ausgerichteten Geräte und Benutzer von jedem visuell und physikalisch zugänglich sein sollen. Aber die Privatsphäre des Einzelnen und den Datenschutzrichtlinien muss eine angemessene Beachtung geschenkt werden.

Eine wichtige Funktion nehmen hier die großen Wanddisplays ein, da hier eine sehr geringe Vertraulichkeit durch die große Sichtbarkeit im Raum gegeben ist. Diese Sichtbarkeit ist im Sinne der kollaborativen Zusammenarbeit gewollt, aber durch die schnelle Erfassbarkeit beim Betreten des Raumes datenschutzrechtlich bedenklich.

Man kann mehrere Vorgehensweisen diskutieren, die im Zusammenhang mit vertraulichen Dokumenten zu leisten sind, um die Vertraulichkeit der Daten zu garantieren. Ein Dokument ist als vertraulich einzustufen, sobald eine Benutzergruppe im Raum keine Leseberechtigung

für dieses Dokument hat. Wenn ein Benutzer versucht ein Dokument dieser Art auf einem öffentlichen Display zu öffnen oder zu veröffentlichen, dann darf das nicht möglich sein. Dies kann man mit einer Fehlermeldung ahnden oder auch mit einem Bestätigungsfenster mit Warnung ob man das wirklich tun möchte. Damit ist sichergestellt, dass ein versehentliches Veröffentlichen der vertraulichen Dokumente nicht möglich ist.

Wenn bereits ein vertrauliches Dokument auf einem öffentlichen Display geöffnet ist und ein Benutzer ohne Leserechte den Raum betreten möchte, dann ist eine Möglichkeit den Raumzugriff komplett zu sperren, solange ein vertrauliches Dokument geöffnet ist. Das widerspricht aber eher dem kollaborativen Arbeiten. Eine andere Möglichkeit ist das Dokument auf dem öffentlichen Display zu schließen oder zu minimieren. Ich würde das Minimieren für sinnvoller erachten, damit ein späteres Bearbeiten eher möglich ist. Die eventuell parallel geöffneten vertraulichen Dokumente auf den persönlichen Geräten bleiben davon unberührt. Hier hat jeder Teilnehmer selbst darauf zu achten, was er sichtbar für die anderen Teilnehmer geöffnet hat und was nicht.

Integrität

Bei der Übertragung von Daten über vielfältige Kommunikationswege muss die Datenintegrität sichergestellt sein. Es darf nicht zu unbemerkten Veränderungen durch das System oder durch Benutzer kommen. Dies stellt nach [Eckert (2006)] eine wichtige Anforderung an die Sicherheit im System dar.

Authentifikation

Nach [Eckert (2006)] stellt die Authentifikationskomponente stellt anhand von Zugangsdaten sicher, dass ein Benutzer das System nur nutzen kann, wenn er seine Identität zweifelsfrei bewiesen hat. Die Zugriffssteuerung sollte in Abhängigkeit vom Zugang eine Vielzahl an Möglichkeiten bieten, dazu gehören diverse Authentifikationsmechanismen:

- Passwörter, TAN-Systeme
- SSL
- Zertifikate
- Smart Card, Token
- Biometrische Systeme, z. B. Fingerabdruckscanner
- HTTP Basic, Formulare

Wichtig in diesem Zusammenhang ist ein gezieltes Sitzungsmanagement und eine Sitzungsverfolgung. Folgende Gesichtspunkte sollten hierbei beachtet werden:

- **Session time-out**, Festlegung einer maximalen Sitzungsdauer
- **Idle time-out**, Inaktive Sitzungen werden verworfen
- **Sitzungsbasierte Aktivitätsverfolgung**

Ziel ist hier ein Single-Sign-On. Benutzer sollen auf unterschiedliche Ressource ohne erneute Anmeldung zugreifen können. Der Konferenzraum wird im Rahmen dieser Arbeit als geschlossenes System betrachtet, physikalische Sicherungsmaßnahmen beim Betreten des Raumes werden erst einmal außer Acht gelassen. Das Betreten des Raumes, physikalisch und virtuell, wird durch die Authentifikation des Benutzers am System ermöglicht. Damit die Authentifikationskomponente die Benutzerdaten prüfen kann, muss die Applikationsschicht Mechanismen zur Authentifizierung bereitstellen.

Die Integration mit bestehenden Anwendungen sollte möglich sein, dazu gehört z. B. das Beziehen der Benutzerdaten von externen Diensten. Dazu gehört auch das Bereitstellen eigener Schnittstellen, um anderen Applikationen die Nutzung eigener Dienste zu ermöglichen.

Auditing

Es muss möglich sein die Abläufe und Nutzungen der Geräte zu überwachen. Dazu gehört in erster Linie das Aufspüren von Sicherheitsverletzungen und Missbrauch von Geräten. Die Protokollierung und Überwachung von Aktivitäten muss unter Berücksichtigung der Datenschutzrichtlinien gewährleistet sein. Dazu gehören das Protokollieren aller Authentifizierungs-, Autorisierungs- und Administrations-Ereignisse (Anfragen und Ergebnisse). Genauso sollte auch eine Protokollierung aller Zugriffe auf nicht geschützte Ressourcen stattfinden, um dann mit einer Ereignisdatenbank eine Auswertung des Benutzerverhaltens vornehmen zu können. Eine Integration in eine bestehende Infrastruktur sollte angestrebt werden, um Schnittstellen der vorhandenen Sicherheitsstruktur nutzen zu können. Dazu gehören z. B. die Nutzung von Intrusion Detection Systemen.

Verschlüsselung

Die Kommunikation sollte auf jeden Fall verschlüsselt ablaufen, um so das Erspähen von Nutzdaten unmöglich zu machen. Hier sollte auf weltweit anerkannte Verschlüsselungsalgorithmen zurückgegriffen werden.

Verfügbarkeit/Ausfallsicherheit

Das System sollte eine hohe Verfügbarkeit besitzen und redundant ausgelegt sein, um eine dauerhafte Verfügbarkeit zu sichern.

3.2.3 Persistenz

Objekte in einer Programmiersprache sind transient, d.h. bei Programmende gehen sie verloren. Allerdings benötigen viele Anwendungen einen dauerhaften Zugriff auf die Objekte, auch über das Programmende hinaus. Typische Objekte sind Benutzer, Rechte, Dokumente, usw. Dies bedingt eine Speicherung der Objekte auf nichtflüchtigen Speichermedien, z. B. in Dateien oder Datenbanken. Dieses Vorgehen nennt man Persistenz.

Der Business Layer soll sich nicht um die physikalische Speicherung der Daten kümmern müssen, sondern nur seine eigentlichen Aufgaben erfüllen. Daher ist es sinnvoll die Datenzugriffe zu kapseln und eine Persistenzschicht einzuführen. Damit wird die Austauschbarkeit der Persistenzschicht spürbar erhöht, siehe Abbildung 3.6.

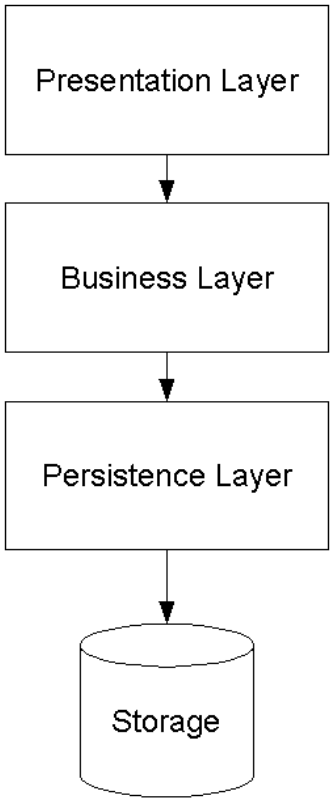

Abbildung 3.6: Four-Tier Architektur

Für die Persistenzschicht ergeben sich eine Reihe von Anforderungen, die ich im Folgenden genauer spezifizieren möchte.

Trennung

Die Zugriffe auf den Datenspeicher dürfen nur durch die Persistenzschicht erfolgen, um die oben beschriebene Kapselung zu erreichen. Im Business-Layer dürfen keine datenhaltungsspezifischen Anweisungen implementiert sein, sonst würden im Falle eines Austausches der Datenhaltung Änderungen in der Businesslogik nötig sein.

optionales Storage

Der Business Logik soll es egal sein, wo die Objekte gespeichert werden und mit welcher Technologie. Es muss also gewährleistet sein, das die Persistenzschicht die Funktionalität für verschiedene Systeme mit unterschiedlichen Versionen bereitstellt.

Transaktionsmanagement

Das gemeinsame Erarbeiten von Inhalten stellt erhöhte Anforderungen an die Kontrolle von Nebenläufigkeiten und an das Transaktionsmanagement. Nach [Schmidt (2006)] muss das Transaktionsmanagement die sogenannten ACID-Eigenschaften garantieren:

- **Atomizität(Atomicity)**, eine Transaktion wird entweder ganz oder gar nicht ausgeführt.
- **Konsistenz(Consistency)**, nach Ausführung der Transaktion ist der Datenbestand nach wie vor in einer widerspruchsfreien Form. Widerspruchsfrei bedeutet, dass die Daten sich in einem konsistenten Zustand befinden. Dies muss auch an unterschiedlichen Orten gewährleistet werden.
- **Isolation(Isolation)**, bei gleichzeitiger Ausführung mehrerer Transaktionen dürfen sich diese nicht gegenseitig beeinflussen.
- **Dauerhaftigkeit(Durability)**, die Auswirkungen einer Transaktion müssen im Datenbestand dauerhaft bestehen bleiben.

Ein weiterer Aspekt ist der Begriff der Nebenläufigkeitskontrolle. Dieser findet auch in der parallelen Verarbeitung und bei verteilten Systemen Anwendung. Das Ziel ist dort, dass nur ein Prozess zurzeit der den gemeinsamen Datenbestand modelliert aktiv sein kann. Die Datenkonsistenz beinhaltet dort, dass alle Manipulationen am Datenbestand durchgeführt wurden und keine neuen Anfragen vorliegen.

Beim interaktiven Konferenzraum gilt es eine synchrone Zusammenarbeit zuzulassen und einen gemeinsamen Datenbestand zu erschaffen, der durch synchrone und asynchrone Modifikationen der einzelnen Teammitglieder entsteht. Durch die Interaktion von Anwendungen und Teammitgliedern im Raum sind die oben genannten Konzepte aus den Fachgebieten nur bedingt anwendbar und besonders auf die menschliche Interaktion muss Rücksicht genommen werden. Man kann es sich nicht leisten, die kollaborative Zusammenarbeit durch lange Reaktionszeiten des Systems oder durch den nachträglichen Abbruch einer ganzen Schrittfolge zu behindern. Das würde der Akzeptanz des Systems deutlich schaden. Bei der Umsetzung einer Nebenläufigkeitskontrolle müssen folgende Anforderungen eingehalten werden:

- **Laufzeit**, die Anwendungen im interaktiven Konferenzraum müssen mit den der hohe Interaktionsrate der Benutzer Schritt halten können. Dazu gehören sehr kurze Antwortzeiten nach irgendwelchen Aktionen und eine schnelle Veröffentlichung an die anderen Teammitglieder im Raum.
- **Gruppenbasiert**, alle Einflüsse auf das Systemverhalten sollen allen beteiligten Benutzern im Raum mitgeteilt werden. Dazu gehören Sperren oder allgemein Wartezeiten, die durch die Nebenläufigkeitskontrolle entstehen. Ausnahmen können bei nichtkritischen Aktionen im Raum gemacht werden.
- **Robustheit**, das Verfahren darf sich nicht durch unvorhergesehene Ereignisse beeinflussen lassen. Dazu gehört z. B. der Ausfall der Netzverbindung, oder das nicht ordnungsgemäße Abmelden eines Benutzers.

Generell gibt es zwei Gruppen von Verfahren, die die Nebenläufigkeitskontrolle genauer spezifizieren, beide Verfahren dienen der Konsistenzsicherung:

- **Optimistische Verfahren**, sie basieren auf der Annahme, dass beim Zugriff keine Inkonsistenzen auftreten werden. Daher ist der Zugriff auf alle Daten jederzeit möglich. Erst wenn eine Inkonsistenz eintritt, werden anschließend Maßnahmen zur Wiederherstellung eines konsistenten Zustands getroffen.
- **Pessimistischen Verfahren**, diese stellen einen Gegensatz zu den optimistischen Verfahren dar. Sie beruhen auf der Annahme, dass Konflikte unvermeidbar sind. Deshalb werden bereits vor dem Zugriff auf die Daten Maßnahmen zur Konsistenzsicherung durchgeführt.

Performanz

Durch das Nutzen einer Persistenzschicht kann es zu Performanceverlusten kommen. Diese Verluste könnten durch das Implementieren eines effektiven Caches gering gehalten werden.

Erweiterbarkeit

Das System sollte Änderungen in der Persistenzschicht ohne Auswirkungen auf die anderen Schichten ermöglichen. Ferner sollten die Änderungen einfach und flexibel durchführbar sein.

Identifizierung

Objekte sollten eindeutig über eine Objekt-ID im ganzen System zugreifbar sein. Diese Objekt-IDs sollten möglichst schlank und einfach zu erzeugen sein.

Transparenz

Die Persistenzschicht stellt für die anderen Schichten eine Blackbox dar, die klar definierte Schnittstellen nach außen anbietet. Wo die Daten gespeichert werden, ist für die anderen Schichten völlig irrelevant.

Skalierbarkeit

Skalierbarkeit bedeutet vor allem, dass Datenvolumina und Anwendungslasten des interaktiven Konferenzraumes in weiten Bereichen ausschließlich durch Einsatz von HW-Ressourcen anwachsen können, ohne dass seine Leistung beeinträchtigt wird. Die Skalierbarkeit bildet die Voraussetzung, dass das einerseits für Anwendungen unterschiedlichster Größe und Leistungsanforderungen nutzen lässt und das das System mit dem interaktiven Konferenzraum wachsen kann.

Schnittstellen

Die Persistenzschicht muss wohldefinierte Schnittstellen zur Nutzung Ihrer Dienste bereitstellen. Dazu gehören Schnittstellen zu Fremdsystemen sowie zu den internen Systemen und Komponenten.

Kommunikationsprotokolle

Das System muss zum einen in der Lage den Netzwerkverkehr zu den Endgeräten so gering wie möglich zu halten, sollte aber für verschiedene Netztopologien Programmierschnittstellen beinhalten.

Online/Offline Szenario

Um das Offline arbeiten mit Daten aus dem interaktiven Konferenzraum zu gewährleisten, kann man sich den Techniken von Replikation und Synchronisation bedienen.

Replikation dient der Einführung von Datenredundanz. Um die durchgeführten Änderungen wieder mit der Datenbank abzugleichen, nutzt man die Synchronisation. Durch das Zusammenwirken dieser beiden Technologien wird das Arbeiten in einem Online/Offline Szenario erst ermöglicht. In einer mobilen Umgebung werden den Clients durch Replikation Daten zur Verfügung gestellt. Durch lokale Speicherung der Daten auf den Clients kann auch im nicht verbundenen Zustand auf die Daten manipulierend zugegriffen werden. Um die Datenkonsistenz im Gesamtsystem zu gewährleisten, müssen die Daten später wieder synchronisiert werden.

Für die Arbeit mit den oben genannten Technologien ergeben sich drei maßgebliche Anforderungen:

- Verfügbarkeit
- Konsistenz
- Performanz

Allerdings gibt es hierbei einen Zielkonflikt, da nicht alle Anforderungen gleichzeitig zu erfüllen sind. Sobald eine Anforderung verstärkt Beachtung findet, werden die anderen Anforderungen abgeschwächt, siehe dazu Abbildung 3.7.

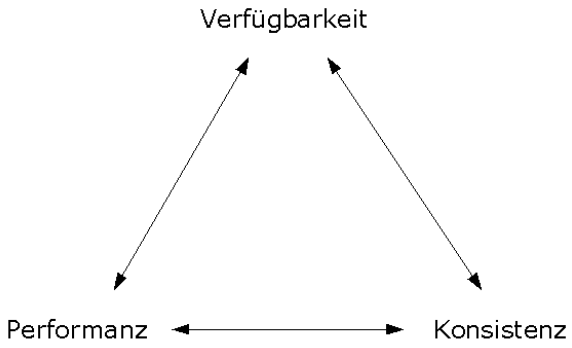

Abbildung 3.7: Zielkonflikt

Allerdings soll das nicht das Thema meiner Arbeit sein, daher verweise ich auf die Diplomarbeit von Aiko Böhm [Böhm (2005)].

Ich möchte trotzdem die Phasen in einem Online/Offline Szenario nochmal genauer spezifizieren, dies ist anschaulich in der Abbildung 3.8 visualisiert.

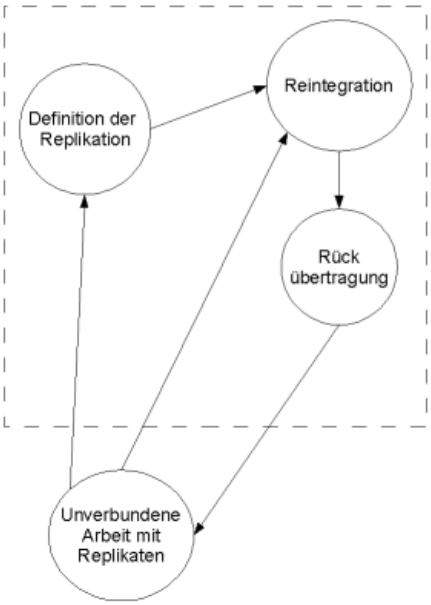

Abbildung 3.8: Replikation und Synchronisation

In der Phase der *Definition der Replikation* wird eine Auswahl getroffen, welche Daten der Replikationsquelle (Persistenzschicht) auf dem Zielsystem (Client) repliziert verfügbar sein sollen. Diesen Replikaten können Zusicherungen mitgegeben werden, d.h. für wen welche Operationen auf den Replikaten gestattet sind und welche Konfliktbehandlung für die replizierten Daten Verwendung finden soll. Die Replikationsdefinition gliedert sich in die Punkte Replikationsschemadefinition und Replikatdefinition auf:

- Das **Replikationsschema** ist der Ausschnitt des Schemas einer relationalen Datenbank auf der Replikationsquelle. Dieser Ausschnitt ist für die Replikation auf mobilen Clients sichtbar, erweitert um Zusatzinformationen. Zum Replikationsschema gehören Zusicherungen für Operationen, z. B. lokal änderbar, sowie alle zur Synchronisation notwendigen Erweiterungen. Diese Festlegung wird über den Administrator getätigt.
- Bei der **Replikatdefinition** werden, bezogen auf ein Replikationsschema, die für eine Aufgabe, einen Zeitpunkt und einen Ort benötigten Daten ausgewählt. Die Definition der Replikate kann durch die Anwendung je nach Benutzerwunsch festgelegt werden.

Ziel bei der Synchronisation ist die Aufrechterhaltung oder Wiederherstellung eines konsistenten Datenbankzustands. Dies wird nötig, wenn auf den redundanten Daten Änderungen stattfinden. Diese Änderungen müssen abgeglichen werden, dazu dient der Synchronisationsprozess. Er besteht aus zwei Phasen: der *Reintegration* und der *Rückübertragung*. Bei der Reintegration findet ein Abgleich der replizierten Daten mit der Replikationsquelle statt. Hierbei muss eine Konfliktbehandlung erfolgen, dabei ist zu beachten, dass Änderungen auf

den Replikaten und auf der Replikationsquelle stattgefunden haben können. Die Rückübertragung überträgt die aktualisierten Daten der Replikationsquelle wieder zurück. Werden die Reintegration und Rückübertragung aufeinander folgend ausgeführt, spricht man von bidirektionaler Synchronisation, wenn nur eine der beiden Phasen ausgeführt wird von unidirektionaler Synchronisation. Dennoch gibt es noch eine gewisse Menge von Teilproblemen, besonders im Bereich der automatischen Konfliktbehandlung, deren Lösung als nicht trivial anzusehen ist.

Metadaten

Bei Metadaten handelt es sich um "Daten über Daten", also Daten, die verschiedene Eigenschaften von Datensätzen beschreiben und den inhaltlichen Kontext beschreiben. Eine offizielle Definition lässt sich im Entwurf der ISO Spezifikation 11179 finden. Metadaten werden dort unter Abschnitt 2 beschrieben als:

> The information and documentation which makes data sets understandable and sharable for users

Die genannte **ISO Definition 11179** ist die wichtigste Definition für Metadaten, Sechs Abschnitte definieren dort die Aufgaben, Prinzipien und Richtlinien für die Klassifizierung, Attributierung, Namenskonvention (Identifikation) und die Strukturierung von Daten, weiteres siehe unter [Gates (2006)].

Nach dieser Definition zu urteilen, handelt es sich bei Metadaten um eine Dokumentation von Daten, die es uns erlauben, diese zu verstehen und damit auch austauschbar mit anderen Benutzern zu machen. Metadaten sind Informationen über Daten, die einen intelligenten und effizienten Zugriff auf die Verwaltung dieser Daten erlauben.

Metadaten sollten erweiterbar und selbstdefinierend sein, da die Masse an Daten immer weiter zunehmen wird und da ansonsten wiederum Metadaten für die vorhandenen Metadaten benötigt werden müssten. Metadaten können in beliebigem Umfang eingesetzt werden, sollten aber sinnvollerweise den Platzbedarf der eigentlichen Daten nicht überschreiten.

Metadaten haben zahlreiche Anwendungsgebiete, man kann diese Bereiche wie folgt unterteilen:

- **Unterstützung der Suche in großen Datenbeständen**, anstatt große Datenbestände mit Hilfe von aufwendiger Volltextsuche zu durchsuchen, ist es effizienter und einfacher, Metadaten anzulegen, die um ein Vielfaches kleiner sind und damit schneller durchsucht werden können. Durch den Einsatz von Metadaten sinkt die durchschnittlich benötigte Datenermittlungszeit deutlich. Besonders in Bezug auf die zunehmende weltweite Vernetzung von Rechnern scheint diese Technik an Bedeutung zu gewinnen.

Sucht man beispielsweise nach bestimmten Informationen, die auf einem entfernten Rechner liegen, möchte man im Normalfall vorher wissen, ob sich diese auch als nützlich erweisen.

- **Datenaustausch**, durch eine weltweite Zunahme des Datenaustausches wird es immer wichtige standardisierte Möglichkeiten zur Kommunikation zu schaffen. Um einen reibungslosen Austausch der Daten zu gewährleisten, bietet sich hier der Einsatz von Metadaten an, Zum einen umfasst diese Mitteilung den syntaktischen Aufbau der übertragenen Datei, zum anderen die eigentliche Interpretation der Daten. Auch diese Aufgabe ist durch Metadaten lösbar. Hier lohnt es sich einen Standard für Metadaten zum geregelten Datenaustausch zu nutzen.
- **Dokumentation der Daten**, Daten bedürfen einer Dokumentation, die Hauptaufgabe der Dokumentation ist bekanntlich die Wissenssicherung. Dieses Wissen über ein System ist erst einmal beim Anwender oder beim Entwickler vorhanden. Die langfristige Werterhaltung kann nur durch gezielte Dokumentation in Form von Metadaten garantiert werden. Es lässt sich damit auch unnötige Redundanz in den Datenbeständen vermeiden, die Integration von Datenbeständen wird damit erleichtert. Durch Metadaten lassen sich auch weitere Qualitätsmerkmale ohne Probleme hinzufügen.

Um den Benutzer nicht unnötig mit dem manuellen Erstellen von Metadaten zu belasten, ist es sinnvoll eine gewisse Anzahl von Metadaten, siehe dazu Tabelle 3.1, automatisch durch das System generieren zu lassen. Dazu gehören alle Daten zum Dokument, die das System selbst ermitteln kann.

NAME	BESCHREIBUNG
id	Eine eindeutige Kennzeichnung
Name	Name des Objektes
Besitzer	Person(en), die Besitzrechte haben
Erstellt(Datum)	Erstellungsdatum des Objektes
Ersteller(Person)	Erzeuger des Objektes
Letzte Änderung(Datum)	Letzte Änderung des Objektes
Beschreibung	Beschreibung des Objektinhaltes
Applikationen	Liste der Applikationen, die dem Objekt zugeordnet sind
Verbunden mit	Liste der Objekte die in Beziehung stehen
Typ	Typ des Objektes

Tabelle 3.1: Automatisch generierte Metadaten

Der Benutzer sollte die Möglichkeit haben weitere Metadaten zum Dokument hinzuzufügen. Entweder erstellt er eigene Beschreibungen oder orientiert sich am Dublin Core Metadata

Element Set, [DCMI (2006)]. Die dort enthaltenen Vorschläge habe ich in einer Tabelle noch mal kurz dargestellt, siehe dazu Tabelle 3.2. Im Sinne des Dublin Core Metadata Element Set wird eine Ressource typischerweise eine Informations- oder Dienstleistungsquelle sein, aber sie könnte auch breiter angewandt werden.

NAME	BESCHREIBUNG
Title	Der an die Ressource vergebene Name
Creator	Das- oder derjenige, der in erster Linie für die Erstellung des Inhalts der Quelle verantwortlich ist.
Subject and Keywords	Das Thema des Inhalts einer Quelle/Ressource.
Description	Erstellungsdatum des Objektes
Publisher	Die Instanz, die für die Veröffentlichung bzw. Verfügbarkeit der Ressource verantwortlich ist.
Contributor	Eine Instanz, die einen Beitrag zum Inhalt der Ressource macht.
Date	Ein Datum, das ein Ereignis im Lebenszyklus der Ressource darstellt.
Resource Type	Die Art bzw. das Genre des Inhalts der Ressource.
Format	Die physische oder digitale Festschreibung der Ressource.
Resource Identification	Eine eindeutige Bezeichnung der Ressource in eienm bestimmten Kontext.
Source	Ein Hinweis auf Quellen, von der die vorliegende Ressource abgeleitet werden kann.
Language	Die Sprache des intellektuellen Inhalts der Ressource.
Relation	Eine Beziehung zu einer verwandten Ressource.
Coverage	Die Reichweite oder der Anwendungsbereich des Inhalts der Ressource.
DC-Rights	Informationen über die Rechte, die in der Ressource und diese betreffend vorhanden sind.

Tabelle 3.2: Metadaten nach dem Dublic Core Metadata Set

3.2.4 Informatik

Laut [Kahlbrandt (1998)] ergeben sich im Rahmen der Informatik für jede Architektur ganz grundsätzliche Anforderungen an das System. Diese werde ich im Folgenden diskutieren.

Modularität

Unter Modularität versteht man das Unterteilen eines Themenkomplexes in mehrere unterschiedliche eigenständige Teile, auch Module genannt. Das macht das Austauschen eines Moduls einfacher und die Wartung wird dadurch auch vereinfacht. Die Änderungen im gesamten System können dadurch gering gehalten werden.

Wiederverwendbarkeit

Wiederverwendbarkeit beschreibt das Einsetzen eines Moduls in einem anderen Produkt oder Projekt. Dadurch lässt sich die Entwicklungszeit eines Projektes signifikant verkürzen, da zum einen nicht neu entwickelt werden muss und zum Anderen das Modul schon getestet wurde. Damit muss das Modul nur noch im Zusammenspiel mit dem neuen Projekt getestet werden.

Wartbarkeit und Erweiterbarkeit

Da die Entwicklung eines Softwareprojektes ein immerwährender Prozess ist, werden an die Wartbarkeit und Erweiterbarkeit hohe Ansprüche gestellt. Es muss zeitnah auf neue Anforderungen oder Fehler reagiert werden, das muss auch ohne sehr tief greifendes Wissen möglich sein. Das Hinzufügen oder Entfernen von neuen Funktionen sollte also flexibel und einfach zu bewerkstelligen sein.

Portabilität

Die Anwendung muss ein hohes Maß an Plattformunabhängigkeit gewährleisten, so dass ein Portieren der Anwendung ohne großen Arbeitsaufwand möglich ist. Somit gehört zur Portabilität nicht nur der Grad der Plattformunabhängigkeit sondern also auch der Aufwand um diese Anwendung Plattformunabhängig zu machen.

Geringe Kosten

Die Entwicklung und Einsatz der Anwendung soll möglichst geringe Kosten verursachen. Deswegen beschränke ich mich bei der Auswahl der Komponenten auf Open-Source Produkte.

3.2.5 Zugriffsrechtemodell

Modelle für Zugriffsrechte aus der Datenbank- und Betriebssystemwelt erfüllen nicht die Anforderungen für kollaborative Umgebungen. In der Vergangenheit wurde nur eine handvoll an Modellen für diesen Bereich entwickelt und umgesetzt. Bei vielen Modellen ist es nur bei theoretischen Überlegungen geblieben, so z. B. Shen und Dewan (1992). Das liegt zum einen daran, dass ein Modell für Zugriffsrechte für ein kollaboratives System sehr komplex sein kann, und damit ist die Implementierung nicht als trivial anzusehen. Es muss auch für den Benutzer einfach zu handhaben und zu verstehen sein. Zum anderen liegt es daran, dass bei der Implementierung eines kollaborativen Systems das Feature der Zugriffskontrolle in der Priorität immer ganz hinten angesiedelt ist, man möchte in den ersten Prototypen das System erst einmal funktionsfähig haben und die Zugriffskontrolle soll später implementiert werden. Ein nachträgliches Erweitern um eine Sicherheitsarchtitektur ist sehr schwierig bis unmöglich. Viele Implementationen von Zugriffsmodellen in kollaborativen Umgebungen kommen oft nicht über den Status eines Prototyps hinaus.

Als grundlegendes Prinzip hat sich nach [Eckert (2006)] bei der Vergabe der Zugriffsrechte das *Need-to-know* Prinzip etabliert, d.h. jeder Benutzer bekommt nur soviel Rechte, wie zur Erledigung seiner Aufgabe nötig ist. Das Prinzip *Separation of Privilege* beschreibt die Unterteilung einer Aktion in mehrere Unteraktionen, die unterschiedlichen Subjekten zugewiesen werden kann. Alle Subjekte müssen zusammen daran arbeiten und damit ist sichergestellt, dass die Sicherheit eines Systems nicht durch den Fehler oder die Kompromittierbarkeit eines einzelnen Subjekts gefährdet werden kann. In wieweit diese Prinzipien umgesetzt werden können, ohne die Kollaborativität im Konferenzraum einzuschränken, muss geprüft werden.

Allgemein lässt sich sagen, das Autorisation 3 Dinge beinhaltet: Ein *Subjekt* hat ein *Recht* eine bestimmte Aktion auf einem *Objekt* auszuführen. Beispiele für Subjekte sind ein angemeldeter Benutzer oder eine Anwendung. Ein Objekt stellt das Ziel einer Aktion eines Subjekts dar, z. B. ein Dokument. Der Begriff Recht bezeichnet hier eine Aktion, die ein Subjekt explizit auf einem Objekt ausführen oder bei einem negativen Zugriffsrecht nicht ausführen darf. Grafisch kann man diese Beziehung zwischen den drei Komponenten Subjekt, Objekt und Zugriffsrecht am einfachsten in einer Lampson-Matrix [Lampson (1974)] darstellen, siehe dazu Abbildung 3.9. Hierbei werden auf der einen Achse alle Objekte und auf der anderen

Achse alle Subjekte des Systems eingetragen. Im Schnittpunkt zwischen einem Objekt und einem Subjekt sind alle Zugriffsrechte aufgeführt, die dem Subjekt für dieses Objekt erteilt wurden.

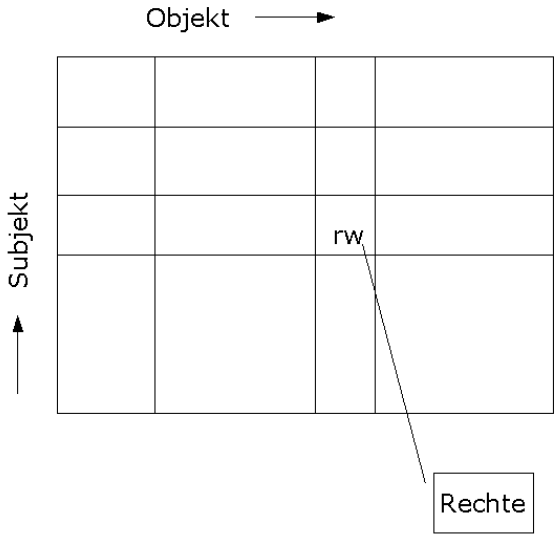

Abbildung 3.9: Lampson Matrix

Die Nachteile dieses Modells sind zum Einen die in heutigen Systemen sehr große Anzahl an Objekten und Subjekten, dadurch erreicht diese Matrix schnell ein unüberschaubare Größe, und zum Anderen sind heutige Systeme von einer sehr großen Dynamik geprägt, d.h. viele Objekte und Subjekte kommen hinzu oder fallen weg. Damit muss eine Spalte und eine Zeile immer neu geschrieben werden. Alle diese Punkte erschweren die Administration dieses Systems erheblich.

Initiiert ein Subjekt einen Zugriff auf ein Objekt, so erfolgt zuerst die Kontrolle, die anhand der vergebenen Zugriffsrechte überprüft, ob der Zugriff gewährt werden kann, und stellt dadurch die Einhaltung der Sicherheitsanforderungen sicher. Um ein IT-System vor unautorisiertem Zugriff zu schützen, müssen mehrere Ebenen betrachtet werden. Zuerst muss eine Sicherheitspolitik beschrieben werden, die genau definiert, wer Zugriff auf welche Prozesse und Objekte haben soll. Eine Sicherheitspolitik kann man durch ein Sicherheitsmodell formalisieren, um dann die geeigneten Sicherheitsmechanismen zu implementieren. Dies soll aber nicht das Thema meiner Arbeit werden. Nichttechnische Aspekte einer Sicherheitspolitik, wie das Verhalten der Benutzer, lassen sich nicht implementieren.

Role-Based Access Control

Der erste Entwurf für ein rollenbasiertes Zugriffsmodell stammt aus dem Jahre 1992 und wurde von David F. Ferrialo und Richard Kuhn et al. [Ferraiolo u. a. (1992)] veröffentlicht. Dieser Entwurf sah eine einfache Zuordnung von Benutzern zu Rollen vor. Dieses Modell wurde RBAC0 genannt. Jede Rolle hat Berechtigungen Operationen auszuführen, siehe dazu Abbildung 3.10.

Abbildung 3.10: RBAC0-Modell

Auch hier kann man wieder das Bank-Beispiel gebrauchen, um das Modell zu verdeutlichen.

> Benutzer A ist Mitglied der Rolle Kassierer. Die Rolle „Kassierer“ hat das Recht abbuchen.

Mit der Erweiterung um Rollenhierarchien wurde das RBAC1-Modell entwickelt, siehe dazu Abbildung 3.11. Rollen können damit in einer Hierarchie abgebildet werden und damit kann die Vererbung genutzt werden.

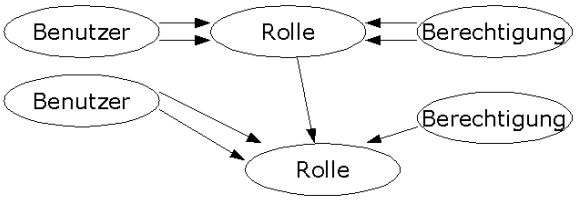

Abbildung 3.11: RBAC1-Modell

> Ein Kassierer ist ein Bankmitarbeiter. Ein Kassenprüfer ist ein Bankmitarbeiter.

Das RBAC2-Modell basiert auf der Definition von Bedingungen, damit ist es möglich Zuweisungen von Berechtigungen zu Rollen an Bedingungen zu knüpfen, siehe dazu Abbildung 3.12. Damit lassen sich Aufgabengebiete in Organisationen klar trennen und der Missbrauch eingrenzen.

> Ein Bankmitarbeiter kann Kassierer und Kassenprüfer sein. Ein Kassenprüfer darf nicht seine eigene Kasse prüfen.

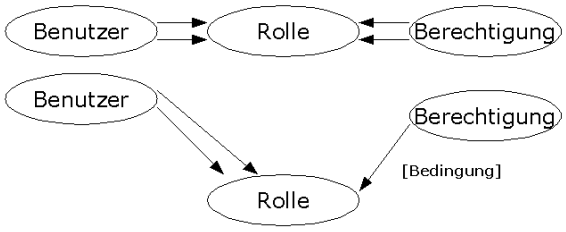

Abbildung 3.12: RBAC2-Modell

Alle Modelle münden dann im RBAC3-Modell, dort werden alle Eigenschaften des vorhergehenden Modells zusammengefasst.

Es gibt eine Reihe von Elementen, die in einem rollenbasierten Zugriffsmodell definiert werden müssen:

- **Benutzer**, ist eine Person
- **Subjekt**, ein aktiver Prozess, der einem Benutzer zugeordnet ist.
- **Rolle**, eine Sammlung von Operationen, die ein Benutzer innerhalb seines Kontextes ausführen darf.
- **Operation**, eine Handlung die auf ein geschütztes Objekt angewendet wird.
- **Funktion**, die Definition einer Geschäftsfunktion, die notwendig wird, um eine Pflichtentrennung vornehmen zu können.
- **Objekt**, auf ein Objekt dürfen autorisierte Operationen ausgeführt werden.

Benutzer sind aufgrund ihrer Position in einer Organisation in Gruppen unterteilt. In unserem Beispiel der Gruppe der Softwareentwickler gehören zu einem Projekt Teilgruppen und verschiedene Benutzer. Jeder Benutzer nimmt dabei eine oder mehrere Rollen im Projekt und im Konferenzraum ein.

Um zuverlässig die Berechtigungen einzelnen Benutzer zuzuordnen, hat es sich nicht als praktikabel erwiesen jedem Benutzer einzeln die Berechtigungen für Operationen und Objekte zuzuweisen. Besonders bei einer sehr großen Anzahl von Benutzern und Berechtigungen ist ein praktikables Arbeiten unmöglich.

Eine Rolle beschreibt Befugnisse und Zuständigkeiten unabhängig von konkreten Akteuren. Nutzer können unter Berücksichtigung der von ihnen wahrzunehmenden Aufgaben Rollen zugewiesen bekommen. Jedem Benutzer können dabei ein oder mehrere Rollen zugewiesen werden. Möchte der Benutzer zur Laufzeit eine bestimmte Aktion ausführen, prüft die Autorisierungskomponente, ob eine der diesem Benutzer zugeordneten Rollen über eine entsprechende Berechtigung verfügt, siehe Abbildung 3.13

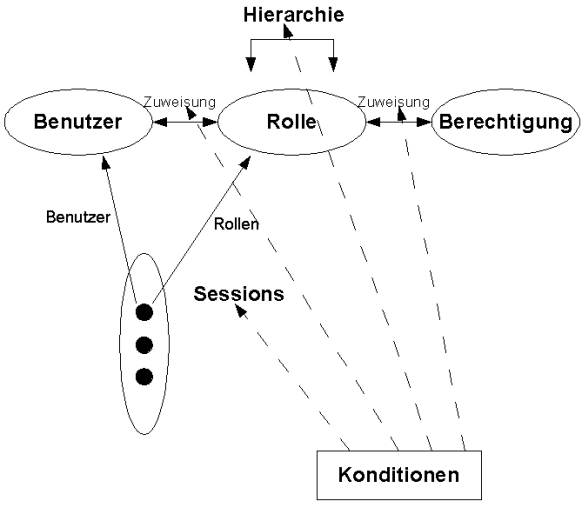

Abbildung 3.13: Rollenbasierte Zugriffsteuerung

Die Zuweisung einer Rolle an einen Nutzer hat keine direkte Auswirkung auf die Privilegien des Benutzers. Damit der Benutzer auch Gebrauch von den Privilegien seiner Rolle machen kann, muss er die Rolle explizit aktivieren. Natürlich kann ein Nutzer nur solche Rollen aktivieren, die ihm zuvor mittels einer entsprechenden Zuweisung verfügbar gemacht wurden. Für ein RBAC-Modell können folgende Sicherheitseigenschaften definiert werden:

- **Rollenhierarchie**, eine Rolle kann eine andere Rolle enthalten, d.h. wenn eine Rolle für eine übergeordnete Rolle autorisiert ist, dann hat sie auch Zugriff auf die untergeordnete Rolle.
- **Rollenautorisierung**, die Rollen, die das Subjekt gerade benutzt, stellen die aktiven Rollen dar, und müssen eine Untermenge der für den Benutzer autorisierten Rollen sein.
- **Rollenkapazität**, für eine Rolle wird eine Mindestkapazität festgelegt, und diese darf nicht überschritten werden.
- **Rollenausführung**, ein Subjekt kann eine Operation nur innerhalb einer Rolle ausführen.
- **Autorisierung der Operation**, ein Subjekt darf nur eine Operation ausführen, wenn diese Operation für seine Rolle zugelassen ist.

In einem RBAC-Modell ist es notwendig, Regeln zur Beschränkung von Rollenmitgliedschaften zu definieren. Damit lässt sich eine Aufgabentrennung gewährleisten, die einen wechselseitigen Ausschluss der Rollenmitgliedschaften zur Folge hat.

- **Statische Pflichtentrennung**, wenn zwei Rollen sich gegenseitig ausschließen, dann kann der Benutzer nur Mitglied der einen Rolle werden.
- **Dynamische Pflichtentrennung**, wenn sich zwei Rollen gegenseitig ausschließen, dann darf das Subjekt die eine Rolle nur aktivieren, wenn er nicht bereits die andere Rolle für sich aktiviert hat.
- **Operative Pflichtentrennung**, jede Operation ist einer Rolle zugeordnet, die einzelnen Operationen einer Geschäftsfunktion dürfen nicht in einer einzigen Rolle ausgeführt werden, da sonst die Gefahr des Missbrauches besteht. Eine Rolle darf nur einer Operation einer Geschäftsfunktion zugeordnet werden,

Diese Einschränkungen bei der Rollenvergabe lassen sich durch ein Beispiel verdeutlichen. Beispiele in einer Bank wären z. B.:

> Kassierer darf nicht Kassenprüfer sein, es gibt genau einen Filialleiter, nur fest Angestellte dürfen als Kundenbetreuer fungieren.

Der Vorteil liegt hier in einer globalen Festschreibung der Sicherheitspolitik, aber sie lässt auch lokal Änderungen zu. Dies hat eine erhöhte Sicherheit bei der Verwaltung zur Folge. Mit einem RBAC-Modell lassen sich Objekte anwendungsspezifisch formulieren und die Berechtigungen für den Zugriff auf Objekte lassen sich aufgabenbezogen vergeben. Mit dieser Strategie lassen sich die Nachteile klassischer objektbezogener Sicherheitsstrategien vermeiden, da sich die Berechtigungsstrukturen von Rollen nur selten ändern. Damit lässt sich auch eine hohe Skalierbarkeit selbst bei dynamisch sich ändernden Umgebungen erreichen. Die RBAC-Modelle eignen sich hervorragend für den Einsatz in verteilten Systemen, der Administrator vergibt global die Berechtigungen für die einzelnen Rollen. Diese Modell wurde durch das [NIST (2000)] im Jahre durch das American National Standards Institute, International Committee for Information Technology Standards (ANSI/INCITS) als ANSI INCITS 359-2004 zertifiziert.

Mandatory Access Control

Nach [Eckert (2006)] ist Mandatory Access Control ein Konzept für die Kontrolle und Durchsetzung von Zugriffsrechten basierend auf systemglobalen Vorgaben. Sie dominieren die benutzerspezifischen Rechtevorgaben, so das der Zugriff aufgrund einer systembestimmten Regel verboten wird, obwohl die benutzerbestimmten Rechte dies zulassen würden. Man kann die benutzerbestimmbaren Rechte dazu nutzen, die systembestimmten Rechte noch weiter einzuschränken, also MAC mit DAC zu ergänzen. Voraussetzung ist, dass die Benutzer niemals direkt, sondern nur durch eine Monitorkomponente auf Objekte zugreifen können. Die vorliegende Infrastruktur muss dementsprechend Maßnahmen und Dienste zur

Verfügung stellen um systembedingte Vorgaben zu implementieren, allerdings bieten aktuelle Betriebssysteme nur durch spezielle Erweiterungen dieses Feature an.

Discretionary Access Control

Benutzerbestimmbare Zugriffskontrollstrategien basieren auf dem Eigentümer-Prinzip, das bedeutet, dass der Eigentümer des Objektes für dessen Schutz verantwortlich ist. Zugriffsrechte werden also pro Objekt vorgenommen, man legt also objektbezogene Sicherheitseigenschaften fest, aber lässt dabei die systemglobalen Eigenschaften außer Acht. Durch das nicht Betrachten von Kommunikationsabhängigkeiten der Objekte untereinander besteht die Möglichkeit eine nicht konsistente Sicherheitsstrategie zu entwickeln. Jeder Benutzer kann eine unternehmensweite Sicherheitsstrategie unterlaufen, in dem er die Rechte an Objekten versehentlich falsch setzt. In großen Systemen kann es für den Administrator sehr umständlich sein, bei jedem neuen Benutzer muss viel eingestellt werden und beim Ausscheiden des Benutzers müssen die Rechte an seinen Objekten neu gesetzt werden. Vor allen Dingen besteht eine sehr große Gefahr, wenn trojanische Pferde im Kontext des Benutzers die Rechte verändern.

Chinese Wall Model

Das Chinese-Wall Sicherheitsmodell wurde ursprünglich entwickelt, um die unzulässige Ausnutzung von Insiderwissen in Bankkreisen zu verhindern. Die grundlegende Idee dieses Modells basiert laut [Eckert (2006)] darauf, dass zukünftige Zugriffsrechte eines Subjektes durch Zugriffe in der Vergangenheit, beschränkt werden. Damit hängt die Zulässigkeit eines Zugriffs immer von der Zugriffshistorie ab. Die Rechte in diesem Modell werden durch eine Zugriffsmatrix realisiert, die durch die Benutzung von universellen Rechten wie *read*, *write*, *execute* festgelegt ist. Die zu schützenden Objekte werden in einem Baum abgelegt, und bestehende Interessenkonflikte werden dort auch abgelegt. Es gibt mehrere Ebenen, Ebene 0 beinhaltet die Objekt, die die Blätter des Baumes sind, Ebene 1 enthält die Zuordnung der Objekte zu unterschiedlichen Unternehmen und Ebene 2 beinhaltet die Interessenskonfliktklassen von konkurrierenden Unternehmen. Das Modell bietet eine anwendungsangepaßte Modellierung von Objekten und Subjekten. Informationsflussbeschränkungen lassen sich sehr umsetzen, aber Vertraulichkeitsaspekte lassen sich nur eingeschränkt modellieren. Die Regeln zum Beschreiben der Zugriffsrechte sind sehr unflexibel und sehr auf spezifische Anwendungsklassen beschränkt.

3.2.6 Datenschutz

Das hamburgische Datenschutzgesetz sieht beim Verarbeiten von personenbezogenen Daten technische und organisatorische Maßnahmen vor. Diese Richtlinien gilt es beim Implementieren der Sicherheitsfunktionalität zu beachten, besonders die Vertraulichkeit spielt im Konferenzraum eine große Rolle. Nach [Hamburger Datenschutzbeauftragter (1990)] sind unter „§ 8 Technische und organisatorische Maßnahmen“ folgende Hinweise zu finden:

> Werden personenbezogene Daten automatisiert verarbeitet, sind technische und organisatorische Maßnahmen zu treffen, die geeignet sind zu gewährleisten, dass nur Befugte die personenbezogenen Daten zur Kenntnis nehmen können.

Der Arbeitskreis *Arbeitskreis eGovernment* hat eine *Orientierungshilfe Datenschutz für Dokumentmanagementsysteme* [Arbeitskreis eGovernment (2006)]herausgebracht. Dort ist aufgelistet, was bei einer Konzeption eines solchen Systemes zu beachten und umzusetzen ist. Es muss also sichergestellt werden, dass die gesetzlichen Anforderungen gegenüber dem Datenschutz im interaktiven Konferenzraum beachtet werden.

3.2.7 Contentzuordnung

Damit das System unterscheiden kann, wer gerade das Dokument modifiziert, muss die Applikation eine Zuordnung zwischen Benutzer und der Änderung herstellen. Dazu kann man diverse Möglichkeiten diskutieren. Eine erhöhte Relevanz gewinnt dieser Punkt, wenn mehrere Personen vor dem öffentlichen Display stehen und gemeinsam Änderungen vornehmen. Arbeiten die Benutzer gemeinsam über die Endgeräte zusammen, so können die Authentifizierungsdaten vom Endgerät mit dem Inhalt übermittelt werden.

RFID

Man kann zum einen die Personen mit RFID-Tags ausrüsten, um so eine maximale Genauigkeit des Standortes zu erreichen. Nun könnte man anhand des Abstandes des Körpers zum öffentlichen Display darauf schließen, das der Benutzer an allen geöffneten Dokumenten mitarbeitet. Nachteil wäre, dass man Änderungen nicht explizit einem Benutzer zuordnen kann, sondern die geöffneten Dokumente allen anwesenden Personen zugeordnet werden. Allerdings kann man als Argument vorbringen, das bei einem gemeinsamen Meeting jeder etwas dazu beisteuert, aber nicht jeder auch am Display steht und die Eingaben machen. Man könnte also denjenigen, der das Dokument geöffnet hat, als Hauptautoren festlegen und den Rest der Teilnehmer als Koautoren. Diese Liste würde sich wahrscheinlich relativ weit eingrenzen, da wahrscheinlich nicht jeder auch Schreibrechte auf das Dokument hat.

Token

Es ist allerdings auch möglich den einzelnen Teilnehmern Token zuzuordnen, z. B. in Form eines Stiftes. Anhand des Tokens kann das System erkennen, wer explizit welche Änderung vornimmt. Das würde die Verknüpfung von Inhalt zu Benutzer deutlich erhöhen, könnte aber zur Folge habe, das eine relativ große Anzahl von Token vorgehalten werden müsste. Außerdem kann es aufwendig sein, bei Beginn des Meetings den Teilnehmer die Token explizit zuzuweisen. Dafür könnte man zwar eine Authentifizierungsstation aufbauen, wo durch Einstecken des Tokens Username und Passwort des Benutzers eingegeben werden kann. Sinnvoller wäre es jedem Benutzer für die Dauer der Tätigkeit für diese Firma ein Token fest zuzuweisen. Damit kann er sich am Raum authentifizieren und es auch gleich für Eingaben nutzen.

Manuelle Zuordnung

Als Lösung kommt auch das manuelle Zuordnen von Autoren zu einem gemeinsam bearbeiteten Dokument in Betracht. Man könnte eine Liste generieren, wo der Autor alle Autoren auswählt und dem Dokument zuordnet. Leider können nun spezielle Teile des Dokumentes einzelnen Autoren nicht mehr zugeordnet werden, was nicht sinnvoll ist.

Fazit

Man sollte den Fall der gemeinsamen Arbeiten vor dem öffentlichen Display noch mal genauer betrachten. Ich gehe davon aus, das die Benutzer keine speziellen Token verwenden, sondern nur z. B. ihre Hände. Wenn sich jeder Teilnehmer für das Bearbeiten des Objekts jedes Mal manuell anmelden müsste, dann kann das eine erhebliche Behinderung der Arbeit nach sich ziehen und das Nichteinhalten der Regel „Erst anmelden, dann editieren“. Vor allen Dingen verschwimmt die Zuordnung zwischen Benutzer und eingegebenen Daten. Hier kann die Lösung nur sein, mit Token zu arbeiten oder mit einer automatischen Zuordnung der Autoren zum Dokument.

4 Design und Realisierung

Der Entwurf steht in diesem Kapitel im Vordergrund. Ausgehend von dem Kapitel Analyse und deren Erkenntnissen, sollen die Anforderungen der hier zugrunde liegenden Softwarekomponenten in einen softwaretechnischen Entwurf umgesetzt werden. Ich werde als Erstes die allgemeine Architektur entwerfen und die notwendigen Komponenten identifizieren und genauer beschreiben. Dann wird das zugrunde liegende Objektmodell genauer spezifiziert, um einen Einblick in eine mögliche Nutzung zu geben. Um grundlegende Abläufe im interaktiven Konferenzraum genauer zu beschreiben, werde ich anhand von Sequenzdiagrammen die grundlegenden Strukturen erläutern. In den beiden letzten Unterkapiteln betrachte ich die beiden Subsysteme, Zugriffsrechteverwaltung und Persistenz, noch einmal gesondert und werde dort Realisierungsmöglichkeiten für die Komponenten aufzeigen. Im Rahmen der Architektur sind einige grundlegende Definitionen zu vereinbaren. Mein Entwurf des Systems findet unter festgelegten Vorbedingungen statt:

- Hardware
  - Öffentliche Displays bestehen aus einem Rechner mit einem großen Display, es muß von mehreren Benutzern gleichzeitig zu nutzen sein.
  - Endgeräte mit Tastatur, Maus und Display müssen vorhanden sein, das wären z.b. Notebooks oder PDA´s.
  - Ein zentraler Server für die Infrastruktur muss vorhanden sein.
- Software
  - Die verwendeten Betriebssysteme sollten multiuserfähig und über TCP/IP Verbindungen aufbauen können. Dies kann drahtlos oder per Kabel geschehen.

4.1 Architektur

Eine Architektur entscheidet mit über die Qualität des Systems. Eine gute Architektur zeichnet sich durch Flexibilität aus, gerade in der heutigen Zeit spielt Anpassbarkeit an ständige Änderungen und neue Ansprüche eine große Rolle. So kann es in einem bestimmten

Kontext beispielsweise sinnvoll sein, ein großes System in mehrere, voneinander unabhängige Komponenten zu zerlegen um die Wartbarkeit der einzelnen Komponenten möglichst zur Laufzeit gewährleisten zu können. Im Zusammenhang mit der Softwareentwicklung von komplexen Systemen werden von den Entwicklern häufig Architekturmuster zur Komplexitätsbewältigung eingesetzt. Architekturmuster liegen auf der höchsten Ebene und sollen das Grundgerüst für eine Systemarchitektur schafften. Auf der darunter liegenden Ebene stehen Entwurfsmuster. Sie dienen der Verfeinerung von Komponenten oder Subsystemen und haben grundsätzlich keine Auswirkungen auf die Struktur des Gesamtsystems. Um eine Architektur für den interaktiven Konferenzraum zu spezifizieren, werde ich im Folgenden einige Architekturmuster vorstellen und auf ihre Tauglichkeit für den interaktiven Konferenzraum prüfen.

- **Client-Server-Architektur**, es gibt hierbei eine Menge von Servern (Prozesse, die Dienste anbieten) und eine Menge von Clients, die die Server kennen und deren Dienste nutzen. Dann wird noch ein Netzwerk benötigt, das Server und Clients verbindet, falls diese auf verschiedenen Rechnern laufen, siehe dazu Abbildung 4.1.

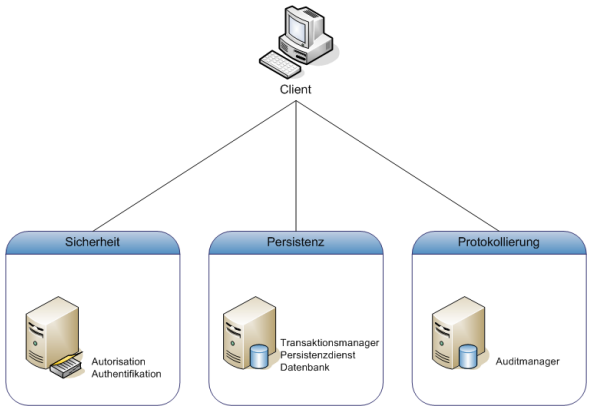

Abbildung 4.1: Client-Server Architektur

Jeder Client greift direkt auf die angebotenen Dienste der Server zu. Beim Objektzugriff wird auf dem Client eine Überprüfung der Autorisation durchgeführt. Dazu muss eine Implementierung eines Autorisations- und Authentifikationsclient auf der Clientapplikation durchgeführt werden. Allerdings sind die Nachteile Effizienzprobleme beim Austausch großer Datenmengen. Jeder Client muss Sicherheits-Prüfungen selbst durchführen, Clients wissen häufig nicht, welche Server es gibt und welche Dienste angeboten werden. Gängige Implementationen für Client-Server-Architekturen sind z. B. Corba oder RMI. Dies hat zur Folge das eine Anwendung alle spezifischen Protokolle der Komponenten selbst implementieren muss. Bei einer Änderung der Schnittstellendefinition einer Komponente müssen alle Anwendungen angepasst werden. Ausserdem muß jeder Client die genauen Standorte der anderen Komponenten kennen.

Durch die Implementation von Adaptern für jeden Dienst, den man nutzen möchte, ist die Erweiterbarkeit sehr eingeschränkt, siehe dazu Abbildung 4.2.

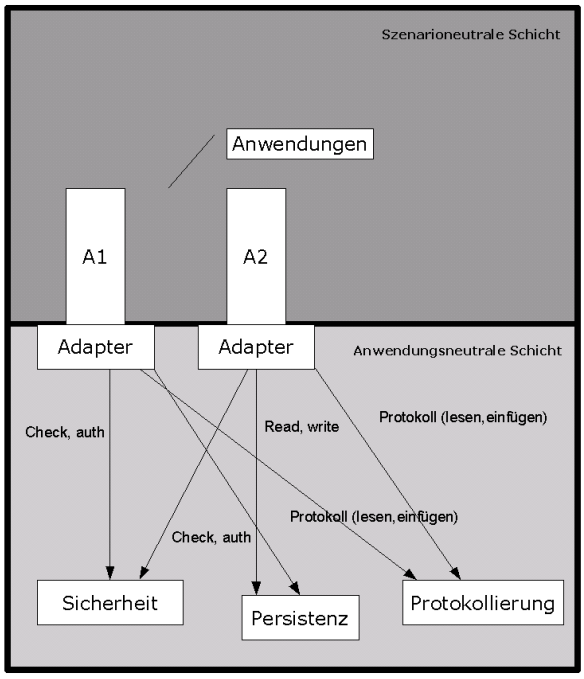

Abbildung 4.2: Mögliche Architektur des Raumes

- **Tupelräume, Blackboard**, es handelt sich hierbei um einen gemeinsam genutzten (virtuell globalen) Speicher, siehe dazu Abbildung 4.3. Er ist die Vermittlungskomponente zwischen Clients mit ihren Dienstanforderungen und den entsprechenden Dienstanbietern. Die anfragende Komponente muss nicht wissen, wo der richtige Anbieter lokalisiert ist, sondern nur der Broker. Die Tupel bestehen aus einer geordnete Menge typisierter Datenwerte. Diese Tupel können von beliebigen Teilnehmern und geschrieben und gelöscht werden. Der Vorteil liegt in einer relativ starken Entkopplung der Teilnehmer. Man kann diese zentrale Komponente passiv oder aktiv vorsehen.
  - **aktiv**, dabei hat die Komponente die Aufgabe die Clients zu registrieren und Aktionen auszuführen. Bei auftretenden Events informiert die zentrale Anwendung die jeweilige Komponente über diesen Fall, siehe dazu Abbildung 4.4. Die zentrale Komponente gewährleistet die Sicherheitsprüfungen und den konsistenten Zugriff auf die Persistenz. Ein Client stellt eine Anfrage für ein Objekt und die zentrale Komponente führt eine Autorisisationsprüfung durch, ob die Anfrage erlaubt ist. Wenn dies der Fall ist, dann wird die Persistenz mit der Anfrage benachrichtigt. Der Client meldet sich für den Empfang der gewünschten Information an - z. B. Abonnement eines Informationskanals (channel) und erhält automatisch

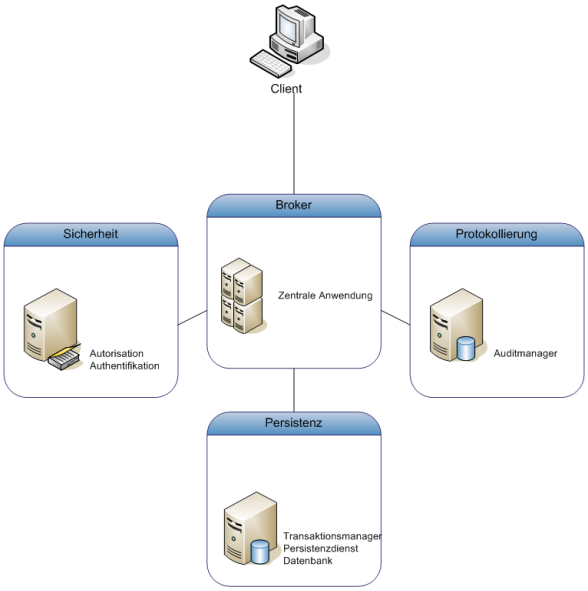

Abbildung 4.3: Architektur mit Broker

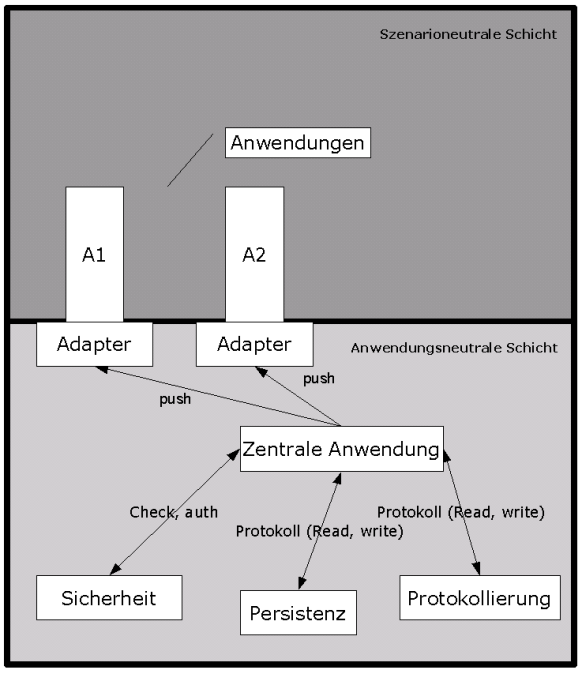

Abbildung 4.4: Aktive Blackboardarchtitektur

(aktualisierte) Information, sobald diese zur Verfügung stehen. Jede Anwendung muss nur einen Adapter für die Kommunikation mit dem Broker implementieren und muss daher nur ein Protokoll definieren, um mit den anderen Diensten zu kommunizieren. Dieser Mechanismus hat den Nachteil, dass die zentrale Komponente wissen muss, welche Anwendungen gerade verbunden sind. Die Anwendungen können dann aber nicht entscheiden, ob ein Mangel an Ereignissen durch einen Absturz des Servers verursacht wurde oder ob nur keine Nachrichten für sie angekommen sind. Wenn die Informationen zu den Clients nicht persistiert werden, dann müssen bei jedem Neustart die Informationen wieder neu gewonnen werden.

– **passiv**, die zentrale Komponente wird als reiner Nachrichtenspeicher genutzt, und besitzt keine weitere Funktionalität. Die Clients müssen die gewünschte Information aktiv anfordern. Da der Client aber nicht weiß, ob bzw. wann sich eine Information geändert hat, wird dadurch periodische Nachfrage beim Server notwendig (polling), siehe dazu Abbildung 4.5. Durch diese Vorgehensweise sind

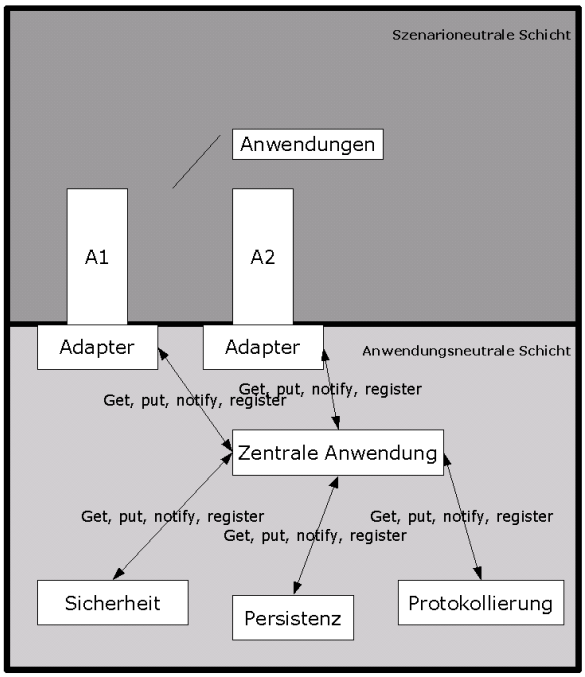

Abbildung 4.5: Passive Blackboardarchitektur

beim Server keine Informationen über Clients notwendig, da diese aktiv Informationen abfragen. Die Clients haben auch keine Schwierigkeiten zwischen einem Serverausfall und einem Mangel an Ereignissen zu unterscheiden.

Für den interaktiven Konferenzraum bietet sich der Einsatz einer zentralen Komponente an, da die Daten und Dienste im interaktiven Konferenzraum verteilt sind und die Aufenthaltsorte sich laufend ändern können. Das System wird laufend erweitert und verändert. Die Komponenten sollten über ein Netzwerk auf entfernte Dienste anderer Komponenten zugreifen können. Für den Zugriff sollte es nicht notwendig sein, implementierungsspezifische Details der Dienste, oder ihre Lokalisierung zu kennen. Außerdem ist der Aufenthaltsort flexibel und somit nicht zwingend bekannt. Die Komponenten im Raum sollten zur Laufzeit ausgewechselt, hinzugefügt, oder entfernt werden können. Dadurch können Anwender und Anbieter von Diensten zur Laufzeit verschwinden oder hinzukommen. Dafür wird eine Vermittlungskomponente zwischen Clients mit ihren Dienstanforderungen und den entsprechenden Servern benötigt. Es muss die anfragende Komponente nicht wissen, wo der richtige Anbieter lokalisiert ist, sondern der Broker. Die Anwendungen melden sich selbständig beim Broker an und stellen so über Schnittstellen ihre Dienste zur Verfügung. Die zentrale Komponente sollte aktiv agieren, und damit z. B. die Prüfung der Berechtigungen für einen Zugriff der abfragenden Anwendung abnehmen. Damit muss der Client keine sicherheitsspezifische Logik implementieren. Es wäre dann möglich soviele Dienste wie möglich an die zentrale Anwendung zu delegieren. Der Client kümmert sich um die Kommunikation mit der zentralen Anwendung und handelt dort ein Protokoll aus. Die Clients sollten aber wie beim passiven Broker ein Polling betreiben, um die Entkopplung zwischen Client und zentraler Komponenten so groß wie möglich zu halten.

Da die Komponenten der Architektur modular und eigenständig sind, ist die Wahl der Architektur vollkommen unabhängig von den Komponenten. Durch wohldefinierte Schnittstelle können die Komponenten mit jeder Architektur betrieben werden.

Die hier betrachtete Architektur besteht aus vier Grundkomponenten, dem Broker, der Sicherheit, der Protokollierung, dem Client und der Persistenz.

1. **Broker**, stellt die grundlegende Kommunikationsdienste für die Komponenten zur Verfügung.
2. **Sicherheit**, die Dienste zur Authentifikation und Autorisation stehen in diesem Bereich zur Verfügung.
3. **Protokollierung**, protokolliert alle Änderungen im Raum und bietet Möglichkeiten zur Filterung.
4. **Persistenz**, ist für die Datenablage zuständig, stellt die Konsistenz der Daten sicher und bietet Schnittstellen zum Zugriff.
5. **Client**, nutzt die angebotenen Dienste im Raum.

Anhand dieser Architektur werde ich die Komponenten jetzt genauer spezifizieren und beschreiben.

4.2 Komponenten

Eine Zerlegung eines Systems in Komponenten reduziert zwar nicht die Komplexität des Gesamtsystems, wird aber deutlich beherrschbarer da über die zusätzliche Abstraktionsebene die innere Struktur der Komponenten in den Hintergrund tritt. Jede Komponente verfügt über definierte Schnittstellen und kann so in mehreren Anwendungskontexten verwendet werden. Die Neustrukturierung eines Systems in kooperierende Komponenten dient hier der Erhöhung der Einsetzbarkeit auf unterschiedlichen Plattformen, damit fällt eine Verteilung auf mehrere Hardwarekomponenten deutlich leichter. Durch die gewonnene Flexibilität kann das System einfacher auf unterschiedliche Nutzungsanforderungen angepasst werden und damit ist die Skalierbarkeit des Gesamtsystems deutlich erhöht. Unter dem Aspekt des Sicherheitsgedanken ist durch die Zerlegung des Systems in Softwarekomponenten ein deutlicher Sicherheitsgewinn zu verzeichnen, da durch die Verteilung auf verschiedene separat abgesicherte Hardwarekomponenten Sicherheitsrichtlinien dedizierter umzusetzen sind. Die Realisierung eines geforderten Sicherheitsniveaus wird durch komponentenbasierte, mehrschichtige Architekturen wesentlich vereinfacht. Durch ein effizientes Zusammenspiel der Komponenten und durch die Möglichkeit, Sicherheitsanforderungen und -maßnahmen isoliert pro Komponente zu definieren, kann das Sicherheitsniveau gesteigert werden. Hierfür können verschiedene Maßnahmen eingesetzt werden, von denen einige im Folgenden kurz angesprochen werden:

- **Segmentierung des Netzwerks, Sicherheitszonen**, durch die Aufteilung des Gesamtsystems in Komponenten ist es möglich eine Segmentierung des Netzwerkes durchzuführen und die Systemkomponenten verschiedenen Sicherheitszonen zuzuordnen. Es bietet sich in unserem Beispiel durchaus an, den Persistenzserver und den Anwendungsserver verschiedenen Zonen zuzuordnen, um einen größtmöglichen Schutz zu gewährleisten. Die einzelnen Netzwerksegmente können beispielsweise durch Firewalls getrennt werden.
- **Trennung von Funktionalitäten**, durch die Zerlegung in Komponenten kann ein hoher Spezialisierungsgrad der einzelnen Module erreicht und eine schmale Schnittstelle realisiert werden. Es wird nur eine geringe Anzahl an Diensten über eine Komponente bereitgestellt, damit ist sie von möglichen Sicherheitslücken anderer Dienste nicht betroffen und bietet selbst wenig Angriffspunkte. Die Kernfunktion der Komponente kann optimal implementiert werden.
- **Modulbasierte Erweiterung bestehender Applikationen um Sicherheitsfunktionalität**, es gibt im System Komponenten, die eine eingeschränkte Sicherheitsfunktionalität besitzen. Durch den modularen Aufbau ist es ohne weiteres möglich, mit geringem Aufwand Sicherheitsfunktionen nachträglich zu implementieren. Die modul-

übergreifende Nutzbarkeit von Diensten kann hier anderen Modulen wichtigen Nutzen bringen und verbessert damit das Sicherheitsniveau der gesamten Architektur.

- **Durchsetzung von dedizierten Sicherheitsrichtlinien**, durch das Aufteilen des Netzwerkes in Sicherheitszonen kann durch z. B. durch eine Firewall aufgrund von Sicherheitsrichtlinien nur definierter Netzwerkverkehr zugelassen werden. So kann verhindert werden, dass z. B. die Datenbank ohne Authentifizierung oder Autorisierung genutzt werden kann.

4.2.1 Autorisationsmanager

Der Autorisationsmanager prüft, ob ein Benutzer, der dem System durch Authentisierung gegenüber dem Authentifikationsmanager bekannt ist, eine bestimmte Aktion ausführen darf oder nicht. Die Komponente benötigt, um dieses erfüllen zu können, Zugriff auf die potentiell zu prüfenden Berechtigungen und eine Zuordnung von Benutzern und Rechten. Da ich in unserer Anwendung ein rollenbasiertes Zugriffmodell verwende, muss der Autorisationsmanager auf die Zuordnung von Rollen zu Benutzern zugreifen können. Die Funktionalität dieser Komponente stellt im Wesentlichen zwei Module zur Verfügung:

- Administration der Autorisationsdaten
- Prüfung der Autorisation

Aus der Sicht anderer Komponenten und Anwendungen ist lediglich die Autorisierungsprüfung relevant, für die Administration der Autorisierungsdaten wird der Zugriff über eine extra Komponente realisiert. Das Modul der Autorisationsprüfung stellt Schnittstellen für andere Komponenten und Anwendungen bereit, die Administration der Autorisationsdaten ist nur für Administrationskomponenten nutzbar, und sollte anderen Komponenten auch nicht zur Verfügung stehen. Die Autorisierungsprüfung bietet eine Schnittstelle zur Überprüfung von Berechtigungen, das heißt Anwendungen und andere Komponenten können vor Durchführung von Aktionen mit Hilfe des Moduls prüfen, ob der jeweilige Nutzer dazu berechtigt ist. Dazu müssen dem Modul folgende Daten zur Verfügung gestellt werden:

- die Berechtigung, die geprüft werden soll,
- das Objekt, auf das sich das Recht bezieht,
- die Identität des Benutzers,
- weitere Daten, die die Session des Benutzers beinhaltet

Im Einzelnen kann die Gruppe der Administratoren folgende Daten pflegen:

- Menge der Rollen einer Anwendung

- Menge der Rechte einer Anwendung
- Zuordnung von Benutzern zu Rollen
- Menge der prüfbaren Berechtigungen
- Zuordnung von Berechtigungen zu Rollen sowie ggf. Abhängigkeiten, die bei einer Berechtigungsprüfung zu berücksichtigen sind.

Es muss möglich sein, dass bei der Durchführung der Berechtigungsprüfung die Attribute der Anwendungs-Session für die Autorisierungskomponente lesbar zugreifbar sind. Dies kann entweder durch eine entsprechende Schnittstelle des Session-Managers ermöglicht werden oder aber durch Übergabe der Daten an die Prüffunktion. Da dieser Bereich besonders sicherheitskritisch ist, muss daher unbedingt manipulationssicher protokollierbar sein, also mittels der Komponente Protokollmanager. Dies gilt im Besonderen für die Administration der Autorisierungsdaten. Das Modul sollte im Sinne der Nutzbarkeit alle möglichen Berechtigungen des Benutzers liefern können, damit die Präsentationsschicht jeweils nur die Flächen verfügbar macht, für die der Benutzer auch die Berechtigung hat. Durch den hohen Schutzbedarf sind auch hier wichtige Sicherheitsanforderungen an diese Komponente zu beachten:

- Die Autorisierungsprüfung darf nicht umgangen werden können.
- Die Integrität der Autorisierungsdaten muss gewährleistet sein.
- Die Komponente selbst ist vor Integritätsverletzungen zu schützen.
- Bei hohem Schutzbedarf ist eine Sonderbehandlung für Administratoren vorzusehen.
- Alle Administrationsvorgänge müssen nachvollziehbar sein.

Der Aspekt der Unumgehbarkeit der Autorisierungsprüfung verdient hier eine gesonderte Betrachtung, da hier eine besonders enge Verzahnung mit der Businesslogik vorliegt. Wichtig ist in diesem Zusammenhang eine klare Trennung der Businesslogik von den Funktionen zur Autorisierungsprüfung. Für einfache Prüfungen ist dieser Aspekt bereits gegeben, da die Autorisierungsprüfung in einer extra Komponente bereitsteht. Wenn für eine spezielle Prüfung in der die Session-Daten mit einbezogen werden, dann muss sichergestellt sein, dass die Prüfroutine komplett im Kontext der Autorisierungskomponente ausgeführt wird.

Durch diese Trennung ist es möglich, den Aufruf der Autorisierungsprüfung aus der eigentlichen Businesslogik zu entfernen. Entweder bietet die Anwendung eine Möglichkeit, selbst diesen Aufruf automatisch vorzunehmen, oder aber das Framework, auf dem die Anwendung basiert, ist entsprechend ausgelegt. Beide Verfahren vermeiden, dass die Autorisierungsprüfung durch fehlerhaft implementierte Businesslogik umgangen wird.

Die Autorisierungskomponente stellt den anderen Komponenten mehrere Dienste zur Verfügung, diese sind exemplarisch in der Tabelle 4.1 beschrieben. Der Dienst *checkDSD()* prüft,

DIENST	BESCHREIBUNG	PARAMETER
hasRole()	Prüft, ob das Objekt Berechtigung für diese Rolle hat	Rollenname, Objektname
getPermissions()	Listet alle Berechtigungen auf, die aus der Rollenzuordnung resultieren	Objektname
getRoles()	Listet alle zugewiesenen Rollen auf	
checkDSD()	Prüft die Abhängigkeit	Rolle1
checkAccess()	Prüft die Berechtigung für eine Operation	Rolle1, Operation, Objekt

Tabelle 4.1: Dienste des Autorisationsmanagers

ob sich zwei Rollen im Rahmen der dynamischen Pflichtentrennung[1] ausschliessen. Ein Benutzer kann zwei sich gegenseitig ausschliessende Rollen nicht gleichzeitig aktivieren.

Zur Administration der Autoriserungsdaten steht ein weiteres Modul zur Verfügung. Das Administrationsmodul bietet eine Reihe von Diensten an, siehe dazu Tabelle 4.2. Diese werden aber nur von einer speziellen Administrationsanwendung genutzt. Bei einem hohen Schutzbedarf sollte die Administration der Rechte des Administrators ebenfalls über das Administrationsmodul der Komponente erfolgen. Dadurch ist es möglich, Administrations-Rechte wirksam von übrigen Benutzerrechten zu trennen, bei Bedarf können auch die Administrations-Rechte auf verschiedene Administrator-Rollen verteilt werden (z. B. können die Aufgaben „Anlegen von Rollen und Rechten“ und „Zuordnung von Benutzern zu Rollen“ an unterschiedliche Personengruppen vergeben werden). Letztlich bedeutet dies eine Einschränkung der Rechte eines Administrators, jeder Administrator ist nur zu denjenigen Aktionen autorisiert, die er für die Erfüllung seiner Aufgaben benötigt. Um dies durchzusetzen, muss natürlich darauf geachtet werden, dass es einem Administrator nicht möglich ist, seine eigenen Rechte zu erweitern.

4.2.2 Authentifikationsmanager

Jede Komponente des Gesamtsystems kann über die Authentifikationskomponente die Dabei werden verschiedene Verfahren unterstützt, die sich in drei Gruppen einteilen lassen:

[1] siehe dazu Kapitel 3.2.5

DIENST	BESCHREIBUNG	PARAMETER
createRole()	Legt eine Rolle neu an	Rollenname
deleteRole()	Löscht eine Rolle	Rollenname
revokePermission()	Entfernt eine Berechtigung	Berechtigungstyp, Objekt
grantPermission()	Erteilt eine Berechtigung	Berechtigungstyp, Objekt
addInheritance()	Erstellt eine Vererbungshierarchie zwischen zwei Rollen	Rolle1, Rolle2
addAscendant()	Eine neue Rolle wird als Nachfolger einer Rolle angelegt	Rolle1
addDSDRole()	Eine Bedingung zur Pflichtentrennung wird angelegt	Rolle1
deleteDSDRole()	Eine Bedingung zur Pflichtentrennung wird gelöscht	Rolle1

Tabelle 4.2: Administrationsdienste des Autorisationsmanagers

- Benutzerkennung / Passwort
- Token
- Biometrische Erkennung

Je nach Anforderungen an die Sicherheit können zwei dieser Verfahren kombiniert werden und damit eine „Zwei-Faktor-Authentifikation“ realisiert werden. Für dieses Verfahren sind einmal der Besitz eines Tokens oder Karte und das Wissen eines Passwortes nötig. Allerdings lässt sich diese Form auch über Einmalpasswörtern und Zertifikaten implementieren. Für die Anwendung des interaktiven Konferenzraumes bieten sich zum einen die Kombination aus Benutzername und Passwort an und zum Anderen Token, die dem Benutzer fest oder für ein Meeting zugeordnet werden. Ziel ist hier ein größtmögliches Maß an Sicherheit zu gewährleisten, aber den Benutzer in seiner Arbeit nicht zu behindern. Die Anmeldung des Benutzers kann zum einen auf dem Endgerät vorgenommen werden, oder an einem gemeinsamen Display. Für den ersten Prototyp werde ich mich rein auf die Benutzername/Passwort Authentifikation beschränken, weitere Authentifizierungsverfahren sind bei Bedarf nachrüstbar.

Diese Komponente implementiert ein Protokollierungsmodul, das Authentifikationsvorgänge erfasst und ablegt. Die Authentifizierungsdaten werden in einer Datenbank abgelegt, allerdings hat aus Sicherheitsüberlegungen nur die Authentifizierungskomponente Zugriff darauf. Die Verwaltung der Authentifizierungsinformationen soll zentral erfolgen, dies ist durch die

Auslagerung als Komponente möglich und die Nutzung einer „Single-Sign-on-Lösung“ ist deutlich vereinfacht. Durch eine einmalige Authentifizierung kann ein Benutzer alle Dienste nutzen. Die Authentifikationskomponente bietet den anderen Komponenten Dienste an, siehe dazu Tabelle 4.3. Die Administration der Benutzerdaten erfolgt über ein seperates Modul,

DIENST	BESCHREIBUNG	PARAMETER
login()	Meldet den Benutzer an und prüft die Identität	Benutzername, Passwort
logout()	Meldet den Benutzer ab	

Tabelle 4.3: Dienste des Authentifikationmanagers

siehe dazu Tabelle 4.4

DIENST	BESCHREIBUNG	PARAMETER
adduser()	Legt einen Benutzer im System an	Zugangsdaten
deleteUser()	löscht einen Benutzer	Benutzername
lock()	Sperrt einen Benutzer	Benutzername
unlock()	Entsperrt einen Benutzer	Benutzername

Tabelle 4.4: Administrationsdienste des Authentifikationmanagers

4.2.3 Auditmanager

Die Komponente Auditmanager hat die Aufgabe festzuhalten, welcher Akteur zu welchem Zeitpunkt welche Aktion durchgeführt hat. Neben der Sicherstellung der Nachweisbarkeit von Vorgängen dienen diese Protokolle auch als wichtige Basis für Revisionen und können dabei helfen, unerlaubten Umgang mit Anwendungen aufzudecken. Der Auditmanager stellt eine spezielle Art der Protokollierung zur Verfügung, die sich durch folgende Punkte auszeichnet:

- Es werden alle Daten protokolliert, die zur Nachvollziehbarkeit eines beliebigen Vorgangs erforderlich sind.
- Die Protokollierung erfolgt in einem strukturierten Format.
- Die Protokolldaten werden in einer relationalen Datenbank abgelegt.
- Zum Zwecke der Langzeitarchivierung werden die Protokolldaten mit Erreichen eines definierbaren Alters an eine Archivierungskomponente übergeben.

Wichtig ist, dass der Auditmanager lediglich die Funktionalität zur Verfügung stellt, eine Protokollierung durchzuführen. Die Verantwortung zur korrekten Protokollierung der Vorgänge im System liegt bei den jeweiligen Anwendungen und Komponenten, die den Auditmanager nutzen. Dies schließt auch die Einhaltung gesetzlicher Regelungen wie des Datenschutzgesetzes ein, die ebenfalls in der Verantwortung der aufrufenden Komponenten liegen, siehe dazu Kapitel 3.2.6. Der Auditmanager kann jedoch unterstützende Mechanismen zur Verfügung stellen. Man kann über Löschkennzeichen eine automatische Löschung veranlassen, um den datenschutzrechtlichen Vorschriften gerecht zu werden.

Da durch die Protokollierung große Datenmengen anfallen können, kann der Auditmanager nicht zur Langzeit-Archivierung genutzt werden. Die Daten werden deshalb mit Erreichen eines konfigurierbaren Alters exportiert und an ein Langzeit-Archivierungssystem übergeben.

Das Auditmanager bietet eine Schnittstelle zum Hinzufügen von Einträgen. Diese Schnittstelle kann sowohl von Anwendungen als auch von anderen Komponenten genutzt werden. Aus Performance-Gründen sollte die Schnittstelle des Auditmanagers asynchron ausgelegt, d.h. die Kommunikation der Anwendungen und Schnittstellen läuft nachrichtenbasiert ab. Der Sender erwartet dabei lediglich eine Bestätigung, dass die Daten entgegengenommen wurden, die eigentliche Verarbeitung der Daten kann zu einem späteren Zeitpunkt erfolgen. Damit die Daten ausgewertet werden können, verfügt der Auditmanager über eine entsprechende Schnittstelle, die autorisierten Personen Auswertungen erlaubt. Dort kann dann mittels geeigneter Metadaten nach den Datensätzen gesucht werden. Die Komponente Auditmanager stellt folgende Dienste bereit, siehe dazu Tabelle 4.5.

DIENST	BESCHREIBUNG	PARAMETER
writeMessage()	Protokolliert eine Nachricht	Nachricht
readMessage()	Liest eine Nachricht aus dem Speicher	Nachrichten-id
filterMessages()	Holt anhand von Metadaten die geeigneten Nachrichten	Metadaten

Tabelle 4.5: Dienste des Auditmanagers

4.2.4 Broker

Das zu entwerfende System besteht aus einem zentralen Server und mehreren Clients. Die Topologie ist sternförmig um einen Server als Mittelpunkt aufgebaut. Der Broker hat in dieser Konstellation folgende Aufgaben zu erledigen:

- **Zentrale Konsistenzsicherung**, alle Zugriffe auf den gemeinsamen Datenbestand erfolgen über den Server.
- **Minimierung der Kommunikationsverbindungen**, die Gesamtanzahl der Verbindungen im System entspricht der Anzahl der Clients bzw. Client-Anwendungen. Dadurch vereinfacht sich ihre Verwaltung.
- **Erreichbarkeit**, der Server sorgt für die Erreichbarkeit und Adressierung der Clients bzw. Client-Anwendungen.
- **Erweiterbarkeit um neue Teilnehmer**, neue Teilnehmer melden sich direkt beim Server an und erhalten von diesem eine Bestätigung.

Die zentrale Anwendung stellt das Kommunikationsmedium für alle angeschlossenen Geräte dar. Die Anwendungen können ihre Kommunikation ausschließlich über die zentrale Anwendung abwickeln und brauchen so nicht mit anderen Geräten im Raum direkt zu kommunizieren. Ich gebe in meiner Diplomarbeit nur einen sehr groben Überblick was der Broker oder die zentrale Anwendung für Dienste bereitstellen soll, weiteres siehe dazu in der Diplomarbeit von Lars Burfeindt [Burfeindt (2006)]. Der Broker bietet den Komponenten des Systems Dienste an, die ich in der Tabelle 4.6 exemplarisch spezifiziere. Der Zugriff auf die zentrale

DIENST	BESCHREIBUNG	PARAMETER
writeMessage()	Schickt eine Nachricht an den Broker	Tuple
listMessages()	Liest alle Nachrichten, die dem Suchparameter entsprechen	search parameter
subscribeMessages()	Abonniert alle Nachrichten, die dem Suchpara- meter entsprechen	search parameter

Tabelle 4.6: Dienste des Brokers

Kommunikationskomponente, das Blackboard, findet über diese Schnittstelle statt. So muss eine Anwendung nicht die Adresse des Blackboards, oder die spezifischen Aufrufe kennen, um es zu nutzen. So könnte es eventuell durch eine andere Lösung ersetzt werden, ohne die Anwendungen zu ändern. In dieser Schnittstelle können auch weitere Funktionen definiert sein, die der tatsächliche Server nicht bereitstellt.

4.2.5 Client

Der Client wird in dieser Diplomarbeit nur am Rande behandelt, weiteres zum Client findet sich in der Diplomarbeit von Roman Bartnik [Bartnik (2006)].

4.2.6 Transaktionsmanager

Ein Transaktionsmanger ist eine Softwarekomponente, welche den atomaren Charakter vieler gleichzeitig ablaufender Transaktionen sicherstellt. In Umgebungen, die durch den gleichzeitigen Zugriff vieler Benutzer über viele Prozesse auf viele Ressourcen gekennzeichnet sind, regelt der Transaktionsmanager den Zugriff.

Im interaktiven Konferenzraum gilt es in erster Linie den Zugriff auf ein Objekt von mehreren Benutzern und Anwendungen zu koordinieren. Wir haben eine Persistenzschicht mit einer physikalischen Datenablage dahinter. Der Transaktionsmanager muss durch sinnvolles Sperren von Objekten ein konsistentes Gesamtbild für alle Teilnehmer erhalten. Generell haben wir im interaktiven Konferenzraum sehr lange Transaktionen zu verzeichnen, da die Teilnehmer ein Dokument öffnen und während der Bearbeitungsphase nicht wieder schließen. Für die Datenbank ist die Transaktion immer aktiv, und für andere Benutzer gesperrt. Dies ist für ein kollaboratives Arbeiten problematisch, siehe dazu Abbildung 4.6.

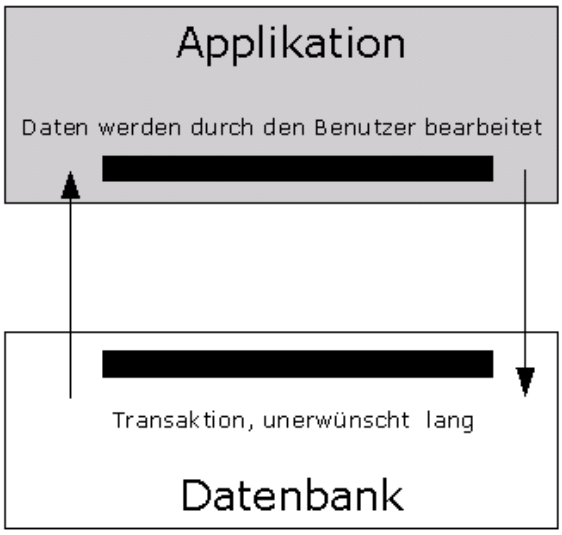

Abbildung 4.6: Lange Transaktion

Die Nebenläufigkeitsstrategie für Änderungen an gemeinsamen Daten kann man wie im Folgenden diskutiert umsetzen. Bei sich überlappenden Transaktionen kann es zu einer gewissen Menge an Inkonsistenzen kommen, z. B. die unbeabsichtigte nachträgliche Änderung

von gemeinsamen Daten. Man kann aber auch nach der Auffassung verfahren, dass die zuletzt gespeicherten Daten eh die neuesten sind. Allerdings gibt es auch Vorgänge (Lagerhaltung, Kontenbewegungen) bei denen das nicht erwünscht ist, da sollte dann bei Konflikten informiert werden. Folgendes Verfahren bietet einen Lösungsansatz: Generell sollte man eine lange Transaktionen vermeiden, und daher die Zugriffe auf kurze Transaktionen verlagern, siehe dazu Abbildung 4.7. Beim Lesen eines Datensatzes wird ein Zeitstempel mitgenommen, welcher den Zeitpunkt der letzten Änderung beinhaltet. Wenn beim Zurückschreiben der Zeitstempel sich geändert hat, dann werden die Änderungen verworfen, sonst werden sie geschrieben. Allerdings ist dieses Verfahren sehr fehlerträchtig, da in der Zwischenzeit ein anderer Benutzer die Ursprungsdaten verändert haben könnte. Daher kann man das Zeitstempelverfahren noch weiter optimieren, man könnte im Falle eines Konfliktes die Applikation oder den Benutzer informieren und er kann dann entscheiden, ob er die Daten trotzdem speichern will. Allerdings können sich die Daten auch wieder ändern, solange er sich entscheidet. Möglich ist auch anstelle eines Zeitstempels die ursprünglich von der Applikation gelesenen Daten aufzubewahren plus die vorgenommenen Änderungen. Zum Zeitpunkt des Zurückschreibens wird geprüft ob die ursprünglich gelesenen Daten noch mit den aktuellen in der Datenbank übereinstimmen. Der Nachteil liegt im erhöhten Speicher- und Zeitbedarf für das Aufbewahren und Vergleichen der ursprünglichen Daten.

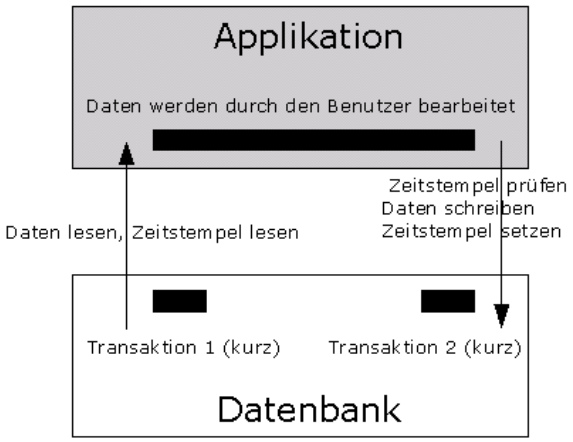

Abbildung 4.7: Kurze Transaktion

Man kann anstelle eines Zeitstempels auch Konfliktprüfungsregeln anwenden, der gleichzeitige Zugriff auf ein Objekt wird erlaubt und durch eine semantische Prüfung über Regeln kann ein gleichzeitiges Schreiben möglich sein, solange sich zwei Aktionen auf ein Objekt nicht ausschließen.

Eine wichtige Größe im Zusammenhang mit Sperrverfahren bei kollaborativen Umgebungen ist die *Sperrgranularität*. Ein Dokument wird von mehreren Benutzern gleichzeitig bearbeitet

und in einzelne Segmente unterteilt. Je kleiner ein Segment ist, desto feiner ist die Sperrgranularität. Mit feiner Granularität steigt die Anzahl der Sperranforderungen und der Grad der Auslastung für den Transaktionsmanager. Je größer die Granularität ist, desto niedriger ist die Auslastung des Transaktionsmanagers. Man kann folgende Größen der Sperrgranualarität diskutieren:

- **Komplettes Dokument**, diese Granularität wirkt der kollaborativen Gruppenarbeit entgegen, denn es kann zu jedem Zeitpunkt nur maximal ein Anwender am Dokument arbeiten.
- **Ein Wort, Satz oder Zeichen**, diese Größe sollte für den kollaborativen Gebrauch zu klein bemessen sein, da hier auch der Verwaltungsoverhead für etwaige Sperren steigt.
- **Durch Benutzer frei wählbar**, prinzipiell ein guter Ansatz, allerdings muss das Einstellen der Grenzen einfach und flexibel für den Benutzer machbar sein. Das Festlegen der Grenzen könnte aber eine nicht unerhebliche Unterbrechung der Arbeit nach sich ziehen und den Sinn des kollaborativen Arbeitens zunichte machen.
- **Innerhalb definierter struktureller Einheiten frei wählbar**, Die Granularität ist abhängig vom Manipulationsort. Standardmäßig wird der kleinste Bereich im Umfeld der Manipulationseinheit des Anwenders gewählt. Somit würde in einer Tabelle gerade der Teil gesperrt, in dem sich der Anwender aufhält. Analoges gilt für ein Kapitel, Unterkapitel oder Absatz. Die Verfahrensweise stellt eine Verbesserung gegenüber der absolut frei wählbaren Granularität dar, da ein impliziter Bezug zu den Daten im Dokument hergestellt wird

Den besten Ansatz scheint die Granularität abhängig vom Manipulationsort und von den Daten zu haben. Für ein Textdokument wäre das z. B. ein Absatz und für ein Bild ein bestimmter Pixelbereich sinnvoll.

Für den interaktiven Konferenzraum bietet sich eine Lösung an, die zwei wichtige Komponenten beinhaltet. Zum einen eine Vermeidung von langen Wartezeiten für die Benutzer durch geeignete Sperrmechanismen und zum anderen die Wahl einer sinnvollen Sperrgranularität. Eine relativ kleine Sperrgranularität verhindert das übermäßige gleichzeitige Zugreifen auf Daten und damit auch die Konflikte die entstehen könnten. Durch das Informieren der Teilnehmer über Dokumente, die in Bearbeitung sind, lassen sich über das Gruppenbewusstsein weitere konkurrierende Zugriffe minimieren. Dies ist natürlich in der Präsentationsschicht explizit umzusetzen.

Im Allgemeinen sollten die Benutzer im interaktiven Konferenzraum untereinander ein soziales Protokoll respektieren. Dieses soziale Protokoll sollte beinhalten, dass die Anwender nicht z. B. versuchen, gleichzeitig ein Wort in einem Text zu verändern. Vor Sitzungsbeginn

oder auch während einer Sitzung sollten die Anwender sich auf verschiedene Arbeitsbereiche einigen.

Innerhalb kollaborativer Texteditoren, die eine logischen Dokumentstruktur verwenden, z.B. ein Datenobjekt pro Abschnitt, kann der Einsatz von Objektsperren ebenfalls sinnvoll sein. Wenn ein Teilnehmer einen Abschnitt verändern will, beantragt er über die graphische Oberfläche die Sperrung dieses Bereiches. Nach Erhalt des Bearbeitungsrechts kann er den Bereich editieren. Die anderen Teilnehmer können zwar die Veränderungen in diesem Abschnitt beobachten, aber keinen Einfluss auf diese nehmen. Wichtig sind auch hier die Granularität und die Dokumentstruktur. Die Komponente Transaktionsmanager sollte folgende Dienste bereitstellen:

DIENST	BESCHREIBUNG	PARAMETER
startTransaction	Startet eine neue Transaktion	Session-id, Transaktions-id
stopTransaction()	Stoppt eine Transaktion	Transaktions-id
commit()	Persistiert die Transaktion	Transaktions-id
rollback()	Führt Rollback durch	Transaktions-id
getOpenTransactions()	Listet alle offnen Transaktionen in der Session auf	

Tabelle 4.7: Dienste des Transaktionsmanagers

4.2.7 Notificationmanager

Im Rahmen der Förderung des Gruppenbewusstseins nimmt der Notificationmanager eine wichtige Rolle ein. Diese Komponente stellt Dienste zur Benachrichtigung der Teilnehmer zur Verfügung. Jeder Teilnehmer soll die Änderungen des unmittelbar beteiligten Teammitgliedes in Echtzeit sehen können. Dazu informiert der Notificationmanager alle beteiligten Clients, die dasselbe Dokument geöffnet haben über Änderungen an dem Dokument. Er wird darüber von der Persistenzschicht informiert, wenn eine Änderung in die Datenbank geschrieben wurde. Die Präsentationsschicht des Clients übernimmt dann die visuelle Darstellung der Änderungen. Es stellt sich die Frage, wie häufig alle Bearbeiter des Dokuments vom Notificationmanager benachrichtigt werden sollen. Dazu stellt der Begriff der Benachrichtigungsgranularität einen wichtigen Baustein dar.

Die Benachrichtigungsgranularität bezeichnet die Segmentgröße des bearbeiteten Dokumentes, das nach einer Modifikation an die anderen Benutzer übermittelt wird. Es kann nicht größer als die Sperrgranularität[2] sein, da der modifizierte Bereich immer innerhalb des

[2]Erklärung des Begriffes Sperrgranularität im Kapitel 4.2.6

gesperrten Bereichs liegen muss. Falls die Benachrichtigungsgranularität der Sperrgranularität entspricht, sehen die anderen Teilnehmer die vorgenommenen Veränderungen nach der Freigabe. Es ist aber auch denkbar, daß die Granularitäten entkoppelt werden und die Benachrichtigungsgranularität kleiner gewählt wird als die Sperrgranularität, um eine frühere Benachrichtigung über Veränderungen zu ermöglichen. Beispielsweise könnte als Granularität der Benachrichtigung ein Wort und als Sperrgranularität ein Absatz gewählt werden. Der Vorteil dieser Verfahrensweise besteht darin, dass eine Modifikation nicht abrupt auf dem Bildschirm erscheint, sondern von den anderen Teilnehmern in ihrem Verlauf beobachtet werden kann, da jede Veränderung ab Wortgröße auch bei ihnen sichtbar wird. Zu klein sollte die Benachrichtigungsgranularität auch nicht gewählt werden, damit die Benutzer in ihrer eigenen Arbeit nicht gestört werden. Vor allen Dingen bei Benutzern, die viele Fehler einstreuen, könnte es für die Mitautoren sehr nervend sein, diese jeweils auch mitzubekommen.

4.2.8 Sessionmanager

Eine Session repräsentiert den angemeldeten Benutzer im System, ihr wichtigster Teil ist die Identität dieses Benutzers. In der Session werden alle Attribute, die den Zustand der Sitzung ausmachen, zusammengefasst. Das Erschaffen einer Session beginnt mit der erfolgreichen Anmeldung eines Benutzers und endet mit dem Abmelden. Das Ende der Session wird dabei entweder explizit durch den Benutzer eingeleitet (Abmelden) oder automatisch nach Ablauf einer gewissen Zeit ohne Aktivität des Benutzers (Time-Out, etc.).

Durch das Verwenden von Sessions wird gewährleistet, dass jedes Modul einer Anwendung, das eine Anmeldung voraussetzt, nur nach erfolgreicher Authentifizierung genutzt werden kann. Um nicht jedes Mal bei Aufruf eines Moduls eine Prüfung der Authentifizierungskomponente durchführen zu müssen, wird eine Benutzersession erschaffen, in der sich der Anwendungsserver die notwendigen Authentifizierungsdaten merken kann.

Jede Session wird durch eine eindeutige Kennung (Session-ID) identifiziert. Jeder Client übergibt bei jedem Aufruf seine Session-ID, dadurch ist eine Prüfung der Session gewährleistet. Es wird geprüft, ob zu dieser Session-ID eine gültige Session existiert. Das Prüfen und Verwalten von Sessions können unabhängig von fachlichen Anwendungen im System durchgeführt werden und sind damit als eigenständige Komponente nutzbar. Der Sessionmanager stellt eine Vielzahl an Diensten bereit:

- Starten einer neuen Session.
- Beenden einer Session (Abmelden, timout)
- Prüfen einer Session / eines Aufrufs

- Speichern und Abruf von Daten in einer Session.
- Absicherung der Sessions gegen unbefugte Übernahme
- Verwaltung von Sessions (Start, Beenden, Liste führen)
- Verwaltung von Anwendungen (Start, Beenden, Liste führen)
- Fehlerbehandlung
- ggfs. Daten aus Offlinesitzungen bereitsstellen

4.2.9 Persistenz

Auswahl des Persistenzmechanismus

Die Objekte müssen persistiert werden, dafür gibt es verschiedene Vorgehensweisen, die mehr oder weniger flexibel sind. Bei der Serialisierung wird der gesamte Graph des Objektes, also die transitive Hülle, in einen Byte-Stream abgelegt und so in Form einer Datei im Dateisystem gespeichert. Serialisierung kann schnell angewandt werden, das Objekt muss nur die Schnittstelle Serializable implementieren, allerdings ist es nur für kleine Anwendungen eine brauchbare Alternative. Bei großen Systemen machen sich die fehlende Transaktionsunterstützung, die fehlende Abfragesprache sowie die fehlende Möglichkeit zum inkrementellen Laden eines Objektes negativ bemerkbar. Auch eine Transparenz ist hier nicht gegeben, da man aufwendig von Hand die Objekte serialisieren/deserialisieren muss. Damit ist diese Methode für unsere Zwecke unbrauchbar.

Eine weitere Möglichkeit besteht im manuellen objekt-relationalen Mapping, dabei werden Klassen und Beziehungen als Relationen abgebildet und die Objekte per JDBC in einem relationalem DBMS gespeichert. Hier resultieren die Probleme daraus, dass das objektorientierte Modell deutlich komplexer als das relationale Datenbankmodell ist, und damit nur eine verlustbehaftete Abbildung möglich ist. Ferner ist ein Problem, dass zusätzliche gewünschte Eigenschaften zur Performanzsteigerung und Effizienz selbst programmiert werden müssen. Die enorme Menge an Quellcode, der hierfür zu entwickeln ist, ist zudem meist starr und muss bei Änderungen des Objektmodelles häufig geändert werden. Letztendlich ist die eigene Entwicklung längst nicht so ausgereift wie vorhandene Persistence-Frameworks.

Die Verwendung eines Persistence Frameworks ist für diesen Kontext die beste Alternative, siehe dazu 4.8. Das Einlesen und Interpretieren der Metainformationen, um zur Laufzeit die benötigten Datenbankzugriffe zu erstellen, bedeuten zwar im direkten Vergleich zu JDBC aus Sicht der Performanz einen geringen Overhead, dieser wird aber durch Funktionalitäten zur Leistungssteigerung mehr als ausgeglichen. Die Implementierung ist ohne Probleme erweiterbar und kann in unterschiedlichen Umgebungen wieder verwendet werden. Die Kom-

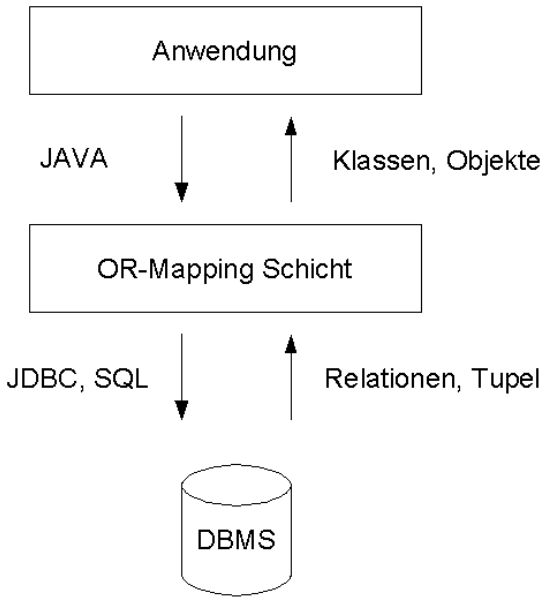

Abbildung 4.8: Hibernate Mapping

ponente Persistenz sollte folgende Dienste, wie in Tabelle 4.8 beschrieben, bereitstellen:

4.3 Sequenzdiagramme

Das Zusammenspiel der einzelnen Komponenten soll nun an einigen Diagrammen verdeutlicht werden. Bei der Anforderung eines Objektes sind eine Vielzahl an Komponenten beteiligt, siehe Abbildung 4.9. Die Anwendung kommuniziert mit der zentralen Anwendung und gibt dort die Objektanforderung weiter. Alle Anfragen müssen zuerst bei der Autorisierungskomponente geprüft werden. Wenn der Zugriff nicht erlaubt ist, dann wird eine Fehlermeldung zurückgegeben. Nach erfolgreicher Prüfung wird an die Persistenz die Anfrage weitergegeben. Bevor ein direkter Objektzugriff stattfindet, prüft der Transaktionsmanager ob keine andere Anwendung das Objekt im Zugriff hat. Sollte dies der Fall sein, dann wird die Anwendung benachrichtigt. Jetzt wird die Referenz auf das Objekt an die zentrale Anwendung geschickt und die anfragende Anwendung kann die Referenz auf dieses Objekt beziehen. Die Speicherung eines Objektes läuft in verschieden Phasen ab, siehe dazu Abbildung 4.10. Die Anwendung sendet die Anfrage an die zentrale Anwendung. Dort wird vorher der Persistenzmanager aktiv wird geprüft, die die Berechtigung zum Speichern des Objektes besteht. Ist die Prüfung nicht erfolgreich, dann wird die Aktion abgebrochen und eine Nachricht zurückgesendet. Wenn die Prüfung erfolgreich war, dann prüft die Persistenz, ob das Objekt

DIENST	BESCHREIBUNG	PARAMETER
getObject()	Holt ein Objekt mit Hilfe der Objekt-Id	Objekt-id
getObject()	Holt ein Objekt aufgrund von Metadaten	Metadaten
saveObject()	Speichert ein Objekt in der Datenbank	Objekt-id
saveObject()	Speichert ein Objekt in der Datenbank	Metadaten
importObject()	Importiert ein Objekt in den Raum	Metadaten
exportObject()	Exportiert ein Objekt zur Nutzung ausserhalb des Raumes	Objekt-id
getObjectIDs()	Holt nur die Objektids zu einer gegebenen Anzahl von Metadaten	Metadaten

Tabelle 4.8: Dienste der Persistenz

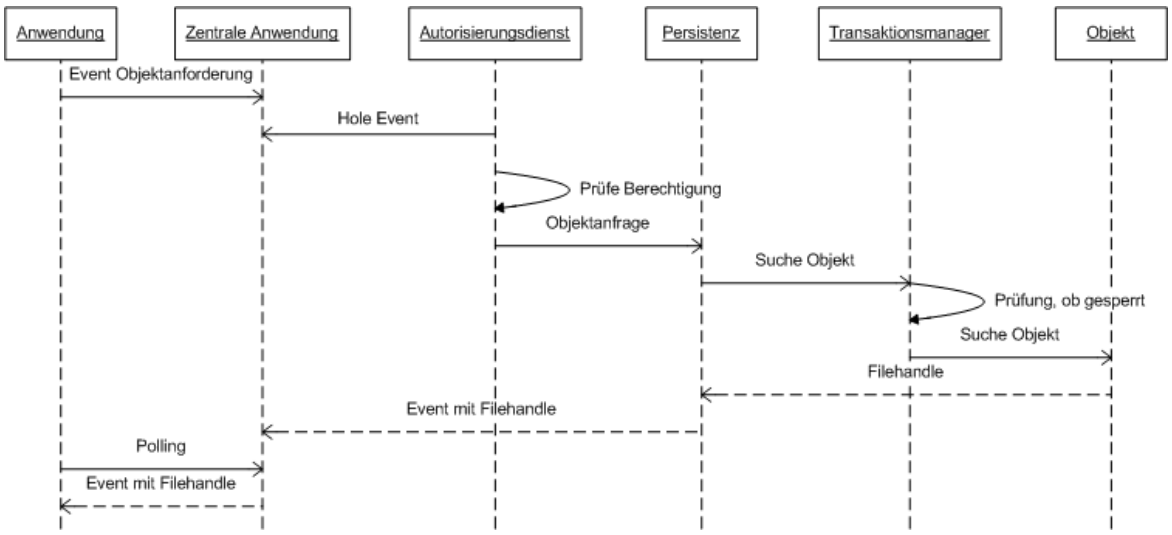

Abbildung 4.9: Sequenzdiagramm Objekt lesen

schon existiert und ggf. gesperrt ist. Wenn die Speicherung erfolgreich war, dann wird eine Nachricht an die zentrale Anwendung zurückgegeben.

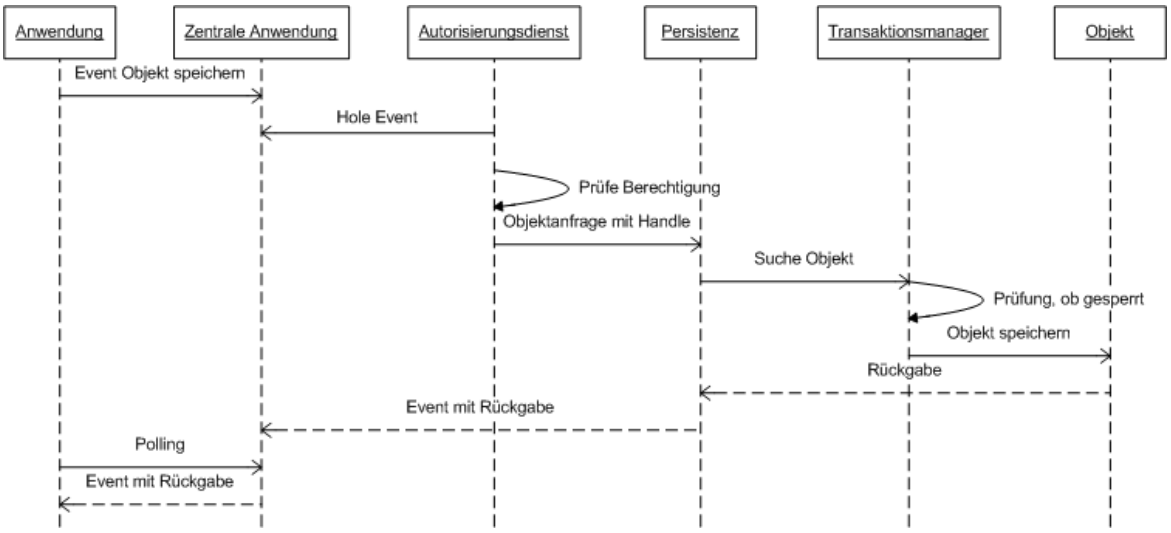

Abbildung 4.10: Sequenzdiagramm Objekt schreiben

4.4 Objektmodell

Bevor man zu einer genauen Spezifikation der Rechte und Benutzer kommt, möchte ich erst einmal das zugrunde liegende Objektmetamodell spezifizieren. Die Idee ist hierbei, die Daten im interaktiven Konferenzraum als ein Graph zu repräsentieren. Ein Graph besteht aus Knoten und Knoten, die Knoten repräsentieren die Daten und die Kanten sind die Beziehungen zwischen den Daten, siehe dazu Abbildung 4.11. Es kann mehrere Knotentypen geben z. B. Textdokument, Bild, UML-Diagramm, eine Applikation, ein Benutzer usw. Jeder Knoten enthält alle notwendigen Daten, die zur Beschreibung seines Typs und zu seinem Verhalten notwendig sind.

Durch Vergabe von Berechtigungen und durch die Vererbung wird der Graph in verschiedene Zugriffszonen aufgeteilt, dort gelten für unterschiedliche Benutzergruppen unterschiedliche Rechte. Sinnvoll ist eine Unterteilung in einen privaten und gemeinsamen Arbeitsbereich. Im privaten Arbeitsbereich hat nur der jeweilige Besitzer Zugriff im gemeinsamen Arbeitsbereich können mehrere Benutzer Zugriffsmöglichkeiten haben, siehe dazu Abbildung 4.12. Jeder Benutzer hat einen persönlichen Arbeitsbereich, in dem er für sich wichtige Daten ablegen kann. Per default hat dort kein anderer Benutzer Zugriff. Dort bekommt er die Rolle des

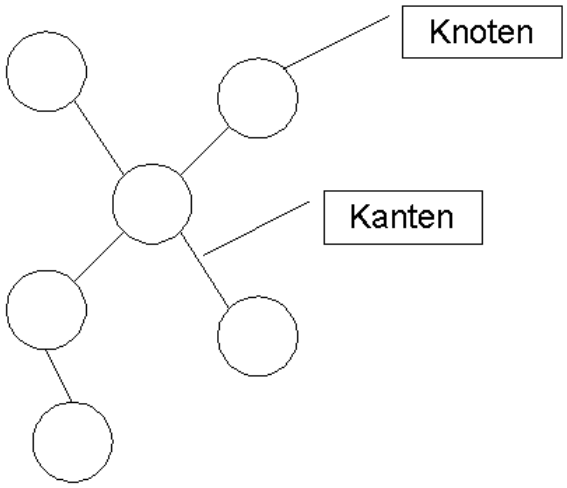

Abbildung 4.11: Objektmodell

Managers zugewiesen. Weiterhin gibt es die Möglichkeit einen gemeinsamen Arbeitsbereich anzulegen, um mit anderen Benutzern gemeinsam Dokumente zu bearbeiten.

Weiteres dazu in den Diplomarbeiten von Carola Neumann [Neumann (2006)] und Petra Rohwer-Ehlers [Ehlers (2006)].

4.5 Persistenz

4.5.1 Versionskontrolle

Bei einem Dokument unter Versionskontrolle wird dessen Inhalt bei Änderungen nicht einfach überschrieben, vielmehr wird eine zusätzliche Version erzeugt, die nun die aktuelle Fassung darstellt. Die früheren Versionen sollen erhalten, ihre Abfolge wird durch Versionsnummer (z.B. 0.1, 0.2, 0.3 etc.) gekennzeichnet. Ohne automatische Versionskontrolle müssten die Teilnehmer viel administrative Routinearbeit zusätzlich erledigen, um die verschiedenen Fassungen der Dokumente in ihrer zeitlichen und inhaltlichen Abfolge auseinander zuhalten.

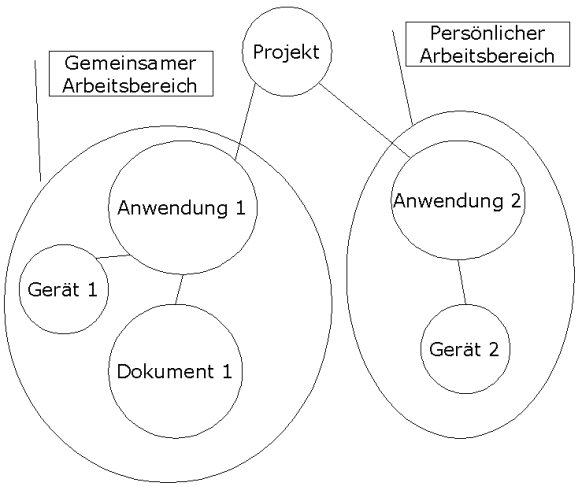

Abbildung 4.12: Arbeitsbereiche

4.5.2 Online/Offline Szenario

Da die Entscheidung des Persistenzdienstes auf eine Datenbank gefallen ist, bietet es sich an bei der Online/Offline Arbeit die Clients mit mobilen Datenbanken auszurüsten. Die Entwicklung in diesem Bereich ist sehr weit fortgeschritten, um beispielsweise PDA´s auch schon mit funktionsfähigen Datenbanken auszustatten, siehe dazu Abbildung 4.13. Über die Replikationsdefinition lassen sich so die Dokumente festlegen die der Benutzer mit der mobilen Datenbank nutzen kann. Diese können je nach Berechtigung geändert oder nur gelesen werden. Allerdings muss das Problem der Rückübertragung und Integration der Daten genauer betrachtet werden, da es zu Situationen kommen kann in denen dieses nicht automatisch funktioniert. Dann sollte der Benutzer beim Betreten des Raumes entscheiden, welche Version die aktuellste ist, weitere dazu in der Diplomarbeit von Aiko Böhm [Böhm (2005)].

4.5.3 Realisierung

In diesem Unterkapitel wird die Umsetzung der, im Architektur Kapitel ausgearbeiteten, Infrastruktur beschrieben. Hierfür werden soweit wie möglich Standardkomponenten eingesetzt. Darauf aufbauend wird eine einfache Beispielanwendung erstellt wurde. Der Fokus liegt hier auf der Komponente für die Persistenz. Der Prototyp zeigt zwei Dienste der Persistenzschicht. Zum einen soll ein Objekt in der Datenbank gespeichert werden und zum anderen wieder ausgelesen werden. Dabei wird mit Hibernate als O-R-Mapper gearbeitet und mit einer Datenbank dahinter.

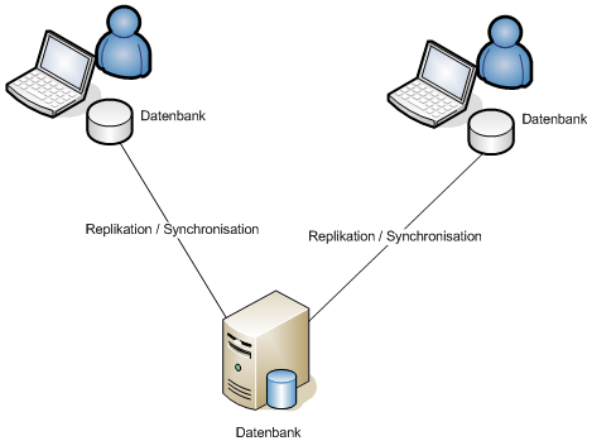

Abbildung 4.13: Online/Offline Szenario

Laboraufbau

Das System wurde in einer beispielhaften Laborumgebung realisiert, die im Folgenden beschrieben wird. Sie weist einige Unterschiede zur späteren Vollausstattung aus. Als Clientrechner habe ich einen handelsüblichen Computer mit Windows XP SP2, Java 1.5 und Hibernate installiert. Als Server kommt wiederum ein handelsüblicher Computer mit dem Betriebssystem Debian Linux Sarge und der Datenbank PostgreSQL zum Einsatz, eine Übersicht findet sich in der Abbildung 4.14.

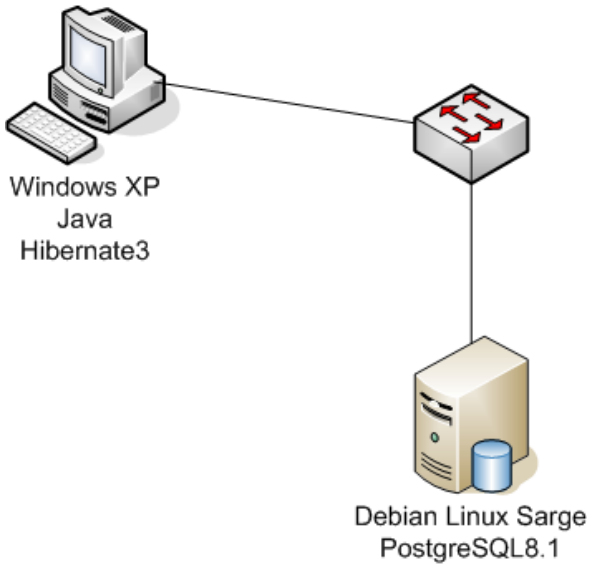

Abbildung 4.14: Laboraufbau des Prototypen

Programmiersprache

Als Sprache der selbst entwickelten Komponenten wird Java gewählt. Nach [Saake und Sattler (2003)] gewährleistet diese Sprache einerseits die Plattformunabhängigkeit des kompilierten Codes, die eine Hardwareunabhängigkeit der Clientkomponenten erleichtert, weiteres zu Java ist unter [Sun (2006)] zu finden. Für die Realisierung des Prototyps habe ich mich für Java als Programmiersprache

Auswahl der Plattform

Damit die Dienste installiert werden können, wird ein Betriebssystem gebraucht. Im Open-Source Bereich fällt die Entscheidung auf Linux[3] Es sind diverse Linuxvarianten am Markt vorhanden. Die Ausrichtung auf die jeweilige Zielgruppe ist sehr unterschiedlich. Ich habe mich im Rahmen der Diplomarbeit für Debian Linux entschieden. Debian [Debian (2006)] ist eine GNU/Linux-Distribution, die ausschließlich aus freier Software besteht. Enthalten ist das Betriebssystem und eine große Auswahl an Anwendungsprogrammen und Utilities, zusammen mit einem passenden Kern. Es ist für diverse Hardwarearchitekturen erhältlich. Debian hat ein effizientes Paket-Managementsystem, damit ist es vergleichsweise einfach, alte Versionen von Debian durch aktuelle Versionen zu ersetzen oder neue Softwarepakete zu installieren. Debian ist momentan in der Version 3.1 *Sarge* die umfangreichste Linux-Distribution. Zurzeit werden von Debian drei Varianten angeboten:

- **stable**, ist die zuletzt veröffentlichte offizielle Version. Dort sind alle Pakete gut getestet und aufeinander abgestimmt und Sicherheitsupdates sind schnell verfügbar. Debian veröffentlicht im Moment nur etwa alle zwei Jahre eine neue Version, so dass Stable für zwei Jahre konstant bleibt. Stable gilt als geeigneter Kandidat für Server-Systeme, die lange Zeit sehr stabil laufen müssen und nicht auf die allerneuesten Programm-Versionen angewiesen sind.
- **testing**, ist der Kandidat für die nächste Veröffentlichung. Testing wird häufig als Betriebssystem für Arbeitsplatzrechner empfohlen, weil man dort etwas weniger auf Stabilität und etwas mehr auf neue Software und Unterstützung aktueller Hardware angewiesen ist.
- **unstable**, landen neue Versionen von Paketen und auch Programme, die neu in Debian aufgenommen wurden. Dort werden sie von denen, die Unstable verwenden, auf Fehler geprüft.

[3] Linux ist ein freies und plattformunabhängiges Mehrbenutzer-Betriebssystem für Computer, das Unix ähnlich ist.

Ich habe für die Diplomarbeit die Version 3.1 *Sarge* (stable) ausgewählt, da sie im Moment die stabilsten Pakete beinhaltet.Die Modularität von Debian und das sehr gute Paketmanagement bieten mir die Möglichkeit die Installation komplett an meine Bedürfnisse anpassen zu können. Die Anzahl der installierten Dienste lassen sich minimal halten und so auch die Anforderungen an die Sicherheit sehr gut umsetzen.

Auswahl des O-R-Mappers

Bei der Wahl des in Kapitel 4 empfohlenen O-R-Mappers ist Hibernate die überzeugendste Alternative. Die Vorteile von Hibernate [Hibernate (2006)] sind u. a.: Datenbankabstraktion, Unterstützung der gängigsten Datenbanken, Objekt-Caching, Connection-Pooling und das Nachladen von referenzierten Objekten beim ersten Zugriff. In einer Java-Umgebung können die einzelnen Objekte ohne Einschränkungen benutzt werden. Das Laden und Speichern von Objekten wird größtenteils transparent von Hibernate übernommen. In einer Java-Anwendung reicht es oft, das Objekt über die aktuelle Hibernate-Session der Methode *save()* oder *find()* zu übergeben. Das Auflösen und Speichern bzw. das Laden und Instanzieren des Objekts wird von Hibernate übernommen.

Die Anwendungen greifen auf Hauptspeicherobjekte zu, die ORM-Schicht erkennt die Änderungen und überträgt diese in die DB, siehe dazu Abbildung 4.15. Allerdings muss beim Zugriff auf von mehreren verteilten Anwendungen eine Instanz für einen geregelten Zugriff sorgen, sonst kann es zu Problemen kommen. In unserem Szenario gibt es den Transaktionsmanager der diese Aufgabe übernimmt.

Hibernate ist ein frei erhältlicher O/R Mapper für Java, mit dem es möglich ist, die objektorientierte mit der relationalen Welt zu verbinden. Mit Hibernate lassen sich Java-Objekte auf einfache Art über XML-Konfigurationsdateien mit relationalen Datenbanktabellen koppeln. Neben einer ODMG28 3.0 konformen Schnittstelle können Datenbankabfragen über SQL oder HQL-Statements ausgeführt werden. Man kann auch über XDoclet [XDocLet (2006)] aus Java-Code direkt die Mapping-Dateien generieren lassen.

Hibernate ist komplett Open-Source und es existiert eine große Community, die eine umfassende Dokumentation bereitstellt. Dank der umfangreichen Werkzeugunterstützung kann man durch dieses Framework die Persistierung schnell und einfach realisieren. Das Open-Source Projekt JBoss hat Hibernate standardmäßig als Persistenzmechanismus integriert. Dies spricht für die hohe Qualität von Hibernate. Daher wird in dieser Diplomarbeit Hibernate für die Implementierung der Persistenzschicht herangezogen.

Design-Patterns In der Persistenzschicht werden einige Design-Patterns verwendet, die ich hier näher erläutern möchte, speziell Hibernate bietet diese Dienste an.

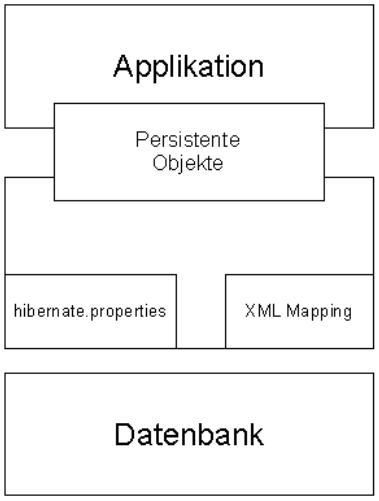

Abbildung 4.15: Hibernate Architektur

Lazy Loading Komplexe Objekte bestehen oft aus einer Vielzahl von zusammengesetzten oder referenzierten Objekten. Stellt man sich hier z. B. ein Dokument vor, so kann dieses sehr komplex und tief geschachtelt sein. Wird beim Laden eines Objektes der gesamte Graph an referenzierten Objekten mit geladen, so kann dies die Geschwindigkeit des Systems sehr beeinträchtigen. Oftmals werden nur einfache Attribute auf oberster Ebene benötigt. Will man z. B. nur die Namen der Objekte in einer Liste anzeigen, so ist es nicht nötig, alle untergeordneten Objekte, z. B. die Binärdaten mit in den Hauptspeicher zu laden. Genau hier kommt die Methodik des „faulen Ladens“ zum Einsatz. Hier werden die konkreten Objekte zwischenzeitlich durch Stellvertreter-Objekte ersetzt, bis die konkreten Objekte tatsächlich benötigt werden. Dies kommt uns vor allen Dingen in unserem Konferenzraum zugute, da die großen Binärobjekte erst beim Zugriff geladen werden. Dies spart Ressourcen und sorgt für eine gute Performanz.

Data Access Object Eine wichtige Anforderung an eine Persistenzschicht ist die Transparenz und Austauschbarkeit. Um diesen Anforderungen gerecht zu werden, sollte man die Geschäftsobjekte und Businesslogik weitestgehend von der Persistenzschicht zu entkoppeln. Dies wird durch Einführung eines Data Access Objektes erreicht, hierbei handelt es sich um eine zusätzliche Schicht, die als Schicht zwischen den Objekten und der Datenbank realisiert wird. So lassen sich die Implementierungsdetails für den Datenzugriff zu verbergen. Es werden in der Regel einfache Schnittstellen für das Laden und Manipulieren von Objekten angeboten. Das Objekt weiß gar nicht, ob es persistiert wird, da es selber keinen Code für den Datenbankzugriff bereitstellt. Damit kann die Art der Persistierung jederzeit ausgetauscht werden, ohne dass Änderungen an der Anwendungslogik vorgenommen werden müssen.

Generierte Objektidentifikation Die Identität der Objekte spielt im interaktiven Konferenzraum eine wichtige Rolle, da jedes Objekt eindeutig identifiziert werden muss. Solange sich das Objekt im Hauptspeicher befindet, ist es durch seine Speicheradresse eindeutig zu identifizieren. Wird dieses Objekt persistiert, dann muss die Persistenzschicht die systemweite Eindeutigkeit garantieren. Um das zu gewährleisten, bietet sich an einen generierten, datenbankweit eindeutigen Schlüssel zu verwenden, der keinerlei Geschäftslogik in sich birgt. Damit wird sichergestellt, dass es bei der Eindeutigkeit des Schlüssels bleibt und man die Datenbank austauschen kann. Man kann folgende Wege der Generierung des Schlüssels diskutieren:

- **High-Low**, hierbei wird eine eigene Tabelle zur Verfügung gestellt, welche eine Sequenznummer beinhaltet. Diese dient als Basis für neue Schlüssel, die algorithmisch errechnet werden.
- **Identity**, dabei werden die Schlüssel von der jeweiligen Datenbank berechnet. Datenbanken wie Hypersonic SQL, MS SQL Server, MySQL und Sybase ASE/ASA unterstützen diese Methodik.
- **Sequence**, der Schlüssel wird auf Basis des propreitären Sequence-Konzepts von Interbase, Oracle, PostgreSQL und SAP DB berechnet.
- **UUID**, Dieser Algorithmus liefert eine weltweit eindeutige ID auf Basis der IP-Adresse, einem Zähler und der aktuellen Zeit in Millisekunden.

Für den interaktiven Konferenzraum wähle ich die High-Low Methode aus, da sie zum einen sehr effizient ist, und zum Anderen die Unabhängigkeit vom Persistenzmechanismus fördert.

Singleton Das Singleton-Pattern soll absichern, dass von einer Klasse maximal ein Exemplar existiert. Dazu wird durch die Klasse, die als Singleton implementiert ist, ein globaler Zugriffspunkt auf die einzige Instanz bereitgestellt. Ein Client kann nur über diesen Zugriffspunkt an eine Instanz der Singleton-Klasse gelangen. Die Schnittstelle zur Persistenzschicht wird mit Hilfe des Singleton-Patterns implementiert, damit die Clients nur über diese Schnittstelle zugreifen können, und damit keine Inkonsistenzen auftreten können, wenn mehrere Instanzen auf die Daten zugreifen.

Auswahl der Datenbank

Als Datenbank wurde PostgreSQL ausgewählt, sie entstand Ende der achtziger Jahre an der Universität von Berkeley. PostgreSQL [PostgreSQL (2006)] zeichnet sich durch eine umfassende Unterstützung des SQL-Standards sowie generell durch eine sehr breite Unter-

stützung gängiger Unternehmensstandards aus. Insbesondere die Techniken *Views*, *Stored Procedures* und *Trigger* sind Bestandteil von PostgreSQL. Weitere technische Fähigkeiten von PostgreSQL sind:

- Referenzielle Integrität
- native Schnittstellen für ODBC, JDBC, C, C++, PHP, Perl, TCL, ECPG, Python und Ruby
- Sequences
- Vererbung
- Outer Joins
- Subselects
- Benutzerdefinierte Datentypen
- Unterstützung des ACID-Prinzips (Atomicity, Consistency, Isolation, Durability) zur Transaktionsverarbeitung.

Während PostgreSQL auf fast allen Unix-Varianten verfügbar ist, läuft PostgreSQL ab Version 8.0 auch auf Microsoft NT-basierten Betriebssystemen wie Windows 2000, XP und Server 2003. PostgreSQL ist Open-Source und hält sich bei der Implementierung an die gängigen Standards aus der Datenbankwelt. Ein wichtiges Auswahlkriterium war die Unterstützung durch Hibernate, und eine sehr hohe Praxistauglichkeit des Produktes.

Implementierung des Prototypen

Der Prototyp wurde durch Realisierung der einen Komponente soweit realisiert, so dass man die Tragfähigkeit des Konzeptes zeigen kann. Es wurde das Speichern eines beliebigen Objektes in der Datenbank implementiert. Als Umgebung auf der Clientseite wurde Java von Sun Microsystems [Sun (2006)] verwendet, dass mit Hilfe von Hibernate in jeder beliebigen Datenbank Objekte speichern kann.

Klassen Das Klassendesign wurde auf Basis des in Kapitel 4.2.1 beschriebenen UML-Modells ausgeführt. Es wurden drei Beispielklassen erstellt, um einen kleinen Überblick über die geplante Funktionalität zu geben, siehe dazu Abbildung 4.16.

Die Klasse *Benutzer* verfügt über rudimentäre Attribute. Dazu gehören Id, FirstName, LastName und *roles*. Das Attribut *roles* ist vom Typ Rolle und beinhaltet alle dem Benutzer zugeordnete Rollen. In der Javaklasse ist dies ein Set von Rollen. In der Klasse *Rolle* werden auf der einen Seite die Mitglieder einer Rolle verwaltet, hier geschehen durch das Attribut

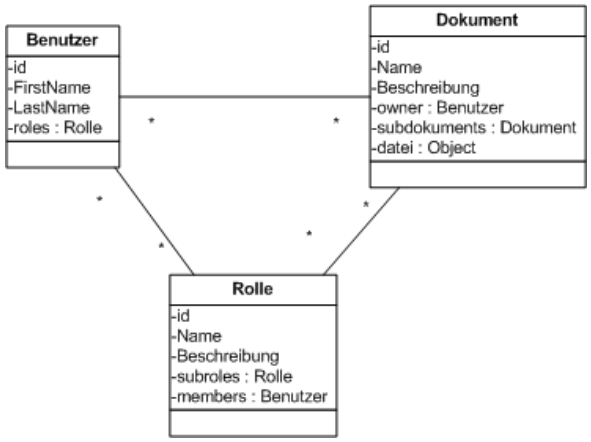

Abbildung 4.16: Klassenmodell des Prototypen

members vom Typ *Benutzer*, und auf der anderen Seite wird die Rollenhierarchie durch das Attribut *subroles* angedeutet. Damit wird die Verknüpfung der Rollen untereinander geschaffen. Die Klasse Dokument besteht aus diversen Attributen und beinhaltet Objekte die mit anderen Klassen verknüpft sind. Als Anforderung wurde formuliert, dass die Objekte untereinander verknüpft sein sollen, dies ist durch das Attribut *subdokuments* gegeben. Dieses Attribut vom Typ *Dokument* enthält alle verknüpften Dokumente. Das Attribut *owner* verknüpft nun wiederum die Klasse Dokument mit der Klasse Benutzer. Dieses Attribut enthält alle Besitzer des Dokumentes. Im Prototyp wird der binäre Inhalt der Datei persistiert, dazu ist noch ein weiteres Feld nötig, das die Binärdaten aufnimmt. Dieses Attribut ist in dieser Klasse *datei*, es ist vom Typ *blob*.

Persistenz Das eben beschriebene Klassenmodell sorgt noch nicht für die Persistierung der Objekte. Dafür ist Hibernate zuständig, durch die Steuerung über xml-Dateien lässt sich sehr flexibel jedes Objekt persistieren. Für den Prototyp habe ich alle drei Klassen persistiert. Für jede Klasse habe ich eine *<Klassenname>.hbm.xml* Datei erstellt, in der die Attribute angegeben werden, die persistiert werden sollen. Für die Klasse Dokument ist ein Auszug im Folgenden zu sehen:

Listing 4.1: Hibernate Dokument.hbm.xml

```
1 <?xml version="1.0"?>
2 <!DOCTYPE hibernate-mapping PUBLIC
3 "-//Hibernate/Hibernate␣Mapping␣DTD␣3.0//EN"
4 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5
6 <hibernate-mapping>
```

```
7   <class name="de.Dokument" table="DOKUMENT">
8    <id name="id" type="int" column="ID" >
9    <generator class="increment"/>
10   </id>
11   <property name="name"/>
12   <property name="beschreibung"/>
13   <set name="allBesitzer" table="own_role">
14     <key column="dok_uid"/>
15     <many-to-many
16      column="role_uid"
17      class="de.Rolle"/>
18      </set>
19    <property name="datei"/>
20  </class>
21 </hibernate-mapping>
```

Dort wird für jedes Attribut in der Klasse ein Eintrag gemacht, entweder übernimmt Hibernate den Typ aus dem Java-Quellcode oder man gibt in dieser Konfigurationsdatei den Typ des Attributes explizit an. Um die Erstellung der *<Klassenname>.hbm.xml* zu automatisieren, bietet sich der Einsatz von XDocLet an, siehe dazu XDocLet (2006). XDocLet generiert aus den Javaklassen anhand von Tags die notwendigen Mappingdateien und vereinfacht so die Nutzung von Hibernate.

Die Beispielanwendung schreibt eine Textdatei in die Datenbank und liest diese wieder aus. Dazu wird die Datei in einen Stream verwandelt und mit einer Hibernate-Methode in ein Blob umgewandelt, siehe dazu das folgende Listing:

Listing 4.2: Beispielanwendung

```
1  SessionFactory sessionFactory = new Configuration().
2  configure().buildSessionFactory();
3  session =sessionFactory.openSession();
4  dateiname = "Z:/tmp/test.txt";
5  File f = new File(dateiname);
6  byte[] buffer = new byte[ (int) f.length() ];
7  InputStream in = new FileInputStream( f );
8  Dokument dok1 = new Dokument();
9  dok1.setName("test.doc");
10 dok1.setBeschreibung("Worddokument");
11 dok1.setDatei(Hibernate.createBlob(in));
12 session.save(dok1);
```

Das Dokument-Objekt kann durch einfache Getter- und Settermethoden zusammengesetzt werden und durch einen einfache save()-Aufruf persistent gemacht werden. Hibernate kümmert sich um die weitere Speicherung des Objektes. Hibernate legt bei Bedarf im ersten Schritt das Datenbankschema an, bei einer Änderung der Datenstruktur wird das Schema neu generiert.

4.6 Zugriffsrechte

Gemäß dem in Kapitel 4.4 beschriebenem Objektmodell werden die Berechtigungen für die jeweiligen Knoten über Rollen festgelegt. Entlang der Kanten können über Vererbung die jeweiligen Rechte weitergereicht werden. Durch die Nutzung von Rollen und Gruppen lässt sich damit das Sicherheitsmodell einfach und effizient umsetzen.

4.6.1 Benutzer und Gruppen

Benutzergruppen können rekursiv definiert werden, das heißt ein Benutzer ist eine Benutzergruppe und mehrere Gruppen sind eine Benutzergruppe. Benutzergruppen können demzufolge Hierarchien bilden. Mitglieder einer Benutzergruppe können also direkt oder indirekt in der Benutzergruppe vorhanden sein. In der Abbildung 4.17 sind die Mitglieder von Team2 die Benutzer *User1*, *User2* und *User3*. Die Untergruppe von Team2 ist *SubGruppe*, *User2* und *User3*. Folgende Operationen sollen auf Gruppen möglich sein.

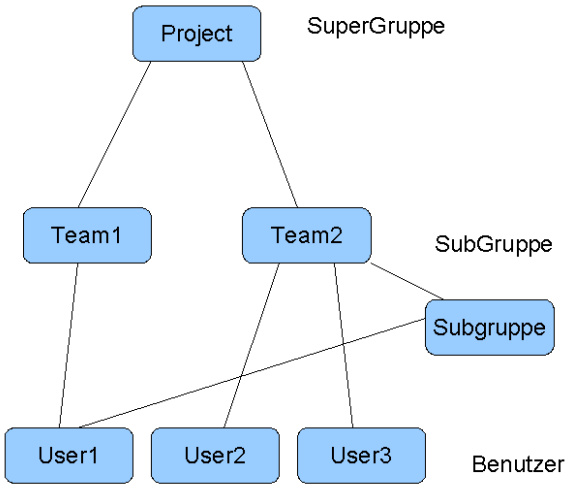

Abbildung 4.17: Gruppenhierarchie

- **NeueGruppe**, fügt eine neue Gruppe in den Baum ein.
- **NeueUnterGruppe**, fügt eine neue Untergruppe hinzu.
- **UntergruppeLöschen**, löscht eine Untergruppe.
- **Gruppe löschen**, löscht eine Gruppe aus dem Baum, die zugehörigen Untergruppen werden damit mitgelöscht. Damit verliert die Obergruppe Mitglieder.
- **GruppeAuflösen**, die Gruppe wird gelöscht und die Untergruppen werden an die Obergruppen der zu löschenden Gruppe angehängt.
- **GruppeUmbenennen**, Gruppe wird umbenannt.
- **GruppeSplitten**
- **GruppeZusammenfassen**
- **GruppeEinfügen**

Um diese Operationen zu veranschaulichen, habe ich die Operationen *Gruppe löschen* und *Gruppe auflösen* visuell dargestellt, siehe dazu Abbildung 4.18.

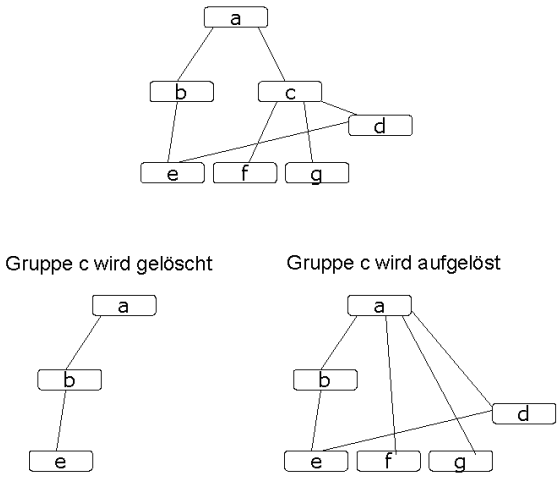

Abbildung 4.18: Löschen von Gruppen

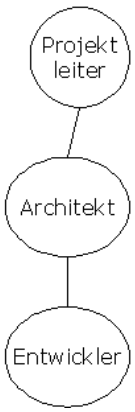

Abbildung 4.19: Hierarchien im RBAC-Modell

4.6.2 Hierarchien

Wie schon im vorgehenden Kapitel beschrieben, kann im RBAC-Modell eine Hierarchie von Rollen aufgebaut werden. Im Szenario der Softwareentwicklung kann es z. B. zu einer Hierarchie, wie in Abbildung 4.19 gezeigt, kommen.

Die Rolle *Projektleiter* bekommt automatisch die Berechtigungen der Rollen *Entwickler* und *Architekt* zugewiesen. Wenn einem Benutzer nun die Rolle *Architekt* zugewiesen bekommt, dann kann er Operationen der Rollen *Architekt* und *Entwickler* ausführen. Man kann dabei erkennen, dass jede Rolle die Berechtigungen der Nachfolger in sich vereint.

4.6.3 Berechtigungsklassen

Berechtigungsklassen vereinfachen die Zuweisung von Rechten für Benutzer oder Rollen. Diese stellen eine vereinfachte Sicht auf die einzelnen Berechtigungen dar und können somit effektiv und fehlerfrei zugewiesen werden.

Die Berechtigungsklasse **Information**, siehe Tabelle 4.9, listet Berechtigungen auf, die rein zum Lesen von Informationen gedacht sind. Die Daten werden nicht manipuliert, sondern nur rein zu Informationszwecken genutzt.

Die Operation **Konvertieren** überführt das Objekt in ein anderes Format, z. B. ein Textdokument in ein PDF-Dokument. Um dem Benutzer das nachträgliche Manipulieren der Daten zu erschweren, würde sich das PDF-Format anbieten, da es umfangreiche Möglichkeiten zur Reglementierung von Inhalten bietet, siehe dazu Abbildung 4.20. Weiteres zu Sicherheitseinstellungen in PDF-Dokumenten unter [Adobe Systems Incorporated (2006)]. Die Operation **Kommentieren** verändert nicht das Objekt, sondern fügt Notizen oder Anmerkungen

OPERATION	BESCHREIBUNG
öffnen	Öffnet ein Objekt
exportieren	Exportiert ein Objekt zur Nutzung ausserhalb des Raumes
drucken	Druckt die Objekt aus
kopieren	Legt eine Kopie eines Objektes an
konvertieren	Konvertiert ein Objekt in ein anderes Format
kommentieren	Erstellt Kommentare für ein Objekt

Tabelle 4.9: Berechtigungsklasse Information

hinzu. Das soll Benutzern ohne Schreibrechte für das Objekt die Möglichkeit eröffnen, Anmerkungen zu machen.

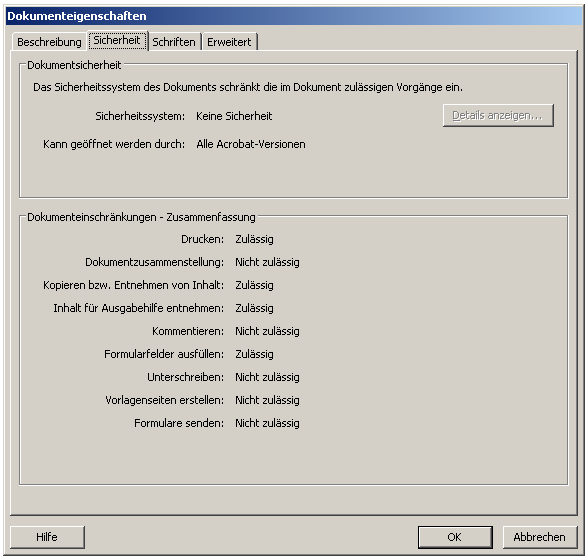

Abbildung 4.20: Sicherheitseinstellungen in PDF-Dokumenten

In der Berechtigungsklasse **Erweiterte Information**, siehe dazu Tabelle 4.10 ist es möglich weitere Informationen über ein Objekt einzusehen. Dazu gehören Berechtigungen der einzelnen Benutzer oder das Anzeigen von weiteren Versionen dieses Dokumentes. Die Berechtigungsklasse **Bearbeitung**, siehe dazu Tabelle 4.11, bietet diverse Möglichkeiten zur Manipulation der Objekte. Es sind alle Berechtigungen vereint, die mit dem Neuanlegen und Ändern von Objekten in Beziehung stehen. Allerdings ist hier bewusst das Löschen von Objekten ausgenommen.

Über das Recht **Metadaten ändern** ist zu regeln, wer die Metadaten des Objektes ändern darf. Der Benutzer kann neue Metadaten erstellen oder gemäß dem **Dublin Core Metadata**

OPERATION	BESCHREIBUNG
Mehr Information	Zeigt Metainformationen zu einem Objekt an
Versionen anzeigen	Zeigt weitere Versionen im Rahmen der Versionshistorie an

Tabelle 4.10: Berechtigungsklasse Erweiterte Information

OPERATION	BESCHREIBUNG
Neues Objekt	Legt ein neues Objekt an
Bearbeiten	Bearbeitet ein Objekt
Umbenennen	Benennt ein Objekt um
Metadaten ändern	Manipuliert die Metadaten des Objektes
Neue Version	Legt eine neue Version des Objektes an
Versionskontrolle einschalten/ausschalten	Schaltet die Versionskontrolle für ein Objekt an
Sperren/Entsperren	Sperrt/Entsperrt das Objekt
importieren	Importiert Objekt in das System
Neue Abhängigkeit	Legt eine Verlinkung auf ein anderes Objekt an

Tabelle 4.11: Berechtigungsklasse Bearbeitung

Set vorgeschlagene Metadaten mit Informationen füllen. Bei eingeschalteter Versionskontrolle werden bei jeder Änderung automatisch neue Versionen generiert. Der Benutzer sollte dazu aufgefordert werden, die Versionsnummer und den Status der Version mit anzugeben. Man kann die nächste freie Versionsnummer vorschlagen und dann kann der Benutzer den Status angeben, das wäre z.b. „vorläufig“ oder „in Bearbeitung“.

Die Berechtigungsklasse **Erweiterte Bearbeitung**, siehe dazu Tabelle 4.12 fasst alle Operationen zusammen, die mit dem Entfernen oder Ausschneiden eines Objektes aus dem jeweiligen Kontext zu tun haben. Dabei werden die Objekte nicht gelöscht, sondern nur der Verweis auf den aktuellen Kontext wir aufgehoben.

OPERATION	BESCHREIBUNG
Ausschneiden	Schneidet ein Objekt aus und kopiert es in eine Zwischenablage
Entfernen	Entfernt das Objekt aus dem Kontext, im System bleibt es noch erhalten

Tabelle 4.12: Berechtigungsklasse Erweiterte Bearbeitung

Die Berechtigungsklasse **Endgültig Löschen**, siehe dazu Tabelle 4.13 enthält die Berech-

tigung ein Objekt endgültig aus dem System zu entfernen. Dies geht allerdings nur, wenn keine Beziehungen zu anderen Objekten bestehen.

OPERATION	BESCHREIBUNG
Entfernen (endgültig)	Entfernt ein Objekt aus dem System

Tabelle 4.13: Berechtigungsklasse Endgültig löschen

Einer der wichtigsten Berechtigungsklassen ist die **Kollaboration**. Sie enthält Berechtigungen zum Einladen von Benutzern und zum Teilen von Dokumenten, siehe dazu Tabelle 4.14.

OPERATION	BESCHREIBUNG
Benutzer einladen	Lädt einen Benutzer in den Kontext ein
Auf öffentlichem Display anzeigen	Zeigt ein Objekt auf einem öffentlichen Display an
Privates Display veröffentlichen	Veröffentlicht das private Display

Tabelle 4.14: Berechtigungsklasse Kollaboration

Die Berechtigung **Benutzer einladen** soll anderen Benutzern ermöglichen Zugriff auf Objekte aus dem jeweiligen Kontext zu bekommen. Jeder eingeladene Benutzer bekommt für den Kontext, in dem er sich bewegt, eine Rolle zugewiesen.

Eine Erweiterung zur Kollaboration ist die **Erweiterte Kollaboration**. Diese Berechtigungsklasse stellt Operationen zum Administrieren von Rollen bereit, siehe dazu Tabelle 4.15.

OPERATION	BESCHREIBUNG
Neue Rolle	Legt eine neue Rolle an
Rolle ändern	Ändert die Berechtigung einer Rolle im Kontext
Rolle zuweisen	Weist Benutzern im Kontext eine Rolle zu
Rolle löschen	Löscht eine Rolle

Tabelle 4.15: Berechtigungsklasse Erweiterte Kollaboration

Die Berechtigungsklasse **Unterschreiben** dient dazu Versionsstände von Objekten verbindlich für alle festzulegen.

Das **Signieren des Objektes** dient zum gemeinsamen Festhalten eines Versionsstandes. Damit stimmen alle signierenden Mitglieder diesem Beschluss zu. Von diesem Objekt können weiteren Versionen angelegt werden oder Änderungen vorgenommen werden.

OPERATION	BESCHREIBUNG
Objekt signieren	Signiert das Objekt

Tabelle 4.16: Berechtigungsklasse Unterschreiben

OPERATION	BESCHREIBUNG
lokales Gerät steuern	Steuert das lokale Gerät
Öffentliches Display steuern	Steuert das öffentliche Display

Tabelle 4.17: Berechtigungsklasse Gerätesteuerung

Die Berechtigungsklasse Geräteadministration ist der Rolle des Administrators in erster Linie vorbehalten, siehe Tabelle 4.18. Dort können Geräte im Raum angemeldet werden und ggf. Anwendung installiert werden.

OPERATION	BESCHREIBUNG
Gerät im Raum anmelden	Gerät dem Raum gegenüber bekanntmachen
Gerät abmelden	Gerät aus dem Raum entfernen
Anwendung installieren	Anwendung installieren

Tabelle 4.18: Berechtigungsklasse Geräteadministration

Jeder Benutzer hat lokal auf seinem Gerät volle Administrationsrechte, außer es handelt sich um fest installierte Geräte im Raum.

4.6.4 Rollen

Es gibt in unserem interaktiven Konferenzraum drei unterschiedliche Arten von Rollen. Diese sind je nach Einsatzzweck fest definiert oder können während der Benutzung des Raumes dynamisch festgelegt werden.

Fachliche Rollen

Die fachlichen Rollen werden je nach Einsatzzweck des Raumes oder gemäß der Organisationsstruktur der jeweiligen Firma oder Gruppe festgelegt. In unserem Szenario der Softwareentwicklungsgruppe sind verschiedene Rollen denkbar. Ich werde mich hier auf die wichtigsten Rollen beschränken und einen kleinen Einblick in mögliche sinnvolle Rechtezuweisungen geben. Generell gilt die Bedingung, dass alles was nicht explizit erlaubt ist, verboten ist. In

der Standardauslieferung eines solchen Systems sollten keine fachlichen Rollen im System schon angelegt sein, sondern das bleibt der Gruppe selbst überlassen, die den Raum nutzt. Es muss nur eine einfach zu benutzende Schnittstelle zur Rollenadministration vorhanden sein, damit besonders die wechselseitigen Ausschlüsse bei der Vergabe von Rollen sinnvoll definiert werden können. Im Szenario Softwareentwicklung würde man die folgenden Rollen typischerweise anlegen:

- **Projektleiter**,
- **Kunde**,
- **Entwickler**,

Diese können in Ihren Rechten je nach organisatorischer Struktur des Unternehmens mehr oder weniger stark variieren.

Administrative Rollen

Die administrativen Rollen sind bereits vordefiniert und können auch verändert werden. Jede Rolle hat eine gewisse Anzahl an Standardberechtigungen. Dies hat den Sinn dem Benutzer den Einstieg in die Rechteverwaltung zu erleichtern und demzufolge die Akzeptanz des Systems zu erhöhen. Dementsprechend sind hier sprechende Rollenbezeichnungen gewählt worden, und ein von den Benutzern akzeptiertes Rechtesystem trägt zur Gesamtsicherheit des Systems bei.

- **Manager**, jeder Benutzer der einen neuen Knoten im System anlegt, wird automatisch zum Manager für dieses Objekt. Er hat damit den Vollzugriff für dieses Objekt. Der Rolle Manager werden folgende Berechtigungsklassen zugeordnet:
  - Information
  - Erweiterte Information
  - Bearbeitung
  - Erweiterte Bearbeitung
  - Kollaboration
  - Erweiterte Kollaboration
- **Teilnehmer**, dies ist die Standardrolle für den interaktiven Konferenzraum. Die zugeordneten Berechtigungsklassen lauten wie folgt:
  - Information
  - Erweiterte Information

  - Bearbeitung
  - Erweiterte Bearbeitung
  - Kollaboration
- **Eingeschränkter Teilnehmer**, diese Rolle darf nur lesend auf die Objekte zugreifen, das Manipulieren ist nicht gestattet.
  - Information
  - Erweiterte Information
- **Unterzeichner**, diese Rolle hat die Fähigkeit ein Objekt vor weiterer Veränderung zu schützen. Das Objekt wird damit verbindlich für alle, und kann nur von der Rolle Manager oder der Rolle Unterzeichner selbst wieder änderbar gemacht werden.
  - Unterschreiben
- **Gast**
  - Information
- **Administrator**, der Administrator hat standardmäßig Zugriff auf alle Objekte und Geräte im Raum.
  - Information
  - Erweiterte Information
  - Bearbeitung
  - Erweiterte Bearbeitung
  - Kollaboration
  - Erweiterte Kollaboration
  - Geräteadministration

Lebenszyklus-Rollen

- **Authentifiziert**, hat eine gültige Benutzername/Passwort Kombination geliefert.
- **Anonym**, ohne Anmeldung ist der Zugriff standardmäßig auf kein Objekt gestattet. Man kann darüber nachdenken, ob man Objekte von außen ohne vorherige Anmeldung zugreifbar macht, z. B. für den Zugriff über einen Webserver.
- **Eigentümer**, für jedes erzeugtes Objekt wird der Erzeuger in die Rolle Eigentümer hinzugefügt. Der Eigentümer eines Objektes hat folgende Rechte:
  - Endgültig löschen

  Er kann allerdings nur dann Objekte löschen, wenn kein anderes Objekt mehr eine Beziehung zu diesem unterhält.

Es gibt die Möglichkeit Rollen selbst zu definieren. Dabei kann man als Vorlage eine bestehende Rolle benutzen oder über die Berechtigungsklassen einfach und flexibel die Rechte selbst vergeben. Diese selbst definierten Rollen sollten nur im jeweiligen Objektkontext gelten, das heißt im man kann in unterschiedlichen Arbeitsbereichen Rollen mit gleichen Namen haben. Diese können aber unterschiedliche Rechte haben, ich denke das macht die Vergabe der Rollen noch etwas mehr flexibler, weil jeder Benutzer in verschiedenen Projekten unterschiedliche Rechte haben kann. Sinnvoll wäre es über eine Wahlmöglichkeit den Geltungsbereich der Rolle auszuwählen. Entweder gilt die Rolle projektübergreifend für alle Objektknoten oder nur in dem jeweiligen Knotenbereich.

Wegen der umfangreichen Rechte ist der Administrator aus Sicherheitsgründen keine Rolle im eigentlichen Sinn des Sicherheitskonzeptes. Es soll vermieden werden, dass über die normale Benutzerschnittstelle an den Administrationsrechten manipuliert werden kann. Um mit Administrationsrechten arbeiten zu können, muss diese Rolle explizit aktiviert werden. Das kann man über die graphische Schnittstelle lösen. Damit soll verhindert werden, dass ein Benutzer die ganze Zeit mit Administrationsrechten arbeitet und damit eine potentielle Gefahr darstellt.

4.6.5 Vererbung

Damit nicht bei jedem neu angelegtem Objekt eine explizite Zuordnung von Rollen notwendig wird, werden die Rollendefinitionen entlang der Objekthierarchie vererbt. Wenn ein neues Objekt in den Objektgraph eingefügt wird, dann erbt dieses Objekt alle sicherheitsrelevanten Rollenzuordnungen. Der Geltungsbereich einer Rolle ist das Objekt, für das der Benutzer eine Rollenmitgliedschaft hat, und alles „unterhalb“ dieses Objekts, und zwar so weit, bis dem Benutzer in der Objekthierarchie eine andere Rolle zugewiesen wird.

Man muss hier aber auf jeden Fall den privaten vom gemeinsamen Bereich trennen. Rollenzuweisungen, die im privaten Bereich vorgenommen wurden, werden nur im privaten Bereich weiter vererbt. Dasselbe gilt auch für gemeinsame Objektbereiche, diese Bereiche können nur von anderen gemeinsamen Bereichen erben. Dies hat den Sinn, Sicherheitskonflikte bei der Rollenvergabe zu minimieren. Folgender Fall wäre sonst möglich, siehe dazu Abbildung 4.21. In diesem Fall würde Objekt C alle Rollenzuweisungen von Objekt A erben. Wenn der Rolleninhaber Manager in seinem Bereich ist, dann würde er auch Manager im gemeinsamen Bereich werden. Dies würde zu einer Sicherheitsverletzung führen, da der Benutzer mehr Rechte bekommt, als ihm eigentlich zusteht, daher die genannte Einschränkung der Vererbung. Die Rolle *Eigentümer* bildet hier eine Ausnahme, da diese Rolle sich nur auf ein-

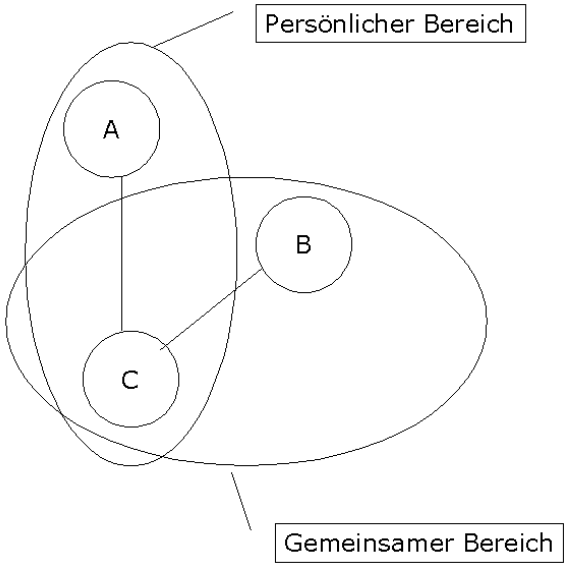

Abbildung 4.21: Vererbung zwischen persönlichen und öffentlichen Bereichen

zelne Objekte bezieht und damit in der Objekthierarchie nicht vererbt wird. Der Eigentümer eines Objektes ist nicht notwendigerweise Eigentümer der davon abgeleiteten Objekte. Bei der Erzeugung eines neuen Objektes wird der Benutzer als Eigentümer in die Metadaten des Objektes eingetragen. Nur er kann bei Bedarf die Eigentümerschaft an andere Benutzer weitergeben. Man könnte darüber bei der Berechnung der Speicherplatzbelegung im Raum den Eigentümer damit belasten. Damit würde sich der Speicherplatz jeden Benutzers daran orientieren, bei welchen Objekten er als Eigentümer eingetragen ist.

4.6.6 Konditionen

Durch die Vererbungshierarchien kann es zu einer Ansammlung von Berechtigungen kommen, die dem Benutzer Zugriffe ermöglichen, die zur Ausführung seiner Aufgaben nicht nötig sind. Um hier Abhilfe zu schaffen, muss das Prinzip der statischen und dynamischen Pflichtentrennung hier angewendet werden. Dazu beschränkt man die Rollenmitgliedschaft der Benutzer. Dazu bedarf es einer Definition der Abhängigkeiten der verschiedenen Rollen. Als beste Möglichkeit hat es sich herausgestellt die Möglichkeit zur dynamischen Pflichtentrennung zu nutzen, da vielfach ausreicht die gleichzeitige Aktivität von Subjekten in unterschiedlichen Rollen zu untersagen. Dazu muss man die gleichzeitige Rollenmitgliedschaft nicht untersagen. Weitere komplexere Konditionen lassen sich bei Bedarf einfach nachrüsten, aber für die erste Ausbaustufe des interaktiven Konferenzraumes reicht fürs Erste die dynamische Pflichtentrennung aus.

4.6.7 Autorisierung

Die Implementation eines rollenbasierten Zugriffsmodells wird auf der Basis von Java vorgenommen. Die Rollen können entweder in einer Datenbank oder in einem Verzeichnisdienst abgelegt werden. Das folgende UML-Modell dient als Basis für die Implmentation, siehe Abbildung 4.22. Ein Benutzer ist Mitglied einer Rolle und diese werden in der Session je nach Bedarf aktiviert. Nicht der Benutzer hat die Berechtigung, sondern die Rolle. Die Rolle hat eine Berechtigung eine Operation auf ein Objekt auszuführen. Es gibt verschiedene Arten von Rollen und diese können sich auch gegenseitig beinhalten. Rollen können in einer Vererbungshierarchie stehen, und mit Bedingungen zur Pflichtentrennung verknüpft sein. Dies ist das Modell, das der Autorisierungsprüfung zugrunde liegt.

Der Autorisationsmanager gibt anhand der Rollenvergabe Tickets für die Nutzung von Diensten an den Client aus.

4.6.8 Authentifikation

Der Client meldet sich im Raum an und wird authentifiziert. Der Authentifikationsmanager prüft die Identität des Benutzers und der Client erhält ein Ticket, das er dem Autorisationsmanager vorlegen kann, um weitere Dienste zu nutzen. Eine gängige Implementation so eines Dienstes ist Kerberos [MIT (2006)].

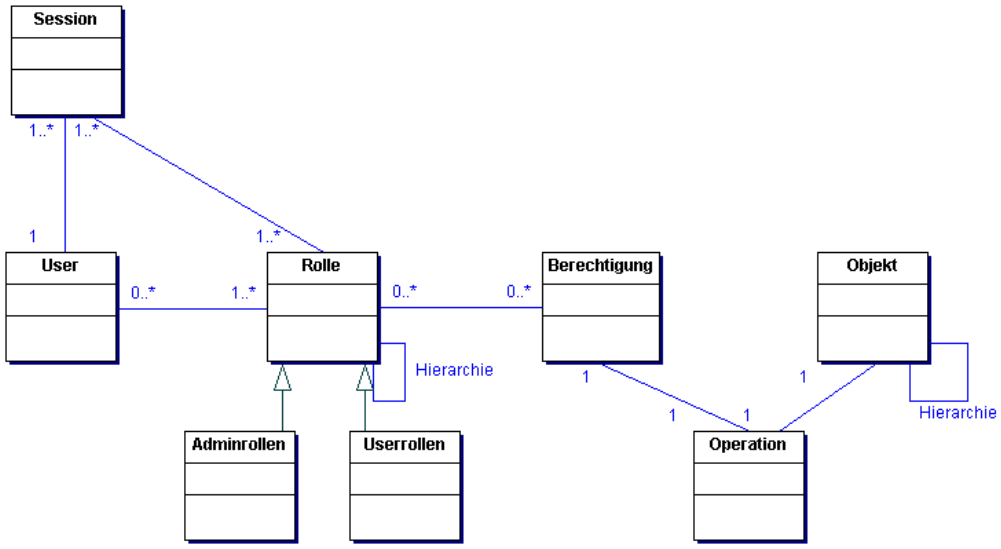

Abbildung 4.22: UML-Modell

Kerberos Kerberos ist ein verteilter Authentifizierungsdienst (Netzwerkprotokoll) zur Authentifizierung, der für offene und unsichere Computernetze von Steve Miller und Clifford Neuman entwickelt wurde. Die zurzeit aktuelle Version ist Kerberos 5. Kerberos bietet eine sichere und einheitliche Authentifizierung in einem ungesicherten TCP/IP-Netzwerk aus sicheren Hostrechnern. Die Authentifizierung übernimmt eine vertrauenswürdige dritte Partei. Diese dritte Partei ist ein besonders geschützter Kerberos 5-Netzwerkdienst. Kerberos unterstützt Single-Sign-On, damit muss sich der Benutzer nur einmal anmelden, und kann dann alle Netzwerkdienste nutzen, ohne ein weiteres Mal ein Passwort eingeben zu müssen. Kerberos übernimmt die weitere Authentifizierung. Es gab mehrere Gründe für die Entwicklung von Kerberos:

- Immer größere Netze
- Schutz von Ressourcen vor nicht authorisiertem Zugriff
- Nicht vertrauenswürdige Netze, z. B. WLAN

Kerberos hat eine Reihe von Vorteilen, die besonders im interaktiven Konferenzraum Anwendung finden. Das Benutzerkennwort wird nur beim Anmelden übertragen, sämtliche Tickets sind verschlüsselt. Aus Sicherheitsgründen ist die Gültigkeit der Tickets beschränkt und durch den zentralisierten Ansatz sind die Sicherheitsmechanismen zentral zu verwalten.

4.6.9 Betrachtung der Kommunikation

Der Benutzer betritt den Raum mit seinem Notebook. Durch Übermittlung seiner Authentifizierungsdaten meldet er sich beim Authentifikationsmanager an. Er bekommt ein Ticket ausgestellt, dass ihn berechtigt Dienste im Raum zu nutzen. Die Kommunikation läuft verschlüsselt ab. Wenn der Client einen Dienst nutzen möchte, dann wird das ausgestellte Ticket dem Autorisationsmanager vorgelegt und anhand der Rollenvergabe wird geprüft, ob der Client den Dienst nutzen darf oder nicht. Dies wird noch einmal in Abbildung 4.23 veranschaulicht.

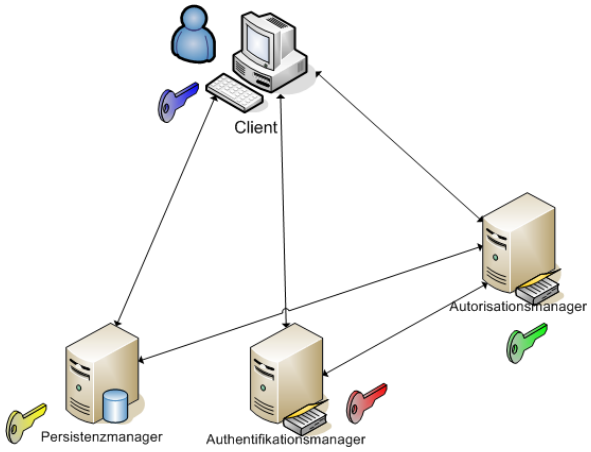

Abbildung 4.23: Ticketbasierte Autorisierung

Zwei Vorgänge bedürfen einer genaueren Betrachtung, die Anmeldung eines Clients und die Autorisation einer Dienstnutzung.

- **Anmeldung**, Der Benutzer meldet sich am Client an. Der Authentifikationsmanager erstellt ein Ticket für den Autorisationsmanager. Der Authentifikationsmanager verschlüsselt das Ticket mit dem geheimen Schlüssel des Autorisationsmanagers und fügt seinen Schlüssel an und verschlüsselt dann beides mit dem geheimen Schlüssel des Benutzers (blauer Schlüssel). Jetzt wird der Benutzer am Client nach seinem Passwort gefragt und dieses wird dann in den Schlüssel des Benutzers umgewandelt. Damit wird dann das Ticket und der äußere Sitzungsschlüssel dechiffriert.
- **Nutzung eines Dienstes**, der Client erstellt dazu einen Echtheitsbeweis und chiffriert diesen mit dem Schlüssel des Authentifikationsmanagers. Der Autorisationsmanager dechiffriert das Ticket und erhält daraus den Echtheitsbeweis und den Schlüssel des Authentifikationsmanagers. Er prüft, ob das Ticket noch gültig ist, und prüft anhand der Rollenzuweisungen, ob der Benutzer den Dienst nutzen kann. Ist das der Fall, dann wird ein neues Ticket mit einem neuen Sitzungsschlüssel erstellt und mit dem

geheimen Schlüssel des Dienstes (hier die Persistenz) verschlüsselt. Dann wird der Sitzungsschlüssel angehängt und beides mit dem Schlüssel des Authentifikationsmanagers verschlüsselt. Der Dienst dechiffriert das Ticket, bekommt den Sitzungsschlüssel, dechiffriert den Echtheitsnachweis mit dem Sitzungsschlüssel und überprüft das Ticket und den Echtheitsbeweis. Wenn die Prüfung erfolgreich war, dann kann der Dienst genutzt werden.

4.6.10 Protokollierung

Die Protokolle der Vorgänge im Konferenzraum werden an einen zentralen Server gesendet, der sich um die Aufbewahrung der Daten kümmert. Dieser ist bei einem hohen Schutzbedarf noch besonders zu sichern. Der Server hat als Betriebssystem Debian Sarge [4] installiert. Die Protkolldaten werden in einer relationalen Datenbank abgelegt, ein geeignetes Produkt ist PostgreSQL [PostgreSQL (2006)][5]. Der Protokollierdienst, der sich um die Annahme der Meldungen kümmert, ist ein Syslog-Dienst. Eine gängige Implementierung ist der Syslog-ng [Balabit (2006)]. Die Ablage der Meldungen in einer Datenbank ist hier als sinnvoll zu erachten, da eine Datenbank umfangreiche Auswertungsmöglichkeiten bietet und damit die Flexibilität deutlich zunimmt. Zur Erhöhung der Sicherheit ist nur der Syslog-ng-Server über das Netzwerk erreichbar, und das Senden der Protokolldaten zum Server ist verschlüsselt vorzunehmen. Dies sichert die Datenintegrität und verhindert ein Verfälschen der Daten. Mit einem Syslog-Dienst lassen sich die Protokollmeldungen aller Dienste und Geräte netzwerkweit einheitlich sammeln. Das erleichtert die Arbeit beim Sichern, Analysieren und Weiterverarbeiten der Logdateien. In der Vergangenheit dienten die Protokolle oft nur dazu, festzustellen, ob das lokale System einwandfrei läuft. Heute muss die Zuverlässigkeit verbessert werden, und die Integrität der Daten gewährleistet sein. Syslog-ng [Balabit (2006)]ist Open-Source und bietet eine Implementation eines Syslog-Servers auf der Linux- und Unix-Plattform an. Es implementiert das syslog-Protokoll und bietet einige Erweiterungen, die bekannte Schwachstellen des Protokolles beheben sollen. Man kann Meldungen wahlweise in Dateien oder in Datenbanken ablegen. Die Menge der unterstützten Plattformen ist sehr umfangreich, und die Syslog-Implementierung ist überall einheitlich. Damit wird der Aufbau eines homogenen Logging-Systems deutlich erleichert. Durch die mittlerweile sehr große Verbreitung wird Syslog-ng von vielen anderen Syslog-Implementierungen als Referenzimplementierung verwendet, damit übt das Programm einen erheblichen Einfluss auf die Weitereintwicklung auch anderer Syslog-Implementierungen aus.

[4] Weitere Informationen zu Debian Sarge sind im Kapitel 4.5.3 zu finden
[5] Weitere Informationen zu PostgreSQL sind Kapitel 4.5.3 zu finden

4.7 Evaluation

Die Implementierung hat gezeigt, dass sich der Entwurf sehr gut in die Praxis umsetzen lässt. Die Zusammenarbeit von Java, Hibernate, und PostgreSQL hat sich als unkompliziert herausgestellt. Da besonders Hibernate und Java sehr gut aufeinander abgestimmt sind, wurde der Prototyp in sehr kurzer Zeit umgesetzt. Da auf der Seite der Anwendungslogik kaum Änderungen vorgenommen mussten, wurden die Anforderungen der Wartbarkeit und Erweiterbarkeit so gut wie erfüllt. Den einfachen Austausch des Datenspeichers wird von Hibernate sehr gut unterstützt, es ist so nur eine Änderung in der Konfiguration von Hibernate nötig. An der Anwendungslogik oder an den Mappingdateien sind keine Änderungen nötig. Besonders das *lazy loading* und das *Caching* von Hibernate ersparen dem Entwickler eine Menge Aufwand bei der Implementation. Das rollenbasierte Zugriffmodell wurde im Prototyp nur sehr einfach umgesetzt, aber hat gezeigt, dass damit eine effiziente Vergabe von Berechtigungen möglich ist. Dieses Modell bietet eine sehr gute Grundlage für die Nutzung in einer kollaborativen Umgebung. Die Wahl der Programmiersprache Java hat sich zudem als überaus sinnvoll erwiesen, da sich dadurch die Software fast komplett auf einem normalen Desktop PC implementieren und testen ließ. Damit lassen sich unterschiedliche Plattformen einsetzen, was die Anforderung der Portabilität erfüllt. Die Installation der genutzten Komponenten lief problemlos ab, da durch eine sehr gute Dokumentation und hohe Standardisierung die Interoperabilität gewährleistet ist. Das rollenbasierte Zugriffsmodell hat auch bereits eine Standardisierung erfahren und ist daher ohne Problem einsetzbar.

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Ziel dieser Arbeit war der Entwurf einer Berechtigungsstruktur für kollaborative Umgebungen. Dieses Ziel wurde mit der vorliegenden Arbeit erreicht. Dazu wurde zuerst ein Szenario festgelegt, anhand dessen die Analyse stattfand. In der Analyse wurden als erstes die Anwendungsfälle ermittelt und die technischen Anforderungen fixiert. Anschließend wurde eine Architektur entwickelt und die beteiligten Komponenten näher beschrieben. Es wurden mögliche Berechtigungsklassen und die Bestandteile des Zugriffsrechtemodell beschrieben. Weiterhin wurde der Persistenzmechanismus festgelegt und eine konzeptionelle Übersicht über das zugrunde liegende Objektmodell gegeben. Darauf basierend wurde ein Prototyp entwickelt, der die wichtigsten Dienste der Persistenz umsetzt und die Machbarkeit des Entwurfes zeigt. Dabei wurde ein Objekt mit Binärdaten in die Datenbank geschrieben und wieder ausgelesen. Dieser Schritt hat gezeigt, dass sich die Anforderungen an die Persistenz mit heutigen technischen Möglichkeiten realisieren lassen.

5.2 Fazit

Diese Arbeit hat gezeigt, dass man Berechtigungsstrukturen für kollaborative Umgebungen modellieren und realisieren kann. Das gilt auch für die Realisierung der Persistenz, die durch die geeigneten Technologien eine einfache und flexible Form zur Datenhaltung gewährleistet. Hibernate bietet einen ausgereiften und stabilen Weg, um Objekte zu persistieren. Diverse Technologien zur Geschwindigkeitssteigerung sind bereits implementiert, besonders der Aspekt des *Lazy Loading* kann bei der vermaschten Objektstruktur einen Geschwindigkeitsvorteil bringen. Um weitere Objekte zu persistieren sind keinerlei Änderungen an der Applikationslogik nötig, damit wird eine große Flexibilität und Erweiterbarkeit erreicht. Durch die Ablage der Daten als Binärdaten in einer Datenbank kann es je nach Größe der Datei zu Geschwindigkeitseinbußen kommen. Geschwindigkeitsmessungen wurden in dieser Arbeit nicht durchgeführt. Dies ist im Rahmen einer anderen Arbeit noch genau zu evaluieren. Die erarbeiteten Rollen und Berechtigungsklassen bieten eine sehr gute Möglichkeit eine

Zugriffskontrolle im interaktiven Konferenzraum zu implementieren. Diese Berechtigungsstrukturen sind szenariounabhängig und lassen sich damit auch für andere Benutzergruppen nutzen. Dies erhöht die Einsetzbarkeit des interaktiven Konferenzraumes auch für andere Benutzergruppen. Ein wichtiger Punkt ist die Qualität der Benutzerschnittstelle. Durch eine hochqualitative und intuitiv zu bedienende Benutzerschnittstelle ist es möglich die Akzeptanz der Berechtigungsstrukturen soweit zu steigern, dass sie von den Benutzern angenommen wird und auch angewendet wird. Dies trägt damit auch zur Sicherheit im Gesamtsystem bei.

5.3 Ausblick

Das Ziel ist es, aufzuzeigen, wie die vorgenommenen Untersuchungen weiter fortgeführt werden können.

Aufgrund der zunehmend überregionalen Firmenkooperation und Internationalisierung wächst der Bedarf zur Durchführung von kollaborativen Aktivitäten ohne physische Anwesenheit aller Beteiligten in einem Raum stark an. Die hohe Verbreitung von Bildschirmarbeitsplätzen und deren leistungsfähige Vernetzung wird es in Zukunft ermöglichen, Tätigkeiten, bei denen die Teilnehmer mittels Audio-/Videokonferenzsystemen interagieren, unabhängig von Zeit und Ort gemeinsam durchzuführen. Allerdings ist dazu eine verteilte Datenhaltung nötig, um den Anforderungen der Verwaltung und der effizienten Nutzung der Netzstruktur gerecht zu werden. Dabei muss ein besonderes Augenmerk der Nebenläufigkeitskontrolle gewidmet werden, da bei weltweit verteilten Daten es eher zu Verbindungsabbrüchen kommen kann. Da gilt es geeignete Konsistenzsicherungsverfahren zu evaluieren, die die Konsistenz sicherstellen. Durch die dann vorhandene Verteilung der Daten kann eine Lösung, wie in dieser Arbeit beschrieben, mit mobilen Datenbanken erreicht werden. Durch diese Trennung der Benutzer bekommt vor allen Dingen die Darstellung der gerade in Bearbeitung befindlichen Teile des Objektes eine wichtige Bedeutung. Es wird dann nicht mehr ausreichen, durch informelle Absprachen die Zugriffe auf Objekte zu regeln. Durch eine mögliche weltweit verteilte Umgebung sind auch an die Sicherheitsstruktur erhöhte Anforderungen zu stellen. Es wird dann auch nötig sein, die Sicherheitsrichtlinien zu verteilen und damit an jedem Ort der Welt die Durchsetzung der Richtlinien zu gewährleisten.

Weitere Untersuchungen müssen im Bereich der Segmentierung der Dokumente getätigt werden, jedes Dokument sollte in seine Bestandteile zerlegbar sein um ein effizientes kollaboratives Arbeiten zu gewährleisten. Es muss geprüft werden, in welcher Form sich das effizient realisieren lässt. Ein besonderes Augenmerk gilt es hierbei auf die Binärformate zu legen. Dort ist zu prüfen, wie performant die Ablage der Daten als Blob in der Datenbank ist oder ob es nötig ist, die Binärdaten woanders abzulegen. Dann würden die Metadaten zum Objekt in der Datenbank liegen und die Binärdaten z. B. auf einem Webserver. In der

Datenbank würde dann der Verweis liegen. Allerdings könnte es schwierig sein, auch auf dem Webserver die Sicherheitsrichtlinien umzusetzen. Darüber müssen dann umfangreiche Geschwindigkeitsmessungen Aufschluss geben.

Da ich in meiner Diplomarbeit von einem gutartigen System ausgegangen bin, ist die Realisierung der Sicherheitsanforderungen sehr gering ausgefallen. In diesem Bereich sind noch genaue Analysen erforderlich, da die Sicherheit im Raum ganzheitlich betrachtet werden muss. Das schwächste Glied bestimmt den Grad der Sicherheit im Raum. Wenn Personen im Raum sehr hohe Anforderungen an die Sicherheit im Raum haben, dann ist über eine gezielte Härtung der befindlichen Komponenten im Raum nachzudenken. In diesem Zusammenhang muss auch über den Schutz der Endgeräte, wie z. B. Notebooks, nachgedacht werden, da diese nicht im direkten Zugriff des Administrators sind. Es ist durchaus denkbar, dass über die Endgeräte schadhafte Programme in den Raum gelangen und dort Sicherheitsverletzungen anstreben.

Literaturverzeichnis

[Adobe Systems Incorporated 2006] ADOBE SYSTEMS INCORPORATED: *Adobe Acrobat Family*. 2006. – URL http://www.adobe.de. – Zugriffsdatum: 03.06.2006

[Arbeitskreis eGovernment 2006] ARBEITSKREIS EGOVERNMENT: Orientierungshilfe Datenschutz bei Dokumentenmanagementsystemen / Bundesrepublik Deutschland. URL http://fhh.hamburg.de/stadt/Aktuell/weitere-einrichtungen/datenschutzbeauftragter/informationsmaterial/informationstechnik/dokumentenmanagement-oh-pdf,property=source.pdf. – Zugriffsdatum: 01.03.2006, 2006. – Forschungsbericht

[Balabit 2006] BALABIT: *IT Security: syslog-ng*. 2006. – URL http://www.balabit.com/products/syslog_ng/. – Zugriffsdatum: 20.06.2006

[Bartnik 2006] BARTNIK, Roman: *Weiterentwicklung einer Technologiebasis für interaktive Gruppenarbeitsräume*, Haw Hamburg, Diplomarbeit, Februar 2006. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/diplom/bartnik.pdf. – Zugriffsdatum: 28.02.2005

[Böhm 2005] BÖHM, Aiko: *Einsatz mobiler Datenbanken in einem Ferienclub-Szenario*, Haw Hamburg, Diplomarbeit, März 2005

[Bundesrepublik Deutschland 2006] BUNDESREPUBLIK DEUTSCHLAND: *Gesetz zur digitalen Signatur (Signaturgesetz - SigG)*. Bundesrepublik Deutschland, 2006. – URL http://www.datenschutz-berlin.de/recht/de/rv/tk_med/iukdg_de.htm#a3. – Zugriffsdatum: 20.05.2006

[Burfeindt 2006] BURFEINDT, Lars: *Konstruktion einer Middleware für computergestützte Gruppenarbeit in ubiquitärer Systemumgebung*, Haw Hamburg, Diplomarbeit, june 2006

[DCMI 2006] DCMI: The Dublin Core Metadata Element Set / Dublin Core Metadata Initiative. URL http://dublincore.org/documents/dcmi-terms/. – Zugriffsdatum: 10.04.2006, 2006. – Forschungsbericht

[Debian 2006] DEBIAN: *Das universelle Betriebssystem*. 2006. – URL http://www.debian.org. – Zugriffsdatum: 10.06.2006

[Eckert 2006] ECKERT, Claudia: *IT-Sicherheit*. Bd. 4. Auflage. Oldenbourg Verlag, 2006. – ISBN 3-486-20000-3

[Ehlers 2006] EHLERS, Petra R.: *Titel steht noch nicht fest*, Haw Hamburg, Diplomarbeit, june 2006

[Ferraiolo u. a. 1992] FERRAIOLO ; KUHN ; GAVRILA: Role-Based Access Control / National Institute of Standards and Technology, Computer Security Division. URL http://csrc.nist.gov/rbac/ferraiolo-kuhn-92.pdf. – Zugriffsdatum: 01.04.2006, 1992. – Forschungsbericht

[Fraunhofer 2006] FRAUNHOFER: *Fraunhofer FIT und OrbiTeam Software GmbH: BSCW (Basic Support for Cooperative Work)*. 2006. – URL http://www.bscw.de/. – Zugriffsdatum: 10.06.2006

[Gates 2006] GATES, Ray: *ISO/IEC 11179, Information Technology – Metadata Registries (MDR)*. 2006. – URL http://metadata-standards.org/11179/. – Zugriffsdatum: 25.03.2006

[Hamburger Datenschutzbeauftragter 1990] HAMBURGER DATENSCHUTZBEAUFTRAGTER: *Datenschutzrecht*. 1990. – URL http://fhh.hamburg.de/stadt/Aktuell/weitere-einrichtungen/datenschutzbeauftragter/datenschutzrecht/hamburgisches-datenschutzgesetz-1990-07-05-pdf,property=source.pdf. – Zugriffsdatum: 03.02.2006

[Hibernate 2006] HIBERNATE: *Relational Persistence for Java and .NET*. 2006. – URL http://www.hibernate.org. – Zugriffsdatum: 10.05.2006

[Intraline 2006] INTRALINE: *Intraline - Die Intranet-Standardsoftware*. 2006. – URL http://www.intraline.de/html/com/. – Zugriffsdatum: 23.05.2006

[Kahlbrandt 1998] KAHLBRANDT, Prof. Dr. B.: *Softwarenengineering*. Springer, 1998. – ISBN 3-540-63309

[Lampson 1974] LAMPSON, Butler W.: Protection / Xerox Corporation. Palo Alto, California, 1974. – Forschungsbericht. – URL http://citeseer.ist.psu.edu/287804.html. – Zugriffsdatum: 05.01.2006

[MIT 2006] MIT: *Kerberos: The Network Authentication Protocol*. 2006. – URL http://web.mit.edu/kerberos/. – Zugriffsdatum: 20.06.2006

[Neumann 2006] NEUMANN, Carola: *Effizienzsteigerung von Diskussionsprozessen in einem neu gestalteten Konferenzraum*, Haw Hamburg, Diplomarbeit, june 2006

[NIST 2000] NIST: *Role Based Access Control*. 2000. – URL http://csrc.nist.gov/. – Zugriffsdatum: 07.02.2006

[PostgreSQL 2006] POSTGRESQL: *The World´s most advanced Open Source Database*. 2006. – URL http://www.postgresql.org. – Zugriffsdatum: 01.05.2006

[habil. Reiner R. Dumke 2006] REINER R. DUMKE, Prof. Dr.-Ing. habil.: *UML-Tutorial*. 2006. – URL http://ivs.cs.uni-magdeburg.de/%7Edumke/UML/. – Zugriffsdatum: 23.06.2006

[Russell und Gossweiler 2001] RUSSELL, Daniel M. ; GOSSWEILER, Rich: On the Design of Personal and Communal Large Information Scale Appliances / IBM Almaden Research Center, 650 Harry Rd, San José, CA, USA, 95120. 2001. – Forschungsbericht

[Saake und Sattler 2003] SAAKE, Gunter ; SATTLER, Kai-Uwe: *Datenbanken & Java: JDBC, SQLJ, ODMG und JDO*. 2. überarbeitete und aktualisierte Auflage. Heidelberg : dpunkt.verlag, 2003. – ISBN 3-89864-228-3

[Schlichter 2002] SCHLICHTER, Johann: *Computergestützte Gruppenarbeit, Vorlesungsskript*. München : Institut für Informatik, TU München,Germany, 2002. – URL http://www.bsi.bund.de/gshb/deutsch/m/m02008.htm. – Zugriffsdatum: 13.03.2006

[Schmidt 2006] SCHMIDT, Prof. Dr. T.: Verteilte Transaktion und Nebenläufigkeitskontrolle / HAW Hamburg. URL http://users.informatik.haw-hamburg.de/~schmidt/vs/07_Transaktionen.pdf. – Zugriffsdatum: 25.03.2006, 2006. – Forschungsbericht

[Shen und Dewan 1992] SHEN ; DEWAN: Access Control for Collaborative Environments / ACM. URL http://citeseer.ist.psu.edu/59575.html. – Zugriffsdatum: 10.10.2005, march 1992. – Forschungsbericht

[Smarttech 2005] SMARTTECH: *Smart Technologies*. 2005. – URL http://www.smarttech.de. – Zugriffsdatum: 29.11.2005

[Stanford 2006] STANFORD: *Interactive Workspaces Project*. 2006. – URL http://iwork.stanford.edu/. – Zugriffsdatum: 10.09.2005

[Stanford University 2005] STANFORD UNIVERSITY, HCI G.: *Human Computer Interaction. Website*. 2005. – URL http://hci.stanford.edu/research/ideas/. – Zugriffsdatum: 10.07.2005

[Sun 2006] SUN: *The Source for Java Developers*. 2006. – URL http://java.sun.com. – Zugriffsdatum: 10.01.2006

[Vollbrecht und Calhoun 2000] VOLLBRECHT, J. ; CALHOUN, P.: AAA Authorization Framework, RFC 2904 / RFC 2904. 2000. – Forschungsbericht

[XDocLet 2006] XDOCLET: *Attribute-Oriented Programming*. 2006. – URL `http://xdoclet.sourceforge.net/xdoclet/index.html`. – Zugriffsdatum: 2006-06-03

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 10. Juli 2006	
Ort, Datum	Unterschrift