Seminararbeit

Konzeption einer Basisarchitektur für kollaborative Applikationen auf Basis von Eclipse und dem IBM Workplace Managed Client

Prof. Dr. Ludwig Nastansky

Sommersemester 2005

Betreuer
Dipl. Wirt. Inf. Ingo Erdmann

vorgelegt von:

Jörg Wiese

Wirtschaftsinformatik

Brüderstraße 36

33098 Paderborn

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
IBM	International Business Machines
RCP	Rich Client Platform
IDE	Integrated Development Environment
JDT	Java Development Tools
TCO	Total Cost of Ownership
UML	Unified Modeling Language
SDO	Service Data Objects
SQL	Structured Query Language
JDBC	Java Database Connectivity
WMC	Workplace Managed Client

1 Einleitung

1.1 Motivation und Zielsetzung

In der heutigen vernetzten Welt versuchen Unternehmen verstärkt ihre vorhandene IT - Infrastruktur zu nutzen, um eine effiziente Plattform für Kommunikation und Kollaboration bereitzustellen. Im Hinblick auf die Globalisierung wird eine fast zwingende Notwendigkeit einer solchen effizienten Umgebung deutlich, da durch die Globalisierung immer häufiger Teams entstehen, die über weite Entfernung oder gar über unterschiedliche Zeitzonen hinweg zusammen arbeiten müssen.[1] IBM verfolgt mit der neuen IBM Workplace Produktfamilie und dem darin enthaltenen Workplace Managed Client das Ziel, eine solche effiziente Plattform für Kommunikation und Kollaboration im Unternehmen zu bieten.[2] Diese Plattform bietet ein breites Spektrum an Anwendung zur Unterstützung von Kollaboration, kann aber nicht alle Einsatzszenarien von kollaborativen Applikationen in einem Unternehmen abdecken. Für spezielle kollaborative Applikationen wird die Entwicklung eigener kollaborativer Anwendungen notwendig.

Ziel dieser Arbeit ist es auf Basis einer Anforderungsanalyse eine Basisarchitektur zu schaffen, die es Entwicklern erleichtert asynchrone kollaborative Applikationen auf Basis des IBM Workplace Managed Client zu entwickeln. Da der Entwurf und die Umsetzung einer komplett ausgereiften Basisarchitektur eine sehr komplexe und umfangreiche[3] Aufgabe darstellt, kann dies nicht Ziel dieser Seminararbeit sein. Vielmehr sollte eine Basisarchitektur geschaffen werden, die leicht erweiterbar ist und schon jetzt den Entwickler in möglichst vielen Entwicklungsaufgaben unterstützt. Zu diesem Zweck sollen insbesondere generisch nutzbare Sicherheitsmechanismen und Datenablagestrukturen von der Basisarchitektur zur Verfügung gestellt werden.

1.2 Aufbau der Arbeit

Zunächst werden in Kapitel 2 wichtige Grundkonzepte für die weitere Arbeit erläutert. Dazu wird der Begriff asynchrone Groupware diskutiert und grundlegende Client Konzepte erläutert.

---

[1] Siehe hierzu *Ozzie und O'Kelly* 2003, S. 5 und *Costanzo und Littlejohn* (2003), S. 3
[2] Siehe hierzu *IBM Lotus News* 2005 und *IBM Whitepaper* (2005), S. 9
[3] Siehe Kapitel 2.3

Im Kapitel 3 werden anfangs Merkmale und Anforderungen asynchroner kollaborativer Applikationen herausgestellt. Darauf folgend wird ein differenzierender Blick auf mögliche Client Plattformen für die in Kapitel 4 entwickelte Basisarchitektur gegeben. Anschließend wird in Kapitel 4 der Aufbau der Basisarchitektur beschrieben und auf die beispielhafte Nutzung dieser, anhand der Entwicklung einer Beispielanwendung, eingegangen. Im Ausblick in Kapitel 5 werden mögliche Erweiterungen der Basisarchitektur dargestellt. Zuletzt wird in Kapitel 6 ein kurzes Fazit gegeben.

2 Grundlagen

2.1 Groupware und asynchrone kollaborative Applikationen

Unter Kollaboration versteht man die Zusammenarbeit mehrerer Individuen. Eine Definition für die Bedeutung des Begriffes Kollaboration im Businesskontext geben Costanzo und Littlejohn: „In the world of business, collaboration involves more than one person networking and working together to achieve a common goal or outcome".[4] Für Groupware lassen sich in der zahlreichen Literatur verschiedene Definitionen finden. Eine einheitliche Definition hat sich für Groupware zum jetzigen Zeitpunkt nicht herausgebildet. In der vorliegenden Arbeit sollen „unter dem Begriff Groupware Applikationen verstanden werden, welche die Computerunterstützung kooperativen Arbeitens ermöglichen".[5] Im Zusammenhang mit dem Begriff Kollaboration können Groupware Applikationen folglich als „collaborative systems"[6] bezeichnet werden.

Eine Möglichkeit kollaborative Applikationen zu kategorisieren ist nach der Raum – Zeit – Matrix von Johansen[7] gegeben. In der Matrix wird eine Kategorisierung nach Zeit - synchron (zur gleichen Zeit), asynchron (zu unterschiedlichen Zeiten) – und nach Raum – gleicher Ort, unterschiedlicher Ort – vorgenommen. Somit sind asynchrone kollaborative Applikationen Anwendungen, die eine Zusammenarbeit zu unterschiedlichen Zeiten von meist räumlich verteilten Nutzern ermöglichen. Hierbei ist zu beachten, dass die Aktionen der Nutzer in erster Linie asynchron stattfinden, gleichzeitige Aktionen aber nicht ausgeschlossen sind.[8] Da aber primäres Ziel dieser Anwendungen die Unterstützung asynchroner Kollaboration ist, werden gleichzeitige Aktionen nur geringfügig unterstützt.

---

[4] Vgl. *Costanzo und Littlejohn* 2003, S. 1
[5] Vgl. *Fischer et al.* 2000, S. 239. Eine genau Diskussion des Begriffes Groupware und verwandte Begriffe kann ebenfalls unter *Fischer et al.* S.239f oder *Borghoff und Schlichter* 1998 S. 91ff gefunden werden.
[6] Vgl. *Guerrero und Fuller* 2001, S. 455
[7] Vgl. *Johansen* 1988
[8] Vgl. *Appelt et al.* 2001, S.194

2.2 Client Konzepte

2.2.1 Eclipse eine Rich Client Platform

Eclipse wurde ursprünglich als eine offene Plattform für Entwicklertools entworfen. Mit der Version 3.0 wurde Eclipse, orientiert an der OSGi Spezifikation[9], zu einer offenen Rich Client Platform weiter entwickelt.[10] Der Ausdruck Rich Client steht hier für reichhaltige Anwendungen, die dem Benutzer eine komfortable Benutzeroberfläche bieten. Die Idee hinter der Rich Client Platform (RCP) ist die Schaffung einer offenen Umgebung, die die Entwicklung jeglicher Art von Anwendungen unterstützt.[11] Die Eclipse RCP sowie die vorangegangenen Versionen bieten, basierend auf Plug-Ins eine leicht erweiterbare Struktur. Plug-Ins sind Komponenten, die die unterliegende Plattform nutzen und sie um eigene Funktionalitäten erweitern. Die folgende Grafik gibt eine Übersicht über die oben beschriebene neue Architektur der Eclipse RCP.

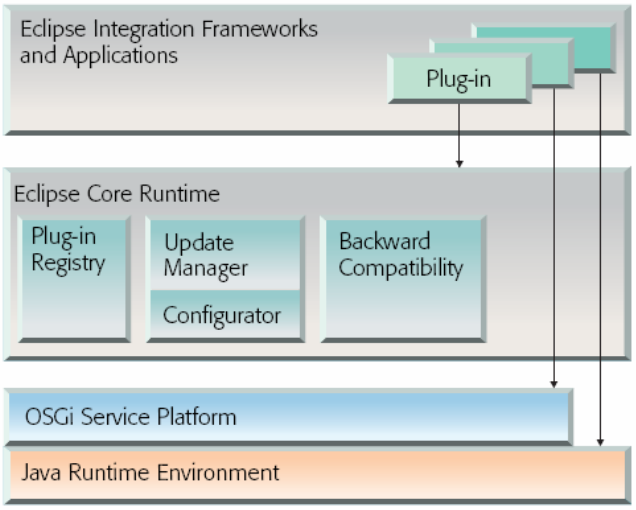

Abbildung 1: Eclipse RCP Architektur[12]

Wie die Abbildung zeigt, setzen die für die RCP entwickelten Anwendungen auf eine Basis Core Runtime auf. Diese Anwendungen bestehen aus einer Sammlung von Plug-Ins, die sich in der Core Runtime registrieren und die gesamte Anwendung bilden. So ist zum Beispiel die aktuelle Eclipse JDT[13] selbst eine solche Sammlung von Plug-Ins, die zusammen eine komplexe Java IDE bilden.

---

[9] Siehe *OSGi Service Platform* 2003
[10] Vgl. *Gruber et al.* 2005, S. 289
[11] Vgl. *Gruber et al.* 2005, S. 297
[12] Vgl. *Gruber et al.* 2005, S. 293
[13] Für genauere Informationen zur JDT siehe http://www.eclipse.org/jdt/index.html

2.2.2 IBM Workplace und IBM Workplace Managed Client

Die neue Produktfamilie IBM Workplace ist eine integrierte Umgebung für verschiedenste kollaborative Applikationen wie z. B. Email, Instant Messaging und Dokumenten Management. Die zentrale Idee beim Workplace Konzept ist, dass dem Benutzer ein effizienteres Arbeiten ermöglicht wird, indem er alle zur seiner Arbeit benötigten Anwendung in einer intrigierten Oberfläche vorfindet. Den Begriff IBM Workplace definiert IBM wie folgt: „IBM Workplace is IBM's initiative to redefine the front-end of computing to make people more productive and offer a better way to organize office work and its enabling technology. The IBM Workplace initiatives are built on IBM Workplace Collaboration Services and IBM Workplace Client Technology".[14] IBM Workplace Collaboration Services ist dabei die zentrale Server Umgebung, die basierend auf einem Portal Server, eine integrierte Plattform mit kollaborativen wieder verwendbaren Komponenten wie z.B. Mail und Dokumenten Management bereitstellt.

Der Zugriff auf diese Server Umgebung kann durch verschiedene vom Server verwaltete Clients erfolgen. Dabei geht das Spektrum der Clients vom browserbasierten Client bis zu dem Rich Client, dem IBM Workplace Managed Client[15] (WMC). Dieser wird von IBM wie folgt beschrieben: „IBM Workplace Client Technology provides a rich user experience within an easy to provision, less expensive than traditional, PC-based solutions environment. It provides higher levels of security, document and data integrity (since everything is stored centrally) and control".[16] Der IBM Workplace Managed Client basiert auf der im Kapitel 2.2.1 beschriebenen Eclipse Rich Client Platform. Die nachfolgende Grafik zeigt detailliert den konzeptionellen Aufbau des IBM WMC.

---

[14] Vgl. *IBM Whitepaper* 2005 S. 9

[15] Hinweis: Der IBM Workplace Managed Client wurde vorher von IBM unter dem Namen IBM Workplace Client Technology geführt.

[16] Vgl. *IBM Whitepaper* 2005 S. 9

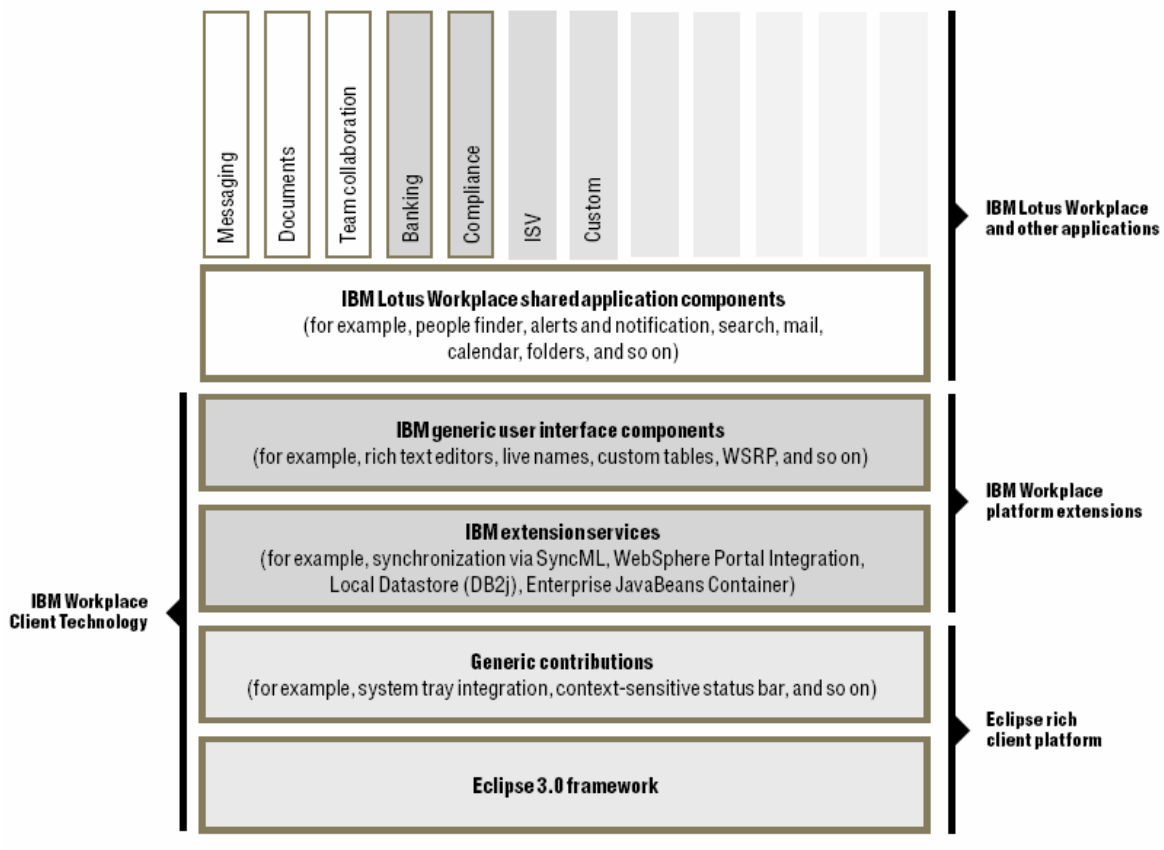

Abbildung 2: Architektur des WMC[17]

Ein zentraler Vorteil des WMC ist, dass er eine reichhaltige Benutzerumgebung liefert, trotzdem aber zentral vom Server verwaltet wird. Dadurch sollen durch den WMC geringere Kosten als bei einem normalen Rich Client entstehen[18]. Darüber hinaus ist durch die auf der RCP basierenden Architektur und des damit verbunden Plug-Ins Konzeptes, eine modulare Struktur gegeben, die wie die Grafik zeigt effizient mit eigenen Anwendungen erweitert werden kann. Daneben sind im WMC viele kollaborative Anwendungen wie Mail, Kalender und Messaging schon enthalten. Des Weiteren liefert er Funktionalitäten und Komponenten, wie den lokalen verschlüsselten Datenspeicher und User Interfaces, die genutzt werden können um eigene Anwendungen zu entwickeln.

2.3 Basisarchitekturen, Frameworks und Entwurfsmuster

Wie in Kapitel 1.1 beschrieben ist das Ziel dieser Arbeit die Entwicklung einer Basisarchitektur für asynchrone kollaborative Applikationen. Mit Hilfe dieser Basisarchitektur soll es Entwicklern später möglich sein, effizient asynchrone kollaborative Applikationen zu entwickeln. Im Kontext der objektorientierten

[17] Vgl. *IBM* 2004 S. 9
[18] Vgl. *IBM* 2004 S. 7

Programmierwelt wird eine komplexe Basisarchitektur auch als Framework bezeichnet. „Ein Framework besteht aus einer Menge von zusammenarbeitenden Klassen, die einen wieder verwendbaren Entwurf für eine bestimmte Klasse von Software darstellen.“ [19] Die Klasse von Software die in dieser Arbeit betrachtet wird, sind somit asynchrone kollaborative Anwendungen. Objektorientierter Entwurf zielt oftmals auf die Wiederverwendung von Komponenten. Frameworks werden explizit für die Wiederverwendung entworfen. Sie legen für ihre Nutzer Entwurfsparameter fest, so dass allgemein wiederkehrende Funktionen vom Framework übernommen werden und der Nutzer nur noch die speziellen Funktionen seiner Anwendung implementieren muss.[20]

Nach Gamma et al. „ist der Frameworkentwurf der komplizierteste Softwareentwurf von allen“[21]. Deshalb sollte bei Ihrer Entwicklung auf Entwurfsmuster zurückgegriffen werden, um einen hohen Grad an Entwurfs- und Codewiederverwendung zu erreichen und um ein möglichst ausgereiftes Framework zu erhalten.[22] Entwurfsmuster (engl. Pattern) beschreiben gut durchdachte Lösungsansätze für häufig wiederkehrende Problemstellungen beim Entwurf von Softwaresystemen. Eine detaillierte Einführung in anerkannte Entwurfsmuster kann in Gamma et al. 2002 gefunden werden.

Für eine Einführung in objektorientierte Programmierung mit Java und für eine Erläuterung der in dieser Arbeit verwendeten Begriffe aus diesem Fachgebiet sei auf Bishop 2001 verwiesen.

3 Konzept der Basisarchitektur

3.1 Merkmale asynchroner kollaborativer Applikationen

In diesem Kapitel sollen die Merkmale kollaborativer Applikationen herausgestellt werden. Hierzu werden zwei typische asynchrone kollaborative Applikationen zur Verdeutlichung herangezogen:

Diskussionsforum

In einem Diskussionsforum können Benutzer Beiträge zu Themen formulieren, die dann allen Benutzern des Forums zugänglich sind. Andere Benutzer können zu den bestehenden Beiträgen Antworten erstellen oder selbst Beiträge formulieren. Durch die

[19] Vgl. *Gamma et al.* 2002 S. 37
[20] Vgl. *Gamma et al.* 2002 S. 37
[21] Vgl. *Gamma et al.* 2002 S. 37
[22] Vgl. *Gamma et al.* 2002 S. 38

Möglichkeit auf Beiträge zu antworten entsteht eine Hierarchie zwischen diesen. Zusätzlich zu den Beiträgen an sich werden noch Metainformationen wie z. B. der Name des Autors, das Erstelldatum oder eine Kategorie des Themas erfasst. Um auf die erstellten Beiträge Zugriff zu haben, werden den Benutzern die Beiträge in Ansichten dargestellt.

Kontrollworkflow

In dieser Anwendung wird das Überprüfen von Dokumenten z. B. eines Artikel für die Veröffentlichung auf einer Homepage unterstützt. Hierzu wird der Artikel in der Anwendung erstellt. Danach wird der Workflow angestoßen, der alle Benutzer involviert, die am Kontrollprozess beteiligt sind. Dazu werden die Benutzer aufgefordert ihre Aktionen (zur Kenntnis nehmen, Freigeben etc.) vorzunehmen. Die beschriebene Anwendung kann sowohl als eigenständige Anwendung angesehen werden, kann aber auch integriert in z. B. einen Gruppeneditor[23] oder einen Diskussionsforum eingesetzt werden.

Ein Hauptmerkmal einer jeden asynchronen kollaborativen Anwendung ist der *gemeinsame Arbeitsbereich* bzw. der *gemeinsame Informationsraum*[24]. Er ist der Einstiegspunkt für die Nutzer um Zugriff auf die gemeinsamen genutzten Datenobjekte zu haben, auf denen sie mögliche Aktionen ausführen können. Die genutzten *Datenobjekte* sind ebenfalls ein wichtiges Merkmal, da sie als zentrale Objekte die gemeinsam genutzten Informationen zur Verfügung stellen. Datenobjekte können sowohl strukturierte Daten als auch unstrukturierte Daten enthalten. Zu den eigentlichen Datenobjekten werden zusätzlich *Metainformationen* erfasst, die die Benutzer in der Koordination ihrer zur erledigenden Aufgabe unterstützen[25]. Ein weiteres Merkmal sind die *Aktionen*, die Nutzer auf den Datenobjekten ausführen können. Verdeutlicht am Beispiel des Forums spiegeln die verschiedenen Ansichten den gemeinsamen Arbeitsbereich wider. Hier sehen die Nutzer die gemeinsamen Informationen in der Form von Beiträgen und Antworten. Diese sind wiederum ein Beispiel für die zentralen Datenobjekte und das Erstellen und Bearbeiten von Beiträgen sind Beispiele für mögliche Aktionen. Metainformationen in einem Forum wären z. B. das Erstelldatum, das Thema, der Autor und eine Kategorisierung, die zusätzlich zu dem eigentlichen Beitrag erfasst werden. In der

[23] Für eine Übersicht über Gruppeneditoren sei auf *Borghoff und Schlichter* 1998, S. 405ff
[24] Vgl. *Appelt et. al.* 2001, S. 195 oder *Borghoff und Schlichter* 1998, S. 125
[25] Vgl. *Appelt et. al.* 2001, S. 195

Workflowanwendung wäre der Kontrollstatus eines Datenobjektes eine wichtige Metainformation.

Zwei weitere Merkmale kollaborativer Applikationen beziehen sich auf die Datenobjekte. Als erstes können Datenobjekte in einer *Hierarchie* zu einander stehen und zweitens können von den Datenobjekten unterschiedliche *Versionen* existieren. Die Hierarchie kann z. B. im Forum durch das Erstellen von Beiträgen und den untergeordneten Antworten dazu entstehen. In der Workflowanwendung können innerhalb des Kontrollprozesses durch Verbesserungsvorschläge der kontrollierenden Personen neue Versionen des Artikels (Datenobjektes) entstehen.

Ein weiteres zentrales Merkmal ist, dass kollaborative Applikationen im Gegensatz zu normalen Anwendungen *Mehrbenutzersysteme* sind. Hieraus ergeben sich wiederum weitere Merkmale. Zum einem existieren mehrere Benutzer mit unterschiedlichen *Benutzerdaten* die verwaltet werden müssen. Des Weiteren haben nicht alle Benutzer die gleichen *Zugriffsrechte* für die Datenobjekte. Zum Beispiel darf in einem Forum nicht jeder Benutzer alle Beiträge bearbeiten, sondern nur die von ihm selbst erstellten. Oftmals haben in kollaborativen Anwendungen mehrere Benutzer dieselben Zugriffsrechte oder andere für alle Benutzer geltenden Eigenschaften. D. h. es können oftmals Benutzer zu *Benutzergruppen* oder zu speziellen *Benutzertypen* zusammengefasst werden. In der Workflow Anwendung könnte es z. B. mehrere Benutzer geben, die dazu berechtigt sind den Artikel freizugeben.

Speziell die Workflow Anwendung macht deutlich, dass es dem System auch möglich seien muss, dem Nutzer *Mitteilungen* zukommen zu lassen. Da der Benutzer z. B. darüber aufgeklärt werden muss, dass er als nächstes einen Artikel kontrollieren muss. Die Tabelle im Anhang unter 8.1 gibt eine Übersicht über die in diesem Kapitel herausgestellten Merkmale.

3.2 Anforderungen asynchroner kollaborativer Applikationen

In dem folgenden Kapiteln sollen in Bezug auf die beschriebenen Merkmale Anforderungen an kollaborativen Applikationen beschrieben werden. Der Focus der Anforderungen liegt hierbei auf funktionale Aspekte und berücksichtigt nicht allgemeine ergonomische Aspekte von Software wie Benutzerfreundlichkeit, weil diese allgemeingültig für alle Anwendungstypen gelten.

3.2.1 Gemeinsamer Arbeitsbereich

Aus dem Merkmal des gemeinsamen Arbeitsbereiches entsteht zunächst die Anforderung, dass eine grafische Benutzeroberfläche existieren muss. Diese Benutzeroberfläche muss den Zugriff auf alle benötigten Informationen gewährleisten. Darüber hinaus muss sie die Benutzer in der Ausführung ihrer Arbeiten bestmöglich unterstützen. Somit müssen die Benutzer über die Oberfläche Zugriff auf alle für sie möglichen Aktionen haben. Den Zugriff auf die Daten sollte die Benutzeroberfläche mit Hilfe von verschiedenen Ansichten gewährleisten, die es ermöglichen die Daten aus verschiedenen Blickwinkeln durch Sortierung etc. zu betrachten. Außerdem müssen die Ansichten eine gute Möglichkeit der Navigation über die Daten bieten, damit der Benutzer möglichst effizient an die benötigten Daten gelangt.

3.2.2 Daten

Persistente Datenhaltung

Da sowohl strukturierte als auch unstrukturierte Informationen gespeichert werden müssen, ergeben sich eine Reihe von Anforderungen für die persistente Datenhaltung. Die Metainformationen und Teile der Datenobjekte liegen in strukturierter Form vor. Daneben besteht der Rest der Datenobjekte aus unstrukturierten Daten, wie z. B. formatierten Text, Bildern oder anderen Dateianhängen. Die kompletten Daten, ob unstrukturiert oder strukturiert müssen persistent gespeichert werden. Um zu gewährleisten, dass geeignete Ansichten und Navigationsmöglichkeiten für die Benutzeroberfläche erzeugt werden können, ist darauf zu achten, dass auf die Daten in geeigneter Weise zugegriffen werden kann. Eine weitere Anforderung an die Datenhaltung ist, dass die Daten durchsuchbar seien sollten. Dies sollte explizit für die Daten gelten die in strukturierter Form vorliegen. Optimalerweise lassen sich auch die Teile der unstrukturierten Informationen durchsuchen, die maschinenlesbaren Text beinhalten.

Die Datenhaltung muss weiterhin sicherstellen, dass alle Beziehungen die zwischen den Datenobjekten bestehen auch mit abgespeichert werden und entsprechend rekonstruiert werden können. Dies betrifft insbesondere bestehende Hierarchien zwischen Datenobjekten. In bestimmten kollaborativen Applikationen besteht wie in 3.1 beschrieben die Möglichkeit mehrere Versionen eines Datenobjektes zu erstellen. Dafür sollte die Datenhaltung Funktionen bereitstellen.

Verteilte Datenhaltung

Bei asynchronen kollaborativen Applikationen arbeiten mehrere Nutzer auf einem gemeinsamen Datenbestand. Auf diesen sollten die Nutzer jederzeit Zugriff haben, um eine möglichst große Nutzbarkeit der Anwendung zu gewährleisten. Eine rein zentrale Datenhaltung auf einem Server[26] wäre von Nachteil, da die Nutzer für den Zugriff eine Verbindung zum Server bräuchten und somit bei fehlender Verbindung kein Zugriff auf die Daten möglich wäre. Bei einer verteilten Datenhaltung[27] in der die Daten bei jedem Nutzer lokal vorliegen ist ein Zugriff jederzeit gewährleistet. Bei gleichzeitigem Zugriff mehrerer Nutzer auf dieselben Daten, muss eine Nebenläufigkeitskontrolle[28] umgesetzt werden, um die Daten konsistent zu halten.

3.2.3 Sicherheit

Da kollaborative Applikationen sensible Daten enthalten können, sollten nur autorisierte Benutzer auf die Anwendung zugreifen können. D. h. eine asynchrone kollaborative Applikation braucht einen Mechanismus zur Authentifizierung autorisierter Benutzer. Des Weiteren sollte bei der Speicherung darauf geachtet werden, dass die Daten nicht in Klartext gespeichert werden und somit von jeder Person - auch nicht autorisierten – gelesen werden können. Auf der Datenobjektebene dürfen nicht alle Benutzer, die Zugriff auf die Anwendung haben auch auf alle Datenobjekte zugreifen. Dafür sollten asynchrone kollaborative Anwendungen Mechanismen liefern, die die Steuerung des Zugriffs von Benutzern auf Datenobjektebene steuern. Oftmals müssen in kollaborativen Applikationen von Nutzern die Zugriffrechte für Datenobjekte geändert werden. Hierfür sollten auch unterstützende Funktionalitäten geliefert werden.

Um eine möglichst fein einstellbare Verwaltung der Zugriffsrechte in asynchronen kollaborativen Applikationen zu erzielen, sollten nicht nur Rechte an Benutzer zugewiesen werden können, sondern auch den in 3.1 beschriebenen Benutzergruppen und Benutzertypen.

---

[26] Für eine kurze Einführung in das Client/Server Modell sei auf *Koch und Schlichter* 2001 S. 119f verwiesen.

[27] Für eine genauere Diskussion des Begriffes Verteilte Datenhaltung sei auf *Koch und Schlichter* 2001 S. 121ff verwiesen.

[28] Vgl. *Koch und Schlichter* 2001 S. 122

3.2.4 Mehrbenutzersystem

Da es sich bei kollaborativen Anwendungen um Mehrbenutzersysteme handelt, ist eine Benutzerverwaltung für sie zwingend erforderlich. Die Benutzerverwaltung sollte die verschiedenen Benutzer und ihre Daten verwalten können und im Zusammenspiel mit der Zugriffsrechteverwaltung auch die Möglichkeit einer Authentifizierung einzelner Benutzer bieten. Die Merkmale „Benutzergruppen“ und „Benutzertypen“ müssen auch in der Verwaltung der Benutzer berücksichtigt werden. Benutzergruppen sollten global verfügbar sein und Benutzertypen sollten für jede Anwendung definierbar sein. Das bedeutet, dass Benutzergruppen für statisch feststehende Summierungen von Benutzern stehen, wie z. B. organisatorische Einheiten in einem Unternehmen. Daneben sind Benutzertypen für sich dynamisch ändernde Strukturen geeignet.[29] Über sie können z. B. aus Benutzergruppen wiederum einzelne Benutzer für eine bestimmte Anwendung zu einem Typ von Benutzern zusammengefasst werden.

Da bei asynchronen Anwendungen die Zugriffe nur bedingt asynchron stattfinden, muss für den Fall eines synchronen Zugriffs mehrerer Benutzer auf das gleiche Datenobjekt eine Regelung gefunden werden[30]. Für Datenobjekte, die nur von einem Benutzer gleichzeitig bearbeitet werden dürfen, muss sichergestellt werden, dass auch nur ein Benutzer das Datenobjekt gleichzeitig zum Bearbeiten geöffnet haben darf.

3.2.5 Zusatzfunktionalität

Neben der reinen asynchronen kollaborativen Applikation, sollten auch kommunikationsorientierte Komponenten die Benutzer in der Koordination ihrer Arbeit unterstützen[31]. Somit können auch synchrone Kommunikationsanwendungen wie Instant Messaging oder aber auch asynchrone wie EMail eine sinnvolle Ergänzung der reinen asynchronen kollaborativen Applikation sein. Diese Komponenten könnten nicht nur zur Kommunikation zwischen Benutzern genutzt werden, sondern auch um Nachrichten von dem System an den Benutzer zu senden.

Die gemeinsame Arbeit der Benutzer kann auch durch die Wahrnehmbarkeit der Aktivitäten anderer Benutzer erleichtert werden. Sie unterstützt die Bildung eines gemeinsamen Arbeitskontextes und wird in der Literatur als Awarness behandelt[32]. Zu

---

[29] Vgl. mit Arbeitsgruppen und Rollen in *Fischer et al.* S. 298 ff.
[30] Vgl. *Guerrero und Fuller* 2001, S. 463
[31] Vgl. *Appelt et al.* 2001, S.194
[32] Vgl. *Appelt et al.* 2001, S.195

normalen Datenbankanwendungen ist die Wahrnehmbarkeit der Aktivitäten anderer Nutzer ein wesentlicher Unterschied kollaborativer Applikationen[33].

3.3 Basisplattformen

Als mögliche Basisplattformen wurden der IBM Workplace Managed Client und die Eclipse Rich Client Platform betrachtet. Da der IBM WMC wie in 2.2.2 beschrieben auf der Eclipse RCP aufsetzt, bieten beide eine ähnliche Basis an grafischen Komponenten um eine Rich Client Anwendung zu entwickeln. Der Unterschied zwischen beiden in diesem Punkt ist die Erweiterung der grafischen Komponenten durch den IBM WMC. Ein Vorteil bei der Entwicklung der Basisarchitektur aufbauend auf der Eclipse RCP ist eine komplett auf Open - Source Software basierende Architektur. Der IBM WMC bietet bereits Funktionalitäten wie den lokalen verschlüsselten Datenspeicher, eine Authentifizierung von Nutzern und einen Richtext Editor. Dies bringt den wesentlichen Vorteil, dass diese Funktionalitäten in die Basisarchitektur intrigiert werden können. Daneben ergänzen die im IBM WMC bereits bestehenden kollaborativen Anwendungen, wie EMail und Instant Messaging, die zu entwickelnden asynchronen kollaborativen Applikationen - wie in 3.2.5 beschrieben - mit Koordinationsfunktionalitäten.

4 Basisarchitektur für asynchrone kollaborative Applikationen auf Basis des IBM WMC

4.1 Aufbau der Basisarchitektur

In den folgenden Kapiteln wird der Aufbau der Basisarchitektur beschrieben. Dabei wird auf verwendete Konzepte eingegangen und versucht deutlich zu machen wie die Anforderungen aus 3.2 umgesetzt wurden.

4.1.1 Gewählte Plattform

Da im Rahmen einer Seminararbeit die Entwicklung einer kompletten Basisarchitektur für jegliche asynchronen kollaborativen Applikationen schwer umzusetzen ist, fiel die Wahl für die Basisplattform auf den IBM WMC. Dieser bietet wie in 3.3 beschrieben nutzbare Funktionalitäten, die in die Basisarchitektur intrigiert werden können, und somit nicht selbst entworfen werden müssen. Im Wesentlichen wird von dem IBM WMC der lokale verschlüsselte Datenspeicher, der Richtext Editor und die Benutzerauthentifizierung verwendet. Bei dem Entwurf der Basisarchitektur wurde darauf

[33] Vgl. *Ellis et al.*1991, S. 40

geachtet, dass die genutzten Komponenten möglichst lose gekoppelt in der Basisarchitektur intrigiert werden und somit auch bei einer Weiterentwicklung für die Eclipse RCP leicht ersetzt werden können.

4.1.2 Daten

Anfangs musste eine Form gefunden werden, wie die Datenobjekte im System repräsentiert werden sollen. In Anlehnung an die Geschäftswelt, werden die Datenobjekte Dokumente genannt. Die Dokumente bestehen aus Standart-Eigenschaften, wie das Erstelldatum und der letzte Bearbeiter, die direkt im Dokument gespeichert werden, sowie aus Attributen die zu dem Dokument hinzugefügt werden können. Der nachfolgende Auszug aus dem UML[34] Klassen Diagramm zeigt den Zusammenhang zwischen Dokumenten, Attributen und Nutzern.

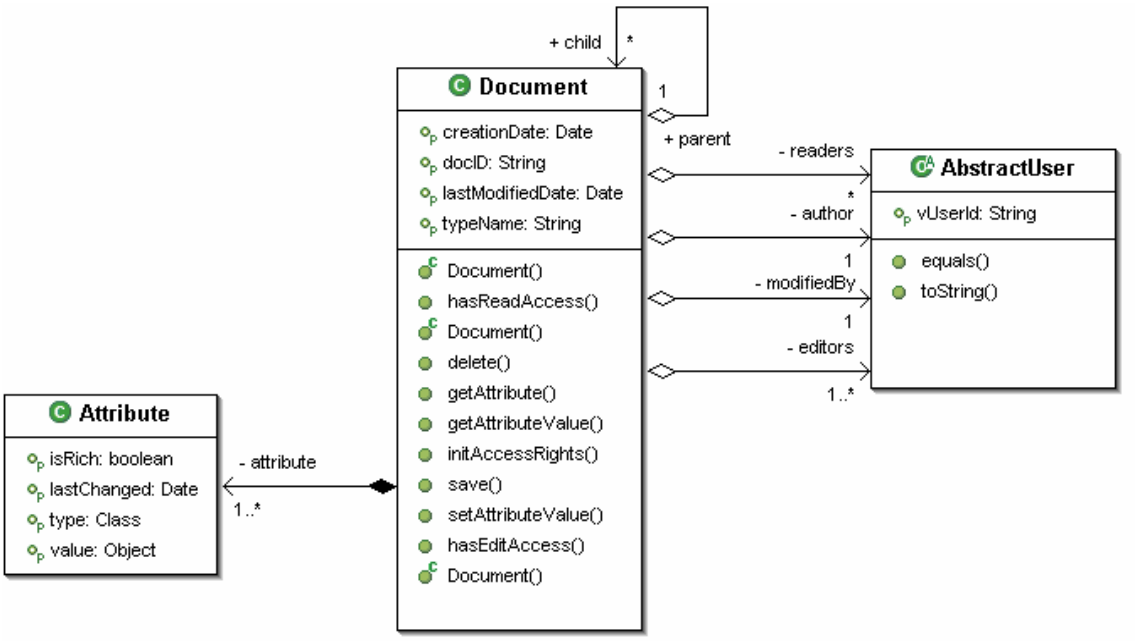

Abbildung 3: Klassendiagramm Dokument

Attribute sind fest an ihre Dokumente gebunden und somit kann auch nur über die entsprechenden Dokumente auf die ihm zugehörigen Attribute zugegriffen werden. Im Dokument werden die Attribute über einen eindeutigen Namen verwaltet, über den der Wert eines Attributes gesetzt oder abgefragt werden kann. Jedes Attribut hat einen Typ der als Java Klasse angegeben wird und auch dazu genutzt wird um den Wert des Attributes auf Typ - Korrektheit zu überprüfen.

---

[34] UML steht für Unified Modeling Language, eine standardisierte Beschreibungssprache für Strukturen und Abläufe in objektorientierten Programmsystemen. Vgl. *Wikipedia* 2005

Eine Hierarchie zwischen Dokumenten ist über die Vater-Kind-Beziehung möglich (siehe Abbildung 3). Hierbei kann ein Dokument mehrere Kinder haben, aber jedes Kind nur einen Vater.

Datenspeicherung

Um die Umsetzung der Datenspeicherung von der restlichen Basisarchitektur zu entkoppeln wurde das abstrakte Fabrik (Abstract Factory) Entwurfsmuster verwendet. [35] Die Repository Klasse, die für die Verwaltung der Dokumente verantwortlich ist, arbeitet nur mit der abstrakten Definition eines DataMangers, der von der AbstractDataManagerFactory erzeugt wird. Die jetzige Implementierung der Basisarchitektur nutzt zur Speicherung der Dokumente die lokale relationale Datenbank des IBM WMC, auf die per JDBC[36] und SDO[37] zugegriffen wird.[38] Da die Repository Klasse aber nur mit dem abstrakten Konstrukt des DataManagers arbeitet, kann leicht eine weitere Art der Speicherung der Dokumente implementiert werden, ohne Änderungen am Rest der Basisarchitektur vornehmen zu müssen.

Da relationale Datenbanken ihre Daten in Tabellen mit darauf definierten Referenzbeziehungen und Integritätsbedingungen abbilden[39], müssen die Dokumente mit ihren Attributen passend in dieses auf Tabellen basierende Schema abgelegt werden. Abhängig von der entwickelnden Anwendung sind die Attribute der Dokumente unterschiedlich. Somit kann ein solches Schema nicht fix für die Basisarchitektur definiert werden. D. h. für jede kollaborative Anwendung, die mit der Basisarchitektur entwickelt wird, muss das Schema automatisch generiert werden. Dies wird über die Schnittstelle IDocumentFactory realisiert, die für jeden in einer Anwendung vorkommenden Dokumententyp definiert werden muss.

Ein Dokumententyp ist ein Dokument mit einer festen Zuordnung von Attributen. Ein Beispiel für einen Dokumententyp wäre ein Forumsbeitrag, der neben den Standarteigenschaften eines Dokumentes wie Autor noch die Attribute Thema, Kategorie und Beitragstext hat. Durch die Schnittstelle IDocumentFactory kann die Basisarchitektur von der auf ihr basierenden Anwendung die Dokumententypen und ihre Attribute erfragen. Diese Informationen nutzt die Basisarchitektur um für die Anwendung das

---

[35] Vgl. *Gamma et al.* 2002 S.107ff
[36] JDBC Java Database Connectivity. Für eine Erläuterung siehe http://java.sun.com/products/jdbc/.
[37] SDO Service Data Objects. Für genauere Informationen siehe *Beatty et al.* 2003.
[38] Siehe Klassendiagramm im Anhang unter 8.2.
[39] Für eine Einführung in die Prinzipien relationaler Datenbanken und der Anfragesprache SQL sei auf *Heuer und Saake* 2000 verwiesen.

Datenbankschema zu erzeugen. Dafür werden für jeden Dokumententyp 4 Tabellen angelegt. Die nachfolgende Grafik zeigt das Schema mit den 4 Tabellen eines Dokumententyps.

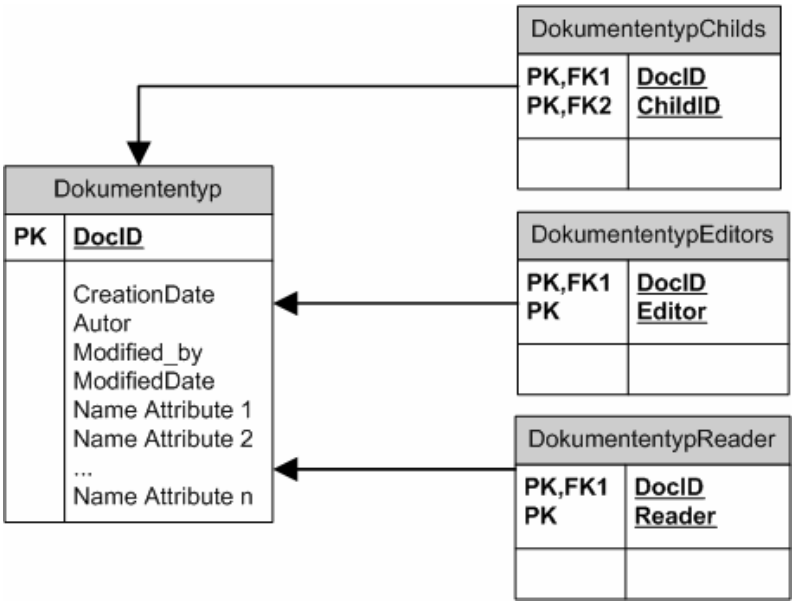

Abbildung 4: Tabellenschema eines Dokumententyps

Wie die Grafik zeigt, werden die Attribute eines Dokumententyps als eine Spalte in die Dokumententypentabelle abgelegt. Der Typ eines Attributes wird in der Basisarchitektur als Java Klasse definiert. Da in relationalen Datenbanken für Spalten ebenfalls ein Datentyp definiert wird und diese Datentypen nicht Java sondern SQL Typen[40] sind, ist eine Abbildung der Java Klassen auf SQL Typen notwendig. Hierzu wird von der Basisarchitektur für häufig vorkommende Java Klassen eine Zuordnung zu SQL Datentypen vorgenommen. Für spezielle Datentypen muss die nutzende Anwendung eine ergänzende Abbildung definieren.

Durch die gerade beschriebenen Verfahren basierend auf Dokumenten und Attributen wird versucht, die in 3.2.2 gestellten Anforderungen möglichst effektiv zu erfüllen. So ist es z. B. möglich, strukturierte als auch unstrukturierte Informationen abzuspeichern und trotzdem den Vorteil relationaler Datenbanken zu haben, direkt auf die Spalten der Tabellen zugreifen zu können. Dadurch besteht auch die Möglichkeit eine Suche über Attribute direkt über relationale Anfragen zu realisieren. In der jetzigen Implementierung besteht der Nachteil, dass nach der automatischen Generierung des Datenbankschemas für eine Anwendung, Änderungen der Attribute eines Dokumentes zu Fehlern führen, wenn

[40] Eine Übersicht über SQL Typen kann unter *IBMa* 2004 S. 145ff gefunden werden.

nicht manuell das Datenbankschema angepasst wird. Somit sollte bei der Entwicklung einer kollaborativen Anwendung mit der Basisarchitektur darauf geachtet werden, dass die Dokumententypen im Vorhinein sorgfältig definiert werden.

4.1.3 Synchronisation

Die Datenspeicherung basiert im Wesentlichen auf die im IBM WMC intrigierte lokale relationale Datenbank. Um die lokalen Datenspeicher der verschiedenen Clients zu synchronisieren, wird eine zentrale relationale Cloudscape[41] Datenbank verwendet. Die folgende Grafik gibt einen Überblick über die verwendete Architektur.

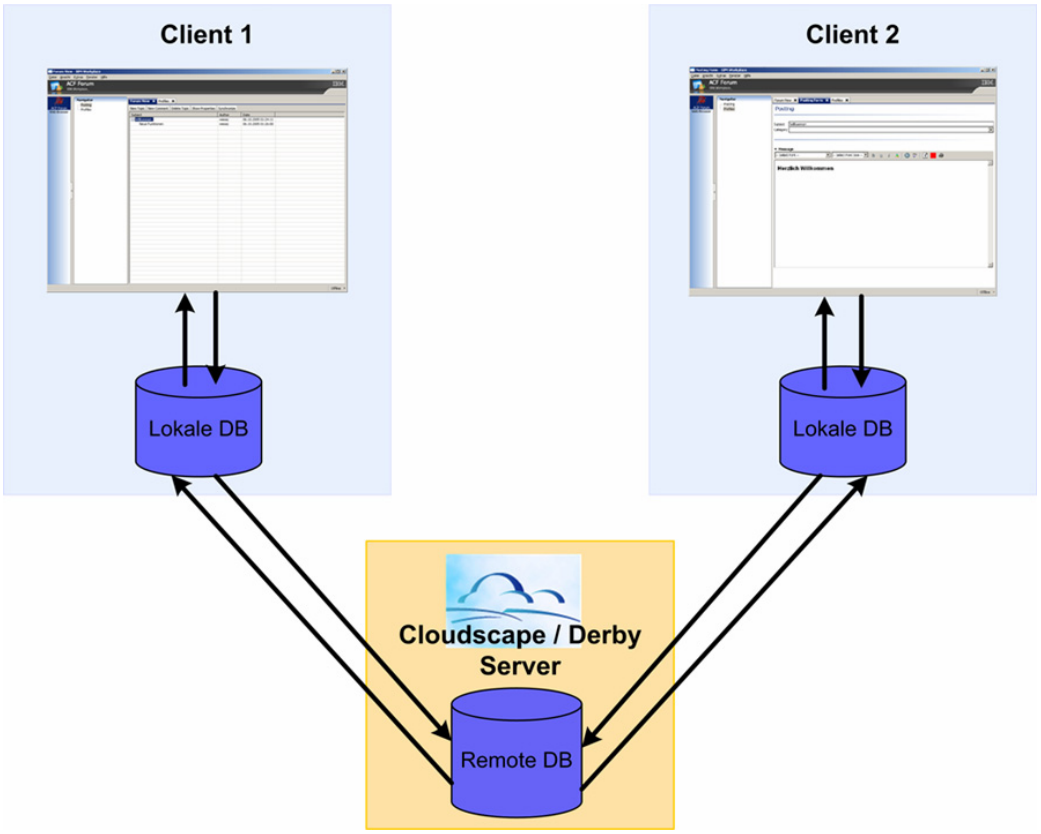

Abbildung 5: Synchronisation

Wie die Grafik zeigt, arbeiten die Clients zunächst auf ihrer lokalen Datenbank. Die zentrale Datenbank dient nur zur Synchronisation der Clients. Um keine eigene zentrale Serverkomponente zu benötigen, wird die eigentliche Synchronisation lokal von der Basisarchitektur übernommen. Hierzu hat jede lokale Datenbank und die zentrale Datenbank eine Aktionen - Tabelle, in der alle Änderungen der Datensätze als Log - Eintrag gespeichert werden. Um die Daten zwischen einer lokalen Datenbank und der zentralen Datenbank abzugleichen, fragt die Basisarchitektur aus beiden Datenbanken die

[41] Vgl. *IBMa* 2004

Log – Einträge aus den Aktionen – Tabellen ab. Diese Log – Einträge werden auf Konflikte überprüft, die - falls vorhanden - bereinigt werden. Die Konfliktbereinigung ist standardmäßig so implementiert, dass der neuere Eintrag übernommen und der ältere verworfen wird. Sind die Log – Einträge bereinigt, werden die darin enthaltenen Informationen genutzt um entweder Daten aus der zentralen Datenbank in die lokale zu schreiben oder umgekehrt.

4.1.4 Sicherheit

Die Basisarchitektur unterstützt eine Zugriffskontrolle auf drei Ebenen. Für die erste Ebene, der Authentifizierung von Benutzern, nutzt die Basisarchitektur die Authentifizierung des IBM WMC. Wird eine kollaborative Anwendung gestartet, wird ein Session - Objekt angelegt, das den aktuell am IBM WMC angemeldeten Benutzer ausliest und speichert. Die Anwendung kann nun über das Session Objekt die Benutzer ID des aktuell authentifizierten Nutzers abfragen.

Die zweite Ebene ist die Zugriffskontrolle auf Anwendungsebene. Hierfür wird eine Zugriffskontrollliste verwendet, in der alle Benutzer aufgelistet sind, die Zugriff auf die Anwendung haben. Für jeden Nutzer kann hier eine der folgenden Berechtigungsebenen festgelegt werden: Leser, Autor, Editor, Manager. Je nach Ebene hat der Benutzer unterschiedliche Berechtigungen. Ist der aktuell am IBM WMC angemeldete Benutzer nicht in der Zugriffskontrollliste aufgeführt, wird die Anwendung mit einen Hinweis an den Benutzer beendet.

Die dritte Ebene ist die Zugriffssteuerung auf Dokumentenebene. Auf dieser Ebene wird für jedes Dokument gesteuert welcher Nutzer Lese- oder Schreibberechtigung für ein Dokument hat. Hierbei spielen zum einen die oben genannten Berechtigungsebenen und zum anderen die Leser- und Bearbeiterliste eines Dokumentes eine Rolle. Die Standartimplementierung in der Basisarchitektur behandelt die Leseberechtigung wie folgt: Sobald mindestens ein Benutzer in der Leserliste aufgeführt wird, haben nur die Nutzer Lesezugriff, die auch in dieser stehen. Wenn kein Benutzer in der Leserliste aufgeführt ist, haben alle Benutzer die zumindest die Berechtigungsebene Leser haben Lesezugriff. Bei der Schreibberechtigung gilt, dass alle Benutzer die Editor und Manager sind immer Editor Rechte besitzen, ansonsten haben nur die Benutzer Schreibberechtigung wenn sie in der Bearbeiterliste aufgeführt werden.

Wie in 4.1.2 beschrieben, wird für die Datenspeicherung die lokale Datenbank des IBM Workplace Managed Clients genutzt. Ein Vorteil der sich durch die Wahl dieser

Speicherungsart für die Sicherheitsarchitektur ergibt, ist die Möglichkeit die Datenbank zu verschlüsseln, um somit die darin enthaltenen Daten vor unberechtigtem Zugriff zu schützen.[42]

4.1.5 Benutzeroberfläche

Die Basisarchitektur unterstützt den Entwickler asynchroner kollaborativer Anwendungen bei der Erstellung der Benutzeroberfläche. Sie bietet schon Standartelemente auf Basis der User Interface (UI) Komponenten des IBM WMC und der Eclipse RCP, die von den Anwendungen unverändert verwendet oder für eigene Zwecke erweitert werden können. Um Dokumente anzeigen oder Bearbeiten zu können wird eine Maske erzeugt, in dem für jedes Attribut eines Dokumentes ein UI Element auf der Maske generiert wird, das passend für den Typ des Attributes ist. Für Attribute die Richtext enthalten können, wird der Richtext Editor des IBM WMC als UI Komponente gewählt. Um eine leicht adaptierbare Maske zu erstellen, muss der Entwickler die Klasse DocumentForm[43] erweitern und speziell vorbereitete Methoden überschreiben.

Eine weitere Standartkomponente der Basisarchitektur ist eine Ansicht, in der Dokumente hierarchisch angezeigt werden können.[44] Über diese Ansicht kann innerhalb der darin enthaltenen Dokumente navigiert werden und entsprechende Aktionen wie Öffnen oder Löschen der Dokumente ausgeführt werden. Weitere nutzbare UI Komponenten der Basisarchitektur sind Elemente zum Editieren der Leser- und Bearbeiterliste.

4.2 Beispielhafte Anwendung der Basisarchitektur bei der Umsetzung eines Forums

In diesem Kapitel sollen kurz die wesentlichen Schritte erläutert werden, die notwendig sind um eine asynchrone kollaborative Anwendung mit der Basisarchitektur zu entwickeln. Dabei wird beispielhaft die Umsetzung eines einfachen Forums (siehe 3.1) mit der Basisarchitektur beschrieben.

Zunächst müssen, wie in 4.1.2 beschrieben, die Dokumententypen definiert werden. Für ein einfaches Forum werden nur Beiträge benötigt, die neben den Standarteigenschaften eines Dokumentes noch die Attribute Thema, Kategorie und Nachricht (Beitragtext)

---

[42] Vgl. *IBM* 2005 S. 144
[43] Siehe Klassendiagramm im Anhang unter 8.2.
[44] Beispielgrafiken für die UI Komponenten können im Anhang unter 8.3 gefunden werden.

beinhalten. Um den Dokumententyp Beitrag zu definieren muss die Anwendung das Interface IDocumentFactory implementieren.

Damit das Framework die Dokumententypen kennt und Einstellungen wie den Namen der zu erzeugenden Datenbank erfragen kann, muss die Anwendung das Interface IDeployConfig implementieren. Beim ersten Start der Forumsanwendung nutzt die Basisarchitektur diese IDeployConfig Implementierung und die darin enthaltene Dokumentendefinitionen um die Datenbank inklusive Datenbankschema für das Forum zu erzeugen. Um eine Ansicht für die Beiträge zu erstellen, muss die abstrakte AbstractHierarchicalDocumentView Klasse erweitert und die darin enthaltenen abstrakten Methoden implementiert werden. Zu der so erstellten Ansicht muss die Klasse DocLabelProvider erweitert werden, die für das Anzeigen der Spalteninhalte verantwortlich ist. Das folgende Klassendiagramm zeigt die gerade beschriebenen Zusammenhänge.

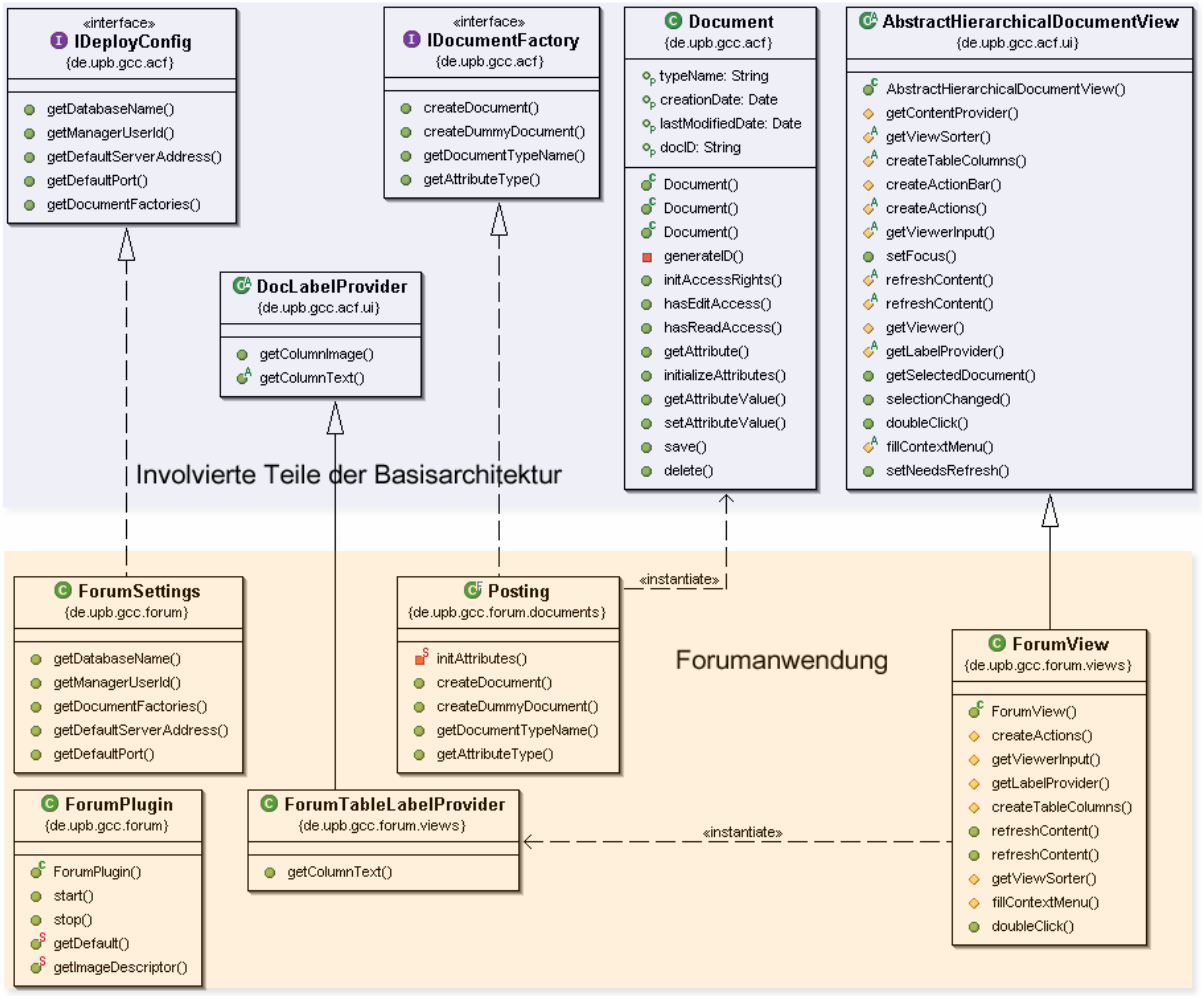

Abbildung 6: Klassendiagramm Forum und Basisarchitektur

Die oben beschriebenen Schritte sind im Wesentlichen die Schritte, die ein Entwickler vorzunehmen hat, wenn er mit der Basisarchitektur eine kollaborative Anwendung entwickeln möchte, die ein Dokumententyp beinhaltet. Für Anwendungen mit mehr als einem Dokumententypen und unterschiedlichen Ansichten sowie individuellen Masken zur Bearbeitung der Dokumente müssen die Schritte wiederholt ausgeführt werden und an einigen Stellen durch weitere Aktionen ergänzt werden.

5 Ausblick

Wie schon in der Einleitung erwähnt, ist es im Rahmen einer Seminararbeit schwer möglich, eine komplett ausgereifte Basisarchitektur zu entwickeln. Somit sollen in diesem Kapitel die möglichen Erweiterungen und Verbesserungen vorgestellt werden.

In den Anforderungen an die Datenspeicherung (3.2.2) wurde die Möglichkeit zur Speicherung mehrerer Versionen eines Datenobjektes gefordert. Für diese gewünschte Funktionalität hält die bisherige Basisarchitektur keine Mechanismen bereit. Hierzu müsste das Datenschema der Architektur um die Möglichkeit der Speicherung mehrerer Versionen eines Dokumentes erweitert werden.

Eine zusätzliche sinnvolle Erweiterung der Basisarchitektur wäre die Integration von Suchmechanismen über die Dokumente. Hier bietet die Datenspeicherung bereits durch das Konzept der Ablage einzelner Attribute die Möglichkeit gezielt nach speziellen Inhalten in Attributen zu suchen. Dafür müsste die bisherige Datenanfragefunktionalität um Funktionen ergänzt werden, die gezielte Anfragen nach Attributen erlauben. Durch die verbesserten Anfragemöglichkeiten wären so umfangreiche Suchmechanismen umsetzbar.

Zusätzlich zu den bereits bestehenden UI Komponenten der Basisarchitektur könnten noch weitere UI Komponenten entwickelt werden, um den Entwickler einen schnelleren und leichteren Entwurf der Benutzeroberfläche zu ermöglichen. Denkbar wäre, neben der hierarchischen Ansicht, eine weitere kategorisierbare Ansicht. Darüber hinaus wäre eine erweiterte Adaptierbarkeit der bestehende UI Komponenten hilfreich, um ein komplexeres UI Design zu ermöglichen. Das genutzte Richtext Element des IBM WMC bietet keine Unterstützung für Dateianhänge. Somit wäre die Entwicklung eines grafischen Elementes, das die Aufnahme von Dateien erlaubt eine mögliche Erweiterung der Basisarchitektur. Damit die Datenspeicherung der Basisarchitektur Dateien behandeln kann muss die in 4.1.2 angesprochene Abbildung von Java Typen nach SQL Typen um einen Dateityp erweitert werden.

Die Synchronisation der Basisarchitektur unterstützt bis jetzt nur eine recht einfache Konfliktbeseitigung. Diese vergleicht nur das Datum der letzten Änderung und übernimmt das neueste Element. Hier wäre z. B. ein Verfahren wünschenswert, das die genaue Ursache des Konfliktes erkennt und gegebenenfalls eine automatische Mischung der Daten der beiden im Konflikt stehenden Dokumente vornimmt.

6 Fazit

Wie das Kapitel 4.2 zeigt, ist es im Rahmen dieser Seminararbeit gelungen eine Basisarchitektur zu entwerfen, die es Entwicklern erleichtert asynchrone kollaborative Anwendungen zu entwickeln. Dabei lag der Focus, aufbauend auf den in Kapitel 3.2 beschriebenen Anforderungen darin, ein möglichst flexibles und sicheres Datenmodell zu entwickeln. Dieses wird, über Mechanismen zur Synchronisation, Ansprüchen einer verteilten Datenhaltung gerecht. Des Weiteren wurden bei der Entwicklung der Basisarchitektur mehrstufige Sicherheitsmechanismen umgesetzt. Sie ermöglichen eine Zugriffssteuerung auf Anwendungs- und Dokumentenebene. Zusätzlich unterstützen schon vorgefertigte UI Komponenten der Basisarchitektur den Anwender bei der Entwicklung seiner kollaborativen Anwendung.

7 Literatur

Appelt, W.; Busbach, U.; Koch, T. (2001): Kollaborationsorientierte asynchrone Werkzeuge. in: CSCW - Kompendium. Hrsg.: Schwabe, G.; Streitz, N.; Unland, R., Springer, Berlin u. a., 2001, S. 194 – 203

Beatty, John; Brodsky,S.; Ellersick,R.; Nally,M.; Patel,R. (2003): Service Data Objects Version 1.0; IBM Corp. und BEA Systems, Inc.; 2003; aus: ftp://www6.software.ibm.com/software/developer/library/j-commonj-sdowmt/Commonj-SDO-Specification-v1.0.pdf (am 10.09.2005)

Bishop, J. (2001): Java lernen. 2 Auflage, Addison Wesley Verlag, München u. a.

Borghoff, U. M.; Schlichter, J. H. (1998): Rechnergestützte Gruppenarbeit. Springer, Berlin, 1998

Constanzo, C.L.; Littlejohn I. (2003): The integration of collaborative technologies. aus: ABInsight November 2003: http://www-03.ibm.com/ibm/palisades/abinsight/issues/2003-Nov/article-2-print.pdf (am 21.05.2005)

Ellis, C.A.; Gibbs, S. J.; Rein, G.L. (1991): Groupware: some issues and experiences. in: Communications of the ACM, Vol. 34 (1991) Nr. 1, S. 39 – 58

Fischer, J.; Herold, W.; Dangelmaier, W.; Nastansky, L.; Suhl, L. (2000): Bausteine der Wirtschaftsinformatik. 2. Aufl., Erich Schmidt Verlag, Berlin 2000.

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. (2002): Entwurfsmuster: Elemente wieder verwendbarer objektorientierter Software. 1 Auflage, 5. korrigierter Nachdruck, Addison Wesley Verlag, München u. a.

Gruber, O.; Hargrave, B.J.; MCAffer, J.; Rapicault, P.; Watson, T. (2005): The Eclipse 3.0 platform: Adopting OSGi technology. in: IBM Systems Journal, Vol. 44, Nr. 2, S. 289 – 299

Guerrero, L.A.; Fuller, D.A. (2001): A pattern system for development of collaborative applications. in: Information and Software Technology, Vol. 43 (2001), Nr. 7, S. 457 - 467

Heuer, A; Saake, G (2000): Datenbanken Konzepte und Sprachen. 2. Auflage, MITP-Verlag, Bonn, 2000

Johansen, R. (1988): Groupware: Computer Support for Business Teams. Free Press, New York, London, 1988

Koch, M.; Schlichter, J. (2001): Verteilung von Daten und Kommunikation. in: CSCW - Kompendium. Hrsg.: Schwabe, G.; Streitz, N.; Unland, R., Springer, Berlin u. a., 2001, S. 116 – 123

Ozzie, R.; O'Kelly, P. (2003): Communication, Collaboration & Technology: Back to the Future. Groove Networks, Inc.,
aus: http://www.groove.net/contact/b2f-download/Back_to_the_Future.pdf
(am 21. 05.2005)

Ohne Verfasser

IBM (2004): IBM Workplace Client Technology - Driving the next generation of network-centric computing. aus: ftp://ftp.software.ibm.com/software/lotus/pub/lotusweb/clienttech/IBMWorkplaceClientTechnology_StrategyWhitePaper_May2004.pdf (am 04.10.2005)

IBM (2005): IBM Workplace Client Technology API Toolkit Users Guide. aus: http://doc.notes.net/uafiles.nsf/docs/wcs251api/$File/lwpapi251_wct_ug.pdf
(am 19.09.2005)

IBM Lotus News (2005): Introducing IBM Workplace Collaboration Services - the product for everyday collaboration. aus: http://www-306.ibm.com/software/swnews/swnews.nsf/n/jmae65wjzf?OpenDocument&Site=lotus
(am 04.10.2005)

IBM Whitepaper (2005): IBM Workplace Client Technology Overview and Business Considerations. aus: ftp://ftp.lotus.com/pub/lotusweb/workplace/Client_Technology_Overview.pdf
(am 10.09.2005)

IBMa (2004): IBM Cloudscape Reference Manual Version 10.
aus: http://publib.boulder.ibm.com/epubs/pdf/c1892480.pdf (am 05.10.2005)

Wikipedia (2005): Unified Modeling Language.
aus: http://de.wikipedia.org/wiki/Unified_Modeling_Language (am 12.10.2005)

8 Anhang

8.1 Tabelle Merkmale kollaborativer Applikationen

Merkmale
gemeinsame Arbeitsbereiche
Datenobjekte
Metainformationen
Versionen
Hierarchien
Aktionen
Mehrbenutzersysteme
Benutzerdaten
Benutzergruppen
Benutzertypen
Zugriffsrechte
Mitteilungen des Systems

Tabelle 1: Merkmale kollaborativer Applikationen

8.2 Klassendiagramm für Teile der Pakete ACF, Data und UI

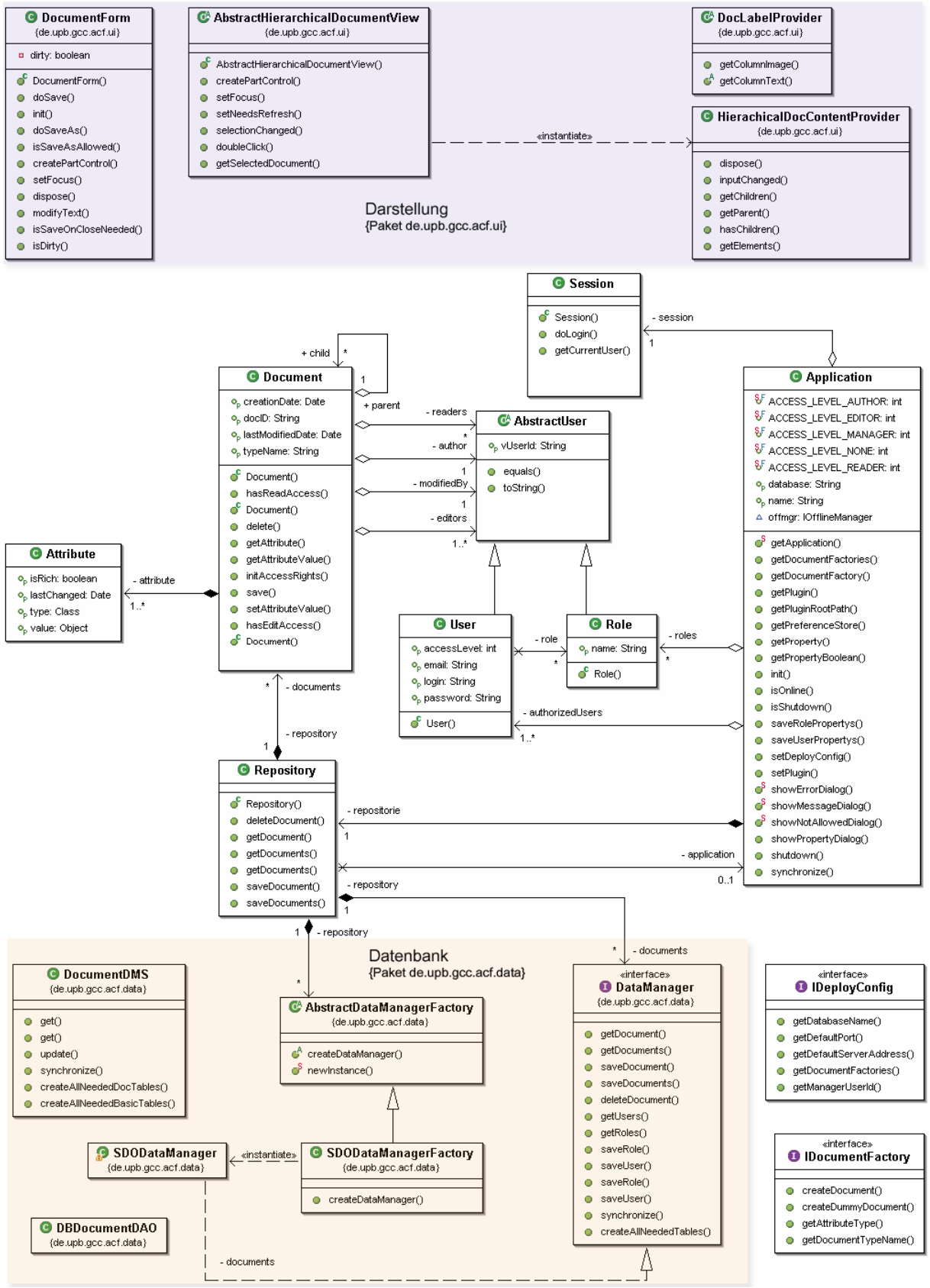

Abbildung 7: Teil des Klassendiagramms

8.3 Beispiele für Standart UI Komponenten der Basisarchitektur

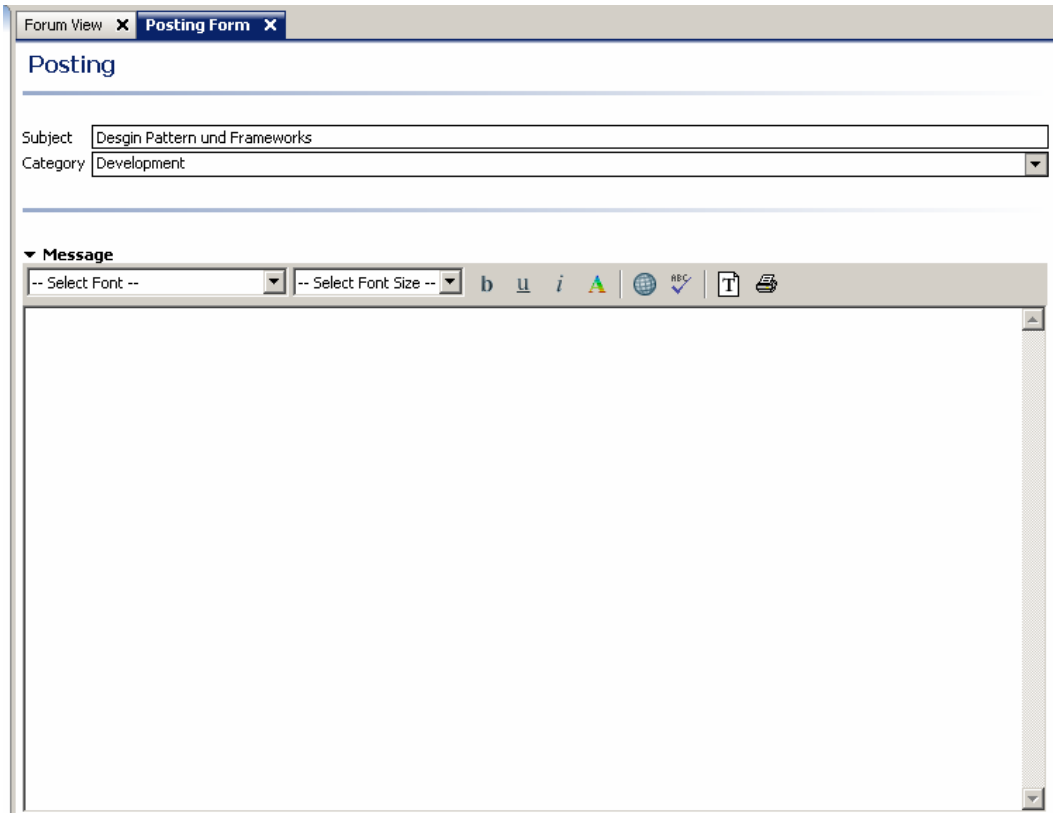

Abbildung 8: Maske für Dokument

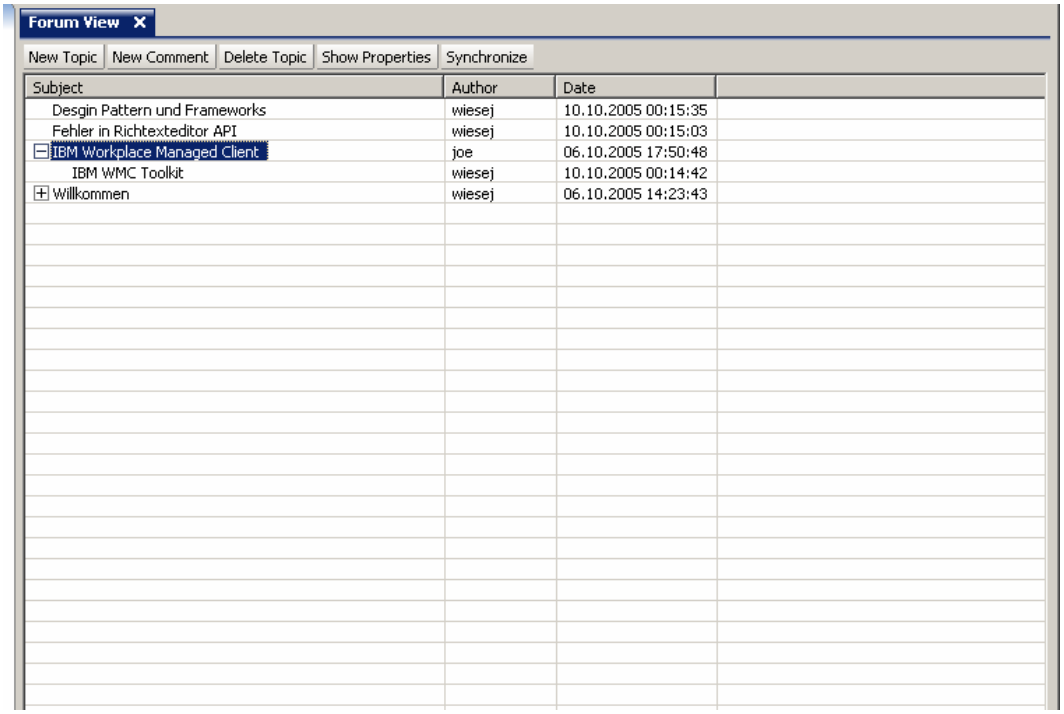

Abbildung 9: Hierarchische Ansicht