

Prüfer: Prof. Dr. Rothermel

Betreuerin: Dr. Cora Burger

Begonnen am: 15.9.1999

Beendet am: 14.3.2000

CR-Nummer: H.3.4,H.4.1,H.5.2

Studienarbeit Nr. 1764

**Einbau weiterer Desktop
Publishing Systeme in DDE
(Distributed Document Environment)**

Bo Wu

Inhaltsverzeichnis

1	Einleitung	4
1.1	Motivation	4
1.1.1	Bedarf	4
1.1.2	Infrastruktur	4
1.2	Beispielszenario	5
1.3	Aufgabenstellung	6
1.4	Überblick	9
2	Autoren Umgebung	11
2.1	Anforderungen	11
2.2	Textverarbeitungssysteme	12
2.2.1	Emacs	12
2.2.2	Microsoft Word	14
2.2.2.1	WordBasic	14
2.2.2.2	Microsoft Word-API	14
2.2.2.3	Bewertung	15
2.2.3	Pocket Word	16
3	Plattformaspekte	17
3.1	JAVA	17
3.2	CORBA	18
3.3	DDE	19
3.3.1	Anforderungen	19
3.3.2	Modell	21
3.3.3	DDE-Architektur	24
4	Analyse	26
4.1	Anforderungen für die Erweiterung der DDE	26
4.2	Erweiterungsmöglichkeiten von DDE	27
4.3	Ergebnis	31
5	Entwurf	33
5.1	Grobentwurf	33
5.2	Feinentwurf	35
5.2.1	Erweiterung der Autoren Umgebung	35
5.2.2	Erweiterung der Dokumenten-Plattform	36
5.2.2.1	Implementierungsaspekte der Dokumenten-Plattform	36
5.2.2.2	Entwurf des Wrapper-Servers	37
5.2.3	Kommunikation zwischen Pseudo-Wrapper und Wrapper-Server	38
5.3	Maßnahmen gegen Inkonsistenzen	39
5.3.1	Beispielszenario	39

5.3.2	Maßnahmen gegen Inkonsistenz in der ursprünglichen DDE.....	40
5.3.3	Maßnahmen gegen Inkonsistenz in der erweiterten DDE.....	41
6	Implementierung.....	42
6.1	Implementierung des Pseudo-Wrappers.....	43
6.1.1	Teilaufgaben.....	43
6.1.2	Der Startvorgang des Pseudo-Wrappers.....	44
6.1.3	Die Benutzungsoberfläche des Pseudo-Wrappers.....	45
6.1.4	Darstellung der Dokumentenstruktur.....	46
6.1.4.1	Dokumentenstuktur in DDE.....	46
6.1.4.2	Implementierung im Pseudo-Wrapper.....	46
6.1.4.3	Eine Alternative.....	47
6.1.4.4	Dateiverwaltung.....	48
6.1.5	Die Funktionen der Knöpfe.....	48
6.2	Implementierung des Wrapper-Servers.....	59
6.2.1	Implementierungsalternativen.....	59
6.2.2	Resultat.....	61
6.2.3	Wrapper-Server.....	62
7	Bewertung und Zusammenfassung.....	63
8	Ausblick.....	65
	Literaturverzeichnis.....	67

1 Einleitung

In diesem Kapitel werden zuerst die Motivationen für ein verteiltes Dokumenten-System kurz besprochen. Dann wird ein Beispielszenario gegeben. Nach der Beschreibung von Einschränkungen des bestehenden verteilten Dokumenten-Systems wird die Aufgabenstellung für die Erweiterung des Dokumenten-Systems vorgestellt. Zum Schluß wird ein Überblick über diese Studienarbeit gegeben.

1.1 Motivation

Die Zusammenarbeit zwischen Menschen in möglicherweise weltweit verteilten Teams sowie ihre Unterstützung durch Rechner (Computer Supported Cooperative Work, CSCW) gewinnen immer mehr an Bedeutung.

Eine Art von dieser Zusammenarbeit ist die gemeinsame Erstellung eines Dokuments von einem verteilten Team. Die Unterstützung solcher Zusammenarbeit durch Rechner ist durch folgende Aspekte motiviert.

1.1.1 Bedarf

- Einerseits sind der Umfang und die Komplexität von Aufgaben oft so groß, daß man sie allein nicht schaffen kann, andererseits sind die Kenntnisse von Menschen heutzutage immer spezifischer. Deswegen bilden Leuten häufig ein Team um gemeinsam ein Dokument herzustellen.
- Die Teammitglieder von einem Team sind oft geographisch weit getrennt, arbeiten in verschiedenen Plattformen, und benutzen möglicherweise unterschiedliche Texteditoren, um ein Dokument zu editieren.

1.1.2 Infrastruktur

- Heutzutage werden fast alle Dokumenten elektronisch erstellt und bearbeitet. Es gibt schon viele unterschiedliche Editoren, um die Erstellung und Bearbeitung von einzelnen Benutzern zu unterstützen.
- Dank der vernetzten Rechner, z.B. Internet, Intranet, und Middleware, z.B. CORBA, und darauf aufgebauter Groupware-Systeme, ist es möglich, ein verteiltes Team bei Koordination und Kooperation zu unterstützen.

1.2 Beispielszenario

Abbildung 1.1 zeigt ein Beispiel einer Teamarbeit, um gemeinsam Dokumente zu erstellen.

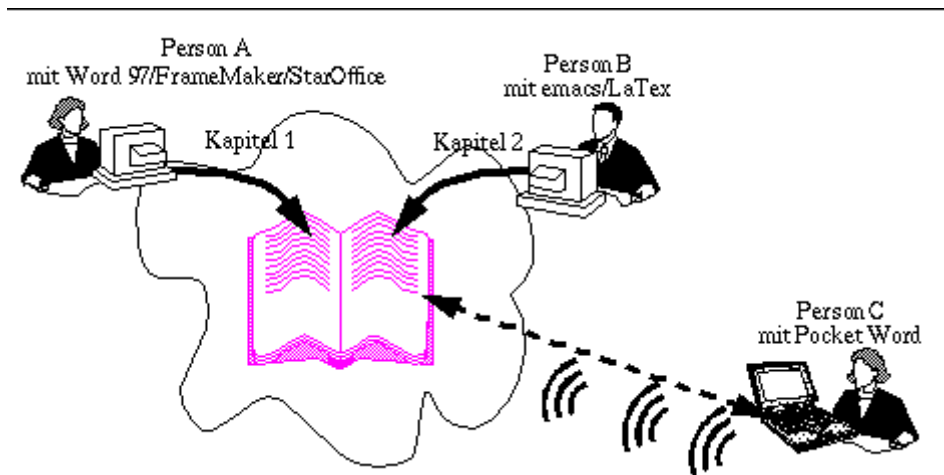


Abbildung 1.1: Gemeinsame Erstellung eines Dokuments

Person A, Person B und Person C bilden ein Team, um gemeinsam ein Dokument zu erstellen. Sie sind möglicherweise geographisch entfernt und spielen verschiedene Rollen in diesem kleinen Team.

Person A ist für das Kapitel 1 zuständig. Sie könnte mit Word 97, FrameMaker oder Staroffice arbeiten.

Person B ist für das Kapitel 2 zuständig. Er bevorzugt, Dokumente mit Emacs zu editieren.

Person C könnte die Leiterin für das Team sein. Sie erstellt selbst kein Dokument, aber sie muß die Arbeit von allen Teammitgliedern kontrollieren können, um den Überblick über das ganze Projekt zu haben. Weil sie oft unterwegs ist, besitzt sie ein Mobilgerät und benutzt Pocket Word als Texteditor.

Ein geeignetes verteiltes Dokumenten-System ist unentbehrlich für eine erfolgreiche Zusammenarbeit dieses Teams.

1.3 Aufgabenstellung

In einer bereits abgeschlossenen Arbeit wurde eine verteilte Dokumentenumgebung (DDE) auf Basis von CORBA entworfen und realisiert sowie über den Emacs im LaTeX-Mode für eine gemeinsame Dokumentenbearbeitung verfügbar gemacht. Im Rahmen dieser Studienarbeit werden die Erweiterungsmöglichkeiten von dieser Dokumentenumgebung untersucht. Die auf dieser Basis erstellten Lösungen sind prototypisch umzusetzen und zu bewerten.

Die bereits verfügbare verteilte Dokumentenumgebung hat einige Einschränkungen:

1. Bis jetzt können Benutzer nur über Emacs mit dieser DDE arbeiten. Es besteht ein Bedarf, weitere Textverarbeitungssysteme, wie z.B. Microsoft Word, Pocket Word, in diese verteilte Dokumentenumgebung zu integrieren.
2. Die Implementierung des Dokumenten-Systems ist auf einem ORB von ORBIX basierend. Da ORBIX keinen POA (Portable Object Adapter) unterstützt, der eine herstellerunabhängige Entwicklung von Servern ermöglicht, ist das Dokumenten-System von ORBIX abhängig.
3. Gleichzeitige Änderungen des gleichen Kapitels durch mehrere Benutzer sind in der ursprünglichen DDE von [Schr99] nicht möglich. In der erweiterten DDE von [Schu99] wird dies unterstützt, muss also in dieser Studienarbeit nicht mehr betrachtet werden.
4. Mit dem zunehmenden Einsatz von mobilen Endgeräten sind die Integrationsmöglichkeiten von mobilen Endgeräten zu betrachten.

Für mobile Endgeräten gibt es zusätzliche Besonderheiten:

- Die Ressourcen, z.B. Tastatur, Bildschirm, Speicher, CPU, usw. sind beschränkt.
- Die Verbindung mit anderen Rechnern besteht meistens über drahtloses Medium. Dies kann einige Probleme mit sich bringen, z.B. häufigere Unterbrechungen, geringere Bandbreite, leichter abhörbar, usw.

Für unsere verteilte Dokumentenumgebung gibt es daher besondere Einschränkungen bei mobilen Endgeräten:

- Nur bestimmte Betriebssysteme sind benutzbar, z.B. Window CE.

Window CE wird häufig bei Mobilgeräten eingesetzt, und die verfügbare Software unter Windows CE ist beschränkt.

- Nur bestimmte Texteditoren sind verfügbar, z.B. Pocket Word.

Weil es keine Möglichkeit gibt, solche Texteditoren zu erweitern, ist eine direkte Integration von solchen Texteditoren nicht möglich.

- Manche Middleware, z.B. CORBA, ist nicht verfügbar.

Man kann daher die Funktionen der corbabasierten DDE-Plattform nicht direkt benutzen.

- Keine permanente Verbindung zum Festnetz bzw. zur DDE-Plattform.

Weil das Mobilgerät keine permanente Verbindung zur DDE-Plattform aufweist, besitzen Benutzer von Mobilgeräten oft lokale Kopien von den Dokumentendaten. Nach der Bearbeitung müssen diese lokale Kopien wieder zur DDE-Plattform übertragen werden und die Konsistenz zwischen den lokalen Kopien und den Dokumentendaten in der DDE-Plattform muß wieder hergestellt werden.

In dieser Studienarbeit werden, mit der DDE als Basis, weitere Funktionalitäten hingefügt. Dabei werden die oben genannten Punkte berücksichtigt.

Es handelt sich hauptsächlich um zwei Dinge:

1. Die Benutzer von anderen Texteditoren zu unterstützen.
2. Die Benutzer von anderen Betriebssystemen zu unterstützen.

1.4Überblick

Im vorigen Abschnitt wurde die Aufgabenstellung von dieser Studienarbeit vorgestellt. Hier wird ein Überblick über die weitere Arbeit gegeben.

Die weitere Arbeit ist in folgende Kapitel untergliedert.

In Kapitel 2 wird die Autorenumgebung betrachtet. Hier werden die häufig benutzten Texteditoren, z.B. Emacs, Microsoft Word und Pocket Word, hinsichtlich des Einbaus in der DDE untersucht.

In Kapitel 3 werden die Plattformaspekte dieser Studienarbeit besprochen. Darunter werden die benutzte Programmiersprache „JAVA“, die zugrundliegende Middleware „CORBA“ und die bereits bestehende verteilte Dokumentenumgebung „DDE“ diskutiert.

In Kapitel 4 werden zuerst die Anforderungen für die Erweiterung der DDE besprochen. Dann werden die 4 Erweiterungsmöglichkeiten analysiert. Nach der Analyse wird eine Schlußfolgerung für die weitere Arbeit gezogen.

Anhand der Schlußfolgerung in Kapitel 4 wird ein Entwurf in Kapitel 5 hergestellt. Hier werden alle Entwurfsaspekte sowohl auf der Serverseite als auch auf der Klientseite betrachtet.

Die Probleme von Inkonsistenzen in diesem Dokumenten-System werden auch in diesem Kapitel betrachtet.

In Kapitel 6 wird die Implementierung von dieser Studienarbeit vorgestellt. Darunter werden sowohl die Implementierung von einem Pseudo-Wrapper als auch die Implementierung von einem Wrapper-Server beschrieben. Die Implementierungsalternativen werden auch kurz besprochen.

In Kapitel 7 wird diese Erweiterung der DDE bewertet, und eine Zusammenfassung der gesamten Arbeit wird gegeben.

Am Schluß wird ein Ausblick auf dieses Gebiet in Kapitel 8 gemacht.

2 Autorenumgebung

In diesem Kapitel wird die Autorenumgebung besprochen. Hier werden zuerst die Anforderungen an das Textverarbeitungssystem für den Einbau in der DDE besprochen. Dann werden die Erweiterungsmöglichkeiten von einigen häufig benutzten Textverarbeitungssystemen von verschiedenen Betriebssystemen hinsichtlich des Einbaus in der DDE diskutiert.

2.1 Anforderungen

Für den Einbau in der DDE stellen sich folgende Anforderungen an ein Textverarbeitungssystem:

- Erweiterung der Benutzungsoberfläche

Die Benutzungsoberfläche des Textverarbeitungssystems muß erweitert werden, um die Dokumentenstruktur der Dokumenten-Plattform darzustellen, zusätzliche Funktionen durchzuführen und verschiedene Informationen dem Benutzer anzuzeigen.

- Zusätzliche Funktionen

Bei dem erweiterten Textverarbeitungssystem werden die Zugriffe auf lokale Dateien und auf entfernte Dateien (Dokumenten auf der Dokumenten-Plattform) unterschieden. Außerdem müssen die Funktionen für die Zugriffe auf die entfernten Dateien neu implementiert werden.

- Kommunikation mit der Dokumenten-Plattform

Es muß eine Möglichkeit geben, von dem Textverarbeitungssystem auf die Funktionen der Dokumenten-Plattform zuzugreifen.

2.2 Textverarbeitungssysteme

2.2.1 Emacs

Emacs ist ein sehr bekannter Texteditor aus der Unix Welt.

Emacs ist ein offenes Text Editor System. Es besteht aus einer Ansammlung von Bibliotheken, die beliebig erweitert werden kann. Die Erweiterung von Emacs wird mit der Programmiersprache Emacs Lisp realisiert.

Emacs Lisp ist eine interpretative, funktionale Sprache.

Emacs Lisp besitzt dynamische Typen- und Variablenbindung. Da Emacs Lisp als Sprache für den Gebrauch in Text Editoren entworfen wurde, verfügt es über spezielle Funktionen zum Lesen und Erkennen von Text, sowie über spezielle Datenstrukturen, die ein einfaches und schnelles Bearbeiten von Text ermöglichen.

Für den Einbau von Emacs in DDE sind die Konzepte von Emacs wie Puffer, Fenster und Modus sehr wichtig.

- Puffer

Ein Puffer ist eine interne Datenstruktur in Emacs. Beim Öffnen einer Datei erzeugt Emacs einen Puffer, und lädt die Datei in den Puffer. Modifikationen werden dann immer an diesem Puffer vollzogen. Am Ende der Bearbeitung wird der Inhalt des Puffers in die Datei zurückgeschrieben. Jedem Puffer wird beim Erzeugen ein eindeutiger Name zugeordnet.

Emacs legt in einen Puffer Text ab und bietet Funktionen an, die eine einfache und schnelle Bearbeitung des Textes ermöglichen.

Der Text kann aus einer Datei oder auch über einen Unix Port gelesen werden. Das ermöglicht, daß ein eigenständiger, von Emacs unabhängiger Prozess, Text in einen Puffer schreiben und lesen kann. Mit einem derartigen zusätzlichen Prozeß kann Emacs Daten von der Dokumenten-Plattform lesen oder zu der Dokumenten-Plattform schreiben.

- Modus

Unter einem Modus versteht man eine Menge von Definitionen, die eine Bearbeitung von Text in einem Puffer in einer sehr komfortablen Art und Weise ermöglicht. Ein Modus wird durch eine Sammlung von Funktionen, die in Emacs Lisp geschrieben sind, realisiert. Eine Funktion kann direkt aufgerufen werden oder sie kann an eine Funktionstaste gebunden werden, so daß bei der

Verwendung der Funktionstaste die daran gebundene Funktion ausgeführt wird. Eine solche Bindung wird durch sogenannte `Keymaps` ermöglicht. Jeder Modus verfügt über eine eigene `Keymap`. Ein Modus kann zu beliebiger Zeit durch den Benutzer ein- und ausgeschaltet werden.

Es gibt zwei Arten von Modi:

Der Hauptmodus wird normalerweise beim Erzeugen eines Puffers diesem zugeordnet. Emacs verfügt standardmäßig über eine Reihe von Hauptmodi, wie z.B. den `latex-mode`. Zu einem Zeitpunkt kann ein Puffer nur über einen Hauptmodus verfügen.

Der Nebenmodus stellt eine Erweiterung des Hauptmodus dar. Er dient einem Benutzer, um individuelle Anpassungen am Hauptmodus durchführen zu können. Ein Puffer kann über beliebig viele Nebenmodi verfügen.

Die Modi sind untereinander priorisiert. Bei der Betätigung einer Funktionstaste werden zuerst alle Nebenmodi durchsucht, danach der Hauptmodus und zum Schluß die Grundeinstellung von Emacs. Durch die Verwendung eines Nebenmodus kann folglich die Funktionalität des Hauptmodus ausgeblendet werden.

- Fenster

Ein Fenster in Emacs ist ein Bereich auf dem Bildschirm, in dem ein Puffer dargestellt werden kann. Mehrere Fenster sind in einem Rahmen plaziert. Normalerweise existiert nur ein Rahmen und innerhalb dieses Rahmens ist immer nur ein Fenster aktiv. Dabei bedeutet aktiv, daß der Puffer, der in diesem Fenster dargestellt wird, potentiell modifiziert werden kann. Das API von Emacs verfügt über Funktionen, um Fenster beliebig erzeugen und löschen zu können.

Emacs erfüllt die Anforderungen für den Einbau in der DDE. Es ist möglich, die Benutzungsoberfläche von Emacs zu erweitern und zusätzliche Funktionen in Emacs hinzufügen. Mit einem zusätzlichen Prozeß kann Emacs auf die Funktionen der Dokumenten-Plattform zugreifen.

In der bereits bestehenden DDE wurde die Emacs Version GNU Emacs 20.2.1 so erweitert, daß man mit Emacs im LaTeX-Mode gemeinsame Dokumente bearbeiten kann.

2.2.2 Microsoft Word

Microsoft Word ist ein häufig genutzter Texteditor aus der PC Welt.

Es gibt folgende zwei Möglichkeiten, Microsoft Word zu erweitern.

- WordBasic
- Microsoft Word-API (Application Programming Interface)

2.2.2.1 WordBasic

WordBasic ist eine strukturierte Programmiersprache, die ursprünglich auf der Grundlage der Sprache Microsoft QuickBasic entworfen wurde. Sie vereint eine Untergruppe von Anweisungen aus standardmäßigen BASIC-Sprachen mit Anweisungen und Funktionen, die auf der Benutzungsoberfläche von Word basieren. Mit WordBasic können wir alle Word-Befehle modifizieren oder unsere eigenen Word-Befehle schreiben. Wir können unsere Makros einem Menü, einer Symbolleiste oder einem Shortcut zuordnen, so daß sie genau wie standardmäßige Word-Befehle erscheinen und funktionieren.

WordBasic stellt die gesamte Funktionalität von Word zur Verfügung und kann unter Windows Funktionen des Windows-API sogar direkt aufrufen.

Mit WordBasic kann man die Benutzungsoberfläche von Microsoft Word auf einfache Weise erweitern und neue Funktionen hinzufügen.

2.2.2.2 Microsoft Word-API

Die Microsoft Word-API (Application Programming Interface) ermöglicht den Zugang zur internen Funktionalität von Microsoft Word, Version 6.0x.

Die Word-API geht über die Möglichkeiten von WordBasic hinaus und ist noch flexibler und leistungsfähiger.

Das Programmieren mit der Word-API bietet eine Reihe von Vorteilen gegenüber WordBasic. Wir können schnellen, effizienten und flexiblen Code schreiben. Außerdem können wir bereits existierende Bibliotheken mit externem Code nutzen oder neuen Code mit einem beliebigen Compiler erstellen, sofern er das Erstellen von Codemodulen unterstützt.

Mit externem Code kann man die Kommunikation zwischen Microsoft Word und der Dokumenten-Plattform implementieren.

2.2.2.3Bewertung

Für den Einbau von Microsoft Word in unsere DDE ist eine Kombination von den oben vorgestellten Methoden geeignet.

- Mit Word Basic können wir einfach die Benutzungsoberfläche von Microsoft Word an unseren Bedarf anpassen, die neuen Arbeitsabläufe automatisieren und die benötigten Funktionen in Microsoft Word integrieren.
- Mit Word-API können wir die Funktionen, die mit der Dokumenten-Plattform kommunizieren, effizient implementieren. Diese Funktionen sind schwer oder gar nicht mit Word Basic zu implementieren.

Damit sind die Anforderungen von Abschnitt 2.1 erfüllt.

Microsoft Word läßt sich gut erweitern. Es ist aber zur Zeit nicht in einem Mobilgerät einsetzbar.

In einem Mobilgerät wird eine vereinfachte Version von Microsoft Word (Pocket Word) benutzt. Wir betrachten Pocket Word im nächsten Abschnitt.

2.2.3 Pocket Word

Pocket Word ist ein häufig genutzter Texteditor in einem Mobilgerät.

Er ist eine einfache Version von Microsoft Word in Window CE, und wird häufig beim Mobilgerät eingesetzt. Mit Pocket Word kann man in derselben Weise Dokumente (.pwd) oder Vorlagen (.pwt) erstellen und bearbeiten wie mit Microsoft Word auf einem Desktop-Computer. Man kann die Dokumente in anderen Dateiformaten speichern, z.B. .doc, .rtf oder .txt. Das Dateiformat .rtf ermöglicht, einen Großteil der Formatierung zu erhalten, während beim Dateiformat .txt die gesamte Formatierung entfernt wird.

Im Gegensatz zu Microsoft Word bietet Pocket Word keine Möglichkeit, um sie direkt zu erweitern. Daher gibt es nur folgende Möglichkeiten, die Benutzer von Mobilgeräten zu unterstützen.

- 1) Man besorgt Quellcode von Pocket Word von Microsoft, und erweitert den Editor wie gewünscht.
- 2) Man entwickelt selbst einen Texteditor für das Mobilgerät.
- 3) Man benutzt einen Pseudo-Wrapper (siehe Kapitel 4)

Die Möglichkeit 1) wäre die ideale Lösung für Endbenutzer, da in diesem Fall die Benutzer wie gewöhnlich mit Pocket Word arbeiten können. Sie brauchen nur zusätzliche Funktionen für die DDE zu lernen, um sich dem verteilten Dokumenten-System anzuschließen. Aber es steht kein Quellcode von Pocket Word zur Verfügung, deswegen ist die Implementierung mit diesem Lösungsansatz ausgeschlossen.

Bei der Möglichkeit 2) hat der Entwickler sowohl den Überblick als auch alle detaillierten Implementierungsaspekte von dem Texteditor. Dies erleichtert die Arbeit für die Integration von Texteditoren in DDE. Aber es gibt folgende Nachteile:

- Es ist aufwendig, einen neuen Texteditor zu entwickeln. Wir möchten uns nur auf das Modul, das für die Kommunikation mit DDE zuständig ist, konzentrieren. Ein eigenständiger Editor mit Funktionen wie Textverarbeitung, Textlayout, usw. würde den Umfang einer Studienarbeit sprengen.
- Die Endbenutzer müssen lernen, mit diesem neuen Texteditor umzugehen.

Die Möglichkeit 3) entspricht der Erweiterungsmöglichkeit 4 in Kapitel 4 und wird dort genauer diskutiert.

3 Plattformaspekte

In diesem Kapitel werden die wichtigen Aspekte der Umgebung für diese Studienarbeit besprochen. Insbesondere werden die benutzte Programmiersprache, die eingesetzte Middleware, die ursprüngliche DDE von [Schr99] und eine erweiterte Version von DDE von [Schu99] vorgestellt.

3.1 JAVA

Java ist eine von Sun entwickelte objektorientierte Programmiersprache, die mit Eigenschaften wie Portabilität, Sicherheit, Robustheit und Multithreading in kurzer Zeit eine weite Verbreitung gefunden hat.

DDE wurde mit Java implementiert. Wegen der Plattformunabhängigkeit von Java ist sie geeignet für diese Arbeit, um das Dokumenten-System so zu erweitern, daß Benutzer von verschiedenen Plattformen auf das Dokumenten-System zugreifen können.

Java verfügt über Funktionalität für die Netzwerkprogrammierung.

Das Java Package `java.net` bietet Klassen für die Implementierung von Netzerkanwendungen. Mit Socket Klassen kann man die Kommunikation mit anderen Servern im Netzwerk durchführen oder einen eigenen Server entwickeln. Es gibt auch eine Menge von Klassen, die die einfache Benutzung von Universal Resource Locators (URLs) ermöglichen.

Außer der direkten Kommunikation unterstützt Java auch die Technologie von verteilten Objekten. Dafür stehen RMI (Remote method invocation) und Java IDL zur Verfügung.

Der wesentliche Nachteil von Java ist, daß es nicht so effizient ist wie C oder C++. Die Schnelligkeit der Programme steht aber in dieser Arbeit eher im Hintergrund.

Die Implementierungsalternativen werden im Kapitel 6 genauer diskutiert.

3.2CORBA

DDE (Distributed Dokument Environment) ist ein corbabasiertes Dokumenten-System. Um DDE zu erweitern ist ein gutes Verständniss von CORBA sehr wichtig.

Die Common Object Request Broker Architecture (CORBA) der Object Management Group (OMG) ist eine standardisierte Spezifikation für die Erzeugung von verteilten, objekt-orientierten Softwaresystemen. Diese Spezifikation definiert u.a. die Funktionalitäten von einem Object Request Broker (ORB). Ein ORB ist eine Softwareschicht, die es Entwicklern ermöglicht, Objekte zu definieren, auf die über ein Netz mit klaren, high-level definierten Interfaces zugegriffen werden kann. Diese Interfaces abstrahieren von den Einzelheiten der Anwendungsimplementierungen, z.B. Programmiersprachen oder Betriebssysteme. Die Interfaces werden durch eine Sprache, die IDL (Interface Definition Language) beschrieben.

Die standardisierte Eigenschaft der CORBA Spezifikation hat wichtige Vorteile. Es gibt viele ORB Implementierungen. Wenn unterschiedliche ORBs miteinander kommunizieren, kann dies Probleme bereiten. Deswegen spezifiziert CORBA, daß alle CORBA kompatiblen ORBs ein standardisiertes Kommunikationsprotokoll (Internet Inter-ORB Protocol, oder IIOP) unterstützen müssen. Solche low-level ORB Kommunikationen sind für Anwendungsentwickler nicht sichtbar.

CORBA ermöglicht die Entwicklung verteilter, objektorientierter Anwendungen. Die Anwendungen können miteinander kommunizieren ohne zu wissen, wo die jeweils anderen Komponenten liegen und wie sie implementiert sind.

Die Object Management Architecture besteht aus 5 Komponenten:

- *Object Request Broker (ORB)* legt den Mechanismus fest, mit dem Objekte in einer verteilten Umgebung kommunizieren.
- *Common Object Services (COSS)* spezifizieren eine Reihe von anwendungsunabhängigen Diensten, die die Funktionalität des ORBs ergänzen, jedoch nicht Teil des ORBs sind.
- *Common Facilities* beschreiben allgemeine, systemspezifische Dienste.
- *Domain Interfaces* beschreiben applikationspezifische Dienste.
- *Application Interface* nehmen die Rolle sogenannter Buisness Objects ein.

3.3DDE

In [Schr99] werden die Anforderungen an ein Dokumenten-System zur gemeinsamen, verteilten Erstellung von Dokumenten vorgestellt. Basierend auf diesen Anforderungen wird dann ein Modell für ein verteiltes Dokumenten-System gegeben. Hier werden zuerst die wesentlichen Ergebnisse von [Schr99] kurz vorgestellt. Dann wird die Systemarchitektur gegeben. Diese Kenntnisse ermöglichen das Verständnis der Erweiterungen des Systems.

3.3.1Anforderungen

1. Team

Ein Team besteht aus mehreren, möglicherweise geographisch verteilten Teamteilnehmern, die über verschiedene Eigenschaften verfügen, und verschiedene Rolle im Team spielen können.

Ein Team im verteilten Dokumenten-System könnte aus „Schreiber“, Korrektor, Kommentator, Herausgeber, usw. bestehen. Die Teamteilnehmer könnten unterschiedliche Rolle spielen und unterschiedlich ausgerüstet werden. Das System soll das Team repräsentieren und die Verwaltung vom Team unterstützen.

2. Dokument

Das System soll folgende Merkmale des Dokuments unterstützen:

- Daten

Diese stellen den Informationsgehalt des Dokuments direkt dar. Dazu gehören Text, Abbildungen, Videosequenzen usw. .

- Struktur

Die Gliederung eines Dokumentes, z.B. Kapitel, Abschnitte, Aufzählungen, Listen, Verweise innerhalb eines Dokumentes, eigentlich alles, was das Auffinden und Verarbeiten der im Dokument enthaltenen Information erleichtert.

- Format

Unter dem Format ist die wahrnehmbare Erscheinung des Dokumentes zu verstehen. Die Formatierung macht die Struktur für den Leser erst wahrnehmbar.

3. Teambewußtsein

Das System soll das Teambewußtsein unterstützen. Das Teambewußtsein hat zwei Aspekte:

- Existenzbewußtsein des Teams

Bei einer gemeinsamen Erstellung von Dokumenten soll der Benutzer sich immer bewußt sein, daß eine Teamarbeit existiert. Der benutzte Texteditor soll dieses Existenzbewußtsein unterstützen, z.B. soll der Benutzer informiert werden, ob er gerade im Einbenutzerbetrieb oder im Mehrbenutzerbetrieb arbeitet.

- Sichtbarkeit von Teamaktivität

Bei einer gemeinsamen Erstellung von Dokumenten soll der Benutzer über die Aktivitäten von anderen Teammitgliedern informiert werden. Insbesondere soll der Benutzer die Änderungen in Dokumenten von anderen Teammitgliedern sehen.

4. Rechtliche Rahmenbedingungen und der Sicherheitsaspekt

Rechtliche Rahmenbedingungen definieren den autorisierten Zugriff auf Information im Dokumenten-System. Das System soll z.B. für die Teamteilnehmern anhand ihrer Rolle entsprechende Zugriffsrechte geben und den Zugriff von Teamteilnehmern anhand ihrer Zugriffsrechte überprüfen.

Auch eine sichere Kommunikation zwischen den System-Komponenten muß unterstützt werden.

5. Integrationsfähigkeit

Es soll die Möglichkeit bestehen, bestehende Autorenumgebungen im Dokumenten-System zu integrieren.

3.3.2 Modell

In der DDE werden die Dokumente, die Teamteilnehmer und die Dienste durch Objekte modelliert.

Das verteilte Dokumenten-System besteht aus:

- einer erweiterten Autorenumgebung
- der verteilten Dokumenten-Plattform

Das Modell wird in Abbildung 3.1 dargestellt.

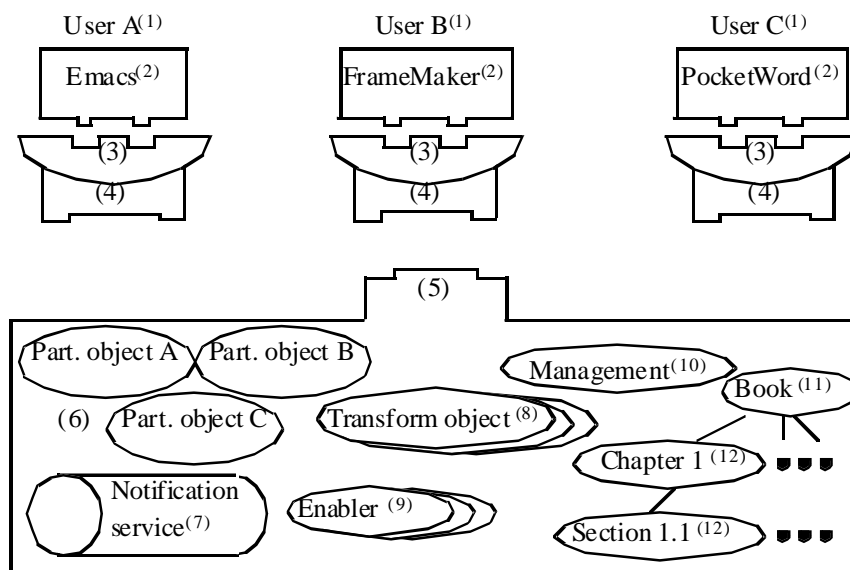


Abbildung 3.1: Modell des Dokumenten-Systems

Im einzelnen enthält die erweiterte Autorenumgebung:

- (1) Teamteilnehmer
- (2) Die gewohnten Textverarbeitungssysteme von Teamteilnehmern
- (3) Erweiterung des Textverarbeitungssystems
- (4) Schnittstelle der Erweiterung der Autorenumgebung

Die verteilte Dokumenten-Plattform enthält die folgende Komponenten:

- (5) Schnittstelle der Dokumenten-Plattform

Die Schnittstelle der Dokumenten-Plattform kapselt die Funktionalitäten der Dokumenten-Plattform, und bietet einen einheitlichen Zugang zur Autorenumgebung.

(6)persönliche Objekte

Teamteilnehmer werden durch die persönlichen Objekte modelliert. Diese stellen die Repräsentanten der Teamteilnehmer dar. Sie sind immer erreichbar, selbst wenn die Teamteilnehmer nicht am Rechner aktiv sind. Mit dieser Eigenschaft können selbst die Benutzer von mobilen Endgeräten gut unterstützt werden.

(7)Benachrichtigungsdienst

Mit dem Benachrichtigungsdienst können die Ereignisse von einem Objekt zu anderen Objekten übermittelt werden. Damit wird das Teambewußtsein unterstützt.

(8)Objekte, die die entsprechenden Formate transformieren

Diese Objekte werden benötigt, da die Teamteilnehmer verschiedene Texteditoren benutzen können, die unter Umständen unterschiedliche Formate von Daten erzeugen.

(9)Enabler

Jedes erzeugte Buch- oder Kapitelobjekt wird mit einigen Enabler Objekten gebunden, die die Rahmenbedingungen, den Sichtbarkeits- und Sicherheitsaspekt gewährleisten.

(10)Document Management Server

Der Document Management Server erzeugt, verwaltet und löscht Bücherregal-, Buch- und Kapitelobjekte.

(11)Bücherregalobjekt

Das Bücherregalobjekt stellt das Wurzelobjekt der Dokumentenstruktur dar.

(12)Buch- und Kapitelobjekte

Die Dokumentenstruktur wird durch verschiedene Objekte, z.B. Buch und Kapitel Objekte modelliert .

3.3.3DDE-Architektur

In diesem Abschnitt wird der prinzipielle Aufbau des bestehenden verteilten Dokumentenumgebung von [Schr99] vorgestellt. Obwohl in der neuen Version von [Schu99] neue Funktionen sowohl zu der DDE-Plattform als auch zu der erweiterten Autorenumgebung (Emacs) eingefügt werden, bleibt die Architektur des DDE-Systems unverändert.

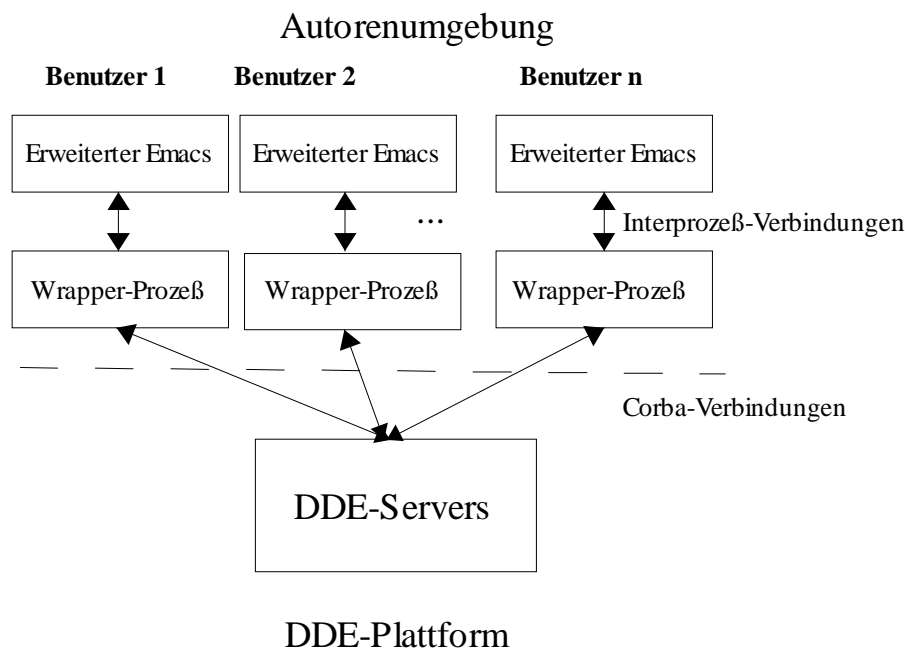


Abbildung 3.2: DDE Architektur

Die DDE-Plattform besteht aus mehreren in Java programmierten editorunabhängigen Corba-Servern, die über eine Corba-Schnittstelle verschiedene Funktionen anbieten.

In der Autorenumgebung wird Emacs in folgender Weise erweitert.

- Der Emacs wird um einige Emacs-Lisp-Funktionen erweitert.

Dazu gehören z.B. die neue Benutzungsoberfläche, um DDE-Funktionen zu benutzen, geeignete Nachrichten für den Wrapper-Prozeß zu generieren und zu schicken, die Nachrichten des Wrapper-Prozesses zu bearbeiten.

- Beim Starten von Emacs wird ein in Java programmierter Corba-basierter Wrapper-Prozeß gestartet. Durch Corba kann der Wrapper-Prozeß die DDE-Server benutzen.

- Emacs baut eine Interprozeß-Verbindung mit diesem Wrapper-Prozeß auf. Die Kommunikation zwischen Emacs und dem Wrapper-Prozeß wird durch vordefinierte Typen von Nachrichten (siehe Abschnitt 5.2.3) realisiert.
- Anhand der Nachrichten von Emacs greift der Wrapper-Prozeß auf die Funktionen der DDE-Server zu und gibt die dabei entstehenden Informationen durch Nachrichten an den Emacs zurück.

Der Wrapper-Prozeß spielt hier die Rolle eines Vermittlers zwischen Emacs und den DDE-Servern. Der Wrapper-Prozeß ist nötig, weil es unmöglich ist, direkt vom Emacs auf die Funktionen der DDE-Server (Corba-Server) zuzugreifen.

4 Analyse

In den vorigen Kapiteln haben wir die Autorenumgebung und die Plattformaspekte der DDE vorgestellt. Hier werden wir zuerst die Anforderungen für die Erweiterung der DDE für die weiteren Text Editor Systeme besprechen. Dann werden die allgemeinen Erweiterungsmöglichkeiten der DDE analysiert. Zum Schluß wird eine Schlußfolgerung aus der Analyse gezogen.

4.1 Anforderungen für die Erweiterung der DDE

In dieser Studienarbeit soll DDE erweitert werden, damit der Benutzer, der andere Texteditoren als Emacs benutzt, die DDE benutzen kann. Insbesondere sollen die Benutzer von Mobilgeräten unterstützt werden. Diese Erweiterung soll folgende Anforderungen erfüllen:

- Unterstützung aller Funktionalitäten der Dokumenten-Plattform

Der Kern von DDE ist die Dokumenten-Plattform. Sie besteht aus verschiedenen CORBA Servern. Die CORBA Server bieten die grundlegenden Funktionalitäten der verteilten Dokumenten-Plattform an, wie zum Beispiel, die Verwaltung der Dokumentenobjekte, Zugriffsrechtprüfung, usw. . Die zugehörigen Schnittstellen wurden in IDL beschrieben.

Bei der Benutzung der Dokumenten-Plattform braucht man nur entsprechende Dokumentenobjekte zu referenzieren, und dann die passende Methode aufzurufen.

Für die Endbenutzer, die über einen bestimmten Texteditor auf die Dokumentenobjekte zugreifen wollen, muß daher die Möglichkeiten gegeben werden, direkt oder indirekt diese Funktionalitäten der CORBA Server zu benutzen.

- Konsistenzerhaltung

Konsistenzerhaltung ist immer ein großes Thema in einem verteilten System. Das Problem wird verschärft, wenn ein Texteditor lose mit der Dokumenten-Plattform gekoppelt wird. Deswegen müssen für diesen Fall Maßnahmen gegen Inkonsistenzen getroffen werden.

- Wartbarkeit

Die Wartbarkeit ist wichtig für ein großes System. Sie ist auch wichtig für die weitere Entwicklung des Systems. Bei der Erweiterung der DDE soll die Wartbarkeit des Systems berücksichtigt werden.

4.2 Erweiterungsmöglichkeiten von DDE

Es gibt 4 Möglichkeiten:

1. Man erweitert den einzelnen Texteditor derart, daß er (über CORBA) die Funktionalitäten von CORBA Servern direkt nutzen kann.
 - + Diese Lösung ist ideal für die Endbenutzer, weil damit die gewohnte Autorenumgebung beibehalten wird.
 - + Der Texteditor ist gut in der Dokumenten-Plattform integriert. Weil der Texteditor eng mit der Dokumenten-Plattform gekoppelt ist, ist die Konsistenz des Systems einfach zu erhalten.
 - Jeder Texteditor muß erweitert werden. Für jede neue Version des Texteditors müssen entsprechende Erweiterungen hinzugefügt werden. Daher ist der Aufwand von Entwurf, Implementierung und Wartung sehr groß.
 - Man muß den Quellcode des Texteditors haben, um den Texteditor CORBA fähig zu machen.
 - Die CORBA Implementierung muß auf der Plattform, auf der der Texteditor läuft, installiert sein.
 - Diese Lösung ist nicht für alle Texteditoren geeignet, z.B. Pocket Word.
2. Man erweitert den einzelnen Texteditor um die Kommunikation mit einem Wrapper-Prozeß. Der Wrapper-Prozeß nutzt (über CORBA) die Funktionalitäten der CORBA Server direkt. Normalerweise wird dieser Wrapper-Prozeß durch den Texteditor gestartet, wie der erweiterte Emacs in [Schr99]. Aber der Wrapper-Prozeß kann im Prinzip ein eigenständiger Prozeß sein.
 - + Die gewohnte Autorenumgebung wird beibehalten.
 - + Man braucht keinen Quellcode von dem Texteditor.
 - + Weil die Zugriffe auf die Dokumenten-Plattform durch eine direkte CORBA-Verbindung stattfinden, ist die Konsistenz des Systems relativ einfach zu erhalten.

- Der Texteditor muß eine API besitzen, um den Texteditor erweitern zu können.
- Jeder Texteditor muß erweitert werden. Für jede neue Version des Texteditors müssen entsprechende Erweiterungen hinzugefügt werden. Daher ist der Aufwand von Entwurf und Wartung sehr groß.
- Der Texteditor muß so erweitert werden, daß er mit dem anderen Prozeß (Wrapper-Prozeß) kommunizieren kann.
- Die CORBA Implementierung muß auf der Plattform, auf der der Wrapper-Prozeß läuft, installiert sein.
- Diese Lösung ist nicht für alle Texteditoren geeignet, z.B. Pocket Word.

Der Emacs der ursprünglichen DDE wurde auf diese Weise erweitert.

3. Texteditor bleibt unverändert. Man schreibt einen Pseudo-Wrapper, der (über CORBA) direkt die Funktionalitäten von CORBA Servern benutzt. Das bedeutet, man schreibt Dokumente mit dem gewohnten Texteditor, und speichert sie in Dateien. Mit dem Pseudo-Wrapper kann man Dateien in die verteilte Dokumenten-Plattform einbringen, oder ein Dokument von der verteilten Dokumenten-Plattform auf eine lokale Datei holen.
 - + Man braucht nur einen Pseudo-Wrapper zu entwickeln. Für die Erzeugung von Dokumenten kann man beliebige Texteditoren benutzen.
 - + Diese Lösungsmöglichkeit ist auch für die Texteditoren, für die es keine Erweiterungsmöglichkeiten gibt, einsetzbar.
 - Der Texteditor ist nicht in der Dokumenten-Plattform integriert. Daher muß man beim Publishing von Dokumenten zusätzliche Arbeit leisten Z.B. um ein Dokument zu editieren, muß der Benutzer mit dem Pseudo-Wrapper das Dokument von der Dokumenten-Plattform zum lokalen Speichermedium holen. Nach der Verarbeitung muß das Dokument wieder in die Dokumenten-Plattform eingebracht werden.
 - Die CORBA Implementierung muß auf der Plattform, auf der der Pseudo-Wrapper arbeitet, installiert sein.
 - Die Gefahr von Inkonsistenzen ist u.U. höher.

4. Texteditor bleibt unverändert, wie in 3. Man schreibt einen Pseudo-Wrapper, der mit einem zusätzlichen Server (Wrapper-Server) auf der Dokumenten-Plattform über einen geeigneten Mechanismus kommuniziert. Der Wrapper-Server benutzt (über CORBA) direkt die Funktionalitäten von CORBA Servern und spielt die Rolle eines Vermittlers zwischen dem Pseudo-Wrapper und der Dokumenten-Plattform. Er nimmt die Anforderungen vom Pseudo-Wrapper, benutzt die Funktionalitäten von CORBA Servern und gibt dem Pseudo-Wrapper die Ergebnisse zurück.
- + Man braucht nur einen Pseudo-Wrapper und einen Wrapper-Server zu entwickeln. Für die Erzeugung von Dokumenten kann man beliebige Texteditoren benutzen.
 - + Diese Lösungsmöglichkeit ist auch für die Texteditoren, für die es keine Erweiterungsmöglichkeiten gibt, einsetzbar.
 - + CORBA-Installation auf dem System, auf dem Texteditor läuft ist nicht erforderlich. Daher ist es für Mobilgeräte gut geeignet.
 - + Durch den Wrapper-Server wird auch die Form der Plattform gekapselt. Damit wird ein Einbau einer anderen Plattform vereinfacht (z.B. mobilen Agenten).
 - In der Dokumenten-Plattform braucht man einen zusätzlichen Server.
 - Ein geeigneter Mechanismus wird benötigt, damit der Pseudo-Wrapper mit der Dokumenten-Plattform kommunizieren kann.
 - Es muß einen Mechanismus für den Pseudo-Wrapper geben, um den zusätzlichen Server finden und mit ihm kommunizieren zu können.
 - Der Texteditor ist nicht in der Dokumenten-Plattform integriert. Daher muß man beim Einbringen von Dokumenten in die Dokumenten-Plattform zusätzliche Arbeit leisten.
 - Die Gefahr von Inkonsistenzen ist u.U. höher.

4.3 Ergebnis

Weil wir eine Lösung für möglichst viele Benutzer finden und nicht auf einen bestimmten Texteditor eingeschränkt sein möchten, sind die Erweiterungsmöglichkeiten 1 und 2 auszuschließen.

Wir möchten auch die Benutzer von Mobilgeräten unterstützen. Zur Zeit ist eine Implementierung von CORBA nicht in allen Mobilgeräten verfügbar. Deswegen kann man von Mobilgeräten die Funktionalitäten von CORBA Servern nicht ohne weiteres benutzen und damit ist auch die Erweiterungsmöglichkeit 3 ausgeschlossen.

Die Lösungsmöglichkeit 4 erlaubt die Benutzung der verteilten Dokumenten-Plattform mit minimalen Anforderungen an die Benutzerumgebung. Daher wird diese Möglichkeit für die weitere Arbeit ausgewählt.

Damit haben wir die folgenden minimalen Anforderungen auf der Benutzerseite:

- Java Umgebung

Der Pseudo-Wrapper wird mit Java implementiert (siehe Kapitel 6). Man braucht die Java Umgebung, um diesen Pseudo-Wrapper auszuführen.

- Physikalische Verbindung mit der Dokumenten-Plattform (dem Wrapper-Server)

Die physikalische Verbindung wird benötigt, um die Kommunikation zwischen dem Pseudo-Wrapper und dem Wrapper-Server durchzuführen. Es spielt keine Rolle, ob man hier ein lokales Netzwerk, Telephoneleitungen oder Funknetz benutzt. Der Netzwerkprotokoll verbirgt die physikalischen Eigenschaften vor der Anwendung.

- Netzwerkprotokoll

Der Netzwerkprotokoll ermöglicht eine einheitliche plattform- und übertragungsunabhängige Kommunikation zwischen den Rechnern. Heutzutage unterstützen fast alle Betriebssysteme TCP/IP (Transmission Control Protocol / Internet Protocol). Damit sind die darauf aufgebauten Anwendungen weit einsetzbar.

Dies schließt die Anforderungen nach einer Integration möglichst vieler anderer Editoren und anderer Plattformen mit ein.

5 Entwurf

Aufbauend auf dem in der Analyse gewonnenen Ergebnis, stellt dieses Kapitel die Entwurfsaspekte der DDE-Erweiterung vor. Zuerst wird ein Grobentwurf gegeben. Dann werden die einzelnen Entwurfsaspekte sowohl auf der Autorenumgebung als auch auf der Dokumenten-Plattform diskutiert. Außerdem werden die Maßnahmen gegen Inkonsistenzen besprochen.

5.1 Grobentwurf

Um die Wartbarkeit des Systems zu erhöhen, wollen wir die bestehenden Funktionen der DDE ausnutzen. Insbesondere sind die Art und Weise, wie Emacs in das System integriert wurde, wichtig für diese Arbeit.

Emacs startet einen CORBA-fähigen Wrapper-Prozeß, um auf die Dokumenten-Plattform zuzugreifen. Der Emacs und der Wrapper-Prozeß kommunizieren über Nachrichtenaustausch. Der Pseudo-Wrapper und der Wrapper-Server arbeiten fast genau auf die gleiche Weise. Die Wiederverwendbarkeit und Veränderungen werden im nächsten Abschnitt genau diskutiert.

Die Endbenutzer schreiben Dokumente mit dem gewohnten Texteditor, und speichern sie in Dateien. Durch die Kommunikation zwischen dem Pseudo-Wrapper und dem Wrapper-Server können Benutzer Dateien auf die verteilte Dokumenten-Plattform verlegen, oder ein Dokument von der verteilten Dokumenten-Plattform auf eine lokale Datei holen. Der normale Arbeitsablauf ist folgender: Der Benutzer kann mit dem Pseudo-Wrapper Bücher im Bücherregal oder Kapitel in einem Buch erzeugen. Wenn er ein Kapitel editieren möchte, holt er das gewünschte Kapitel in eine lokale Datei. Dann kann er die Datei mit einem gewohnten Texteditor editieren. Danach kann er die lokale Datei mit dem Pseudo-Wrapper auf die Dokumenten-Plattform zurückspeichern. Weil möglicherweise mehrere Benutzer das gleiche Kapitel editieren wollen, müssen Maßnahmen gegen die Inkonsistenz getroffen werden.

Daher haben wir für die Erweiterung der DDE die folgenden Teilaufgaben:

- Erweiterung der Dokumenten-Plattform (Wrapper-Server zu entwickeln)
- Erweiterung der Autorenumgebung (Pseudo-Wrapper zu entwickeln)
- Die Kommunikationen zwischen dem Pseudo-Wrapper und dem Wrapper-Server abzustimmen
- Maßnahmen gegen die Inkonsistenz

5.2 Feinentwurf

Hier werden die Entwurfsaspekte von einzelnen Teilaufgaben genauer diskutiert.

5.2.1 Erweiterung der Autorenumgebung

Wie im vorigen Kapitel diskutiert, wollen wir auf der Benutzerseite einen Pseudo-Wrapper entwickeln. Der Pseudo-Wrapper verfügt über die Benutzerinformationen (user profile) z.B. Benutzername, Email-Adresse und die Systeminformationen z.B. Name der Maschine, auf der der Wrapper-Server läuft.

Der Pseudo-Wrapper wird durch den Benutzer gestartet, und durch die Benutzerinformationen zu einem bestimmten Benutzer zugeordnet.

Der Pseudo-Wrapper soll folgende Anforderungen erfüllen:

- 1) Der Pseudo-Wrapper soll beim Starten die Verbindung mit dem Wrapper-Server aufbauen.
- 2) Der Pseudo-Wrapper kann Nachrichten an den Wrapper-Server schicken und Nachrichten vom Wrapper-Server empfangen.
- 3) Der Pseudo-Wrapper verwaltet die Benutzerinformationen und die Systeminformationen. Z.B. soll der Pseudo-Wrapper dem Benutzer ermöglichen, die Benutzerinformationen und die Systeminformationen einzustellen. Diese Informationen sollen danach persistent gespeichert werden, damit das Programm beim nächsten Starten diese Informationen wieder laden kann.
- 4) Der Pseudo-Wrapper soll eine graphische Benutzungsoberfläche (GUI) besitzen. Die Benutzungsoberfläche soll folgende Funktionen besitzen.
 - Mit der Benutzungsoberfläche können Benutzer Befehle angeben, z.B. ein Book Objekt erzeugen oder Kapitel holen bzw. einbringen.
 - Die Benutzungsoberfläche kann die Dokumentenstruktur (Titel von Büchern bzw. Titel von Kapiteln) darstellen und ermöglicht dem Benutzer, ein Buch oder ein Kapitel auszuwählen.
 - Die Benutzungsoberfläche soll dem Benutzer die zugehörigen Informationen anzeigen, z.B. die Ergebnisse einer Aktion.
 - Mit der Benutzungsoberfläche können die Benutzer die Benutzerinformationen oder die Systeminformationen eingeben.

5.2.2 Erweiterung der Dokumenten-Plattform

In diesem Abschnitt werden zuerst die Implementierungsaspekte der Dokumenten-Plattform von DDE vorgestellt. Dann wird der Entwurf des Wrapper-Servers besprochen.

5.2.2.1 Implementierungsaspekte der Dokumenten-Plattform

Die Dokumenten-Plattform stellt die grundlegende Funktionalität zur Erstellung gemeinsamer, verteilter Dokumente zur Verfügung. Das Modell in Abschnitt 3.3.2, Abbildung 3.1 spezifiziert die zu realisierenden Komponenten:

- Verwaltung der persönlichen Objekte
- Verwaltung der Bücherregal-, Buch- und Kapitelobjekte
- Transformationsobjekte
- Awareness-Objekt
- Enabler-Objekte
- Benachrichtigungsdienst

Eine weitere Komponente, die den Kontrollfluß zwischen den oben genannten Komponenten steuert, ist der Mediator Server.

Die von den einzelnen Komponenten verwalteten Objektreferenzen werden in einem Name Server verwaltet.

Die Dokumenten-Plattform ist eine verteilte, corbabasierte Anwendung. Jede Komponente dieser Dokumenten-Plattform spezifiziert ihre Eigenschaft durch eine in OMG IDL definierte Schnittstelle. Damit sind die Schnittstelle und ihre Implementierung getrennt.

Diese Komponenten sind alle Corba-Objekte (Corba-Servers). Sie werden beim Starten der DDE automatisch instanziiert.

In dieser Studienarbeit wird ein zusätzlicher Wrapper-Server implementiert, der beim Starten der DDE mit den anderen Servern mitstartet.

5.2.2.2 Entwurf des Wrapper-Servers

In Kapitel 4 wurde erwähnt, daß der Emacs der ursprünglichen DDE nach Möglichkeit 2 erweitert wurde. D.h. der Emacs benutzt nicht direkt die Funktionalitäten von CORBA Servern, sondern kommuniziert mit einem Wrapper-Prozeß. Der Wrapper-Prozeß benutzt die Funktionalitäten von CORBA Servern direkt. Dieser Wrapper-Prozeß spielt die gleiche Rolle wie der Wrapper-Server hier. Es wäre wünschenswert, diesen Wrapper-Prozeß benutzen zu können.

Es gibt aber einige Aspekte, die einer direkten Benutzung des Wrapper-Prozesses als Wrapper-Server entgegenstehen.

- Der Wrapper-Prozeß wird durch den Emacs auf der Benutzerseite gestartet.

In unserem Fall sollte der Wrapper-Server auf der Dokumenten-Plattform laufen und automatisch oder per Hand mit den DDE-Servern gestartet werden.

- Jeder Emacs startet einen eigenen Wrapper-Prozeß. Der Wrapper-Prozeß kommuniziert nur mit dem Emacs, der den Wrapper-Prozeß startet. (Diese Arbeitsweise wird auch benötigt im Fall der Mole-Plattform (siehe [Lukas]).

In unserem Fall wird der Wrapper-Server aus Gründen der Systembelastung nur einmal auf der Dokumenten-Plattform gestartet und bietet Dienste an alle Pseudo-Wrapper an.

Aus diesen Gründen muß ein eigener Corba-Server (Wrapper-Server) implementiert werden.

Die Kommunikation zwischen Emacs und Wrapper-Prozeß wird durch Nachrichtenaustausch realisiert. Ein Nachrichtenformat wurde definiert. Dieses Nachrichtenformat kann hier benutzt werden, um die Kommunikation zwischen Pseudo-Wrapper und Wrapper-Server durchzuführen.

5.2.3 Kommunikation zwischen Pseudo-Wrapper und Wrapper-Server

Wie schon gesagt, wird die Kommunikation zwischen Emacs und Wrapper-Prozeß durch Nachrichtenaustausch realisiert. Der Informationsfluß zwischen Emacs und dem Wrapper-Prozeß besteht aus Informationseinheiten. Das Format einer Informationseinheit (IE) wird in BNF definiert. Der Einfachheit halber benutzen wir hier auch dieses Format, um die Kommunikation zwischen Pseudo-Wrapper und Wrapper-Server durchzuführen. Die Definition einer Informationseinheit in BNF wird im folgenden gegeben:

```
<IE> ::= "<" <ID> "><" <IETYPE> "><" <IECONTENT> ">"  
      <NEWLINE>
```

```
<IETYPE> ::= <EMACS2WRAPPER> | <WRAPPER2EMACS>
```

```
<EMACS2WRAPPER> ::= "init" | "reset" | "msg" | "openBookShelf" |  
                    "createBookShelf" | "destroyBookShelf" |  
                    "closeBookShelf" | "openBook" |  
                    "createBook" | "destroyBook" |  
                    "closeBook" | "openSection" |  
                    "createSection" | "destroySection" |  
                    "closeSection" | "BookShelfTOC" |  
                    "BookTOC"
```

```
<WRAPPER2EMACS> ::= "bookshelf" | "book" | "section" | "event"
```

```
<ID> ::= ASCII Zeichen 65-90 und 97-122
```

```
<IECONTENT> ::= ASCII Zeichen 32-127
```

```
<NEWLINE> ::= ASCII Zeichen 10
```

Das Nichtterminal "<EMACS2WRAPPER>", repräsentiert den IE Typ für IEs vom Emacs zur Wrapperumgebung, steht hier aber für den IE Typ für IEs vom Pseudo-Wrapper zum Wrapper-Server. Das Nichtterminal "<WRAPPER2EMACS>", repräsentiert den IE Typ für IEs von der Wrapperumgebung zum Emacs, steht hier aber für den IE Typ für IEs vom Wrapper-Server zum Pseudo-Wrapper.

Auf eine Anforderung vom Pseudo-Wrapper reagiert der Wrapper-Server mit einem entsprechenden Methodenaufruf der Schnittstelle der Dokumenten-Plattform. Die Informationen der Dokumenten-Plattform, z.B. der Inhalt eines Kapitels, wird durch eine IE an den Pseudo-Wrapper weitergeleitet.

5.3 Maßnahmen gegen Inkonsistenzen

Wie schon erwähnt, könnte es Inkonsistenzen geben bei gleichzeitigen Änderung der Struktur eines Buches oder desselben Kapitels durch mehrere Benutzer. In [Schu99] werden alle mögliche Inkonsistenzen diskutiert. Während der gemeinsamen Erstellung von Dokumenten sind die Dokumentenstrukturen relativ statisch, z.B. welche Bücher geschrieben werden und welche Kapitel in einem Buch sind. Die Dokumentenstrukturen werden nur selten während des gesamten Erstellungsvorgangs geändert. Deswegen ist nur der Fall der gleichzeitigen Änderung desselben Kapitelinhalts besonders kritisch. Daher wird wegen des Aufwands nur dieser Fall in [Schu99] behandelt.

In diesem Abschnitt werden solche Inkonsistenzen zuerst durch ein Beispiel vorgestellt. Weil die Maßnahmen dagegen von der DDE-Plattform abhängig sind, werden sie im Kontext der DDE von [Schr99] und der DDE von [Schu99] betrachtet.

5.3.1 Beispielszenario

Das Zeitablaufdiagramm in Abbildung 2.2 zeigt ein typisches Szenario, in dem eine Inkonsistenz entstehen kann.

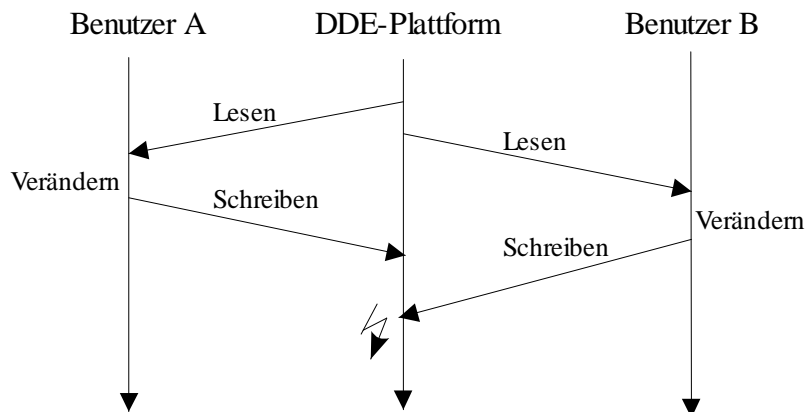


Abbildung 5.1: Beispielszenario

In diesem Beispiel holt der Benutzer A zuerst den Inhalt eines Kapitels zum lokalen Rechner (mit Emacs oder mit dem Pseudo-Wrapper), um den Inhalt dieses Kapitels zu ändern. Der Benutzer B will auch das selbe Kapitel bearbeiten, und holt den Inhalt des Kapitels zu seinem lokalen Rechner. Der Benutzer A bringt zuerst seine geänderte Version des Kapitels in die DDE-Plattform ein. Wenn der Benutzer B seine geänderte Version einbringen will und die beiden Benutzer keine gleichen Änderungen vorgenommen haben, entsteht eine Inkonsistenz. Wenn dem Einbringwunsch von Benutzer B zugestimmt wird, werden die Änderungen von Benutzer A überschrieben. Wenn der Einbringwunsch von Benutzer B verweigert wird, werden aber die Veränderungen von Benutzer B nicht durchgeführt.

Es gibt verschiedene Verfahren, um die Konsistenzen zu behandeln. In [Schu99] werden einige davon vorgestellt und die Einsetzbarkeit in DDE analysiert. Im folgenden werden die Maßnahmen gegen Inkonsistenz in DDE in der ursprünglichen DDE von [Schr99] und in der erweiterten DDE von [Schu99] vorgestellt, und die Einflüsse auf den Pseudo-Wrapper betrachtet.

5.3.2 Maßnahmen gegen Inkonsistenz in der ursprünglichen DDE

In der ursprünglichen DDE wird ein pessimistisches Verfahren, das „Sperrverfahren“, benutzt, um die Konsistenz des Systems zu erhalten, d.h. während des Änderungsvorgangs von einem Benutzer haben die anderen Benutzer keinen Änderungszugriff auf dasselbe Objekt.

In der ursprünglichen DDE wird dieser Mechanismus durch einen Enabler Server realisiert. Bei jedem Zugriffswunsch prüft der Enabler Server, ob dieser Zugriff im Konflikt zu anderen steht. Im Konfliktfall wird der Zugriffswunsch zurückgewiesen.

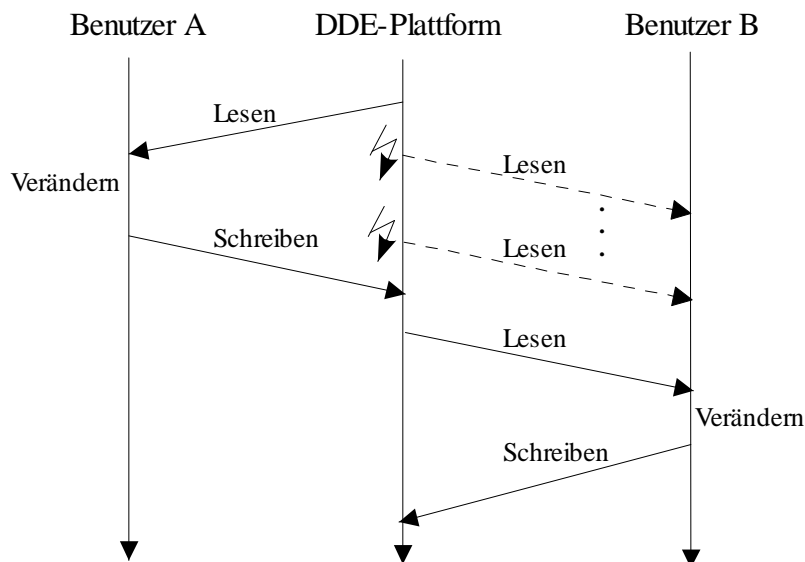


Abbildung 5.2: Konsistenzenerhaltung in der ursprünglichen DDE am gleichen Beispielszenario wie in Abschnitt 5.3.1

Das Beispiel in der Abbildung 5.2 zeigt, wie die Konsistenz erhalten wird. Der Benutzer A hat zuerst das Zugriffsrecht auf ein Objekt. Wenn Benutzer B auf dasselbe Objekt zugreifen will, wird der Zugriffsvorgang verweigert. Erst wenn der Benutzer das Zugriffsrecht wieder zurückgibt, kann Benutzer B mit diesem Objekt weiter arbeiten.

Weil bei der Verarbeitung von Objekten Sperren gesetzt werden, muß der Pseudo-Wrapper gewährleisten, daß alle eingesetzten Sperren nach der Verarbeitung wieder freigegeben werden.

5.3.3 Maßnahmen gegen Inkonsistenz in der erweiterten DDE

In [Schu99] wird ein optimistisches Verfahren benutzt, um die Konsistenz zu erhalten. Die Benutzer können gleichzeitig auf dasselbe Objekt zugreifen. Das System soll die Inkonsistenzen bei gleichzeitigen Änderungen durch verschiedene Benutzer erkennen, und Unterstützung beim Zusammenführen der Änderungen bieten. Dazu werden die Änderungen von verschiedenen Benutzern unterschiedlich (z.B. mit verschiedenen Farben) dargestellt. Der Benutzer, der zuletzt speichern will, erhält diese Darstellung des Objekts, und entscheidet sich fallweise für eine endgültige Version (Reintegration).

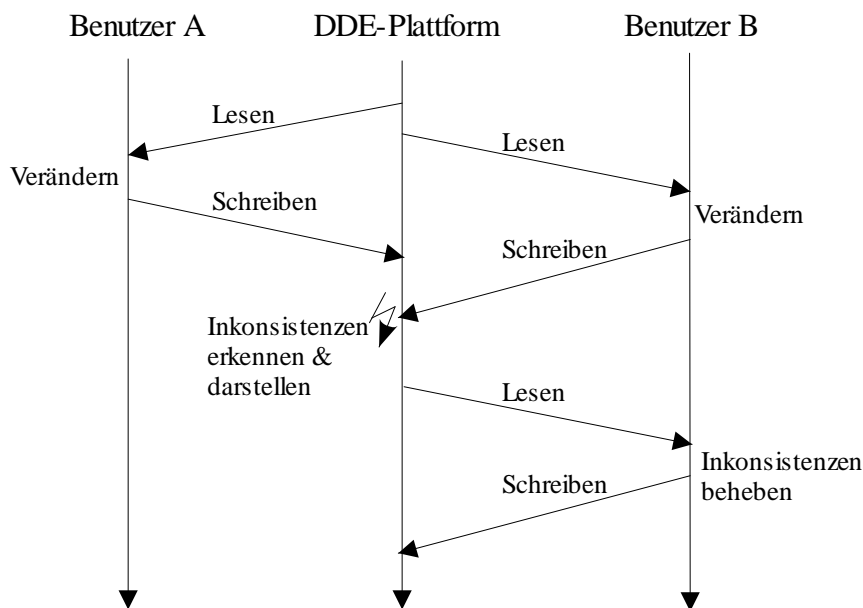


Abbildung 5.3: Konsistenzhaltung in der erweiterten DDE am gleichen Beispielszenario wie in Abschnitt 5.3.1

Das Beispiel in Abbildung 5.3 zeigt, wie die Konsistenz erhalten wird. Benutzer A hat zuerst ein Objekt geändert. Benutzer B kann auch dasselbe Objekt ändern. Beim Einbringen von Benutzer B vergleicht das System die beide Versionen. Wenn sie unterschiedlich sind, erkennt das System die Inkonsistenzen. Das System erzeugt eine neue Version von diesem Objekt. Der Benutzer B kann die Inkonsistenzen sehen, und sie beheben. Bei identischen Änderungen von Benutzer A und Benutzer B entsteht keine Inkonsistenz.

Jeder Benutzer, auch der, der an einem Pseudo-Wrapper hängt, benutzt einen eigenen Editor, um den Kapitelinhalt darzustellen und zu editieren. Deswegen muß der Editor die Inkonsistenzen darstellen können. Dies kann durch eine Konvertierung der Kapitelinhalte in ein bestimmtes Format erreicht werden. Z.B. kann man die Darstellungen von Inkonsistenzen in XML geeignet definieren. Um die Inkonsistenzen im eigenen Editor darzustellen, wird der Kapitelinhalt von XML zu einer bestimmten Format transformiert.

6 Implementierung

Im vorigen Kapitel haben wir die Entwurfsaspekte für die Erweiterung von DDE diskutiert. In diesem Kapitel wird die Implementierung der einzelnen Komponenten vorgestellt.

Die Erweiterung der DDE wird mit der Programmiersprache Java implementiert. Die Gründe für die Entscheidung für Java sind unter anderem die Plattformunabhängigkeit und die gute Netzwerkfähigkeit von Java.

Hier wird Java 2 SDK Version 1.2.2 benutzt.

Die Implementierung wird in folgende zwei Teile aufgeteilt:

- Implementierung des Pseudo-Wrappers
- Implementierung des Wrapper-Servers

Abbildung 6.1 zeigt die Systemarchitektur.

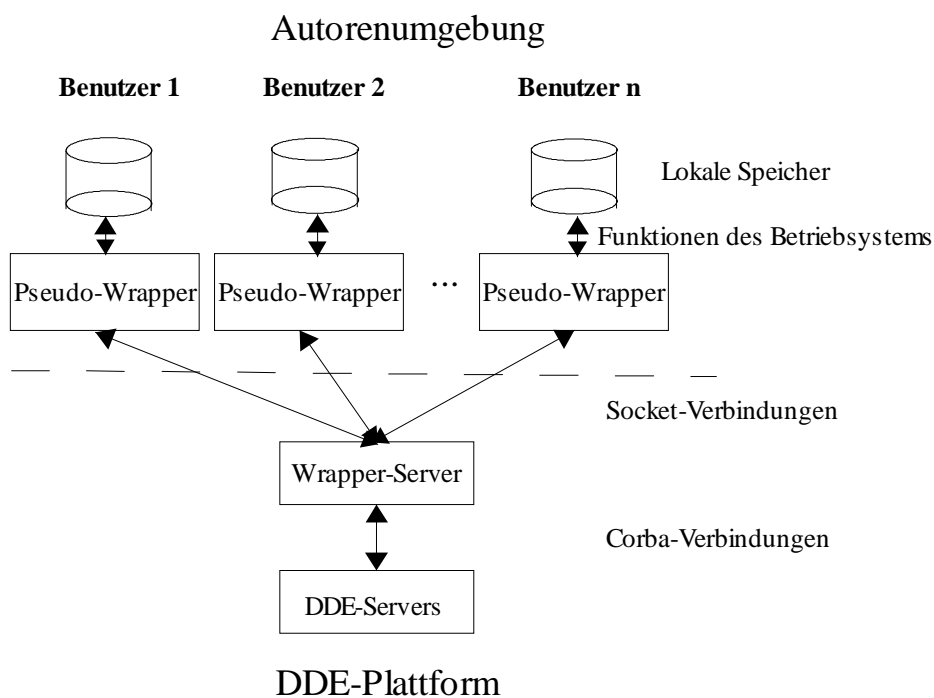


Abbildung 6.1: Systemarchitektur

6.1 Implementierung des Pseudo-Wrappers

6.1.1 Teilaufgaben

Der Pseudo-Wrapper hat folgende Teilaufgaben:

- Die graphische Benutzungsoberfläche (GUI)

Man kann AWT (Abstract Window Toolkit) oder Swing für die Implementierung von einer graphischen Benutzungsoberfläche benutzen. Alle beide sind Teile der Java Foundation Classes (JFC).

Die AWT Komponenten werden von der Plattformen JDK 1.0 und 1.1 angeboten. Obwohl die Java 2 Plattform die AWT Komponenten auch unterstützt, wird empfohlen, die Swing Komponenten zu benutzen. Deswegen benutzen wir hier JFC/Swing für die Implementierung der Benutzungsoberfläche des Pseudo-Wrappers.

- Die Verbindung mit dem Wrapper-Server verwalten

Die Verbindung mit dem Wrapper-Server wird mit Socket wie folgt aufgebaut:

```
ddeSocket = new Socket(ServerName, 8888);
```

ServerName ist der Name oder die IP-Adresse der Maschine, auf der der Wrapper-Server läuft. 8888 ist die Port Nummer, die vom Wrapper-Server benutzt wird. Der Benutzer kann diese Werte neu einstellen (siehe Abschnitt 6.2.5 2 „Settings“ Knopf).

Wenn die Verbindung richtig aufgebaut wird, kann man mit dem Objekt `ddeSocket` Nachrichten von dieser Verbindung lesen oder auf diese Verbindung schreiben. Die Verbindung wird automatisch abgebaut, wenn die Anwendung beendet wird.

- Die Nachrichten verarbeiten

Mit dem im vorigen Kapitel definierten Nachrichtenformat werden die einzelnen Nachrichtentypen eindeutig bestimmt. Damit kann der Pseudo-Wrapper jede einzelne Nachricht behandeln.

- Dateiverwaltung

Die Kapitel Objekte müssen lokalen Dateien zugeordnet werden (siehe Abschnitt 6.2.4.4).

6.1.2 Der Startvorgang des Pseudo-Wrappers

Beim Starten des Pseudo-Wrappers werden hauptsächlich folgende Schritte durchgeführt:

1. Das Programm wird zuerst versuchen, im lokalen Verzeichnis, wo das Programm gestartet wird, die Datei `settings.dat` zu öffnen, und die Einstellungen für das Programm einzulesen. Wenn die Datei `settings.dat` nicht da ist, werden die vordefinierten Einstellungen für das Programm eingesetzt. Diese Funktion wird durch die Java Hilfsklasse `Properties` unterstützt (siehe Abschnitt 6.1.5 „Settings“ Knopf).
2. Das Programm erzeugt alle benötigten Swing Komponenten, z.B. Fenster, Knöpfe, usw.
3. Das Programm versucht, anhand der Einstellungen eine Socket-Verbindung mit dem Wrapper-Server aufzubauen.

Im erfolgreichen Fall werden die Vorbereitungen für das Lesen von dieser Socket oder das Schreiben zu dieser Socket getroffen.

Im Fehlerfall wird eine Fehlermeldung an den Benutzer ausgegeben. Der Benutzer kann die Einstellungen für das Programm überprüfen, und gegebenenfalls ändern. Der Benutzer kann dann die Verbindung mit dem Wrapper-Server neu aufbauen und im erfolgreichen Fall kann das Programm diesen Vorgang fortsetzen.

4. Nachdem die Verbindung richtig aufgebaut wird, schickt das Programm die Initialisierungsinformationen (Benutzername, Email Adresse) und die Anforderung für die Informationen des Bücherregals zum Wrapper-Server.

Der Wrapper-Server prüft die Initialisierungsinformationen. Wenn sie richtig sind, schickt der Wrapper-Server eine entsprechende Nachricht an den Pseudo-Wrapper. Diese Nachricht enthält auch die Namen der aktuellen Bücher im Bücherregal. Der Pseudo-Wrapper kann diese Informationen entsprechend darstellen.

5. Das Programm empfängt die Informationen vom Bücherregal und stellt sie in der Benutzungsoberfläche des Pseudo-Wrappers dar.
6. Das Programm ist bereit, die Eingaben von Benutzern oder weitere Nachrichten vom Wrapper-Server zu bearbeiten.

6.1.3 Die Benutzungsoberfläche des Pseudo-Wrappers

Abbildung 6.2 zeigt die Benutzungsoberfläche des Pseudo-Wrappers bei einem erfolgreichen Starten. In diesem Beispiel wird gezeigt, daß es drei Bücher im Bücherregal gibt.

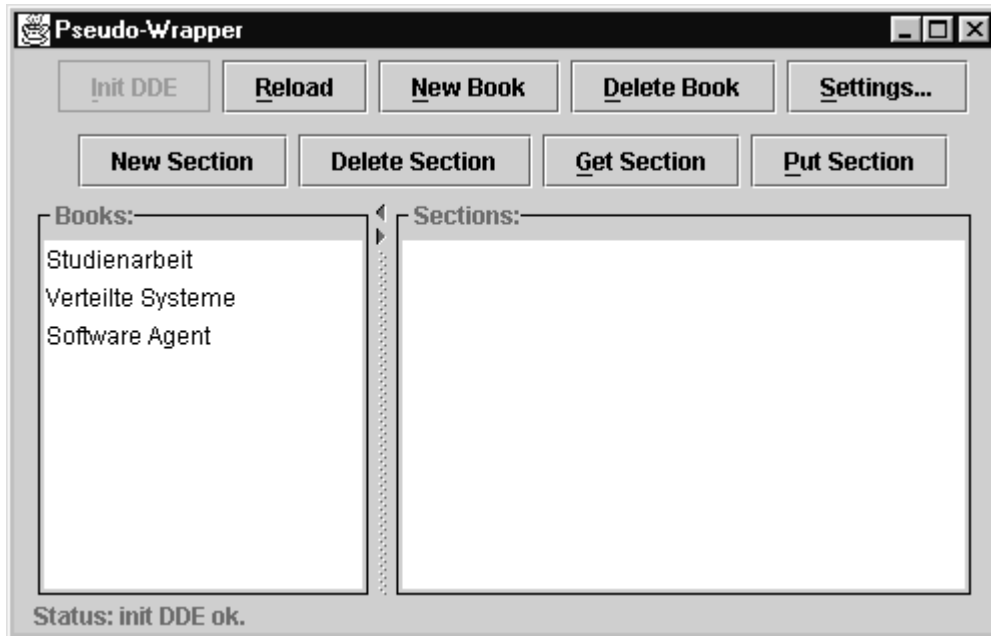


Abbildung 6.2: Benutzungsoberfläche des Pseudo-Wrappers

Die Benutzungsoberfläche des Pseudo-Wrappers ist in drei Bereiche aufgeteilt:

- Im oberen Bereich sind verschiedene Knöpfe, mit denen man bestimmte Funktionen des Pseudo-Wrappers durchführen kann. Diese Knöpfe werden in Abschnitt 6.1.5 genau beschrieben.
- Im mittleren Bereich wird die Dokumentenstruktur dargestellt. Es ist ein geteiltes Feld, d.h. das Feld besteht aus zwei nebeneinander liegenden Felder. Im linken Feld wird die Liste von Büchernamen dargestellt. Man kann mit der Maus das einzelne Buch auswählen. Im rechten Feld wird die Liste von Kapitelnamen, die zu dem ausgewählten Buch gehört, dargestellt. Im nächsten Abschnitt wird die Darstellung der Dokumentenstruktur genau beschrieben.
- Im unteren Bereich gibt es ein Feld mit nur einer Zeile (Status-Zeile). In dieser Status-Zeile werden verschiedene Informationen gezeigt, z.B. welche Funktion gerade durchgeführt wird, die Ergebnisse einer Aktion, u.s.w.

6.1.4 Darstellung der Dokumentenstruktur

In diesem Abschnitt wird zuerst die Dokumentenstruktur in der ursprünglichen DDE vorgestellt. Dann wird die Implementierung der Dokumentenstruktur im Pseudo-Wrapper besprochen. Am Schluß wird eine Alternative gegeben.

6.1.4.1 Dokumentenstruktur in DDE

In der DDE wird eine vereinfachte baumartige Dokumentenstruktur implementiert. Die Wurzel dieser Struktur ist „Bücherregal“. Alle Bücher gehören zu dieser Wurzel. Die Kapitel sind jeweils einem Buch zugeordnet.

Abbildung 6.3 zeigt die Dokumentenstruktur in der DDE.

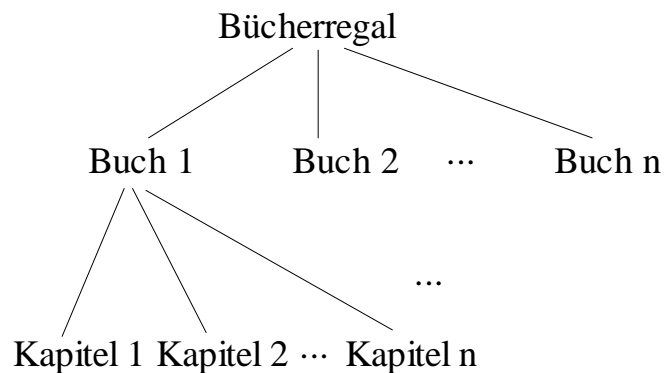


Abbildung 6.3: Dokumentenstruktur

6.1.4.2 Implementierung im Pseudo-Wrapper

Wie in Abschnitt 6.1.3 besprochen, wird die Dokumentenstruktur im mittleren Bereich der Benutzungsoberfläche des Pseudo-Wrappers durch zwei Listen (Buchnamenliste und Kapitelnamenliste) dargestellt.

Um manche Funktionen des Pseudo-Wrappers durchzuführen, muß man zuerst ein Buch oder ein Kapitel auswählen, z.B. um ein Buch zu löschen oder ein Kapitel zu laden. Man kann mit der Maus das einzelne Buch in der Buchnamenliste auswählen. Das ausgewählte Buch wird dann farbig markiert und die Kapitelnamen dieses Buchs werden in der Kapitelnamenliste dargestellt. Man kann dann mit der Maus auch ein einzelnes Kapitel auswählen. Das ausgewählte Kapitel wird ebenfalls farbig markiert.

Abbildung 6.4 zeigt die oben beschriebene Situation.

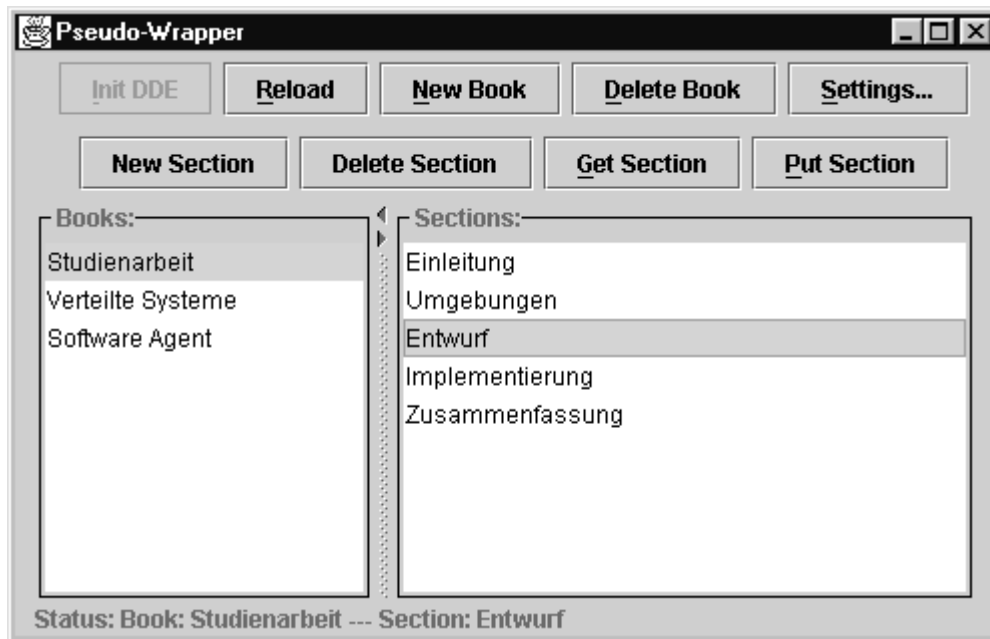


Abbildung 6.4: Darstellung der Dokumentenstruktur

In diesem Beispiel gibt es drei Bücher. Das Buch „Studienarbeit“ wird ausgewählt und die fünf Kapitel in diesem Buch werden in der Kapitel-Liste dargestellt. Das Kapitel „Entwurf“ dieses Buchs wird ausgewählt. Dann kann man z.B. mit dem „Get Section“ Knopf das ausgewählte Kapitel in einer lokalen Datei speichern (siehe Abschnitt 6.1.5 „Get Section“ Knopf).

Diese Informationen werden auch in der Status-Zeile gezeigt.

6.1.4.3 Eine Alternative

Im vorigen Abschnitt wurde die Darstellung der Dokumentenstruktur durch zwei Listen vorgestellt. Diese Implementierung ist nur geeignet für diese vereinfachte Dokumentenstruktur. Weil die normale Dokumentenstruktur baumartig ist und mehr als zwei Schichten hat, wäre eine alternative Implementierung mit Swing Komponent Tree besser. Die Darstellung wäre etwa wie „Windows Explorer“ von MS Windows.

Prinzipiell ist die Implementierung mit Swing Komponent Tree möglich. In Anlehnung an Emacs werden aber in dieser Studienarbeit zwei Listen benutzt. Die Darstellung mit solchen Listen im Pseudo-Wrapper reicht für unseren Fall auch aus.

6.1.4.4 Dateiverwaltung

Es gibt zwei Möglichkeiten für die Dateiverwaltung:

1. Die Objektenstruktur (Bücherregal - Buch - Kapitel) wird genau in ein lokales Verzeichnis abgebildet. Z.B Kapitel1 von Buch1 wird zu der Datei ~/Buch1/Kapitel1 zugeordnet.
2. Die Zuordnung von Kapitel Objekten zu den lokalen Dateien wird vom Benutzer selbst festgelegt.

Bei der ersten Möglichkeit gibt es eine eindeutige Abbildung zwischen den Kapitel Objekten und den lokalen Dateien. Bei der zweiten Möglichkeit hat der Benutzer mehr Flexibilität.

In dieser Studienarbeit wird die Dateiverwaltung mit der zweiten Möglichkeit implementiert. Damit hat der Benutzer mehr Flexibilität. Durch die geeignete Benennung von lokalen Dateien und Verzeichnissen kann man auch hier die eindeutige Zuordnung zwischen den Kapitel Objekten und den lokalen Dateien erreichen.

6.1.5 Die Funktionen der Knöpfe

In diesem Abschnitt werden die einzelnen Knöpfe im oberen Bereich der Benutzungsoberfläche des Pseudo-Wrappers vorgestellt.

1. „Init DDE“ Knopf

Mit dem „Init DDE“ Knopf kann man eine Socket-Verbindung mit dem Wrapper-Server neu aufbauen.

Der „Init DDE“ Knopf wird nur benötigt, wenn beim automatischen Aufbau einer Socket-Verbindung mit dem Wrapper-Server während des Programmstarts ein Fehler auftritt. Deswegen wird beim erfolgreichen Starten des Programms der „Init DDE“ Knopf, wie in der Abbildung 6.2 gezeigt, grau dargestellt. Das bedeutet, dieser Knopf ist momentan nicht ausführbar.

Es gibt einige Situationen, in denen eine Socket-Verbindung mit dem Wrapper-Server nicht möglich ist:

- Der Wrapper-Server ist gerade nicht am Laufen. Der Wrapper-Server muß manuell gestartet werden.

- Die gesetzte Adresse (IP Adresse oder Name) der Maschine, in der der Wrapper-Server läuft, ist falsch.
- Die gesetzte Portnummer des Wrapper-Servers ist schon besetzt, damit kann der Wrapper-Server nicht richtig gestartet werden.

Weil ohne eine Socket-Verbindung mit dem Wrapper-Server der Pseudo-Wrapper weitere Funktionen nicht durchführen kann, sind bei einem Fehlstarten des Pseudo-Wrappers nur der „Init DDE“ Knopf und der „Settings...“ Knopf ausführbar.

Mit dem „Settings...“ Knopf kann der Benutzer die Einstellungen für den Pseudo-Wrapper ändern. Der „Settings...“ Knopf wird später genau beschrieben

Die häufige Ursache für einen Fehlstart sind falsche Einstellungen. Nachdem man mit dem „Settings...“ Knopf die Einstellungen geändert hat, kann man mit dem „Init DDE“ Knopf eine neue Socket Verbindung mit dem Wrapper-Server aufbauen.

Abbildung 6.5 zeigt die Situation bei einem Fehlstart.

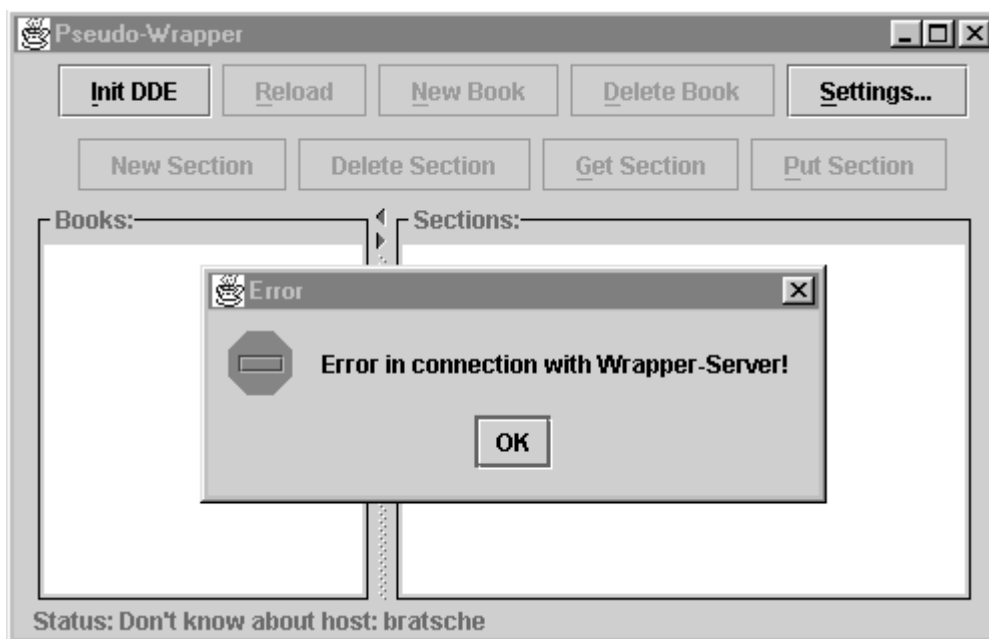


Abbildung 6.5: Fehlstart des Pseudo-Wrappers

Die Information für den Fehlstart wird auch in der Status-Zeile angezeigt.

2. „Settings“ Knopf

Wie im vorigen Abschnitt schon gesagt, ist auch der „Settings“ Knopf beim Fehlstart ausführbar. Mit dem „Settings“ Knopf können die Benutzer die vordefinierten Einstellungen oder die Einstellungen einer vorigen Sitzung für den Pseudo-Wrapper ändern. Wenn Benutzer den „Settings“ Knopf klicken, erscheint das Einstellungsfenster „Settings Dialog“.

Abbildung 6.6 zeigt das Einstellungsfenster.

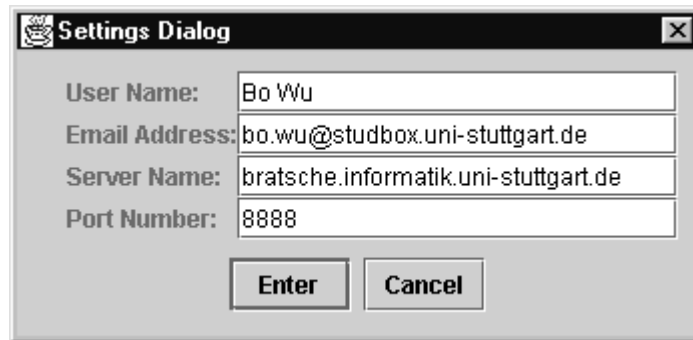


Abbildung 6.6: Einstellungsfenster

Im Einstellungsfenster werden zuerst die aktuellen Einstellungen gezeigt. Beim ersten Starten des Pseudo-Wrappers werden die vordefinierten Einstellungen gezeigt. Man kann hier die aktuellen Einstellungen ändern. Die neuen Einstellungen werden dann in der Datei `settings.dat` im lokalen Home-Verzeichnis gespeichert. Beim nächsten Starten des Pseudo-Wrappers werden diese Einstellungen vom Programm automatisch geladen und für eine Socket-Verbindung mit dem Wrapper-Server benutzt.

Die Funktion des „Settings...“ Knopfs wird durch die java Hilfsklasse `Properties` implementiert.

3. „Reload“ Knopf

Mit dem „Reload“ Knopf kann man den Inhalt der Bücher-Liste aktualisieren.

Das vorgestellte Dokumenten-System ist ein Mehrbenutzersystem. Das bedeutet, die Zustände des Systems können sich ständig ändern. Die Benutzer sollen die Möglichkeit haben, immer die aktuellen Zustände des Systems zu sehen. Es gibt mehrere Techniken, um dieses Ziel zu erreichen:

- Wenn ein Benutzer eine Änderung im Dokumenten-System gemacht hat, schickt der Wrapper-Server eine Nachricht an alle mit der DDE-Plattform verbundenen Pseudo-Wrapper. Wenn ein Pseudo-Wrapper diese Nachricht bekommt, holt er die aktuelle Informationen ab. Wenn die Änderung nicht groß ist, kann diese Änderung mit der Nachricht mitgeschickt werden.

Dieser Ansatz löst das Problem gut, ist aber relativ schwer zu implementieren.

- Der Pseudo-Wrapper fragt periodisch nach aktuellen Informationen.

Dieser Ansatz ist relativ einfach zu implementieren, aber es ist problematisch, den Zeitabstand zu bestimmen. Der optimale Zeitabstand ist von der konkreten Anwendung abhängig. Wenn der Zeitabstand klein ist, wird der Aufwand groß. Wenn der Zeitabstand groß ist, fehlt es an Genauigkeit.

- Vor wichtigen Entscheidungen holen die Benutzer per Hand die aktuelle Informationen.

Weil die Dokumentenstruktur relativ stabil ist, reicht hier eine manuelle Aktualisierung. Beim Klicken auf den „Reload“ Knopf wird folgende Nachricht zum Wrapper-Server geschickt.

```
<BOOKSHELF><E2WBOOKSHELFTOC>< >
```

4. „New Book“ Knopf

Mit dem „New Book“ Knopf kann man ein neues Buch erzeugen.

Beim Klicken auf den „New Book“ Knopf erscheint das Fenster „New Book Dialog“. In diesem Fenster kann man den Namen des neuen Buches eingeben, dann mit „Enter“ die Eingabe bestätigen oder mit „Cancel“ das Fenster schließen ohne ein neues Buch zu erzeugen.

Abbildung 6.7 zeigt das „New Book Dialog“ Fenster.

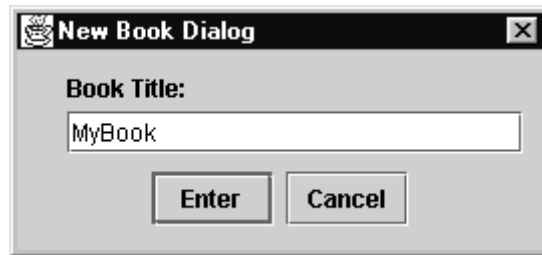


Abbildung 6.7: Das „neues Buch“ Fenster

Wenn man den Buchnamen MyBook eingibt, und die Eingabe mit „Enter“ bestätigt, erzeugt der Pseudo-Wrapper folgende Nachricht:

```
<BOOKSHELF><E2WCREATEBOOK><MyBook>
```

Wenn der Wrapper-Server diese Nachricht empfängt, erzeugt er ein Buchobjekt mit dem Namen MyBook. Nach dem Erzeugen eines Buchs wird die Bücher-Liste automatisch aktualisiert.

5. „Delete Book“ Knopf

Mit dem „Delete Book“ Knopf kann man ein ausgewähltes Buch löschen.

Um ein Buch zu löschen, muß man dieses zuerst auswählen. Dann drückt man auf den „Delete Book“ Knopf. Es wird dann ein Fenster mit Warnung erscheinen. Man kann mit dem „Yes“ Knopf den Löschwunsch bestätigen, oder mit dem „No“ Knopf den Löschvorgang unterbrechen.

Abbildung 6.8 zeigt diese Situation.

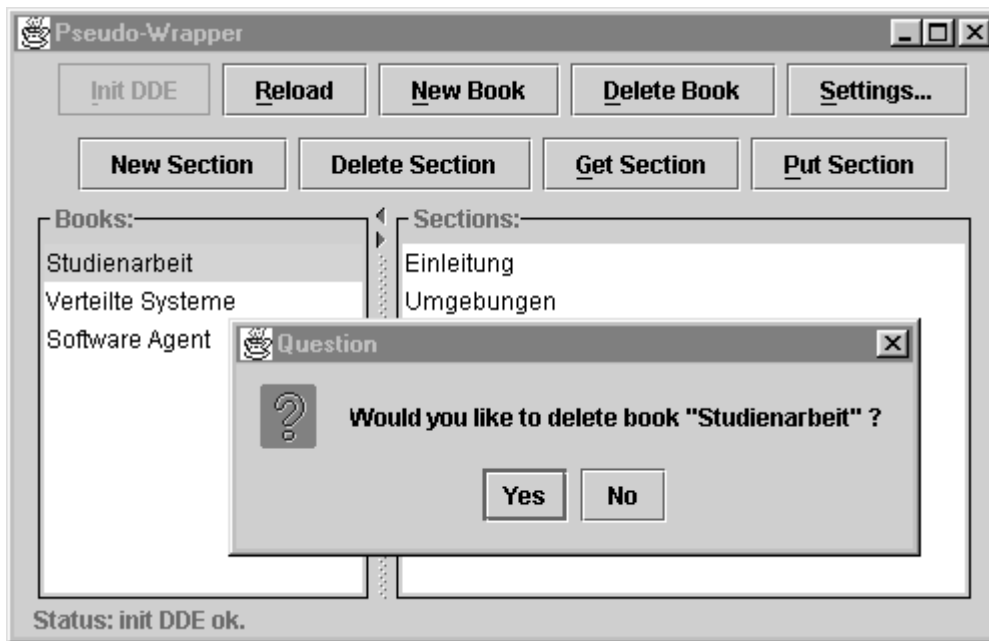


Abbildung 6.8: Ein Buch löschen

Das Beispiel in Abbildung 6.8 zeigt, daß der Benutzer das Buch Studienarbeit löschen möchte.

Das Buch Studienarbeit wird zuerst ausgewählt. Dann wird der „Delete Book“ Knopf gedrückt und das Warnungsfenster erscheint.

Nach der Bestätigung mit dem „Yes“ Knopf im Warnungsfenster wird folgende Nachricht erzeugt und an den Wrapper-Server geschickt:

```
<BOOKSHELF><E2WDESTROYBOOK><Studienarbeit>
```

Nachdem ein Buch gelöscht wird, wird die Buchname-Liste wieder neu aktualisiert.

6. „New Section“ Knopf

Mit dem „New Section“ Knopf kann man zuerst ein Kapitel erzeugen und in ein Buch einfügen.

Ein Kapitel gehört immer zu einem Buch. Deswegen muß man zuerst ein Buch auswählen, um ein neues Kapitel zu erzeugen und dann dieses Kapitel in dieses Buch einzufügen. Dann kann man den „New Section“ Knopf klicken, um ein neues Kapitel in diesem Buch zu erzeugen. Dann erscheint das Fenster „New Section Dialog“. In diesem Fenster kann man den Namen des neuen Kapitels eingeben, dann mit dem „Enter“ Knopf die Eingabe bestätigen oder mit dem „Cancel“ Knopf das „New Section Dialog“ Fenster schließen, ohne ein neues Kapitel zu erzeugen.

Die Abbildung 6.9 zeigt das „New Section Dialog“ Fenster.

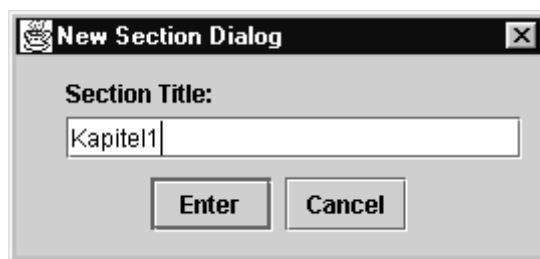


Abbildung 6.9: Ein Kapitel erzeugen

Das Beispiel in Abbildung 6.9 zeigt, daß der Benutzer ein Kapitel mit dem Namen `Kapitel1` erzeugen möchte. Der Inhalt von diesem Kapitel ist zuerst leer. Man kann dann mit dem „Put Section“ Knopf den Inhalt des Kapitels hinzufügen. Die Beschreibung des „Put Section“ Knopfs gibt es in Abschnitt 9.

In einem normalen Texteditor kann man beim Erzeugen eines Kapitels den Inhalt gleich einschreiben. Weil in unserem Pseudo-Wrapper der Inhalt eines Kapitels immer von einer Datei ausgelesen wird, ist diese Anforderung nicht notwendig. Hier könnte man die Möglichkeit anbieten, einen lokalen Dateinamen anzugeben und der Inhalt des Kapitels direkt zur Dokumenten-Plattform einzubringen. Bei der Bestätigung mit dem „Enter“ Knopf wird der Name des ausgewählten Buches in der Bücher-Liste gelesen, danach folgende Nachricht erzeugt und an den Wrapper-Server geschickt:

```
<Verteilte Systeme><E2WCREATESECTION><Kapitel1>
```

`Verteilte Systeme` ist der Name des Buches, in dem ein Kapitel `Kapitel1` eingefügt wird. Nachdem ein Kapitel erzeugt wird, wird die Kapitel-Liste des ausgewählten Buchs neu aktualisiert.

7. „Delete Section“ Knopf

Mit dem „Delete Section“ Knopf kann man ein ausgewähltes Kapitel in einem Buch löschen. Der „Delete Section“ Knopf funktioniert ähnlich wie der „Delete Book“ Knopf. Man muß zuerst das zu löschende Kapitel auswählen.

Das Beispiel in Abbildung 6.10 zeigt, daß der Benutzer das Kapitel „Umgebungen“ in dem Buch „Studienarbeit“ löschen möchte.

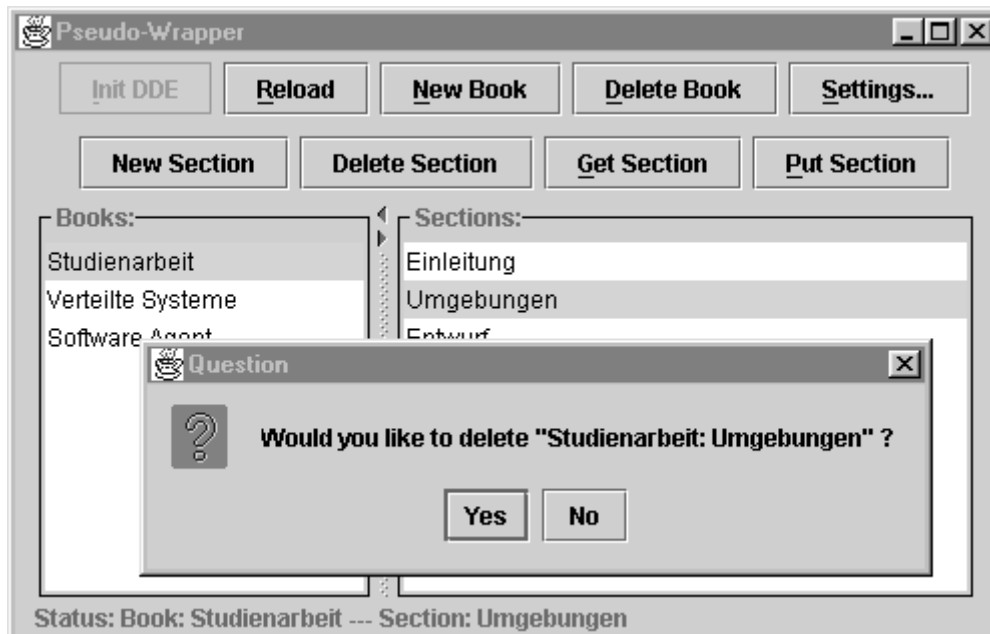


Abbildung 6.10: Ein Kapitel löschen

Nach der Bestätigung mit dem „Yes“ Knopf im Warnungsfenster wird folgende Nachricht erzeugt und an dem Wrapper-Server geschickt:

```
<BOOKSHELF><E2WDESTROYSECTION><Umgebungen>
```

Nachdem ein Kapitel gelöscht wird, wird die Kapitel-Liste neu aktualisiert.

8. „Get Section“ Knopf

Mit dem „Get Section“ Knopf kann man ein Kapitel von einem Buch in einer lokalen Datei speichern. Diese lokale Datei kann schon existieren, dann wird sie überschrieben. Ansonsten wird sie erzeugt.

Man muß zuerst das gewünschte Kapitel in der Kapitelnamen-Liste auswählen, dann den „Get Section“ Knopf drücken. Ein „Get“ Fenster, wie in der Abbildung 6.11 dargestellt, erscheint.

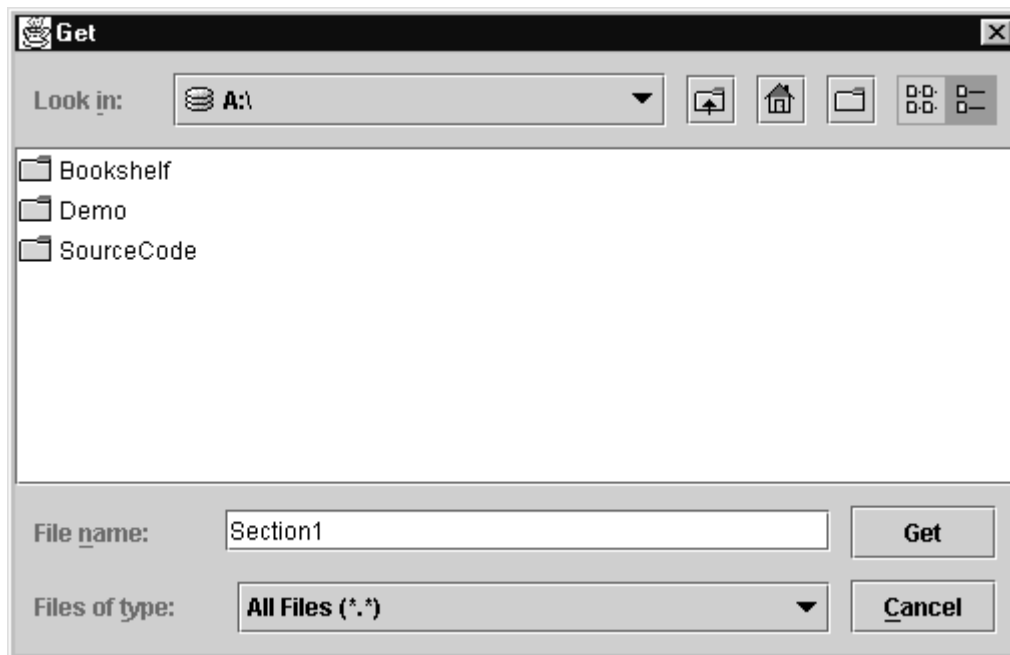


Abbildung 6.11: Ein Kapitel in einer lokalen Datei speichern

Dieses Fenster ist ein Java File Chooser Fenster. Mit diesem kann man in verschiedene Verzeichnisse der lokalen Speichermedien navigieren, eine existierende Datei auswählen oder einen Dateinamen für die neue Datei in dem Eingabefeld „File name“ eingeben. Dann kann man mit dem „Get“ Knopf den Speichervorgang durchführen oder mit dem „Cancel“ Knopf den Speichervorgang unterbrechen.

Wenn man den „Get“ Knopf drückt, wird folgende Nachricht erzeugt und an den Wrapper-Server geschickt.

```
<section><E2WREAD><>
```

Hier steht `section` für den Namen des ausgewählten Kapitels. Der Wrapper-Server schickt dann den Inhalt des Kapitels zurück, und der Pseudo-Wrapper speichert ihn in der lokalen Datei.

9. „Put Section“ Knopf

Mit dem „Put Section“ Knopf kann man den Inhalt einer lokalen Datei in ein Kapitel in einem Buch auf der Dokumenten-Plattform einbringen.

Man muß zuerst das Kapitel auswählen und dann den „Put Section“ Knopf drücken. Das folgende „Put“ Fenster erscheint.



Abbildung 6.12: Ein Kapitel auf der Dokumenten-Plattform verlegen

Dieses „Put“ Fenster ist auch ein Java File Chooser Fenster. Man kann in verschiedene Verzeichnisse der lokalen Speichermedien navigieren und die gewünschte lokale Datei aussuchen. Dann kann man mit dem „Put“ Knopf den Einbringvorgang durchführen oder mit dem „Cancel“ Knopf den Einbringvorgang unterbrechen.

Wenn man den „Put“ Knopf drückt, wird der Inhalt der ausgewählten Datei in einen String `content` gelesen. Dann wird folgende Nachricht erzeugt und an den Wrapper-Server geschickt.

```
<section><E2WRITE><content>
```

Der Wrapper-Server empfängt diese Nachricht, und ruft die entsprechende Methode auf, um den Inhalt des Kapitels zu ändern. Das Ergebnis wird dann zurück zu dem Pseudo-Wrapper geschickt und in der Status-Zeile dem Benutzer gezeigt.

6.2 Implementierung des Wrapper-Servers

In diesem Abschnitt werden zuerst die Implementierungsalternativen des Wrapper-Servers hinsichtlich der Kommunikation mit dem Pseudo-Wrapper diskutiert. Dann wird ein Resultat aus der Diskussion gegeben. Zum Schluß wird die eigentliche Implementierung des Wrapper-Servers vorgestellt.

6.2.1 Implementierungsalternativen

Für die Implementierung des Wrapper-Servers wird das Java Paket `java.net` benutzt. Das Java Paket `java.net` bietet die Klassen für die Implementierung von Netzwerk Anwendungen an. Durch die Klassen von `java.net` können die Entwickler TCP oder UDP zur Kommunikation über das Netzwerk benutzen.

Es gibt drei grundlegende Mechanismen für die Netzwerkprogrammierung:

URLs, Sockets, und Datagrams.

Außerdem unterstützt Java auch die Technologie von verteilten Objekten. Für die Anwendungen von verteilten Objekten kann man RMI (Remote method invocation) oder Java IDL benutzen.

- URL (Uniform Resource Locator)

Ein URL ist die Adresse einer Resource im Internet. URLs werden benutzt, um auf die Informationen im Internet zuzugreifen. Man kann eine Verbindung zu einem URL aufbauen, und dann von dieser Verbindung lesen oder zu dieser Verbindung schreiben.

URLs bieten einen relativ hohen Mechanismus für den Zugriff auf Ressourcen im Internet an (Dateien). Aber in unserer Implementierung müssen einzelne Nachrichten bearbeitet werden. Deswegen brauchen wir hier eine relativ niedrige Netzwerkkommunikation für die Klient-Server Anwendung.

- Datagramm

Ein Datagramm ist eine unabhängige, vollständige Nachricht. Sie wird über das Netzwerk geschickt. Ihre richtige oder rechtzeitige Ankunft ist nicht garantiert. Dieses Modell von Kommunikation über Netzwerk wird durch das UDP Protokoll unterstützt. Die `DatagramPacket` and `DatagramSocket` Klassen von Java Paket `java.net` implementieren diese system-unabhängige Datagrammkommunikation.

Mit Datagrammen kann man unsichere Klient-Server Anwendungen implementieren. Die korrekte Übertragung von Nachrichten ist wichtig für das verteilte Dokumenten-System, weil das erstellte Dokument durch die Nachrichtenübertragung in die Dokumenten-Plattform eingebracht wird. Deswegen brauchen wir hier eine zuverlässige Kommunikation zwischen dem Klient und dem Server.

- Socket

Ein Socket ist ein Endpunkt eines bidirektionalen Kommunikationskanals zwischen zwei auf dem Netzwerk laufenden Programmen. Hier wird TCP benutzt, um einen zuverlässigen, Punkt-zu-Punkt Kommunikationskanal herzustellen.

Mit Sockets kann ein Programm mit einem anderen Programm über das Netzwerk kommunizieren. Jedes Programm verbindet sich mit dem Socket an seinem Ende des Kanals. Der Klient und der Server lesen von oder schreiben auf ihr Socket, um miteinander zu kommunizieren.

Weil Sockets eine sichere Kommunikation zwischen zwei Programmen ermöglichen, sind sie für diese Studienarbeit geeignet.

- RMI (Remote method invocation)

RMI first erschien erstmals in JDK 1.1 und ermöglicht Java Anwendungen durch den entfernten Methodenaufruf über Netzwerk zu kommunizieren. Dieser Methodebasierende Ansatz zur Netzwerkkommunikation lässt einen Zugriff auf ein entferntes Objekt wie den auf ein lokales Objekt erscheinen. Wegen der Ähnlichkeit mit Java IDL wird die Einsetzbarkeit in DDE im nächsten Abschnitt mit Java IDL zusammen besprochen.

- Java IDL

Java IDL fügt der Java Platform CORBA Fähigkeit hinzu. Java IDL ist ähnlich wie RMI. RMI unterstützt nur Java geschriebene Objekte. Die von Java IDL unterstützten Objekte sind sprachunabhängig. Java IDL ist neu in JDK 1.2. Mit der von der Object Management Group (OMG) standardisierten interface definition language (IDL) und Internet Inter-ORB Protocol (IIOP) können Anwendungen transparent Operationen von entfernten Netzwerk Diensten durchführen.

Weil die Dokumenten-Plattform aus mehreren Objekten besteht, sind die Technologien für verteilte Objekte (RMI, Java IDL) in Java gut geeignet für die Erweiterung von DDE. Man braucht hier nur das passende Objekt zu

referenzieren, und dann die entsprechende Methode des Objekts aufrufen. Sie werden aber aus folgenden Gründen nicht in dieser Studienarbeit benutzt.

- DDE benutzt einen eigenen produktspezifischen Nameserver, um die Objekte zu referenzieren. Deswegen ist es problematisch, mit RMI oder Java IDL diese CORBA-Objekte zu referenzieren.
- Zur Zeit ist nur JDK 1.0 in Mobilgeräten mit Windows CE verfügbar. Deswegen können RMI und IDL nicht in Mobilgeräten eingesetzt werden.

6.2.2 Resultat

Sockets werden aus folgenden Gründen für diese Studienarbeit benutzt:

- Sockets ermöglicht eine sichere Kommunikation zwischen zwei Programmen.
- Die Kommunikation wird durch den Nachrichtenaustausch realisiert.
- Sockets sind im weiteren Umfang einsetzbar. Insbesondere können Sockets auch beim Mobilgerät eingesetzt werden.

6.2.3 Wrapper-Server

Im vorigen Abschnitt wurden die Alternativen für die Implementierung des Wrapper-Servers hinsichtlich der Kommunikationsmechanismen diskutiert, und wir haben uns für die Socket-Programmierung entschieden. In diesem Abschnitt wird der eigentliche Wrapper-Server vorgestellt.

Der Wrapper-Server ist ein typisches Server-Programm unter der Verwendung von Socket. Der Wrapper-Server hat folgende zwei Aufgaben:

- Verwalten der Socket-Verbindungen mit den Pseudo-Wrappern.
- Implementieren des Kommunikationsprotokolls mit den Pseudo-Wrappern.

Der Wrapper-Server versucht ein neues ServerSocket Objekt mit folgender Anweisung zu erzeugen, um auf dem spezifischen Port zu horchen.

```
ServerSocket serverSocket = new ServerSocket(port);
```

Wenn der Wrapper-Server das ServerSocket Object erfolgreich erzeugt hat, wartet der Wrapper-Server auf einen Verbindungswunsch von einem Pseudo-Wrapper mit folgender Anweisung.

```
Socket clientSocket = serverSocket.accept();
```

Die accept Methode wartet darauf, daß ein Pseudo-Wrapper eine Verbindung anfordert. Wenn eine Verbindung angefordert und erfolgreich aufgebaut wird, gibt die accept Methode ein neues Socket-Object zurück, das mit einem neuen Port verbunden ist. Der Wrapper-Server kann mit dem Pseudo-Wrapper durch diese neue Socket kommunizieren und den Horchvorgang mit dem originalen Port für mögliche weitere Anforderungen fortsetzen.

Der Pseudo-Wrapper und der Wrapper-Server können jetzt durch Schreiben zu oder Lesen von ihren Sockets kommunizieren. Der Pseudo-Wrapper implementiert das Kommunikationsprotokoll wie folgt:

- Der Pseudo-Wrapper baut Corba-Verbindungen mit den DDE-Servern auf.
- Anhand der Nachricht von einem Pseudo-Wrapper ruft der Wrapper-Server die entsprechende Funktion des DDE-Servers auf.
- Die Ergebnisse werden als Nachricht an den Pseudo-Wrapper über die Socket-Verbindung zurückgeschickt.

7 Bewertung und Zusammenfassung

Diese Studienarbeit beschäftigte sich mit der Erweiterung der DDE der Art, daß Benutzer mit anderen Texteditoren als Emacs von verschiedenen Betriebssystemen DDE benutzen können. Insbesondere wurde der Einsatz von Mobilgeräten betrachtet. Mit dieser Erweiterung werden für die Benutzung von DDE nur folgende Anforderungen benötigt:

1. Java Umgebung, die auch auf Windows CE lauffähig ist
2. Eine physikalische Verbindung mit der Dokumenten-Plattform
3. Netzwerkprotokoll für die hardwareunabhängige Kommunikation

Für die Erweiterung wurden in der Autorenumgebung ein Pseudo-Wrapper und auf der Dokumenten-Plattform ein zusätzlicher Server „Wrapper-Server“ entwickelt.

Diese Erweiterung hat zwei wesentliche Nachteile:

1. Der Benutzer kann nicht direkt über den gewohnten Texteditor DDE benutzen, sondern muß einen Umweg über den Pseudo-Wrapper nehmen.
2. Weil der Pseudo-Wrapper CORBA nicht benutzt, ist das System nicht mehr rechnertransparent. Der Pseudo-Wrapper muß wissen, auf welchem Rechner der Wrapper-Server gerade läuft.

8 Ausblick

Die schnelle Entwicklung von Rechnernetzen (Internet, Intranet) verbindet heutzutage fast jeden Computer auf dem Arbeitsplatz oder zuhause. Es ist möglich, daß Menschen weltweit durch Rechnernetze miteinander kommunizieren. Die effiziente Nutzung der Rechnernetze wird aber noch nicht erreicht.

Der Einsatz von Groupware, z.B. Mehrbenutzereditoren, Telekonferenzsystemen, Workflow Management Systemen, usw., ermöglicht große Fortschritte in der effizienten Nutzung der Rechnernetze. Bessere Groupware Systeme in verschiedenen Anwendungsbereichen werden in Zukunft entwickelt.

Ein besseres Groupware System basiert auf besserer Middleware und darunterliegenden Netzwerktechnologien. Zum Beispiel, die Unterstützung von Multicast, Mobil IP in IP Version 6 bietet neue Möglichkeiten für die Entwicklung von Middleware bzw. Groupware und Anbindung von mobilen Geräten.

DDE kann aus meiner Sicht in folgenden Aspekten weiter entwickelt werden.

- Algorithmen für die Konsistenzerhaltung

Viele Verfahren gegen syntaktische Inkonsistenz haben unterschiedliche Vor-/Nachteile. Es ist wünschenswert, daß das System mehrere Verfahren anbietet.

Die Behandlung von semantischen Inkonsistenzen wird immer noch nicht gelöst.

- Dokumentenstandard

Die Benutzung von XML wird in DDE vorgesehen, aber noch nicht tatsächlich realisiert. Es scheint, daß XML sich in der Praxis durchsetzen wird. Es soll in dieser Richtung weiter entwickeln werden.

- Behandlung von Multimedia-Daten

Heutzutage benutzt man immer mehr Multimedia-Daten. Es besteht der Bedarf, gemeinsam Multimedia-Daten zu bearbeiten.

Man kann Multimedia-Dateien in DDE gleich wie Text-Dateien behandeln. Es werden jedoch andere Maßnahmen zur Konsistenzerhaltung benötigt.

- Sicherheitsmaßnahmen

In einer vernetzten Rechnerumgebung sind die Sicherheitsaspekte immer wichtig. Insbesondere dann, wenn man das System in der Praxis einsetzt. Eine schwache Stelle für die Sicherheit ist die Verbindung zwischen dem Pseudo-Wrapper und dem Wrapper-Server. In dieser Stelle sollen Sicherheitsmaßnahmen eingebaut werden.

- Mobilgeräte

Die Hardware und Software von Mobilgeräten werden weiter verbessert. Wegen der Mobilität und verbesserten Fähigkeiten werden Mobilgeräte in Zukunft immer mehr eingesetzt, auch in Groupware Systemen.

DDE basierend auf CORBA zeigt einige Probleme, z.B. die große Anzahl von Objekten und Schwierigkeiten bei der Teamteilnehmerverwaltung. Eine DDE, die auf Mobile Agenten Systeme basiert, wird gerade in einer Diplomarbeit entwickelt und untersucht. Die Vor- und Nachteile von dem Einsatz des Mobile-Agenten-Systems sind in [Lukas] zu finden.

Groupware verändert die traditionale Arbeitsweise von Menschen. Es ist aber zu beachten, daß die normale menschliche Kommunikation nicht ganz durch Rechner ersetzt werden kann.

Literaturverzeichnis

- [Bur97] Burger, Cora (1997)
Groupware - Kooperationsunterstützung für Verteilte Anwendungen
dpunkt, Heidelberg, 1997
- [GNU91] GNU (1991)
GNU Emacs Manual Edition 2.0
- [Mic95] Microsoft Corporation (1995)
Microsoft Word Developer's Kit
Microsoft Press Deutschland
- [Schr99] Schramm, Oliver (1999)
Entwurf eines verteilten, corbabasierten Dokumentensystems
Universität Stuttgart, Fakultät Informatik, Diplomarbeit Nr. 1708
http://www.informatik.uni-stuttgart.de/cgi-bin/ncstrl_rep_view.pl?inf/ftp/pub/library/medoc.ustuttgart_fi/DIP-1708/DIP-1708.bib
- [Schu99] Schurr, Peter W. (1999)
Erweiterung von DDE (Distributed Document Environment) um Konsistenzbehandlung in mobiler Umgebung
Universität Stuttgart, Fakultät Informatik, Diplomarbeit Nr. 1778
http://www.informatik.uni-stuttgart.de/cgi-bin/ncstrl_rep_view.pl?inf/ftp/pub/library/medoc.ustuttgart_fi/DIP-1778/DIP-1778.bib
- [Sun99] Sun Microsystems, Inc. (1999)
Java-Online-Documentation
<http://java.sun.com/doc/>
- [Tan96] Andrew S. Tanenbaum (1996)
Computer Networks 3rd Ed
Prentice Hall, Inc.
- [Lukas] Lukas Weberruss (2000)
Erweiterung von MoleOffice um gemeinsame Dokumentenbearbeitung
Diplomarbeit

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfaßt und
nur die angegebenen Hilfsmittel verwendet habe.

(Bo Wu)