Konzeptionierung und Implementierung einer Client-Server-Umgebung für Projektdokumentationen

Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Informatiker

an der
Fachhochschule für Technik und Wirtschaft Berlin

Fachbereich Wirtschaftswissenschaften II
Studiengang Angewandte Informatik

1. Betreuer: Prof. Dr. Albrecht Fortenbacher
2. Betreuer: Dipl.-Inf. (FH) Torsten Buller

Eingereicht von Matthias Kraft
am 11. Mai 2001

Danksagung Ich möchte mich an dieser Stelle ganz herzlich bei all den netten Menschen bedanken, die mich in den vergangenen Monaten auf die unterschiedlichste Art und Weise unterstützt haben. Zuerst sind da meine beiden Betreuer zu nennen: Prof. Dr. Albrecht Fortenbacher von der FHTW Berlin und Dipl. Inf. (FH) Torsten Buller von BertelsmannSpringer Science+Business Media. Die beiden haben diese Arbeit überhaupt erst ermöglicht! Ferner möchte ich meinen Kollegen bei BertelsmannSpringer für die vielen Diskussionen über das Design und zu verwendende Technologie danken. Besonders Andre Anneck, Dr. Dragan Macos und Dipl. Inf. Stefan Fabricius haben mir immer wieder geduldig geholfen! Fürs Korrekturlesen und viele wertvolle Anregungen möchte ich mich außerdem bei meinen Eltern bedanken! Ganz besonders bedanken möchte ich mich bei meiner Freundin Gesine Schröter. Sie hat unglaublich viel Geduld mit mir gehabt und viele Tips und Diskussionen zum Thema beigesteuert!

Satz: mit LaTeX 2_ε . Verwendung fanden unter anderem die Pakete `fancyhdr`, `pdftex`, `natbib` und `bibtex`. Als Bibliographiestyle kommt `dinat` zum Einsatz. Die PDF-Version dieser Arbeit ist - dank `hyperref` - vollverlinkt. Die UML-Diagramme wurden aus `TogetherJ 4.2` übernommen. Andere Grafiken wurden zum größten Teil mit `StarOffice 5.2` erstellt.

Vorwort

Bereits vor meiner Diplomarbeit war ich etwa $1\frac{1}{2}$-Jahre für die Firma BauNetz GmbH & Co. KG (im weiteren Text kurz BauNetz genannt (Web:BauNetz)) tätig. Die Firma gehört zur Verlagsgruppe BertelsmannSpringer Science+Business Media (kurz BertelsmannSpringer (Web:BSSBM)) und hat ihren Sitz im gleichen Haus, wie auch die zentrale Technik des Bereichs eBusiness der Verlagsgruppe. Aus der Tätigkeit bei BauNetz (hauptsächlich Entwicklung diverser dynamischer Websysteme) kristallisierte sich dann die Aufgabenstellung (ab Seite 1) heraus. Für die Durchführung der Diplomarbeit wechselte ich deshalb in die Zentraltechnik von BertelsmannSpringer.

Nach Schulz (1999) gehört BertelsmannSpringer

> [...] zu den weltweit führenden Verlagsgruppen für Wissenschafts- und Fachliteratur und ist in Deutschland mit Abstand die Nummer 1. Mit 70 Verlagen und Niederlassungen ist die Gruppe in 18 Ländern in Europa, den USA und Asien präsent.
>
> [...] Die Produktpalette spiegelt das gesamte Medienspektrum wider: von der Fachzeitschrift über Bücher, Newsletter, Seminare, Kongresse und Fernkurse bis zu CD-Roms, Datenbanken und Online-Diensten. [...]
>
> Das Verlagsprogramm umfaßt rund 25.000 lieferbare Buchtitel und 700 Zeitschriften.

Derzeit arbeiten ca. 5000 Mitarbeiter für die Verlagsgruppe. Das Unternehmen erwirtschaftete 1999 ca. 1,34 Mrd. DM Umsatz.

Die Verlagsgruppe BertelsmannSpringer ist erst zum 1. Januar 1999 entstanden, als Bertelsmann 86,5% des wissenschaftlichen Springer Verlags übernahm. Zur Struktur des Unternehmens heißt es weiter:

> Seit dem 1. Oktober 1999 sind die beiden Verlagsgruppen organisatorisch unter einem Dach zusammengeführt. Das Unternehmen ist in zwei Bereiche gegliedert - STM (Science, Technology and Medicine) und B-to-B (Business to Business). STM umfaßt die Wissenschaftsverlage Springer, Birkhäuser und Steinkopff, Medizinverlage wie Urban & Vogel, Groupe Impact Médecin oder die

Ärzte Zeitung, die GWV Fachverlage Gabler, Westdeutscher Verlag und Vieweg sowie die Newsletter Fuchsbriefe oder Platow Brief. Im Unternehmensbereich Business to Business wurden alle Bau- und Verkehrsverlage wie beispielsweise Bertelsmann Fachzeitschriften, Heinze, Heinrich Vogel und Codes Rousseau zusammengefaßt.

Im Bereich eBusiness sind alle Online-Dienste und -Aktivitäten der Fachverlagsgruppe konzentriert. Zu den bekanntesten Marken gehören LIFELINE, BauNetz, Autohaus Online und das Science-Portal LINK.

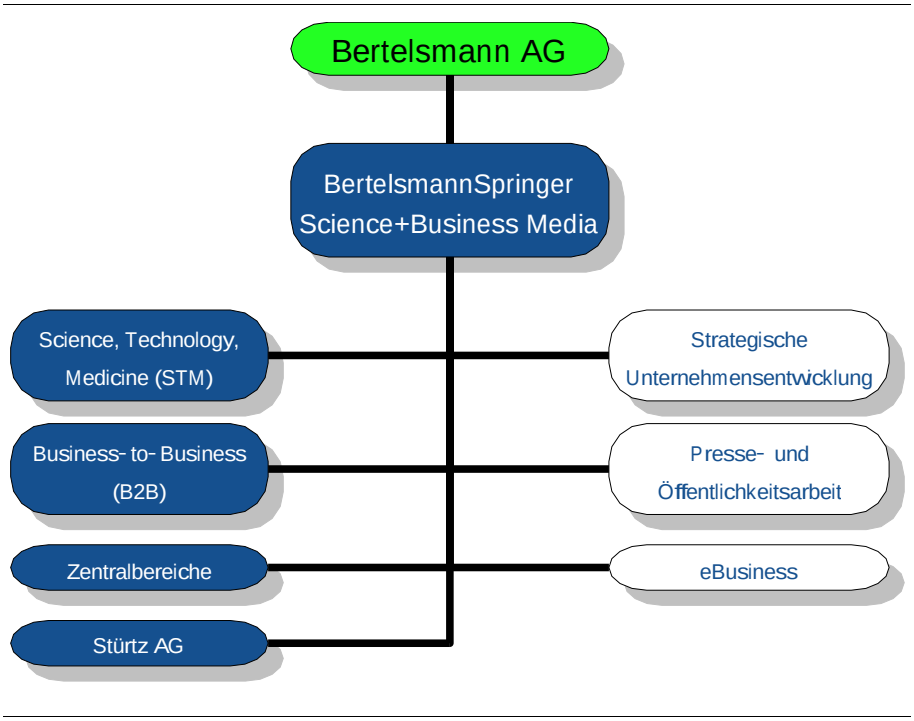

Abbildung 1: BertelsmannSpringer Unternehmensstruktur

Die Abbildung 1 umreißt die grobe Unternehmensstruktur. Auf der linken Seite finden sich die angegliederten Verlage und die zentralen Bereiche der Verwaltung. Auf der rechten Seite schließlich sind eigenständige Unternehmensbereiche die Resourcen für alle Unternehmensteile zur Verfügung stellen, wie zum Beispiel auch eBusiness.

Ein Auszug aus der Chronik von BertelsmannSpringer (siehe Schulz 1999):

1953 Gründung des Verlags "Bertelsmann Fachzeitschriften".

1970 Aufbau des Fachbereichs Verkehr durch Übernahme des Verlags Heinrich Vogel.

1973 Einstieg in den Bereich Medizin durch Akquisition des Münchener Medizin Verlags.

1978 Eingliederung des auf Wirtschaftsthemen spezialisierten Gabler Verlags.

1985 Entwicklung der ersten elektronischen Medien.

1990 Erster Schritt zur Internationalisierung mit Gründung des spanischen Fahrschulverlags Etrasa.

1996 Einrichtung einer Internetplattform und Aufbau des ersten Online-Dienstes (BauNetz).

1998 Ausgliederung aus der Buch AG. Die "Bertelsmann Fachinformation" ist jetzt eigenständiger Geschäftsbereich und dem Vorstandsvorsitzenden der Bertelsmann AG direkt zugeordnet.

1999 Erwerb der Mehrheitsbeteiligung am wissenschaftlichen Springer-Verlag, Berlin / Heidelberg.

Inhaltsverzeichnis

1 Aufgabenstellung

Ziel der Diplomarbeit ist es, eine Umgebung für die Dokumentation von Projekten zu erstellen. Die Lösung soll dabei die verschiedenen Projektteilnehmergruppen, wie z. B. Projektmanagement, Softwareengineering und Administration, entlasten. Damit soll ermöglicht werden, sich in einem vorgegebenen Rahmen auf das Wesentliche – nämlich die Texteingabe – zu konzentrieren. Damit sollen vor allem die Softwareentwickler entlastet werden, die den größten Anteil an Dokumentation zu schreiben haben.

Die zu erstellende Umgebung soll anschließend in eine größere allgemeine Dokumentenverwaltung eingebunden werden. Letztere ist jedoch nicht Bestandteil der Diplomarbeit.

Um die Dokumentationen projektübergreifend zu vereinheitlichen, ist es notwendig verbindliche Richtlinien vorzuschreiben. Darin wird geregelt, was von wem in welchem Umfang zu dokumentieren ist. Die zu erstellende Umgebung soll die vorgegebenen Richtlinien berücksichtigen. Die Erstellung dieser Richtlinien ist nicht Bestandteil dieser Diplomarbeit.

Da bereits Dokumentationen zu abgeschlossenen bzw. noch laufenden Projekten vorhanden sind, ist zu prüfen, ob diese sinnvoll in die zu erstellende Umgebung überführt werden können.

Für die im Folgenden genannten Eckpunkte soll die Umgebung eine Lösung bereit stellen:

- Erstellung und Speicherung der Dokumentationen in XML-Formaten[1] mit vorgegebener Struktur,
- Revisionssichere Speicherung,
- Regelung des Zugriffs auf die Dokumente zentral über eine Datenbank,

[1] Abkürzung: E**X**tensible **M**arkup **L**anguage (Definition siehe Eckstein 1999).

- Sicherstellung einer korrekten Authentifizierung,
- Gewährleistung einer gesicherten Datenübertragung mittels Verschlüsselung.

Besondere Aufmerksamkeit ist der Validierung der XML-Daten zu widmen. Die Validierung stellt sicher, dass die Dokumente nicht nur korrekt nach der XML-Spezifikation sind (*well-formed*), sondern sich auch an die vorgegebene Struktur halten (*valid*; siehe auch McLaughlin 2000). Es ist zu prüfen, auf welchem Wege die Validierung zu erreichen ist und ob bereits vorhandene DTDs[2] genutzt werden können.

Aus dem Vorangegangenem soll ein einfaches Werkzeug resultieren, das die Nutzer an die Hand bekommen, um damit auf die Dokumente zuzugreifen und diese editieren zu können. Das Werkzeug soll sie möglichst durch die, per Richtlinie vorgeschriebenen, zu dokumentierenden Abschnitte führen und selbstständig gültige XML-Dokumente erstellen. Für Nutzer mit entsprechendem Hintergrundwissen ist eine Import/Export-Funktion vorzusehen, die ihnen Zugang zu den XML-Rohdaten eines Dokuments ermöglicht.

[2] Abkürzung: **D**ocument **T**ype **D**efinition; In Eckstein (1999): "[A DTD] provides grammar rules for the document [...] ".

2 Einführung

Die Aufgabenstellung entspringt zu einem großen Teil dem Wunsch, dem aktuell unbefriedigenden Zustand, in dem sich die Dokumentationen zu Projekten befinden, abzuhelfen. In die Aufgabe sind einerseits eigene Erfahrungen und Erfahrungen anderer Mitarbeiter und andererseits Erkenntnisse aus Büchern wie Heitig (1984) und David (1996) eingeflossen.

2.1 Eigene Erfahrungen

Die Arbeit bei BauNetz hat unter anderem auch Erfahrungen mit der Realität von Projektarbeit und -dokumentation mit sich gebracht. Auffallend dabei waren vor allem die geringe Wertschätzung von Dokumentationsarbeit und die daraus folgende lückenhafte und teilweise auch fehlerhafte Dokumentation der Projektinhalte und -ergebnisse. Ursachen für fehlerhafte Dokumentation lagen neben der geringen Wertschätzung auch in ständig wechselnden Projektinhalten und dem engen Zeitrahmen der Projekte. Ein typisches Projekt lief meistens folgendermaßen ab:

1. Auftrag erteilt (meist per Email oder mündlich)
2. Bei größeren Projekten: Meeting, um die Ziele festzuhalten und die technischen Möglichkeiten auszuloten
3. Termin festgesetzt, zu dem Produktionsreife erreicht sein soll
4. Bearbeitung (Design, Programmierung)
5. Sobald die Software einen Grad der Benutzbarkeit erreicht hatte, wurde das nächste Projekt beauftragt.

Zeit für Tests und Dokumentation war fast nie gegeben. Auch wurde wegen enger Terminsetzung die Designphase meist abgekürzt.

Diese Art der Projektabwicklung ist jedoch sehr unbefriedigend: Wenn die Projekte überhaupt dokumentiert werden, dann immer im Nachhinein, nur unvollständig, in keinem vorgegebenen Rahmen, unterschiedlich umfangreich oder auch in proprietären Dateiformaten wie z. B. Microsoft Word. Problematisch wird die unzureichende Dokumentation, wenn nach einiger Zeit Änderungen an den Projektergebnissen (wie z. B. einer erstellten Software) erforderlich werden. Ist keine Dokumentation vorhanden und der Entwickler gerade nicht oder nicht mehr verfügbar, wird das gesamte Projekt oftmals mit den neuen Anforderungen neu aufgesetzt.

Solche Beobachtungen hat offenbar auch David gemacht. Er schreibt in David (1996, S. 18):

> Unter den allgegenwärtigen Pressuren der Fachbereiche und der Präsenz des Anwendungsstaus verlangt das Management ein schnelles Ergebnis und übersieht dabei die Konsequenzen, die sich für die Weiterentwicklung aus einer nicht vorhandenen Dokumentation ergeben. Diese Handlungsweise ist fatal, weil die Einarbeitung in die Software unter diesen Umständen mehr Zeit in Anspruch nimmt und zusätzliche Kosten verursacht.

Diese Praxis ist aber auch einem Mangel an Entwicklern geschuldet und beginnt sich bei BertelsmannSpringer erst jetzt – mit der Gründung und dem Wachsen einer zentralen Entwicklungsabteilung – langsam zu entspannen. Für die Bestätigung der geschilderten Beobachtungen wurden einige Interviews mit Mitarbeitern aus der Entwicklungsabteilung und dem Projektmanagement geführt.

2.2 Zusammenfassung der Mitarbeiterinterviews

Mit Hilfe der Interviews sollte analysiert werden, wie weit an Dokumentation bei der Beauftragung bzw. Initiierung von Projekten gedacht wird.

Befragt wurden sechs Mitarbeiter, drei Projektmanager (darunter der Leiter der Abteilung), zwei Entwickler (inklusive dem Leiter) und der Leiter der gesamten Technik. Der Leiter der Projektmanager und der Leiter der Entwicklungsabteilung waren relativ neu im Team, so dass nach ihren Erfahrungen in ihren früheren Anstellungen gefragt wurde. Insgesamt wurden damit 13 unterschiedliche Projekte untersucht (inklusive sechs noch nicht abgeschlossenen Projekten). Die Befragung wurde vor Einführung der Dokumentationsrichtlinien (siehe "2.4 Arten von Projektdokumentation" auf der Seite 9) durchgeführt.

Neben dem Aspekt ob und wie Dokumentation beauftragt wird, wurde auch nach der Größe der Projekte gefragt. Interessant war dabei die Anzahl der Projektteilnehmer und die geplante bzw. tatsächliche Laufzeit der Projekte. Zusammenfassend lässt sich dazu feststellen, dass – mit einer Ausnahme – alle Projekte mit mindestens zwei und maximal 10 Teilnehmern durchgeführt wurden.

Eine Ausnahme bildet ein Projekt des Leiters der Entwicklungsabteilung, an dem dieser beteiligt war, bevor er zu BertelsmannSpringer wechselte. Das Projekt umfasste mehr als 200 Teilnehmer. Dieses Projekt fällt mit 2,5 Jahren auch im Laufzeitumfang auf. Die durchschnittliche geplante Projektdauer der restlichen Projekte beträgt etwa 4,3 Monate. Bei mindestens vier Projekten wurde der gesteckte Zeitrahmen überschritten. Bei zwei in der Zeit fertig gewordenen Projekten wurden die Testphasen gekürzt bzw. die Dokumentation aus der Abnahme ausgeklammert.

Dokumentationen waren bei fast allen Projekten in der einen oder anderen Form vorhanden. Bei fünf Projekten wurde sie nicht beauftragt und war daher nicht oder nur spärlich (z. B. in Form von Sourcecode-Kommentaren oder Online-Hilfe) vorhanden. Dies betraf vor allem die Projekte, die vollständig im Haus durchgeführt wurden.

In den anderen Projekten wurden sehr unterschiedliche Dokumentationen beauftragt (in Klammern die Anzahl der Projekte für die diese Art der Dokumentation beauftragt wurde):

- Handbücher für Konfiguration (1), Nutzer (8) oder Administration (1)
- Technische Dokumentation (8)
- Kommentierter Sourcecode (4)
- Online-Hilfe (2)
- Consultingleistungen (3)
- Sonstige, z. B. Präsentationen, Protokolle o. ä. (4).

Auf Grund der nicht vorhandenen Richtlinien wurde dem jeweiligen Auftragnehmer bei den Punkten "Technische Dokumentation" und "Kommentierter Sourcecode" überlassen, wie und was er dokumentiert. Lediglich ein Projektmanager gab an, dass er die Dokumentationen vor Abschluss des Projekts durchgesehen und Nachbesserungen eingefordert hat.

Diese Verhaltensweisen sind offenbar schon ziemlich alt. Heitig schrieb bereits 1984:

> Häufig ist derzeit - wenn überhaupt - die Dokumentation zu Datenverarbeitungsanwendungen (Software) nur in Teilbereichen realisiert. Was fehlt, ist die Durchgängigkeit von der Entwicklung der

> Software her bis zu deren Einsatz auf den Datenverarbeitungsanlagen und dem Gebrauch durch die Anwender.

Dies wirft direkt die Frage nach Form und Art der Dokumentation auf.

2.3 Anforderungen an die Dokumentation

Heitig macht außerdem noch einen Mangel an Einsicht in die Notwendigkeit der Erstellung von Dokumentation als eine entscheidende Ursache für die niedrige Dokumentationsdisziplin aus. Er schreibt dazu in Heitig (1984, S. 147):

> Ein weiterer Grund für die so häufig fehlende Dokumentation scheint auch noch darin zu liegen, daß sie nicht beim aktuellen täglichen Geschehen, sondern zumeist erst im Fehlerfall oder bei sonstigen Störungen herangezogen wird und sich dann in der Regel als entweder nicht vorhanden oder als nicht aktuell, nicht aussagefähig, schlecht lesbar und für den Zweck daher als nicht verwendbar darstellt.

Diese Aussage lässt vermuten, dass der Glaube vorherrscht, dass in der Realität Fehler und Störungen in den Projektresultaten (wie z. B. Software) so selten auftreten, dass nicht verwendbare Dokumentation nur unbedeutende negative Auswirkungen auf den Betriebsablauf hat. Dem ist jedoch nicht so und zu dieser Einsicht gelangen nach und nach auch die Entscheider, weshalb bei BertelsmannSpringer nun auch an verbindlichen Dokumentationsrichtlinien gearbeitet wird. Ferner nennt Heitig in dem Zitat schon die Grundanforderungen an eine Dokumentation, die erfüllt sein müssen, damit sie überhaupt brauchbar ist:

- aktuell,
- aussagefähig und
- lesbar.

Bei einer weitergehenden Analyse unterscheidet er die Anforderungen, die sich von "gesetzlichen Vorschriften" ableiten, von denen die aus "betrieblichen Erfordernissen" entstehen.

2.3.1 Gesetzliche Vorschriften

An gesetzlichen Vorschriften führt Heitig § 44 Abs. 1 HGB und § 147 Abs. 1 AO an, nach denen die Ordnungsmäßigkeit gefordert wird. § 44 HGB ist in der

Zwischenzeit aufgehoben worden. Da zum Inhalt des Paragraphen nur die Ordnungsmäßigkeit erwähnt wird, kann auch § 238 Abs. 1 in Zusammenhang mit § 241 Abs. 2 HGB herangezogen werden. Die verweisen darauf, dass die Pflicht zur ordnungsmäßigen Buchführung über Vermögen und Geschäftsvorfälle besteht und dass jedes Inventar nach Art, Wert und Menge feststellbar sein muss. Software gehört zweifelsohne zum Inventar einer Firma. Die Einschätzung von Art und Wert einer Software kann nur erfolgen, wenn eine Dokumentation vorhanden ist, aus der der Umfang und die Funktionalität der Software hervorgeht. Das gilt erst recht, wenn es sich nur um Software-Komponenten handelt. Im Wortlaut heißt es im oben angeführten Gesetz (Auszug aus dem Handelsgesetzbuch (HGB) vom 10. Mai 1897 (RGBl S. 219), zuletzt geändert durch Art. 1 Nr. 7a des Gesetzes vom 19. Dezember 1998 (BGBl. I S. 3836)):

> § 238 Buchführungspflicht
> (1) Jeder Kaufmann ist verpflichtet, Bücher zu führen und in diesen seine Handelsgeschäfte und die Lage seines Vermögens nach den Grundsätzen ordnungsmäßiger Buchführung ersichtlich zu machen. Die Buchführung muß so beschaffen sein, daß sie einem sachverständigen Dritten innerhalb angemessener Zeit einen Überblick über die Geschäftsvorfälle und über die Lage des Unternehmens vermitteln kann. Die Geschäftsvorfälle müssen sich in ihrer Entstehung und Abwicklung verfolgen lassen.
>
> § 241 Inventurvereinfachungsverfahren
> (2) Bei der Aufstellung des Inventars für den Schluß eines Geschäftsjahrs bedarf es einer körperlichen Bestandsaufnahme der Vermögensgegenstände für diesen Zeitpunkt nicht, soweit durch Anwendung eines den Grundsätzen ordnungsmäßiger Buchführung entsprechenden anderen Verfahrens gesichert ist, daß der Bestand der Vermögensgegenstände nach Art, Menge und Wert auch ohne die körperliche Bestandsaufnahme für diesen Zeitpunkt festgestellt werden kann.

Weiterhin stellt Heitig die Zulässigkeit von Dokumentation durch § 6 Anlage 1, Ziffer 7 BDSG und §§ 26, 29 BDSG fest. Jedoch hat es auch hier in den letzten Jahren Überarbeitungen gegeben. Mit "§ 6 Anlage 1, Ziffer 7" ist der 7. Punkt der so genannten "10 Gebote der Datenverarbeitung gemeint". Man findet ihn heute in der Anlage zu § 9 BDSG. § 26 BDSG ist ebenfalls nicht mehr zutreffend. An seine Stelle treten – auf Grund der Feststellung der Zulässigkeit – die §§ 27-29 BDSG. Da laut Gerling (1999) das BDSG im Zuge einer Anpassung an europäische Richtlinien demnächst wiederum grundlegend überarbeitet werden soll, sind die genannten Paragraphen im Anhang unter "B Ausgewählte Paragraphen des BDSG" auf der Seite 79 ff. aufgeführt.

Heitig hält zusammenfassend fest, dass "vorrangig die Ordnungsmäßigkeit (HGB, AO) und die Zulässigkeit (BDSG) gefordert" wird. Er führt weiter aus:

> Der Gesetzgeber nennt dabei oftmals konkret keine besonderen Formen oder Inhalte der Dokumentation. [...] [Der Dokumentationspflichtige] kann davon ausgehen, daß er sein Dokumentationssystem so aufbauen darf, wie er es für zweckmäßig erachtet. Sein wesentlicher Vorteil ist demnach der, daß er völlig frei in der Wahl seiner Mittel und Wege ist, solange er zur Datenverarbeitungsanwendung den Nachweis
>
> - vollständig
> - schlüssig
> - in angemessener Zeit
> - und in einer Art und Weise, daß ein sachverständiger Dritter folgen kann
>
> zu führen in der Lage ist.

2.3.2 Betriebliche Erfordernisse

Die betrieblichen Erfordernisse ergeben sich Heitigs Ausführungen zu Folge "aus den Zielen des Unternehmens, der Abhängigkeit von der Datenverarbeitung, der Branche, dem Wettbewerb gegenüber Mitkonkurrenten und ähnlichen Überlegungen".

Resultierend aus den oben formulierten Anforderungen stellt er dann eine Kriterienliste auf, an der sich die Dokumentation orientieren sollte. Es seien an dieser Stelle nur die wichtigsten Punkte aus dieser Liste genannt:

aktuell Die Beschreibung der Software entspricht exakt dem Stand der beschriebenen Software.

eindeutig Die Dokumentation enthält keine Widersprüche und bedarf zum Verständnis auch keiner weiteren Interpretation oder sonstiger Erklärungen.

knapp Es sind nur jene Informationen enthalten, die notwendig sind.

selbstbeschreibend Die Dokumentation enthält alle Informationen, die erforderlich sind, um sie verstehen zu können. Dies können sein:

- selbsterklärende Überschriften
- Inhaltsverzeichnis
- Stichwortverzeichnis
- Begriffsbestimmungen.

übersichtlich Der Aufbau der Dokumentation und die Formen der Darstellung sind so gestaltet, dass einem Benutzer die Möglichkeit eröffnet wird, sich in kurzer Zeit einen Überblick zu verschaffen.

vollständig Vollständigkeit ist gegeben, wenn die Bedürfnisse aller an einer Dokumentation interessierten Stellen durch die vorhandenen Informationen erfüllt sind.

zugriffsgerecht Die Unterlagen sind geordnet an bestimmten Stellen vorhanden.

David geht einen Schritt weiter und sieht die Dokumentation bereits als integralen Bestandteil der Software. Er ordnet die "Dokumentation und Dokumentenmanagement als gleichwertig neben den Software Engineering Methoden" ein und stellt in David (1996, S. 49 ff.) fest:

> Dokumentation ist für ein Softwaresystem integraler Bestandteil des Produktes – sie ermöglicht die Transparenz der Entwicklung(en) insbesondere auch die der temporär versetzten – und ist auf keinen Fall nur zeitraubende, unproduktive Beschäftigung. Dokumentation von Software muß im Vorgehensmodell und Erstellungsprozeß entsprechend eingebunden und beachtet werden. Ein innovatives Dokumentenmanagement muß über die "klassischen" Dokumente des Software Engineering hinaus auch die Dokumente der Kooperation enthalten und unterstützen.

Er schließt seine Betrachtungen zum Dokumentenmanagement mit einer wichtigen weiteren Anforderung. Dazu zitiert er aus Ayer und Patrinostro (1992): "[. . .] it should be documented as the system is being developed, not retrospectively[. . .] ".

Damit wurden nun eine Reihe von Anforderungen genannt. Nach einer kurzen Einführung über allgemeine Arten der Dokumentation, zeigt der folgende Abschnitt die bei BertelsmannSpringer schon gültigen Vorschriften zur Dokumentation von Software. Verglichen mit den oben genannten Anforderungen, lagen die Schwerpunkte bei der Erstellung der Richtlinien in der Eindeutigkeit, Knappheit, Übersichtlichkeit und Vollständigkeit.

2.4 Arten von Projektdokumentation

In David (1996, S. 61 ff.) werden allgemeine Arten von Softwaredokumenten genannt. David stützt sich dabei auf die "Softwaredokumentation nach Sommerville" und erweitert die von Sommerville vorgeschlagene Klassifizierung in

- Softwareentwicklungsdokumente und

- Benutzerdokumente um
- Administrationsdokumente.

Laut David sind die *Softwareentwicklungsdokumente* alle Dokumente "die der Softwareentwickler zur Herstellung und Bearbeitung von Software benötigt". Er nennt dazu eine Reihe von Dokumenten, die die verschiedenen Phasen eines Vorgehensmodells widerspiegeln. Seiner Meinung nach "ist die schrittweise Verfeinerung der Ergebnisse über die Phasen" hinweg kennzeichnend für diese Art der Dokumente. Daraus resutiert auch eine "enge Verknüpfung der Dokumente miteinander".

In den *Benutzerdokumenten* sind "die Informationen zum sicheren Betreiben und Umgang mit dem Softwaresystem" enthalten. Er nennt zum Beispiel Tutorial und Hilfesystem und anders als seine oben genannte Einteilung vermuten lässt, auch die Dokumente für die Systemadministratoren (z. B. Installations- und Einführungsanweisungen etc.).

Die *Administrationsdokumente* sind deshalb "die Papiere und Dokumente, die die Durchführung der Softwareerstellung steuern und verwalten und der Kommunikation im Team dienen". Als Beispiele für diese Art der Dokumente führt er u. a. den Projektauftrag, Projektsteuerungs- und -planungsunterlagen, Notizen und Protokolle auf.

Die nun folgenden BertelsmannSpringer Richtlinien für die technische Dokumentation in Softwareprojekten beschreiben *Softwareentwicklungsdokumente*.

Diese Richtlinien, die an dieser Stelle kurz aufgezählt und erläutert werden, wurden vom Leiter der Entwicklungsabteilung in Zusammenarbeit mit den Projektmanagern und der Geschäftsführung erarbeitet und sind bindender Bestandteil jedes neuen Softwareprojekts.

Die BlackBox-Dokumentationsrichtlinie wird in Macos (2000b) vollständig beschrieben und unter "2.4.2 Die BlackBox-Dokumentationsrichtlinie" auf der Seite 13 ausführlicher vorgestellt. Die Funktionalität, der aus dieser Arbeit resultierenden Software, wird anhand dieser Richtlinie entworfen und implementiert.

Die White Box Dokumentationsrichtlinie wird in Macos (2000a) vollständig beschrieben. Die folgenden Punkte sind eine knappe Zusammenfassung:

- Beschreibung der Abhängigkeit von anderen Komponenten.
- Design-Aspekte sollen durch folgende Diagramme verdeutlicht werden:
  - Package Diagramme,
  - Klassendiagramme,

  - State-Diagramme und
  - Activity-Diagramme.

- Darstellung der Reaktionen der Komponente auf unterschiedliche Interface-Szenarien durch Sequence-Diagramme.
- Definition der für die Komponente spezifizierten Fehler.
- Beschreibung eventueller Erweiterungspläne bzw. Refactoringmaßnahmen.
- Beschreibung der Einschränkungen der Komponentenverwendung bzw. des möglichen Mißbrauchs.

Kommentierter Sourcecode beinhaltet nicht nur die Kommentare im Source selbst, sondern auch allgemeinere Anforderungen. Ausführlich wird diese Richtlinie in Fabricius (2000) erläutert. Hier nur ein Überblick über die wichtigsten Punkte:

- Lesbarkeit – Bei der Programmierung ist der ausführliche Weg, der von der größtmöglichen Anzahl von Softwareentwicklern gelesen werden kann, dem individuellen Stil vorzuziehen.
- Wiederverwendbarkeit – Im Rahmen der Möglichkeiten der Programmiersprache, sind logische und strukturelle Konzepte so einzusetzen, dass ein maximaler Wert an Wiederverwendbarkeit gewährleistet wird.
- Objektorientierte Programmierung – Wenn eine Sprache OOP-Konzepte zur Verfügung stellt, sollten diese auch eingesetzt werden.
- Dokumentation im Sourcecode – Es wird verlangt, dass innerhalb eines Projektes der verwendete Stil dokumentiert und eingehalten wird. Ferner sind folgende Hinweise zu berücksichtigen:
  - Der Name einer Variablen soll immer beschreibend auf den Verwendungszweck hindeuten. Jede Variable ist zusätzlich zu kommentieren.
  - Am Anfang jeder Sourcecode-Datei ist eine kurze aber genaue Beschreibung der in dieser Datei realisierten Logik zu erstellen. Die Verwendung von RCS-Schlüsselwörtern[1] wird vorgeschrieben. Im Kopf sind `$Id$`, `$Revision$` und `$Author$` einzusetzen. Der Einsatz weiterer Schlüsselwörter steht dem jeweiligen Entwickler frei. Das Schlüsselwort `$Log$` darf jedoch nur am Ende einer Datei vorkommen.
  - Sektionen innerhalb einer Datei sollten durch einleitende Kommentare identifizierbar voneinander getrennt sein.

[1] Abkürzung: **R**evision **C**ontrol **S**ystem: Vorgänger und Basis von CVS (siehe auch Fogel 1999).

  - Für die Namensgebung bei Funktionen bzw. Methoden gelten die gleichen Richtlinien, wie für die Variablen (siehe oben). Zusätzlich sind Methodennamen aus einem Verb (Beschreibung der Tätigkeit) und einem Substantiv (Gegenstand der Tätigkeit) zusammengesetzt.
  - Schleifen und Bedingungen müssen einleitend und abschließend kommentiert werden.

- Verwendung von Style Guides – Für die in den Projekten am häufigsten auftretenden Sprachen, Java und PHP, sind die vorgegebenen Style Guides einzuhalten. Für Java wird in diesen Style Guides z. B. der Einsatz von JavaDoc™ vorgeschrieben und für PHP sind dort u. a. Beispiele für Sourcecode-Formatierungen enthalten.

2.4.1 Vorschriften

Die oben genannten Richtlinien spiegeln das wünschenswerte Level an Dokumentation zu einem Projekt aus Sicht der Firma wider. Je nach Projekt kann dieses Level zu viel oder auch zu wenig sein. Es liegt dann an der Projektleitung, gemeinsam mit den Projektteilnehmern zu Beginn des Projektes festzulegen, wieviel davon notwendig ist bzw. was gegebenenfalls noch ergänzt werden muss.

Wichtig ist in jedem Fall, dass diese Vorschriften dann auch angewandt und umgesetzt werden. Sie müssen verstanden werden und ihre Notwendigkeit muss eingesehen werden. Dazu müssen Schulungen erfolgen, die diese Notwendigkeit allen Teilnehmern verdeutlichen.

An diesem Punkt kommt auch die zu entwickelnde Dokumentationsumgebung zu ihrer Berechtigung, weil sie eine Erleichterung im Umgang mit diesen Vorschriften verspricht. Sie gibt jedem Teilnehmer ein Tool an die Hand, mit dem er in bereits vorgegebenen Abschnitten den Text der Dokumentation anlegen und pflegen kann und nimmt ihm z. B. die folgenden Aufgaben ab:

- Festlegen der Dokumentationsthemen.
- Festlegen des Umfangs einzelner Abschnitte.
- Revisionssichere Speicherung auf einem zentralen Server.
- Ermöglichen der Navigation im Dokument.
- Ferner entfallen Layouten, aufwändige Umformatierungen und ähnliches im Dokument.

Die Benutzung des Tools muss in den Projektablauf und mindestens im Fall der BlackBox-Richtlinie auch in den Entwicklungszyklus integriert werden.

2.4.2 Die BlackBox-Dokumentationsrichtlinie

Diese recht kurz gehaltene Richtlinie (siehe im Anhang unter "A.1 BlackBox-Dokumentationsrichtlinie" auf der Seite 74 ff.) wird in dieser Arbeit als Referenz benutzt. In den folgenden Abschnitten werden die wichtigsten Bestandteile kurz vorgestellt.

2.4.2.1 Inhalt der Richtlinie

Neben einigen Metadaten zum Dokument selbst, enthält die Richtlinie vor allem Vorschriften zum Beschreiben der Außenwirkung einer Komponente. So muss die zu dokumentierende Komponente kurz beschrieben und eine Einordnung in einen gegebenen Kontext erfolgen. Anschließend ist sehr ausführlich das Interface der Komponente zu beschreiben. So werden eine Methoden-Beschreibung, untergliedert nach Art der Methoden, eine Datenstruktur-Beschreibung und eine Beschreibung des Interfaceprotokolls verlangt.

2.4.2.2 Extraktion der wichtigsten Bestandteile

Die Richtlinie gliedert sich in vier wichtige Sektionen:

1. **Titelseite** Auf der Titelseite eines BlackBox-Dokuments werden verschiedene Daten zum Dokument selbst festgehalten. Dies sind u. a. der Name der beschriebenen Komponente, der Autor des Dokuments, die Version des Dokuments und die Projektzugehörigkeit.

2. **Verteilung** Die zweite Seite enthält Angaben, an wen diese Dokumentation verteilt werden soll, ob und wer sie abgenommen hat und schließlich auch einen Änderungsnachweis.

3. **Inhalt** Ein Verzeichnis über den Inhalt des nächsten Punktes.

4. **Kapitel** Die einzelnen vorgeschriebenen Kapitel mit ihren entsprechenden Abschnitten. Die BlackBox-Richtlinie schreibt hier vier Kapitel vor: "Kurzbeschreibung", "Hintergrund, Motivation und Zweck", "Komponentenkontext" und "Interface". Wobei die beiden zuletzt genannten Kapitel noch weitere Gliederungen enthalten.

Die zu entwickelnde Software muss ein solches Dokument entsprechend aufbrechen können und dann einfachen Zugang zu den Daten der einzelnen Sektionen ermöglichen.

2.5 Weitere Aspekte

Im Folgenden werden weitere, für die Entwicklung der Dokumentationsumgebung zu berücksichtigende, Aspekte erläutert. Es wird kurz das Sicherheitsbedürfnis definiert, erläutert, warum XML als Austauschmedium genommen wird, und wieso Revisionssicherheit für Dokumentation wünschenswert ist.

2.5.1 Zugangskontrolle und Verschlüsselung

Es ist geplant, dass das zu erstellende Dokumentationssystem für alle zukünftigen Projekte in der Firma eingesetzt wird. Diese Projekte beinhalten nicht nur komplett interne Entwicklungen, sondern auch Projekte, die zum Teil oder auch komplett an externe Dienstleister beauftragt werden. Da auch diese Dienstleister die Software zur Dokumentation benutzen sollen, müssen Vorraussetzungen, wie Zugangskontrolle und Verschlüsselung des Übertragungsweges, geschaffen werden.

Der Zugriff auf ein Dokument ist nur einem vorher identifizierten Teilnehmer in der für ihn eingetragenen Weise zu erlauben. Das heißt, er muss sich gegenüber einer Nutzerverwaltung authentifizieren können und soll dann im Gegenzug Zugang zu bestimmten Dokumenten erhalten. Außerdem soll die Art des Zugangs kontrolliert werden. Nicht jeder Teilnehmer darf das Dokument ändern. Die Arbeitsschritte beim Anlegen eines neuen Projektes und der entsprechenden Einteilung der Zugriffe zeigt das Aktivitätendiagramm 2.1 auf der anderen Seite.

Weiterhin muss sicher gestellt sein, dass kein unberechtigter Dritter Zugriff auf ein Dokument erhält. Die Übertragung über ein Netzwerk ist daher – wenn möglich – zu verschlüsseln. Dies beinhaltet auch die Übertragung der Authentifizierungsdaten des Teilnehmers.

Mit Einführung einer PKI[2] ist ein Dokument, wenn es temporär auf dem Rechner des Teilnehmers gespeichert werden soll, auch bei der Speicherung zu verschlüsseln. Das ist notwendig, weil die meisten Rechner sich ständig in einem Netzwerk befinden und möglicherweise externer Zugriff auf die Speichermedien dieses Rechners besteht. Es ist dabei eine Vertreterregelung vorzusehen, so dass im Notfall auch ein vorher benannter Vertreter auf das Dokument zugreifen kann. Im Rahmen dieser Diplomarbeit ist die Einführung einer PKI jedoch nicht vorgesehen.

2.5.2 XML als Austauschmedium

Es gibt verschiedene Gründe, XML als Austauschmedium vorzusehen. Einer davon ist, dass XML inzwischen ein verbreiteter offener Standard ist. Der beschreibt einen sehr einfachen Regelsatz, der es den Anwendern erlaubt, ihre

[2] Abkürzung: **P**ublic **K**ey **I**nfrastruktur

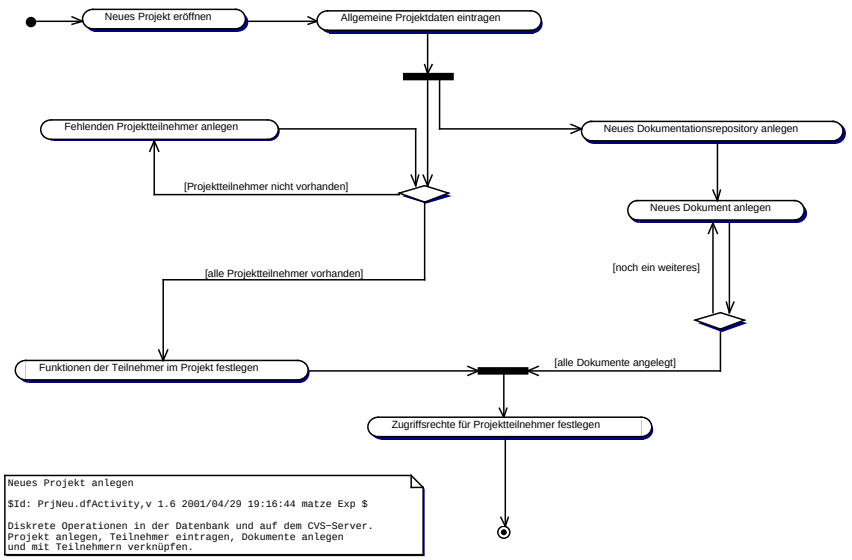

Abbildung 2.1: Neues Projekt anlegen

eigene Markup Language für ihre speziellen Probleme zu kreieren. Dabei entsteht eine hierarchische Dokumentenstruktur, die immer auf die gleiche Art und Weise mit einem Parser verarbeitet werden kann. Ein XML-Dokument kann jederzeit auf Gültigkeit überprüft werden. Dazu muss eine DTD oder ein XML-Schema definiert werden, in dem Elemente, Attribute und Zusammenhänge im XML-Dokument beschrieben werden. Damit definieren eine DTD bzw. ein XML-Schema genau diese Markup Language (Holubek 1999; McLaughlin 2000).

Ein weiterer Grund ist die strikte Trennung der Information von der Darstellung. Diese Idee ist keineswegs neu, bereits SGML[3] sah 1986 diese Trennung vor. Da SGML jedoch zu umfangreich war, wurde 1996 XML als Untermenge von SGML aus der Taufe gehoben. Die anfänglichen 70 Seiten der XML-Definition nehmen sich dann auch sehr bescheiden gegenüber den 500 Seiten der SGML-Definition aus (Großwendt 2000). Eine XML-Datei enthält nur die Information, jedoch keinen Hinweis darauf, wie diese Information dargestellt werden soll. Dazu gibt es dann die Stylesheets der XSL-Dokumente[4], die beschreiben, wie ein XML-Dokument so umgewandelt wird, dass es für einen bestimmten Zweck sinnvoll dargestellt werden kann (Holubek 1999).

Reichel (2000) stellt XSL zum Beispiel wie folgt vor:

[3] Abkürzung: **S**tandard **G**eneralized **M**arkup **L**anguage
[4] Abkürzung: E**X**tensible **S**tylesheet **L**anguage

> XSL ist eine Sprache, die von Dokumentenverfassern eingesetzt werden kann, um die in XML-Dokumenten enthaltene Information zu filtern oder aufzubereiten, sodass unterschiedlichen Bedürfnissen hinsichtlich der Art und Weise der Präsentation Rechnung getragen wird.

Zur Zeit gibt es neben kommerziellen auch einige freie Implementierungen von XSL-Prozessoren, die XML-Dokumente zum Beispiel für die Präsentation auf einem Webserver in HTML umwandeln. Als Beispiel sei hier das Xalan-Projekt (Web:Xalan) genannt. Andere Projekte, wie das FOP-Projekt (Web:FOP), beschäftigen sich mit der Aufbereitung der Dokumente in PDF für die Ausgabe auf einem Drucker.

Die hierarchische Strukturierung der Daten in einem XML-Dokument lässt sich sehr einfach in einem Objektmodell nachbilden. Eine Kombination von XML und Java bietet sich daher an. Eine andere Gemeinsamkeit besteht in der Portabilität. McLaughlin schreibt dazu (frei übersetzt): "[. . .] , was Java für die Portabilität von Code ist, beansprucht XML für die Portabilität von Daten."

Ferner garantiert man mit einer XML-Schnittstelle die Portierbarkeit der Daten zwischen den verschiedenartigsten Anwendungen. Dieser Vorteil wird noch deutlicher, wenn man betrachtet, dass zum Zeitpunkt der Konzeptionierung dieser Schnittstelle noch nicht einmal bekannt sein muss, welche Anwendungen, außer der eigenen, möglicherweise einmal darauf zugreifen werden. Um noch einmal zum Thema "Präsentation" zurückzukommen: Ist eine XML-Schnittstelle vorhanden, kann eine Applikation, die zur Präsentation von Daten existiert (z. B. Webserver), einfach über diese Schnittstelle die Daten abrufen und dann aufbereiten. Es ist nicht notwendig, eine weitere Schnittstelle für die Präsentation zu implementieren.

Natürlich könnte an die Stelle von XML auch ein proprietäres Dateiformat rücken, jedoch schränkt das den Verwendungsbereich ein, da in jede Anwendung, die dieses Format verarbeiten können soll, eine Schnittstelle bzw. ein Filter für dieses Dateiformat implementiert werden muss. Der Aufwand wird sich bei einem Dokumentenserver, mit dem viele Anwendungen kommunizieren müssen, irgendwann nicht mehr rechnen. Wohingegen heute schon viele Anwendungen eine Schnittstelle zu XML implementiert haben.

Ferner wurde bei BertelsmannSpringer schon vor Beginn dieser Arbeit XML als universelles Austauschmedium für alle neuen Applikationen festgelegt.

Bestandteil dieser Arbeit ist daher auch, ein XML-Validierungsdokument zu erstellen, anhand dessen die Dokumentenstruktur auf ihre Gültigkeit überprüft werden kann.

2.5.3 Revisionssicherheit

In David (1996, S. 119) ist eine Definition aufgeführt, die die Notwendigkeit der revisionssicheren Speicherung von Softwareentwicklungsdokumenten beschreibt. Dort heißt es:

> In der Software-Entwicklung sind diejenigen Einheiten, über die Entwickler miteinander kommunizieren, nicht auf Texte beschränkt, sondern umfassen, neben Quellprogrammen, Handbüchern, Testdaten etc., auch beispielsweise grafische Entwurfsbeschreibungen und ausführbare Programme. Desweiteren handelt es sich hier um Einheiten, die Gegenstand einer Entwicklung sind und die daher fortwährend geändert werden können. Aus diesem Grund müssen relevante Zustände durch das Sichern von Versionen dokumentiert werden. Für den Bereich der Software-Entwicklung wird daher ein *Dokument* als eine versionierbare Einheit definiert, die der Dokumentation von Entwicklungsergebnissen dient. Ein Dokument besteht aus einem Profil (Menge von Attributen, die das Dokument beschreiben) und einer Menge von Versionen. Eine Version besitzt ebenfalls ein Profil und einen Inhalt, das abgelegte Entwicklungsergebnis.

Neben der Notwendigkeit, die sich aus dem obigen Zitat ableitet, lassen sich auch die Vorteile revisionssicher gespeicherter Dateien aufzählen. Zu allererst steht da die Möglichkeit die Evolution einer Datei verfolgen zu können (Wer hat welche Änderungen warum durchgeführt?). Das ist etwas allgemeiner und kürzer ausgedrückt der Inhalt des obigen Zitats. Außerdem bieten Versionierungssysteme einen zentralen Platz zur Speicherung der Dateien und ermöglichen, zumindest bei CVS (Fogel 1999), verschiedenen Entwicklern die Arbeit auf einem Repository, ohne sich Gedanken über gesperrte Dateien machen zu müssen. Die Änderungen werden einfach zusammengeführt.

In Fogel (1999, S. 13) ist ein Anwendungsfall beschrieben, der sich zwar auf Sourcecode bezieht, aber durchaus auch auf Dokumentation übertragbar ist. Fogel schreibt dort:

> [. . .] , in the normal course of implementing a new feature, a developer may bring the program into a thoroughly broken state, where it will probably remain until the feature is mostly finished. Unfortunately, this is just the time when someone usually calls to report a bug in the last publicly released version. To debug the problem (which may also exist in the current version of the sources), the program has to be brought back to a useable state.

Eine Dokumentation muss nicht in einen ausführbaren Zustand zurückgebracht werden. Aber dieses Zurückbringen des Programms kann zur Folge haben, dass

die relevante Dokumentation ebenfalls auf diesen Stand zurückgebracht werden muss.

3 Projektplanung für die Arbeit

Im folgenden Kapitel wird eine kurze Übersicht über die geplanten Schritte im Projekt gegeben. Diese Planung soll später in der Zusammenfassung (siehe "7 Ausblick" auf der Seite 69) mit dem tatsächlichen Ablauf verglichen werden.

3.1 Termine

In Tabelle 3.1 auf der folgenden Seite wird eine Übersicht über die Termine des Projektplans gegeben.

3.2 Milestones

Die folgenden zwei Abschnitte erläutern in ein paar knappen Sätzen, wie die Milestones aussehen sollen. Für das Erreichen eines Milestones ist jeweils ein Monat vorgesehen.

3.2.1 Milestone 1 (Prototyp)

Der erste Milestone "Prototyp" stellt eine grundlegende Serverfunktionalität her. Der Server soll in der Lage sein, die Metadaten der Dokumente aus der Datenbank zu lesen und Dokumente aus dem CVS-Repository auszuchecken. Ferner soll das XML-Schema für die BlackBox-Dokumente erstellt werden. Es soll auch schon das komplette Netzwerkinterface mit Protokoll, Authentifizierung des Clients und Verschlüsselung implementiert werden.

Zur Unterstützung der Implementierung des Serverprototypen soll im Prinzip das Netzwerkinterface des Clients implementiert werden. Zusätzlich soll der Client schon in der Lage sein, das Netzwerkprotokoll soweit zu bedienen, dass

Termine	Titel	Beschreibung
10.11.2000 - 26.01.2001	Theoretische Ausarbeitung	Vorstellen der Diplomfirma (siehe " Vorwort" auf der Seite iii). Aufgabenstellung (S. 1 ff.), Einführung (S. 3 ff.) und Ansätze (S. 23 ff.) ausarbeiten. Projektplanung dokumentieren. Grobdesign der gesamten Architektur.
29.01.2001 - 05.03.2001	Konstruktionsphase 1	Feindesign für die erste Iteration. Erstellung der Datenbank und Einrichten des CVS-Repositories. Implementierung der Serverfunktionalitäten: • Persistenz der Objekte mit JDBC (nur lesen), • Persistenz der Objekte mit XML (nur erstellen, XML-Schema definieren), • Speicherung des Dokuments im CVS-Repository (nur checkout / update), • Netzwerkkommunikation (Protokoll definieren, Authentifizierung und Schnittstelle zur Verschlüsselung). Implementierung eines Clientstubs: • Verbindung zum Server (Protokoll implementieren, Passwort Authentifizierung, Blowfish Verschlüsselung) und • Dokument vom Server herunterladen. Tests und Dokumentation der ersten Iteration (siehe "5 Der Dokumentationsserver" auf der Seite 35).
05.03.2001	Milestone 1 (Prototyp)	Release des Serverprototypen.
06.03.2001 - 09.04.2001	Konstruktionsphase 2	Feindesign der zweiten Iteration. Implementierung der Serverfunktionalitäten: • vollständige Objektpersistenz mit JDBC, • vollständige Objektpersistenz mit XML und • Erweiterung der CVS-Funktionalität auf commit, update, status, tag, (add), (remove). Implementierung des Clients: • Objektpersistenz mit XML (einschließlich Validierung), [weiter in Tabelle 3.2 auf der anderen Seite]

Tabelle 3.1: Projektplan – Teil 1

Termine	Titel	Beschreibung
		• Dateihandhabung (Speichern / Laden von XML-Dokumenten ins / aus dem lokalen Filesystem) • grafische Benutzeroberfläche (Editor) Tests und Dokumentation der zweiten Iteration (siehe "6 Der Client" auf der Seite 55).
09.04.2001	Milestone 2 (MiniDok)	Release des MiniDok-Systems.
10.04.2001 - 12.04.2001	Nacharbeiten	Auffangen von Terminverzug und Debugging.
17.04.2001 - 09.05.2001	Fertigstellung der Diplomarbeit	Zusammenfassung und Ausblick (ab Seite 69) ausarbeiten. Rechtschreibung und Formatierungen überprüfen.
09.05.2001 - 11.05.2001		Druck und Abgabe der Diplomarbeit

Tabelle 3.2: Projektplan – Teil 2

ein Dokument vom Server heruntergeladen werden kann. In dieser Iteration ist noch keine Interaktion mit dem Nutzer vorgesehen.

3.2.2 Milestone 2 (MiniDok)

Der zweite Milestone "MiniDok" konzentriert sich vorwiegend auf den Client. Jedoch soll der Server soweit vollendet werden, dass er nun auch Elemente in der Datenbank speichern bzw. updaten kann. Die CVS-Funktionalität soll ebenfalls diejenigen Kommandos umfassen, mit denen die Dokumente im Repository gespeichert werden können.

Der Client wird zu einer vollständigen Applikation. Er soll seine grafische Benutzeroberfläche mit Editorfunktionalität für die BlackBox-Dokumente erhalten. Er soll ferner in der Lage sein, diese Dokumente lokal zu exportieren und validierend zu importieren.

4 Auswahl der Technologien

Aus der Aufgabenstellung und den einführenden Betrachtungen leiten sich verschiedene zu untersuchende Problematiken ab.

In der Aufgabenstellung war die Rede von zentraler Speicherung der Dokumente und auch davon, dass dem Nutzer ein Tool an die Hand gegeben werden soll, mit dem er an die Dokumente herankommt und diese Bearbeiten kann. Dieser Anforderung kann am sinnvollsten mit einer Client- / Server-Architektur nach Art der verteilten Datenhaltung begegnet werden (Meyer 1993). Auf dem Client wird das Userinterface und der Anwendungskern (Bearbeitung der Dokumentation) liegen und der Server ist verantwortlich für die Datenhaltung.

4.1 Anforderungen an den Client

Die folgenden Abschnitte beschreiben die allgemeinen Anforderungen an den Client. Es wird festgestellt, dass ein plattformunabhängiges Offline-Tool benötigt wird und dass es einige Argumente gegen ein Applet gibt.

4.1.1 Netzwerkkonzept

Die Vergangenheit hat gezeigt, dass ein großer Teil der Aufträge an externe Dienstleister abgegeben wird. Kleinere Firmen haben dabei meist keine Standleitung ins Internet, sondern nutzen eine Wählverbindung und zahlen somit für jede Minute, die sie online sind. Da auch diese Dienstleister die zu erstellende Software für ihre Dokumentationsarbeit nutzen sollen, müssen sie die Möglichkeit haben, offline zu arbeiten und nur zur Synchronisation mit dem Server eine Verbindung ins Internet herstellen zu müssen.

4.1.2 Plattform

Bedingt durch die Struktur in der Firma und durch die Services, die angeboten und benötigt werden, liegt eine sehr heterogene Rechnerlandschaft vor. Die Server sind zum überwiegenden Teil SUN Solaris-Rechner. Entwickler und Administratoren haben vielfach PC-Systeme mit dem Betriebssystem Linux. Redaktionen und Designer arbeiten zum Teil an Apple Macintosh-Rechnern und den größten Teil machen PC-Systeme mit diversen Microsoft Windows-Versionen aus. Bei den externen Dienstleistern sieht es ähnlich aus.

Alle diese Mitarbeiter bzw. Dienstleister sollen die zu erstellende Software nutzen. Um den Aufwand der Implementierung überschaubar zu halten und nur einmal zu entwickeln, bietet sich nur Java als plattformunabhängige Programmiersprache an.

Zudem ist mit Swing unter Java ein Toolkit gegeben, das eine schnelle und einfache Entwicklung einer grafischen Benutzeroberfläche gewährleistet. Aktuelle Java-Entwicklungsumgebungen, wie z. B. Forte for Java (Web:Forte4J) oder JBuilder (Web:JBuilder) unterstützen die Oberflächenprogrammierung mit Swing durch sehr einfach zu bedienende Layout-Tools, so dass nur noch die funktionalen Zusammenhänge implementiert werden müssen.

4.1.3 Applikation vs. Applet

Es gibt mehrere Gründe die gegen ein Applet und für eine Applikation sprechen. Einer der ersten ist dabei, dass es bei einem Applet mehr Komponenten gibt, die Einfluss auf das Verhalten der Software nehmen können: der Webserver und der Webbrowser. So muss ein fehlerhaftes Verhalten oder eine Instabilität auch daraufhin untersucht werden, ob sie aus dem Verhalten einer dieser beiden Komponenten resultiert.

Ein anderes Problem ergibt sich, wenn man die verwendeten JVM's[1] in den aktuell am weitesten verbreiteten Webbrowsern ansieht. Es wird überall nur eine JVM der Version 1.1 mitgeliefert. Netscapes Communicator 4.76 zum Beispiel enthält eine Version 1.1.5 und Microsofts Internet Explorer 5 enthält laut des Knowledge-Base Artikels "INFO: JDK Compatibility for Microsoft VM" (Web:Q214828) eine JVM der Version 1.1.4. In Java 1.1 ist aber noch kein Swing-Toolkit enthalten, so dass entweder mit dem Applet auch das etwa 2 MByte große Toolkit verschickt, ein spezielles Java 1.2 Plugin auf jedem Rechner installiert oder der Editor mittels des recht rudimentären AWT[2] implementiert werden müsste.

Letzteres bedeutet einen zu großen Aufwand, da die Funktionalität, die die Swing-Klassen schon beinhalten, erst noch nachgebaut werden müsste. Das

[1] Abkürzung: **J**ava **V**itrual **M**achine
[2] Abkürzung: **A**bstract **W**indow **T**oolkit

Verpacken der Swing-Klassen mit dem zu erstellenden Applet bedeutet eine zu lange Downloadzeit bei jedem Aufruf. Die Software würde in dem Fall keine Nutzerakzeptanz erfahren. Übrig bleibt die Installation des Java 1.2 Plugins. Dagegen spricht nur, dass unter "4.1.1 Netzwerkkonzept" auf der Seite 23 festgestellt wurde, dass ein Offline-Tool die bessere Variante wäre. Ein Applet ist jedoch auf die Verbindung zum Server angewiesen.

Ein weiterer Punkt gegen ein Applet ist das vorhandene Stabilitätsrisiko bei den Webbrowsern. Stürzt der Browser ab, sind die Daten verloren. Sie können mit einem Applet nicht lokal gespeichert werden. Das betrifft zumindest die Daten, die seit der letzten Synchronisation mit dem Server bearbeitet wurden. Zudem könnten vorhandene Sicherheitslöcher in den Browsern genutzt werden, um diese oder sogar das Betriebssystem zum Absturz zu bringen oder – entsprechende kriminelle Energie vorrausgesetzt – Daten auszuspionieren.

Demgegenüber kann eine Applikation sauber installiert werden. Sie kann ihre eigene JRE[3] mitbringen, wenn sie zum Beispiel auf einem Rechner mit einer älteren JRE installiert werden muss. Die einzigen Daten, die über eine Netzwerkverbindung ausgetauscht werden müssen, sind die zu bearbeitenden Dokumente. Die Applikation kann auf das lokale Filesystem zugreifen und somit zum Beispiel in regelmäßigen Abständen Sicherheitskopien anlegen. Außerdem geht eine einzelne Applikation mit dem Speicher genügsamer um, als ein Applet, das in einem Webbrowser läuft.

4.2 Anforderungen an den Server

Die folgenden Abschnitte geben einen Überblick über die verwendeten Technologien beim Server. Es wird erläutert, warum der Server nicht als Teil eines Applicationservers implementiert wird und warum CVS und MySQL als Services für das Revisionsverwaltungssystem und die Datenbank zum Einsatz kommen.

4.2.1 Eigenständiger Server

Im Verlauf der Diplomarbeit entschied BertelsmannSpringer einen Applicationserver anzuschaffen. In Folge dessen kam die Frage auf, ob es nicht eventuell sinnvoll wäre den Server, der aus dieser Arbeit hervorgehen soll, nach der EJB[4] Spezifikation auf den Applicationserver zu implementieren?

Nach einer zum Thema "Enterprise JavaBeans" einführenden Lektüre von Monson-Haefel (2000) zeigte sich, dass dies zu diesem Zeitpunkt nicht mehr sinnvoll umzusetzen ist, denn als die Frage aufkam, war das Design bereits zum größten Teil fertig. Eine Integration auf den Applicationserver hätte eine größere

[3] Abkürzung: **J**ava **R**untime **E**nvironment
[4] Abkürzung: **E**nterprise **J**ava**B**ean

Umstrukturierung bedeutet und kam daher auf Grund des engen Zeitplans nicht in Frage.

Zudem hätte der angeschaffte Applicationserver für die zu erstellende Software nur die Persistenzschicht zur Datenbank bieten können. Die Verbindung zum CVS-Server und die XML-Serialisierung hätten weiterhin implementiert werden müssen. Außerdem blieb ungeklärt, was für Möglichkeiten der Einflussnahme die Applikation auf die Verschlüsselung der Kommunikation zwischen Client und Server gehabt hätte.

Bei den Untersuchungen zu dem Thema wurde die Java-Library Castor (Web:Castor) entdeckt. Diese implementiert eine Persistenzschicht zu relationalen Datenbanken und ermöglicht das Serialisieren von Java-Objekten in XML-Dokumente und umgekehrt. In der FAQ[5] zu Castor (siehe Web:Castor) heißt es:

> With Castor, incoming XML messages can be unmarshaled into data objects. Required information can be obtained from a database using JDO in form of data objects. With this approach, all data manipulation can be done in an object-oriented way. Changes to JDO data objects can be committed transactionally, and result data objects can be marshaled into XML and returned to the client.

Die Castor-Library bringt noch weitere Funktionalitäten (wie z. B. einen Zugriff auf LDAP[6] Verzeichnisse) mit, die für die Diplomarbeit keine Rolle spielen, aber im Hinblick auf die Zugangsverwaltung zu einem späteren Zeitpunkt interessant werden könnten.

4.2.2 CVS als Versionierungssystem

Um die Speicherung der mit dieser Software angelegten Dokumentation revisionssicher zu halten, kommt CVS[7] (Web:CVS) zum Einsatz. CVS unterscheidet sich von anderen Versionsverwaltungssystemen dadurch, dass es das verbreitete "lock – modify – unlock"-Modell nicht unterstützt. Dieses Modell funktioniert nur so lange, wie die Entwickler sich kennen und schnell miteinander kommunizieren können, um Zugangsschwierigkeiten auf Dateien auszuräumen (Fogel 1999). Fogel führt dazu weiter aus:

> However, if the developer group becomes too large or too spread out, dealing with all the locking issues begins to chip away at coding time; it becomes a constant hassle that can discourage people from getting real work done.

[5] Abkürzung: **F**requently **A**sked **Q**uestions
[6] Abkürzung: **L**ightweight **D**irectory **A**ccess **P**rotocol (siehe auch Yeong u. a. 1995)
[7] Abkürzung: **C**oncurrent **V**ersions **S**ystem

CVS benutzt dagegen das "copy – modify – merge"-Modell, welches Fogel in einem Beispiel wie folgt beschreibt:

> 1. Developer A requests a working copy (a directory tree containing the files that make up the project) from CVS. This is also known as "checking out" a working copy, like checking a book out of the library.
> 2. Developer A edits freely in her working copy. At the same time, other developers may be busy in their own working copies. Because these are all separate copies, there is no interference - it is as though all of the developers have their own copy of the same library book, and they're all at work scribbling comments in the margins or rewriting certain pages independently.
> 3. Developer A finishes her changes and commits them into CVS along with a "log message", which is a comment explaining the nature and purpose of the changes. This is like informing the library of what changes she made to the book and why. The library then incorporates these changes into a "master" copy, where they are recorded for all time.
> 4. Meanwhile, other developers can have CVS query the library to see if the master copy has changed recently. If it has, CVS automatically updates their working copies. [...]

CVS ist auch in der Lage, die aus diesem Modell entstehenden Konflikte, aufzulösen. Folgendes Beispiel soll dies verdeutlichen:

Zwei Entwickler arbeiten an der gleichen Stelle in der gleichen Datei. Entwickler A checkt seine Änderungen in das CVS-Repository ein. CVS akzeptiert die Änderungen. Entwickler B versucht wenig später seine Änderungen in das Repository zu übertragen. CVS erkennt einen Konflikt und fordert Entwickler B auf, diesen – in der Datei vom CVS gekennzeichneten – Konflikt aufzulösen.

CVS unterstützt also die Kooperation zwischen den Projektteilnehmern, weil es sie in die Lage versetzt, zur gleichen Zeit an den selben Dateien zu arbeiten. Außerdem besitzt es ein Versionierungssystem, über das die Evolution einer Datei verfolgt werden kann und das die Herstellung früherer Stände ermöglicht. Ferner wurde CVS gewählt, da es als Lizenzmodell die GPL[8] benutzt und daher lizenzkostenfrei ist.

4.2.3 MySQL-Datenbank für die Zugriffsverwaltung

Um möglichst flexibel auf zukünftige Entwicklungen reagieren zu können, soll die Persistenzschicht zur Datenbank unabhängig vom verwendeten DBMS[9] sein.

[8] Abkürzung: GNU **G**eneral **P**ublic **L**icense (siehe auch Web:GPL)
[9] Abkürzung: **D**ata**B**ase **M**anagement**S**ystem

Die oben genannte Castor-Library bringt Implementierungen für verschiedene DBMS' mit. Die Wahl fällt daher eher aus zwei pragmatischen Gründen auf MySQL (Web:MySQL):

1. es müssen keine großen Vorbereitungen (Installation, Einarbeitung, etc.) getroffen werden, da das System bereits verwendet wird, und
2. MySQL (verwendete Version 3.23.32) benutzt die GPL als Lizenzmodell, daher fallen auch hier keine Lizenzkosten an.

4.3 Zusammenspiel von Client und Server

Die folgenden Abschnitte beschreiben die Kommunikation zwischen Client und Server. Ferner werden Details zu den verwendeten Verschlüsselungskomponenten und zur Zugriffsrechteverwaltung erläutert.

4.3.1 Netzwerkkommunikation

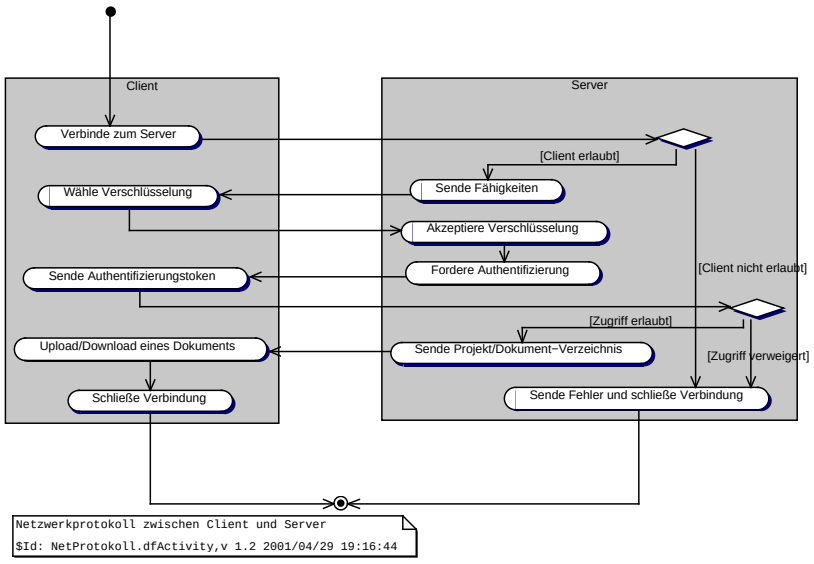

Abbildung 4.1: Handshake zwischen Client und Server

Abbildung 4.1 zeigt die Verbindungsaufnahme zwischen Client und Server. Der Client soll zuerst ein "Hallo!" senden. Der Server überprüft daraufhin, ob

der Client zugelassen ist und antwortet mit seinen Fähigkeiten oder einer Fehlermeldung. Der Client wählt aus der Liste der Fähigkeiten zum Beispiel die gewünschte Verschlüsselung und handelt mit dem Server eine sichere Verbindung aus. Danach fordert der Server den Client auf, sich zu authentifizieren und überprüft die vom Client gesendeten Daten. Der Server antwortet mit einer Liste der für den Client zur Verfügung stehenden Dokumente oder einer Fehlermeldung. Der Client kann dann vom Server ein Dokument anfordern oder die Verbindung beenden.

Die Kommandos des Netzwerkprotokolls sollen als kleine Objekte realisiert werden. Sie können dann mittels der Castor-Library einfach nach XML serialisiert und zwischen Client und Server ausgetauscht werden.

4.3.2 Verschlüsselung der Client- / Serverkommunikation

An zwei Punkten muss Verschlüsselung in die Applikation integriert werden. Der erste und wichtigste ist die Verschlüsselung der Kommunikation zwischen Client und Server. Der zweite Punkt ist die lokale Speicherung der Dokumente beim Client.

Für eine allgemeine parametrisierbare Verschlüsselung wird die JCE-Library[10] benutzt. Eine andere Möglichkeit, die nur die Übertragung über das Netzwerk mittels SSL[11] verschlüsselt, besteht in der JSSE-Library[12]. Beide Möglichkeiten werden in der zu erstellenden Applikation implementiert. Für den Prototypen, der nur Verschlüsselung auf dem Übertragungswege vorsehen soll, reicht die Verwendung von JSSE. Auf der Clientseite muss aber mindestens für die lokale Verschlüsselung auch JCE zum Einsatz kommen.

Die JCE-Library ist SUNs Referenzimplementierung zu dem gleichnamigen Framework. Ein Kryptographiepaket, das JCE ersetzen könnte, ist Cryptix (Web:Cryptix), das sich jedoch in der derzeit aktuellen Version 3.2.0 nicht an die Definitionen des JCE-Frameworks hält. Des Weiteren enthält Cryptix Algorithmen, die in manchen Ländern patentgeschützt sind (z. B. IDEA) und daher erst lizensiert werden müssten.

JSSE ist SUNs Implementierung von SSL. Es stellt im Prinzip Wrapperklassen für das `java.net`-package zur Verfügung, die die Verschlüsselung in die Sockets implementieren. Somit ist ein relativ geringer Aufwand notwendig, wenn eine Applikation von unverschlüsselten Sockets zu SSL migriert werden soll.

Jedoch ist nur im JCE die Einflussmöglichkeit gegeben, die Stärke der Verschlüsselung – in den durch die Algorithmen festgelegten Grenzen – zu bestimmen. Die Tabelle 4.1 auf der folgenden Seite enthält die für die aktuelle

[10] Abkürzung: **J**ava **C**ryptography **E**xtension (siehe Web:JCE)
[11] Abkürzung: **S**ecure **S**ocket**L**ayer
[12] Abkürzung: **J**ava **S**ecure **S**ocket **E**xtension (siehe Web:JSSE)

Implementierung (JCE Version 1.2.1) gültigen Schlüsselstärken (nach Knudsen 1998; Sun Microsystems, Inc. 2000) und stellt diese der Implementierung in der JSSE-Library gegenüber.

	JSSE	JCE
RC4	40 Bit, 128 Bit	—
DES	40 Bit, 56 Bit	56 Bit
TripleDES	112 Bit	112 Bit, 168 Bit
BlowFish	—	32 - 448 Bit

Tabelle 4.1: Implementierte Schlüsselstärken verschiedener Algorithmen

Für die zu erstellende Software werden daher drei Möglichkeiten für die Netzwerkübertragung vorgesehen:

Plain : Keine Verschlüsselung (benutzt den `NullCipher` aus den JCE).

Blowfish : 56 Bit standard Blowfish Cipher aus den JCE.

SSL : Standard SSL aus den JSSE.

4.3.3 Verwaltung der Zugriffsrechte

Der Client wird vom Server aufgefordert, sich zu authentifizieren. Er hat daraufhin ein Login mit dem zugehörigen Passwort an den Server zu senden. Der Server erkennt daran den Nutzer und ist in der Lage die Zugriffsrechte des Clients einzelnen Dokumenten zuzuordnen.

Die Zugriffsrechte werden in der Datenbank verwaltet. Dort wird festgelegt, wer Zugriff auf welches Dokument hat und ob das eine "nur lesen"- oder "lesen und schreiben"-Zugriffberechtigung ist. Außerdem wird die Datenbank die Nutzerverwaltung (Logins und Passwörter für die Authentifizierung) enthalten.

Die Zugriffsrechte sollen so vergeben werden, dass eine Art Baumstruktur entsteht. Der Administrator für den Dokumentationsserver wird an alle Dokumente herankommen. Dann folgen die Projektleiter, die jeweils auf alle Dokumente des Projekts Zugriff erhalten sollen. Danach wird für die restlichen Mitglieder des Projektteams der Zugriff für jedes Dokument innerhalb des Projektes definiert.

4.4 Validierung der Dokumente

Um die Dokumente validieren zu können, ist entweder eine DTD oder ein XML-Schema notwendig. Die Castor-Library lässt beide Arten für die Validierung zu.

Obwohl XML-Schemas noch nicht offiziell vom W3C[13] als Standard verabschiedet worden sind, empfiehlt McLaughlin (2000, S. 10) XML-Schemas wie folgt:

> The most significant fact about XML Schema is that it brings DTDs back into line with XML itself. That may sound confusing; consider, though, that every acronym we have talked about uses XML documents to define its purpose. XSL stylesheets, namespaces, and the rest all use XML to define specific uses and properties of XML. But a DTD is entirely different. A DTD does not look like XML, it does not share XML's hierarchical structure, and it does not even represent data in the same way. This makes the DTD a bit of an oddball in the XML world, and because DTDs currently define how XML documents must be constructed, this has been causing some confusion. XML Schema corrects this problem by returning to using XML itself to define XML. We have been talking about "defining data about data" a lot, and XML Schema does this as well. The XML Schema specification moves XML a lot closer to having all its constructs in the same language, rather than having DTDs as an aberration that has to be dealt with.

Seit dem 24.10.2000 liegt XML-Schema als "Candidate Recommendation" beim W3C vor (siehe auch Web:W3C, /XML/Schema) und wird vermutlich nur noch geringe Änderungen bis zur Verabschiedung als Standard erfahren.

Neben dem Vorteil, dass ein XML-Schema leichter verständlich ist als eine DTD, kann ein XML-Schema von Castor benutzt werden, um Persistenzklassen zu generieren. Das erspart einige Arbeit, denn auch wenn der generierte Sourcecode nicht direkt genutzt werden kann, kann er durchaus als Basis für die weitere Entwicklung dienen. Castor generiert darüber hinaus die Klassen so, dass ein Dokument anhand dieser Klassen validiert werden kann. Es ist also kein extra Validierungslauf notwendig, um feststellen zu können, ob ein Dokument der XML-Schema-Spezifikation entspricht.

Für den aus dieser Arbeit resultierenden Prototypen ist nur anhand der Richtlinie für die BlackBox-Dokumente ein XML-Schema zu erstellen.

4.5 Altdatenübernahme

Beim Ausarbeiten der Aufgabenstellung wurde von BertelsmannSpringer auch die Übernahme von bereits bestehenden Dokumentationen in den Server gewünscht. Dies ist jedoch nicht ohne weiteres möglich. Die schon bestehenden

[13] Abkürzung: **W**orld **W**ide **W**eb **C**onsortium (siehe auch Web:W3C)

Dokumentationen halten sich nicht an die jetzt gültigen Richtlinien. Sie müssten daher erst vervollständigt und an diese angepasst werden. Dafür sind jedoch zusätzliche Recherchearbeiten nötig, um die fehlenden Informationen zu sammeln. Eine Übertragung in XML-Dokumente, die von der zu erstellenden Applikation verarbeitet werden können, ist daher nur von Hand möglich.

Die Dokumentationen, die in den laufenden Projekten erstellt werden, passen jedoch schon in den vorgeschriebenen Rahmen, daher kann man diese – möglicherweise automatisiert – in XML-Dokumente überführen.

Wenn die Dokumentationsumgebung produktiv eingesetzt wird, sollten die

- Dokumente für neue Projekte ausschließlich mit dieser Software erstellt,
- Dokumente von laufenden Projekten zuerst in diese Dokumentationsumgebung übernommen und
- Dokumente von abgeschlossenen Projekten, die bereits nach den Richtlinien erstellt wurden, nach und nach in diese Dokumentationsumgebung überführt werden.

Alte Dokumentation, die nicht getreu den Richtlinien geschrieben wurde, sollte besser nur zentral an einem Ort gesammelt werden, da der Aufwand für die Übertragung sehr hoch ausfallen wird. Dieser Ort könnte zum Beispiel der CVS-Server sein, der auch für die neuen Dokumente genutzt wird.

4.6 Entwicklungsprozess

Für die Entwicklung der Software wurde ein iterativ inkrementeller Prozess ausgewählt (nach Fowler und Scott 2000, Seite 11 ff.). Mit dieser Art von Software-Engineering-Prozessen rückt ein Ergebnis schneller in greifbare Nähe und nachträgliche Änderungen oder neue Anforderungen können – bis zu einem gewissen Grad – in den laufenden Prozess integriert werden.

Der ausgewählte Prozess wurde vom RUP[14] abgeleitet. Im Unterschied zum RUP, enthält dieser Prozess weniger Formalien und ist daher für sehr kleine Entwicklungsteams geeignet. Abbildung 4.2 auf der anderen Seite zeigt einen Überblick über den Prozess.

Der Prozess ist in vier Hauptphasen unterteilt:

- Der *Einstieg* soll die Absicht des Projekts aufzeigen.
- Während der *Ausarbeitung* werden die Anforderungen, Analyse und Entwurf für die grundlegende Architektur zusammengestellt.

[14] Abkürzung: **R**ational **U**nified **P**rocess (siehe auch Web:RUP)

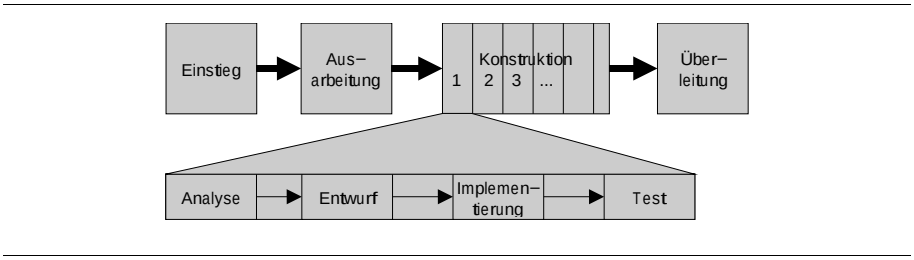

Abbildung 4.2: Inkrementell-iterativer Entwicklungsprozess

- Die anschließende *Konstruktion*sphase besteht aus den Iterationen, die schrittweise die Funktionalität in die Software integrieren. Jede Iteration besteht aus dem üblichen Lebenszyklus (Analyse, Entwurf, Implementierung und Test) und resultiert in einer funktionierenden Software.
- Die *Überleitung*sphase schließlich beinhaltet die Arbeiten, die erst am Ende des Projekts durchgeführt werden können: Benutzertraining, Installation, etc. (Fowler und Scott 2000).

5 Entwicklung des Dokumentationsservers

In diesem Kapitel wird die Entwicklung des Servers der Dokumentationsumgebung beschrieben. Fertiggestellt werden konnte nur die erste Iteration (dazu mehr unter "5.2.3 Test" auf der Seite 53).

5.1 Vorbereitung

In die Vorbereitungphase für die Serverimplementierung fallen das Design und die Erstellung der Metadaten-Datenbank, die Erstellung eines Beispiel-CVS-Repositories und die Festlegung der Schnittstellen.

5.1.1 Datenbank-Struktur

Das Entity-Relationship-Diagramm in Abbildung 5.1 auf der folgenden Seite zeigt den Aufbau der Datenbank "Projektverwaltung". Die Datenbank soll in Zukunft die komplette Projektverwaltung unterstützen, zu diesem Zeitpunkt liegt der Schwerpunkt jedoch erst einmal auf dem Speichern der Metadaten der Dokumente, der Speicherung der Teilnehmerdaten und der Verknüpfung der beiden, um eine Zugriffs-Zuordnung zu erreichen.

Der Server soll die Teilnehmer gegen die Datenbank authentifizieren. Daher gibt es eine Tabelle "Teilnehmer", in der Daten, wie Login und Passwort verwaltet werden. Das Feld "T_Login" ist der Primärschlüssel für die Tabelle und stellt damit auch gleich sicher, dass die Einträge in diesem Feld `unique` (eindeutig) sind. Ein Login-Feld ist auf 48 Byte begrenzt. Das ist ausreichend für ein eindeutiges Login und nimmt weder in der Tabelle selbst, noch im Primärschlüssel-Index viel Platz ein.

Für das Passwort ist als Typ `varchar(33)` vorgesehen. Es wird erwartet, dass die Passwörter mit MD5 "verschlüsselt" werden. Der MD5-Algorithmus

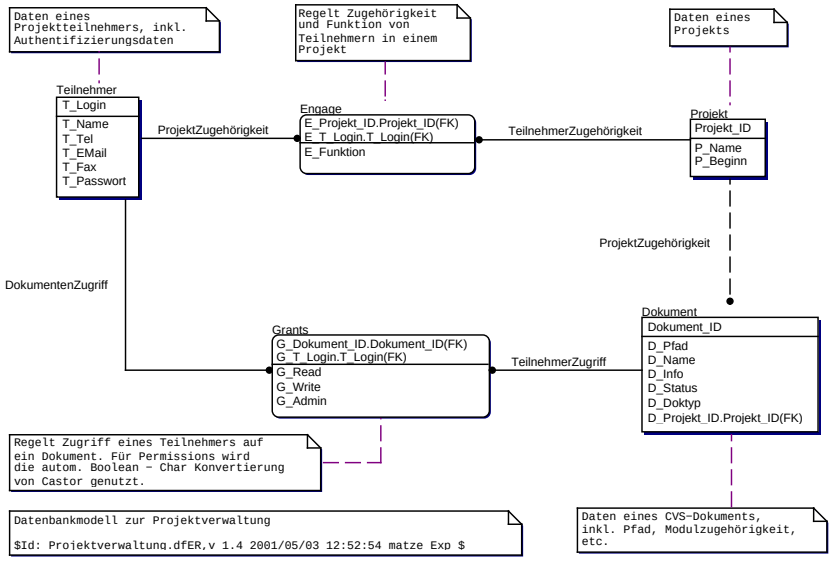

Abbildung 5.1: Datenbankmodell zur Projektverwaltung

"verschlüsselt" einen Eingabewert allerdings nicht, sondern liefert einen 128 Bit langen Hashwert über einen beliebig langen Eingabewert (siehe Rivest 1992; Knudsen 1998, S. 25). Aus diesem Hashwert lässt sich das Klartext-Passwort nicht mehr generieren. Vielmehr soll das Passwort bei Authentifizierung ebenfalls "verschlüsselt" und mit dem im Feld "T_Passwort" gespeicherten Wert verglichen werden. Die Hexadezimal-Darstellung des Hashwertes soll in dem Passwort-Feld gespeichert werden.

Die Metadaten der Dokumente werden in der Tabelle "Dokument" gespeichert. Die Dokumente erhalten eine automatische ID, die als Primärschlüssel fungiert. Der Server findet die Dokumente im CVS-Server anhand der Angaben in "D_Pfad" und "D_Name". Der gespeicherte Pfad ist relativ zum CVS-Repository und wird um den Namen ergänzt, um einen vollständigen Pfad zu erhalten. Die Angabe im Feld "D_Doktyp" ist für den Client wichtig, er erkennt an diesem Eintrag, welcher Dokumenten-Typ übertragen wurde. Zur Zeit kann dies nur "BlackBox" sein.

Die Tabelle "Grants" stellt eine Relation zwischen "Teilnehmer" und "Dokument" mit zusätzlichen Feldern dar. In diesen Feldern werden die Rechte, die ein Teilnehmer auf ein Dokument hat, festgelegt. Die Felder "G_Read", "G_Write" und G_Admin" können jeweils die Werte `0` bzw. `1` annehmen. Diese Werte repräsentieren die Zustände `false` bzw. `true`.

Alle im ER-Diagramm vorhandenen, aber hier nicht genannten Felder, werden von der Software zur Zeit noch nicht ausgewertet. Da die Entsprechungen der Tabellen in der Persistenzschicht (siehe "5.1.3 Schnittstellen") jedoch ebenfalls diese Felder enthalten, müssen in den Datenbanksätzen gültige Werte für alle Felder vorhanden sein.

5.1.2 Repositorystruktur

Der für die Versionskontrolle zuständige CVS-Server kann theoretisch eine beliebige Repository-Struktur aufweisen, da in der Datenbank der Pfad zu jedem Dokument relativ zum Repository angegeben wird.

Es ist jedoch anzuraten, eine logische Konstruktion aufzubauen, die es gegebenenfalls später ermöglicht, den Pfad aus weiteren Metadaten zu konstruieren und das Pfad-Feld einer anderen Funktion zuzuordnen. In diesem Sinne ist folgende Struktur zu favorisieren:

```
<cvsrepository>/<projektid>/<doktyp>/<pfad>/<name>.
```

So kann in großen Projekten das Pfad-Feld genutzt werden, um zum Beispiel nach Java-Packages zu unterscheiden.

Da im Repository für den Prototypen noch keine Dokumente aus aktuellen Projekten verwendet wurden, wurde eine vereinfachte Struktur verwendet, die nicht `<projektid>/<doktyp>/` enthält.

5.1.3 Schnittstellen

In diesem Abschnitt folgt ein kurze Beschreibung der definierten Schnittstellen zu den einzelnen externen Komponenten (vgl. auch Komponentendiagramm C.1 auf der Seite 86).

5.1.3.1 ... zum CVS

Die CVS-Komponente zur Steuerung des CVS-Kommandozeilentools hält ein Interface für den Server bereit, über das die CVS-Befehle abstrahiert werden. Die Implementierung für die erste Iteration sieht nur die Unterstützung des "login"- und "checkout"-Kommandos von CVS vor.

Auf Grund der Implementierung des CVS-Kommandozeilentools, ist das "login"-Kommando nicht in der Lage einen neuen Login durchzuführen. Es überprüft lediglich, ob bereits ein Login durchgeführt wurde und diese Daten weiter verwendet werden können. Ist das nicht der Fall, fordert es den Administrator auf, manuell ein Login durchzuführen und stellt die benötigten Daten dazu bereit.

5.1.3.2 ... zur Datenbank

Die Inhalte der Datenbank werden über eine Persistenzschicht angesprochen. Die Steuerung der Persistenzschicht wird über ein Interface abstrahiert. Benötigt und unterstützt wird das Auslesen der Tabellen "Teilnehmer", "Dokument" und "Projekt". Die ausgelesenen Datensätze werden jeweils als Java-Bean zurückgegeben. Außerdem kann ein Objekt vom Typ `java.util.Vector` mit den für einen bestimmten Teilnehmer lesbaren Dokumenten instanziiert werden. Zusätzlich ist es auch möglich den Datenbankteil eines Dokumentencontainers (siehe Abbildung 5.2 auf der Seite 40) – den Parametern entsprechend – zu initialisieren.

5.1.3.3 ... zum Client

Die Kommunikation mit dem Client wird über IP-Sockets (`java.net.Socket`) abgewickelt. Da der Datenaustausch möglichst verschlüsselt stattfinden soll, ist eine SSL-Implementierung vorhanden und Klassen zur parametrisierbaren Verschlüsselung. Das implementierte Protokoll ist ein Challenge- / Response-Protokoll (siehe Abbildung 4.1 auf der Seite 28) auf der Basis von XML-Dokumenten. Das heißt, dass zum Beispiel der Client auf sein "Hallo!" zu Beginn der Kommunikation eine Antwort vom Server erwartet. Diese Antwort sollte "Capabilities" lauten, kann aber auch "Error!" heissen, wenn ein Fehler aufgetreten ist.

Folgendes XML-Schema beschreibt die Dokumentenstruktur der Protokollelemente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
            elementFormDefault="qualified">
  <xsd:element name="XMLCommand">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="command"/>
        <xsd:element ref="params" minOccurs="0"
                     maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="command" type="xsd:string"/>
  <xsd:element name="params" type="xsd:string"/>
</xsd:schema>
```

Die Parameter richten sich jeweils nach der Implementierung des Kommandos. Für die erste Iteration werden folgende Kommandos benötigt (sortiert nach der Reihefolge des Auftretens im Protokoll): "Hallo!", "Capabilities", "Authenticate", "DocList", "Fetch", "Send", "Bye!" und "Error!".

Eine Ausnahme gibt es vom Gebrauch dieses Protokolls: wenn die parametrisierbare Verschlüsselung verwendet wird, einigen sich Server und Client mittels des Diffie-Hellmann-Protokolls. Das Diffie-Hellmann-Protokoll ist speziell für den Austausch eines geheimen Schlüssels über einen unsicheren Kanal entworfen worden (Knudsen 1998, S. 27).

5.1.3.4 ... zur Konfiguration

Die Konfiguration wird mittels Java Properties (`java.util.Properties`) und Kommandozeilenargumenten durchgeführt. Eine Datei kann Voreinstellungen in Form der üblichen Property-Darstellung (*key*=*value*) enthalten. Die Einstellungen in der Konfigurationsdatei werden von eventuell angegebenen Kommandozeilenargumenten überschrieben. Alle möglichen Einstellungen können sowohl in der Datei als auch an der Kommandozeile angegeben werden.

Die Konfiguration umfasst Einstellungen zum Ansprechen des CVS-Servers (z. B. Servername, Repository, etc.), zur Verbindung mit der Datenbank (Angaben zur Konfiguration von Castor JDO) und zur Parametrisierung der Netzwerkverbindung (Portnummer, SSL, Castor XML-Konfiguration, etc.).

Die einzelnen Serverkomponenten sprechen nur ihre jeweiligen Konfigurationen via Interfaces mit Getter- und Setter-Methoden an, so dass eine spätere Implementierung in einer anderen Technologie keine Probleme bereiten sollte (z. B. Konfiguration via XML).

5.1.4 Aufbereiten des Dokuments

Das Dokument, das den eigentlichen Inhalt ausmacht – also zum Beispiel ein BlackBox-Dokument – findet beim Server überhaupt keine Beachtung. Dieses Dokument ist für den Server nur "Inhalt", der vom CVS-Server zum Client, bzw. umgekehrt, transportiert werden muss. Wenn also auf der Serverseite von einem Dokument die Rede ist, ist eigentlich der Transportcontainer (siehe Abbildung 5.2 auf der folgenden Seite) für diesen "Inhalt" gemeint.

Das Dokument auf der Serverseite ist eine Komposition aus zwei Teildokumenten. Der erste Teil enthält die Metadaten aus der Datenbank und der andere Teil Informationen und den "Inhalt" vom CVS-Server. Dieses Dokument ist als serialisierbare Java-Bean aufgebaut, damit es mittels der Castor XML Komponenten nach XML umgewandelt und über das Netzwerk zum Client transportiert werden kann.

Das Aktivitätendiagramm 5.3 auf der folgenden Seite zeigt die Schritte, die der Server zu tun hat, um eine Anforderung von einem Client zu bedienen.

Im Laufe einer Session fordert der Client mittels des Fetch-Kommandos ein Dokument an. Der Server hat nun zu überprüfen, ob der Client mindestens

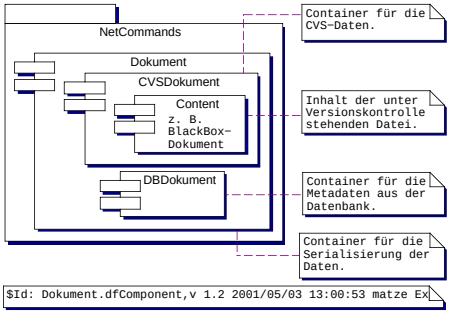

Abbildung 5.2: Der Dokumentencontainer

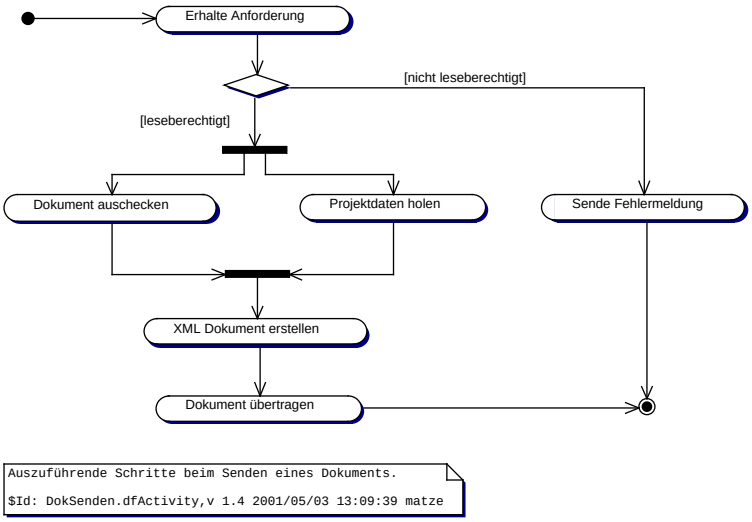

Abbildung 5.3: Die Dokumentenanforderung

die Zugriffsberechtigung für "Lesen" auf dieses Dokument hat. Ist diese Zugriffsberechtigung nicht gegeben, sendet der Server eine Fehlermeldung an den Client zurück. Anderenfalls füllt der Server das Dokument mit den Metadaten aus der Datenbank und den Informationen und dem Inhalt des angeforderten Dokuments aus dem CVS-Server. Nach dem Zusammenführen in ein Dokument-Objekt, wird dieses nach XML serialisiert und an den Client verschickt. Ist eine Aktion unvollständig, fehlerhaft oder gar nicht, durchgeführt worden, ist das Ergebnis eine an den Client verschickte Fehlermeldung und der Abbruch der Aktivität.

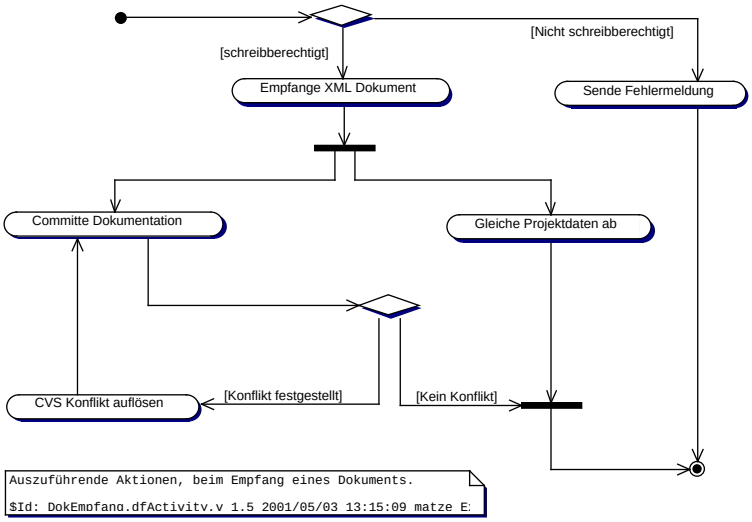

Abbildung 5.4: Der Dokumentenempfang

Das Diagramm 5.4 zeigt die Aktivitäten, die der Server zu durchlaufen hat, wenn er von einem Client ein Dokument erhält. Diese Aktivität wird in der zweiten Iteration implementiert.

Der Client muss zunächst seine Absicht, ein Dokument an den Server zu schicken, dem Server kundtun und die Genehmigung des Servers abwarten. Der Server überprüft nach der Anfrage zuerst, ob der Client schreibberechtigt ist und sendet entsprechend ein "OK" oder eine Fehlermeldung an den Client. Danach erwartet der Server vom Client das Dokument.

Ist das Dokument eingetroffen und deserialisiert, beginnt der Server die Metadaten mit der Datenbank abzugleichen und den Inhalt in das CVS Repository einzuchecken. Tritt beim Einchecken am CVS Server ein Konflikt auf, sendet der Server das Dokument mit den, vom CVS gesetzten, Konfliktmarkern an den Client zurück und fordert ihn auf, den Konflikt zu lösen. Tritt kein Konflikt auf,

bzw. ist der Konflikt vom Client aufgelöst worden, ist die Aktivität beendet.

5.2 Erste Iteration

Im ersten Entwicklungszyklus sollte ein Prototypen fertiggestellt werden, der in der Lage ist, eine Datei aus dem CVS auszuchecken, aus der Datenbank Informationen zu lesen und eine Verbindung zum Client ("6.2 Erste Iteration" auf der Seite 58) herzustellen.

5.2.1 Design

Das Komponentendiagramm des Servers im Anhang "C Diagramme" auf der Seite 85 gibt einen Überblick über das Design des gesamten Servers. Die folgenden Abschnitte und Diagramme erläutern die Funktion der einzelnen Komponentengruppen.

5.2.1.1 Startkomponente `PDServer`

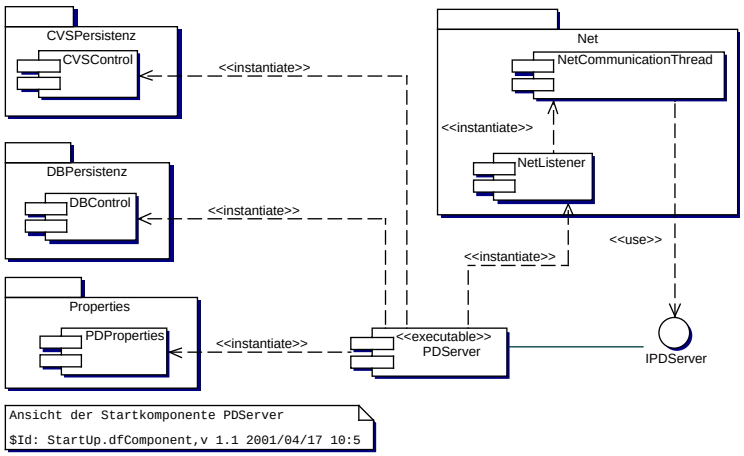

Abbildung 5.5: Ansicht der Startkomponente

Wie im Komponentendiagramm 5.5 zu sehen, instantiiert und initialisiert die Startkomponente `PDServer` die Hauptkomponenten in den vier Subsystemen "Properties", "DBPersistenz", "CVSPersistenz" und "Net". Anschließend wird

die Kontrolle an die `NetListener`-Komponente abgegeben. Die vier genannten Subsysteme werden in dieser Reihenfolge initialisiert.

5.2.1.2 Die Properties

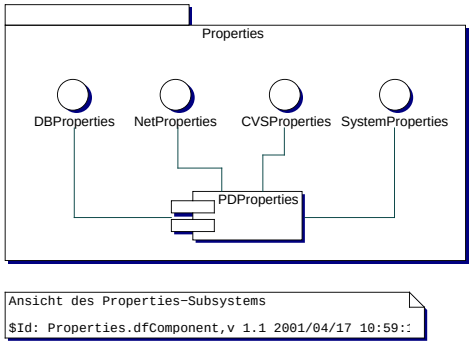

Abbildung 5.6: Ansicht des Properties-Subsystems

Die im Komponentendiagramm 5.6 gezeigte `PDProperties`-Komponente ist von `java.util.Properties` abgeleitet. Sie erstellt und speichert die für den gesamten Server notwendige Konfiguration. Damit die einzelnen Komponenten des Servers darauf zugreifen können, implementiert sie verschiedene Interfaces. Diese Interfaces stellen Getter- und Setter-Methoden für die einzelnen Konfigurationspunkte bereit. Zum Beispiel `String getCVSUser();` bzw. `void setCVSUser(String name);` zum Konfigurieren des zu verwendenden CVS Logins. Die anderen Interfaces erben vom Interface *SystemProperties*, um allgemeine Einstellungen in jeder Komponente zur Verfügung zu haben (siehe auch Klassendiagramm 5.12 auf der Seite 48).

5.2.1.3 Das DBPersistenz-Subsystem

Das Interface *IDBControl* im Diagramm 5.7 auf der folgenden Seite stellt für den Server Methoden für die einfache Datenbankabfrage bereit. Die Methoden liefern meist Objekte aus dem "Projektverwaltung"-Subsystem zurück. Die Klassen in dem Paket wurden von der Castor-JDO Bibliothek[1] generiert. Der Zugriff auf diese Objekte erfolgt ebenfalls über Funktionen dieser Bibliothek. Dieser Zugriff wird von `DBControl` für den Server abstrahiert.

[1] Abkürzung: **J**ava **D**ata **O**bjects; Persistenzschicht zu relationalen Datenbanken.

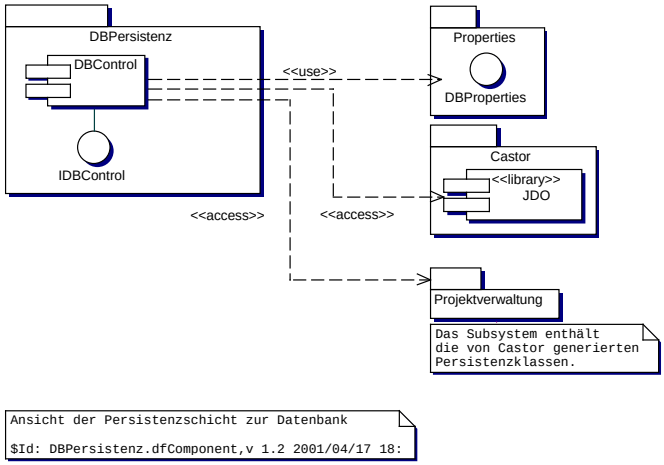

Abbildung 5.7: Die Datenbank-Persistenzschicht

Begonnen wurde das Design mit dem Entwurf einer JDBC[2]-Persistenzschicht. Nach der Entdeckung der Castor-Bibliothek wurde dieser jedoch verworfen und die Anbindung an die Datenbank mittels der JDO-Komponente von Castor realisiert.

5.2.1.4 Die Persistenzschicht zum CVS

Das Interface *ICVSControl* im Diagramm 5.8 auf der anderen Seite stellt für den Server Methoden zur Bedienung des CVS-Servers bereit. Die Methoden abstrahieren die Aufrufe des CVS-Kommandozeilentools unter Unix.

Im weiteren Verlauf der Entwicklung des Servers wäre zu prüfen, ob an die Stelle der Bedienung des Kommandozeilentools die Integration einer bestehenden Java-CVS-Implementierung treten kann. Mit jCVS II (Web:JCVS) existiert bereits ein sehr ausgereifter Java-CVS-Client. Die Klassen stehen jedoch in der "GNU General Public License" (Web:GPL), weshalb sie bisher nicht für eine Integration in Betracht kamen.

5.2.1.5 Die Netzwerkkomponenten

Die `NetListener`-Komponente in Diagramm 5.9 auf der anderen Seite initialisiert den ServerSocket und übergibt bei einer Client-Verbindung die Kontrolle an den `NetCommunicationThread`. Der `NetCommunicationThread` greift über

[2]Abkürzung: **J**ava **D**ata**B**ase **C**onnectivity (siehe auch Web:JDBC)

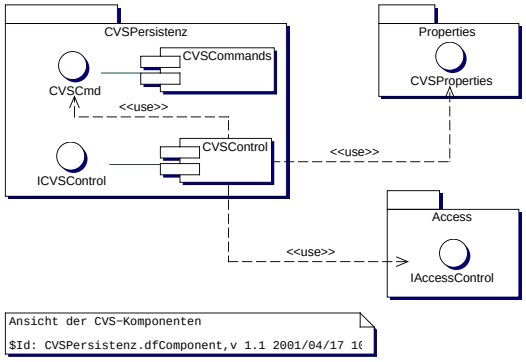

Abbildung 5.8: Die CVS-Persistenzschicht

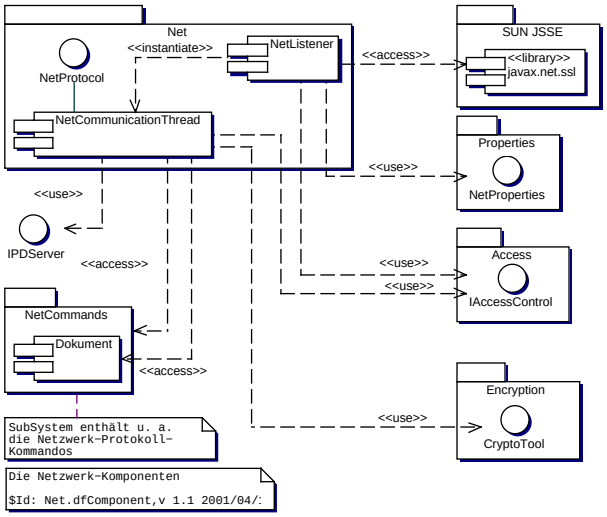

Abbildung 5.9: Die Netzwerkkomponenten

das Interface der Startkomponente auf die jeweiligen Subsysteme zu. Außerdem greift diese Komponente noch über die entsprechenden Interfaces auf die Subsysteme "Access" und "Encryption" zu. `NetCommunicationThread` ist bis jetzt auch die einzige implementierte Komponente, die *IAccessControl* nutzt. Sie führt darüber die Teilnehmerauthentifizierung durch. `AccessControl` greift dafür über die "DBPersistenz" auf die Teilnehmerdaten in der Datenbank zu.

`NetListener` soll in der nächsten Iterationsstufe eine IP-basierte Zugangskontrolle nutzen und `CVSControl` soll den Administrationszugang über "Access" abwickeln. Im Prototypen wird keine Filterung der IP-Adressen vorgenommen und der Administrationszugang zum CVS ist noch nicht implementiert.

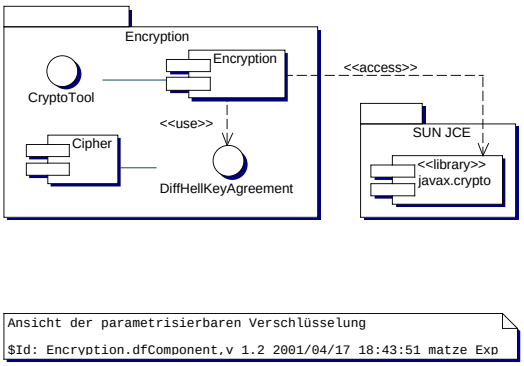

Abbildung 5.10: Die parametrisierbare Verschlüsselung

Das Subsystem "Encryption" (Diagramm 5.10) abstrahiert die parametrisierbare Verschlüsselung des Netzwerkverkehrs. Dazu werden einige Ciphers (zur Zeit nur "Blowfish" und "Null") und das Diffie-Hellmann-Keyagreement aus der JCE-Referenzimplementierung von Sun (Web:JCE) eingebunden. "Encryption" wird jedoch nur benutzt, wenn `NetListener` keine SSL-Sockets benutzt. Dies ist zum Beispiel sinnvoll, wenn keine gültigen SSL-Zertifikate vorliegen oder eine stärkere Verschlüsselung benötigt wird, als SSL bieten kann. `NetListener` greift auf die SSL-Implementierung von Sun's JSSE-Referenzpaket (Web:JSSE) zu.

Der `NetCommunicationThread` steuert auch das `Dokument` und veranlasst zum Beispiel den Abgleich der Metadaten im Dokument mit den Daten in der Datenbank über *IDBControl*.

Das Subsystem "NetCommands" greift außerdem auf die Castor-XML Bibliothek zu, um die Objekte in XML-Dokumente zu serialisieren, bevor sie über das Netzwerk geschickt werden können.

5.2.2 Implementierung

In den folgenden Abschnitten und Klassendiagrammen wird gezeigt, wie die Vorgaben aus dem Design umgesetzt wurden. Im Anhang "C Diagramme" auf der Seite 85 befindet sich noch eine Gesamtübersicht über die Serverklassen.

5.2.2.1 Die Startup Klassen

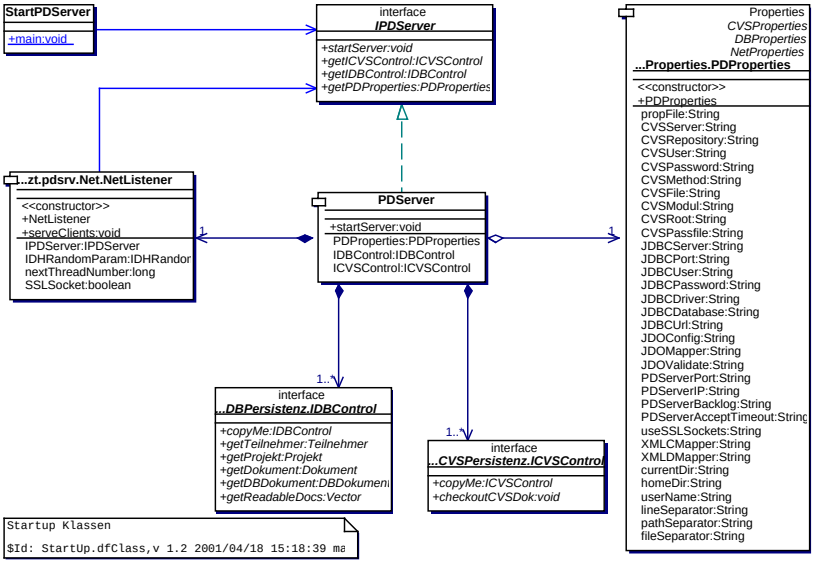

Abbildung 5.11: Die Startup Klassen

Wie im Klassendiagramm 5.11 zu sehen, enthält `StartPDServer` die `main()`-Methode. Die Klasse enthält eine Singelton-Implementierung, um sicher zu gehen, dass nur eine Instanz von `PDServer` erstellt wird (Gamma u. a. 1996, S. 157 ff.). Es werden ebenfalls die Argumente der Kommandozeile weitergereicht.

`PDServer` instanziiert im Konstruktor zuerst `PDProperties` und reicht für die Initialisierung des Objekts die Kommandozeilenargumente weiter. Nach erfolgreichem Erstellen der Properties werden die Peristenzschichten zur Datenbank (`DBControl`) und zum CVS-Server (`CVSControl`) instanziiert. Danach folgt die Instanziierung von `NetListener` und die Übergabe der Kontrolle an dieses Objekt.

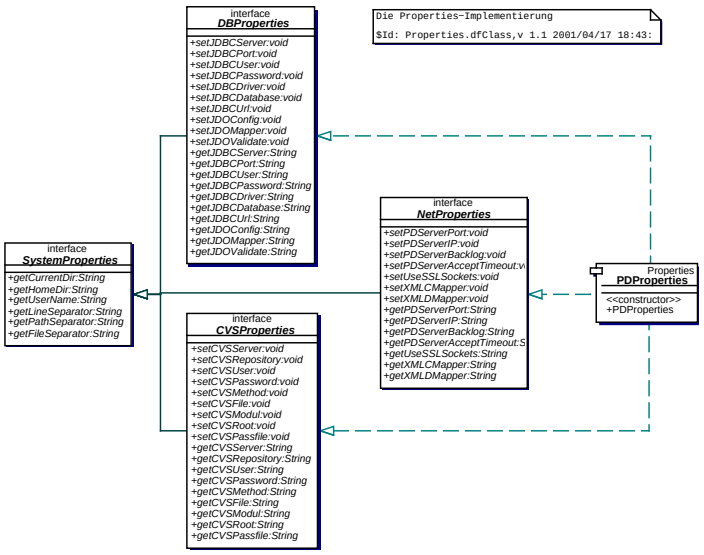

Abbildung 5.12: Das Properties package

5.2.2.2 Die Properties

Das Klassendiagramm 5.12 zeigt die Implementierung der Properties. `PDProperties` implementiert die Getter- und Settermethoden für die CVS-, DB- und Netzwerkkonfiguration und durch Vererbung innerhalb der Interfaces auch das Interface *SystemProperties*. Die Klasse hat auch noch einige private Methoden, die vom Konstruktor aufgerufen werden, um zum Beispiel die Kommandozeilenargumente zu parsen oder eine Properties-Datei zu laden.

5.2.2.3 Die Datenbank-Persistenz

Im Klassendiagramm 5.13 auf der anderen Seite ist die Umsetzung des Designs aus dem Komponentendiagramm 5.7 auf der Seite 44 zu sehen.

`DBControl` implementiert das Interface *IDBControl* und stellt darüber für den Server Methoden zur Abfrage der Datenbank bereit. Konkret können bis jetzt die Tabellen "Teilnehmer", "Projekt" und "Dokument" abgefragt werden. Zusätzlich gibt es noch "Bequemlichkeits"-Methoden, um zum Beispiel einen `Vector` aller Dokumente, auf die ein Teilnehmer lesenden Zugriff hat, aus der Datenbank zu erhalten. Die Methode `copyMe()` klont das aktuelle Objekt oder, falls das nicht möglich ist, erstellt ein neues Objekt mit der gleichen Konfigura-

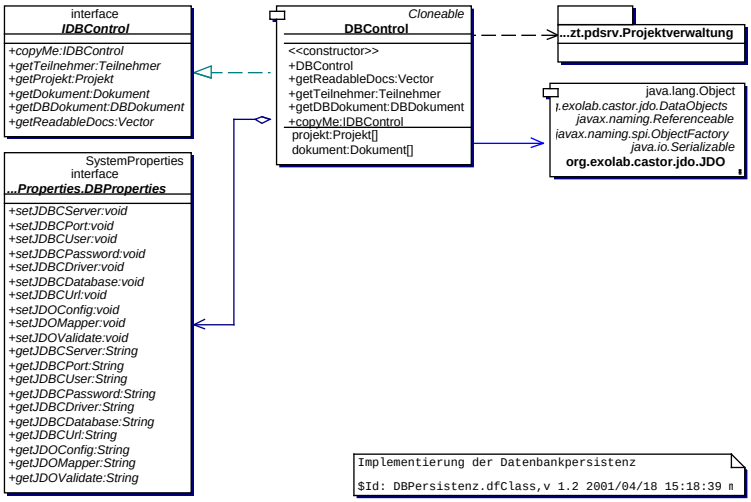

Abbildung 5.13: Die Implementierung der DB-Komponenten

tion, wie das aktuelle.

`DBControl` konfiguriert sich bisher lediglich aus den JDO-Methoden des *DBProperties*-Interfaces.

Castors `JDO`-Klasse wird benötigt, um die Persistenzobjekte aus dem `Projektverwaltung`-package zu bedienen. Diese Objekte haben getter- und setter-Methoden für jede Tabellenspalte in der Datenbank. Es ist jedoch zu beachten, dass die Objekte transient werden, sobald sie außerhalb der Persistenzschicht existieren. Transient heißt, dass sich Änderungen am Objekt nicht in der Datenbank widergespiegeln. Erst das Zurückführen der Objekte in die Persistenzschicht macht auch Änderungen wieder persistent.

5.2.2.4 Die CVS-Persistenz

`CVSControl` im Klassendiagramm 5.14 auf der folgenden Seite wird hauptsächlich über die Methoden `getCVSRoot()`, `getCVSFile()` und `getCVSPassfile()` konfiguriert. Ist kein CVSROOT angegeben, ist der Konstruktor in der Lage aus den Angaben zu Methode, User, Server und Repository ein CVSROOT zu erstellen. Ein CVSROOT hat das Format `:Methode:User@Server:Repository` (Fogel 1999). Ist als Methode "pserver" angegeben, überprüft der Konstruktor außerdem, ob er in der Lage ist, sich auf dem CVS-Server mit der angegebenen Konfiguration einzuloggen.

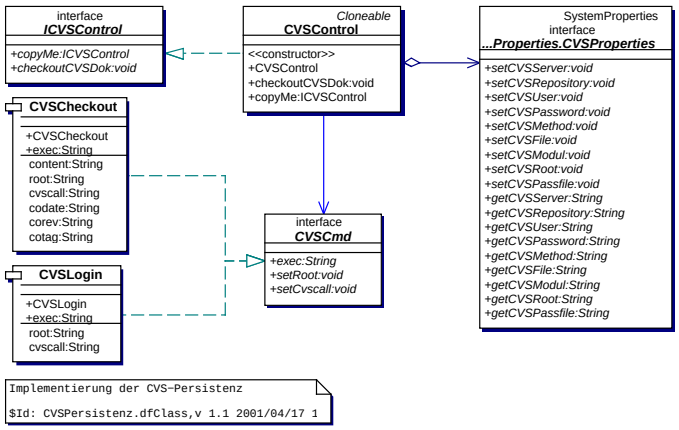

Abbildung 5.14: Die Implementierung der CVS-Komponenten

Die CVS-Kommandos (Objekte der das Interface *CVSCmd* implementierenden Klassen) erhalten über ihren Konstruktor das CVSROOT und den Systemaufruf des CVS-Kommandozeilentools (genannt `cvscall`) und müssen anhand des über die Methode `exec()` übergebenen Arguments einen gültigen Toolaufruf durchführen. Das Muster so eines Aufrufs sieht wie folgt aus:

```
<cvscall> -d <cvsroot> <Optionen> <Kommando> <Optionen> <Argument>
```

Die erstgenannten "Optionen" stellen allgemeine Optionen des CVS-Tools dar, während die folgenden "Optionen" abhängig vom "Kommando" sind. Benötigt das Kommando weitere Optionen, müssen entsprechende Methoden zur Konfiguration vorgesehen sein (wie zum Beispiel beim `CVSCheckout`). Gleichzeitig ist eine vorgegebene Konfiguration sicherzustellen, so dass ein bloßes Instanziieren mit anschließendem `exec()`-Aufruf ebenso funktionieren kann. Für diesen Fall sieht der CVS-Aufruf von `CVSCheckout` zum Beispiel so aus:

```
<cvscall> -d <cvsroot> checkout -pf <Argument>
```

Dieser Aufruf checkt die aktuelle Headrevision des in "Argument" angegebenen Moduls aus dem Server aus. Wenn jedoch eine andere Revision ausgecheckt werden soll, muss diese bevor `exec()` aufgerufen wird mit `setCorev()` eingestellt werden. `CVSCheckout` fügt dann selbstständig noch ein "`-r <corev>`" in den Aufruf ein. Anschließend kann mit `getContent()` das ausgecheckte Modul ausgelesen werden. `exec()` liefert an dieser Stelle nur Meldungen vom CVS-Server zurück.

Andere Kommandos können auch den Rückgabewert von `exec()` für das Ergebnis des Aufrufs vorsehen. Hier ist abzuwägen, wie das Kommando am

sinnvollsten gekapselt werden kann. Für jedes Kommando ist im Interface *ICVS-Control* nur ein Aufruf mit möglichst wenigen Argumenten vorzusehen.

5.2.2.5 Netzwerk- und Verschlüsselungsklassen

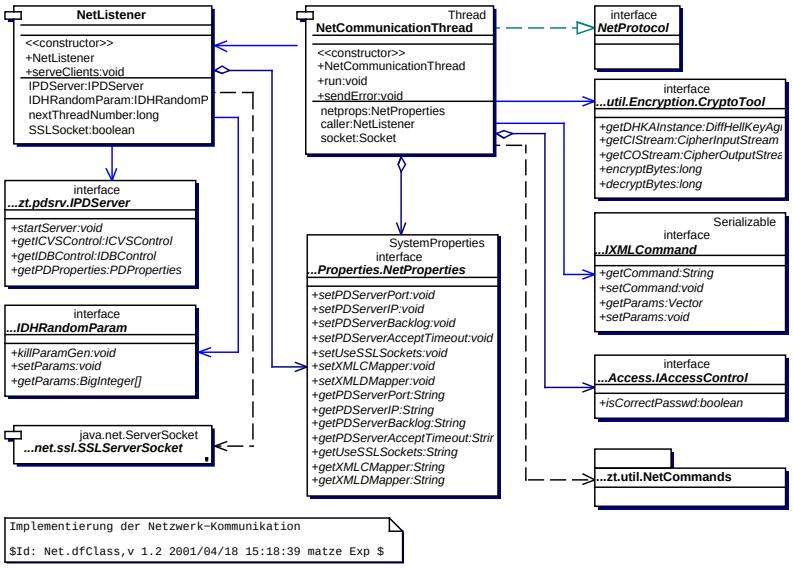

Abbildung 5.15: Die Implementierung der Netzwerk-Kommunikation

Das Klassendiagramm 5.15 zeigt die Implementierung der Netzwerkschicht. `NetListener` erhält von `PDServer` die Kontrolle und richtet einen ServerSocket ein. Ist in der Konfiguration SSL gewünscht, nutzt es die `SSLServerSocket-Factory` aus den `javax.net.ssl`-package, um einen ServerSocket zu erstellen. Andernfalls wird schlicht `java.net.ServerSocket` instanziiert.

Konfiguriert werden kann neben der IP-Addresse, an der der Server lauschen soll, auch die Portnummer, das BackLog und ein AcceptTimeout. Letzteres wird dazu benutzt die ThreadGroup der verbundenen Clients aufzuräumen.

Ist kein SSL gewünscht, wird ein paralleler Thread gestartet, der für die parametrisierbare Verschlüsselung Seed-Werte[3] generiert (siehe Interface *IDHRandomParam* und damit zusammenhängende Klassen in Abb. 5.16 auf der folgenden Seite). Dieser Vorgang ist deshalb in einen zusätzlichen Thread ausgelagert, weil das Generieren eines Wertes relativ lange dauert (bis zu 5 Minuten auf einem

[3] Anfangswerte für den Zufallsgenerator.

Intel Pentium III 700MHz / Linux 2.4 / J2SDK[4] 1.3). Um einen "Vorrat" an Wertepaaren zu haben, läuft dieser Thread so lange, bis 10 Wertepaare in einer Queue vorhanden sind. Danach wird er schlafen gelegt und wieder aufgeweckt, wenn die Queue weniger als 8 Paare aufweist. Jede Abfrage eines Wertepaares entfernt dieses aus der Queue – es sei denn, es ist das letzte Paar in der Queue.

Bei einer Clientverbindung wird ein Thread (`NetCommunicationThread`) gestartet, der diese Verbindung handhabt. In dem Thread wird das Protokoll abgearbeitet und es werden die Clientanfragen bedient. Das Protokoll besteht aus den Elementen Begrüssung, parametrisierbare Verschüsselung (nur wenn kein SSL), Authentifizierung, Senden einer Dokumentenliste, Reaktion auf Clientanfrage durch Senden eines Dokuments und Verabschiedung.

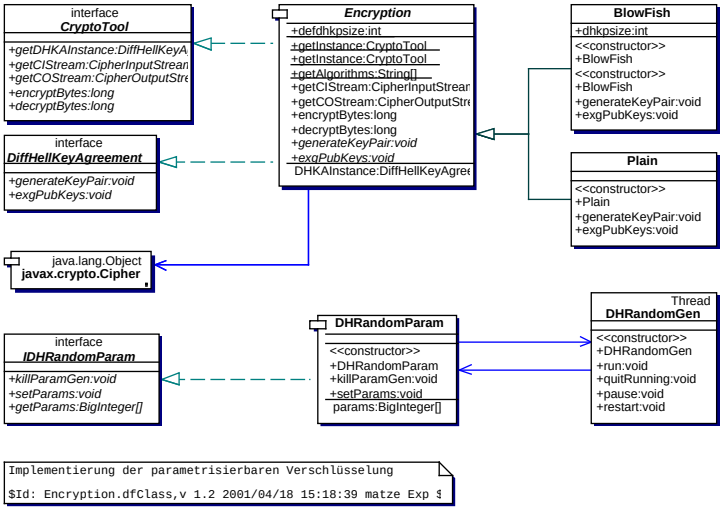

Abbildung 5.16: Implementierung der parametrisierbaren Verschlüsselung

Die Implementierung der parametrisierbaren Verschlüsselung ist im Klassendiagramm 5.16 abgebildet. Diese Klassen kommen nur zum Zug, wenn keine SSL Sockets benutzt werden. Die Implementierung sieht vor, dass kein Dokument über eine unverschlüsselte Verbindung gesendet werden darf. Da es jedoch vorkommen kann, dass Verschlüsselung nicht erlaubt ist, gibt es die Klasse `Plain`, die den `javax.crypto.NullCipher` des JCE-Frameworks benutzt. Dieser Cipher implementiert keine Verschlüsselung. Für das Dokument sieht es jedoch so aus, als wäre eine verschlüsselte Verbindung geschaffen.

Als Verschlüsselung wird bisher nur die JCE-Implementierung des freien

[4] Abkürzung: **J**ava **2** **S**oftware **D**evelopment **K**it

Blowfish-Algorithmus' von Bruce Schneier (Schneier 1994; Berg 1999) verwendet. Auch ist noch keine Parametrisierung des Ciphers implementiert. Benutzt wird die voreingestellte Schlüssellänge von 56 Bit.

5.2.3 Test

Am Server wurden bisher nur Smoketests zusammen mit dem Client durchgeführt. Diese Tests dienen zur Überprüfung der Funktionalität. Kompatibilitäts- und Profiling-Tests, zur Untersuchung der Speicher und Rechenzeitnutzung, stehen noch aus.

Für das Testen des Servers wurde ein Clientstub geschrieben, der das Netzwerkprotokoll bedienen konnte, jedoch keine interaktiven Elemente besaß. Mit dieser Testkonstellation konnte festgestellt werden, dass die Castor-Library noch einige grobe Ungereimtheiten aufweist. Auch wurde eine fehlerhafte Implementierung der parametrisierbaren Verschlüsselung offenbar.

Das letztere Problem konnte durch eine Überarbeitung des Sources beseitigt werden. Das Problem in der Castor-Library jedoch ließ sich nur durch einen Workaround teilweise beheben.

Eine Abfrage einer Tabelle in der Datenbank ließ sich nicht durchführen, obwohl eine Abfrage nach gleichem Muster auf einer anderen Tabelle funktionierte. Als Workaround blieb nur die Aufsplittung in zwei Teilabfragen. Diese Aufsplittung funktioniert zur Zeit allerdings nur, weil der Prototyp eine Relation in der Datenbank noch nicht benötigt. Im Laufe der weiteren Entwicklung ist mit neueren Versionen der Castor-Library zu überprüfen, ob dieser Workaround wieder entfernt werden kann.

Sollte sich bei dieser Überprüfung herausstellen, dass der Fehler immer noch existiert, ist eine Einarbeitung und Fehlersuche in den Sourcen von Castor angebracht, um die genaue Fehlerursache herauszufinden und möglicherweise auch zu entfernen.

Die Bearbeitung der beiden oben genannten Fehler dauerte jeweils fast vier Tage. Außerdem kamen noch einige kleinere Fehler zum Vorschein die jeweils relativ schnell beseitigt werden konnten. Die gesamte Testphase dauerte aber sehr viel länger als geplant, wodurch die Implementierung des Clients um mehr als zwei Wochen in Verzug geriet.

6 Entwicklung des Clients

Auf den folgenden Seiten wird ein Überblick über das Design und den Stand der Implementierung des Clients gegeben. Da bei der Implementierung des Clients bereits klar war, dass die zweite Iteration in der vorgegebenen Zeit nicht zu schaffen ist, wurden Teile der zweiten Iteration noch mit in die erste übernommen, um zumindest einen Überblick über angedachte Funktionalitäten geben zu können.

6.1 Vorbereitung

Zur Vorbereitung gehörten, neben den Überlegungen zu den externen Schnittstellen des Clients, auch das Design der grafischen Oberfläche. Die zu verwendenden Technologien waren weitgehend durch die Implementierung des Servers vorgegeben, so dass nur noch die Übertragung auf die Bedürfnisse des Clients notwendig war.

6.1.1 Schnittstelle zum Server

Diese Schnittstelle gehört zu jenen Technologien, die durch die Implementierung am Server bereits vorgegeben waren.

Der Client benötigt als Schnittstelle zum Server ein Netzwerkinterface und Verschlüsselung. Zur Verschlüsselung sollen entweder SSL oder die Klassen der parametrisierbaren Verschlüsselung (siehe auch "5.2.2.5 Netzwerk- und Verschlüsselungsklassen" auf der Seite 51) eingesetzt werden.

Das Netzwerkinterface muss die Client-Seite des Protokolls implementieren und in der Lage sein die XML-Kommandos entsprechend zu interpretieren.

6.1.2 Lokale I/O-Schnittstelle

Der Client ist als Abstraktionstool zu den XML-Dokumenten (wie zum Beispiel BlackBox-Dokumente) gedacht. Es ist aber durchaus denkbar, dass der eine oder andere Nutzer weiß wie er mit XML-Dokumenten umgehen muss und daher möglicherweise lieber mit einem anderen Tool arbeiten möchte. Deshalb soll es eine Schnittstelle geben, die das Exportieren des XML-Dokuments in das lokale Filesystem erlaubt. Ebenso soll ein Import vorgesehen sein, der auch eine Validierung des Dokuments durchführen können muss.

Für die Validierung des BlackBox-Dokuments wurde ein XML-Schema erstellt. Eine grafische Repräsentation ist im Anhang unter "C Diagramme" auf der Seite 85 zu finden. Dieses Schema wird auch für die Generierung der Klassen des BlackBox-Dokuments durch die Castor-Library benötigt.

Für die Erstellung des XML-Schemas wurde die BlackBox-Richtlinie als Vorlage genommen. Als Root-Element wurde "Dokument" mit dem Namen des Dokuments und dem Typ "BlackBox" als Attribute gewählt. In der nächsten Hierarchiestufe wurden die vier Teile "Titelseite", "Historie", "Inhalt" und "Kapitel" geschaffen. Die ersten drei Teile repräsentieren jeweils die ersten drei Blätter der Richtlinie.

Die "Titelseite" enthält hauptsächlich Einzeiler, die das Dokument näher beschreiben.

Unter "Historie" wird zusätzlich zur Evolution des Dokuments, noch die "Verteilung" und die "Abnahme" dokumentiert.

Der "Inhalt" enthält eine Liste der Kapitel und Unterabschnitte, jeweils mit Namen und Hierarchiestufe. Der Inhalt sollte möglichst automatisch generiert werden.

Unter "Kapitel" schließlich, sind die vier Kapitel mit ihren jeweiligen Unterabschnitten beschrieben. Die Kapitel und Abschnitte sind dabei nicht namentlich genannt, die Festlegung der Überschriften muss in der Applikation oder im XML Template für die BlackBox-Dokumente geschehen. Unterhalb von Kapitel sind noch weitere Elemente – meist für Textauszeichnungen (z. B. "Schrift", "Liste", etc.) – definiert. Ferner können Methodenbeschreibungen und URL's mit eingebunden werden.

6.1.3 Graphische Benutzeroberfläche

Die im Screenshot 6.1 auf der anderen Seite abgebildete Oberfläche ist die des Clients nach dem Start. Das Swing-Layout wurde mit dem JBuilder 4 (Web:JBuilder) erstellt. Das Layout teilt sich in vier Bereiche: Menü, Symbolleiste, kombinierter Navigator / Editor und die Statuszeile. Außerdem wurden noch ein paar Dialoge entworfen, zum Beispiel zur Eingabe der Login-Informationen für den Server.

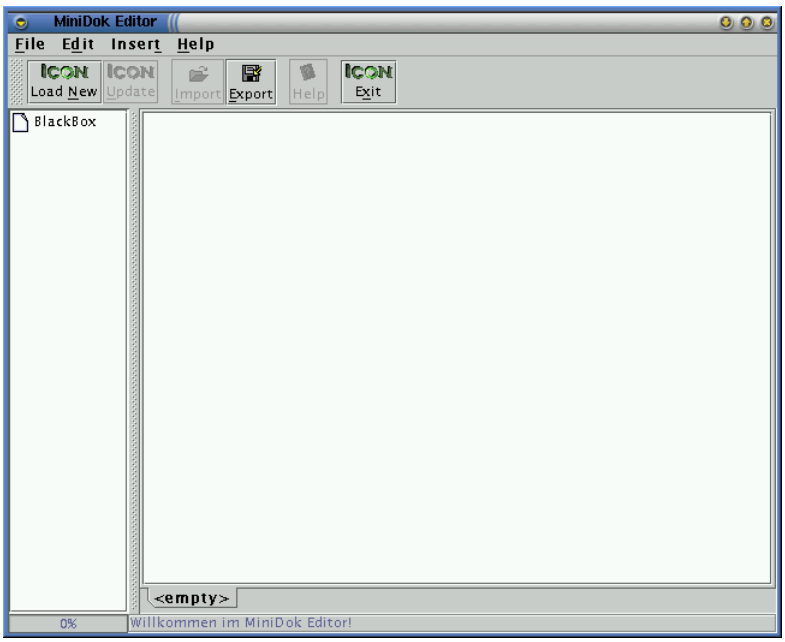

Abbildung 6.1: GUI nach dem Aufruf

6.1.3.1 Die Menüleiste

Die Menüleiste vereint alle Ein- und Ausgabefunktionen im Menü "File". Dazu gehören sowohl das Laden bzw. Updaten eines Dokuments vom bzw. zum Server, das Im- und Exportieren eines Dokuments vom bzw. in das lokale Filesystem als auch das Beenden des Programms.

Im Menü "Edit" ist bisher nur der Aufruf eines Konfigurationsdialogs ("Options...") vorgesehen.

Unter "Insert" können Editoren für spezielle Elemente der Kapiteltexte aufgerufen werden. Nach dem Ausfüllen oder Editieren der Elemente werden diese im Text eingefügt. Zu diesen speziellen Elementen gehören Methoden, URL's und Grafiken.

Das Menü "Help" schließlich enthält einen Punkt der einen Webbrowser mit den Hilfeseiten aufruft ("Content..."), einen Punkt der Informationen über das geladene Dokument preisgibt ("About Dokument...") und "About MiniDok..." mit Versionsinformationen zum Client.

6.1.3.2 Die Symbolleiste

Die Symbolleiste verkürzt den Weg zu den wichtigsten Menüpunkten. Aktiv sind wegen des aktuellen Standes der Implementierung (siehe "6.2.2 Implementierung" auf der Seite 62) nur die Punkte "Load New", "Export" und "Exit".

Die Icons in "Load New" und "Exit" sind Dummy-Icons und müssen noch ersetzt werden. Das Icon unter "Export" ist ein vom JBuilder generiertes Standardicon.

6.1.3.3 Navigator / Editor

In der linken Hälfte wird das geladene Dokument als Baumstruktur dargestellt und als Navigator benutzt. Für einen schnellen Zugriff können auch einzeilige Strings direkt dort editiert werden. Die rechte Seite stellt den Raum für den Editor zur Verfügung. Beide Teile sind abhängig von der Implementierung des Objektmodels aus dem XML-Schema des Dokumententyps (für den Typ "BlackBox" siehe "6.2.2.4 Der Dokumenteneditor" auf der Seite 66).

6.1.3.4 Statuszeile

Die Statuszeile enthält zwei Komponenten. Links ist eine Prozentbalkenanzeige (`javax.swing.JProgressBar`) und rechts eine Anzeige für Statusmeldungen. Diese beiden Komponenten sind von außerhalb der Oberfläche steuerbar und sollen zum Beispiel den Fortschritt beim Laden eines Dokuments anzeigen.

6.2 Erste Iteration

Der erste Entwicklungszyklus beim Client ist in engem Zusammenhang mit dem des Servers zu sehen. Der Client stellt für den Server in dieser Phase nicht viel mehr als ein Testtool bereit. Der Client wird sich mit dem Server verbinden, beim Server authentifizieren und ein Dokument herunterladen können.

6.2.1 Design

Das Komponentendiagramm des Clients im Anhang "C Diagramme" auf der Seite 85 gibt einen Überblick über das Design des gesamten Clients. Die folgenden Abschnitte und Diagramme erläutern die Funktion der einzelnen Komponentengruppen.

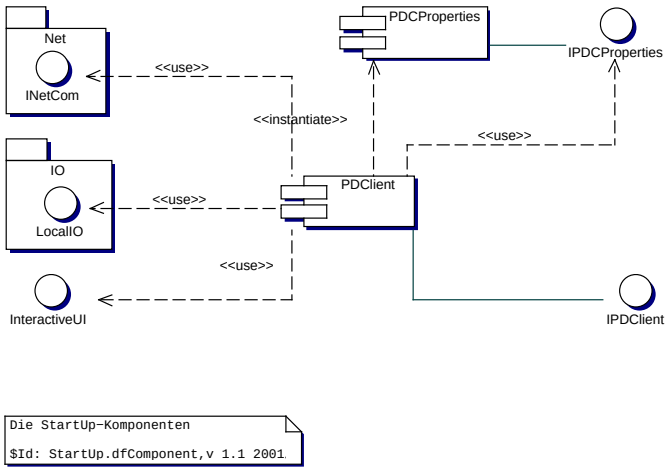

Abbildung 6.2: Die StartUp-Komponenten des Clients

6.2.1.1 Startkomponenten

Das Komponentendiagramm 6.2 zeigt die beim Start des Clients wirksamen Komponenten. Nach der Instanziierung von `PDClient` sorgt dieser dafür, dass sich der Client aus einer Properties-Datei und eventuell übergebenen Kommandozeilenargumenten konfiguriert.

`PDClient` ist die zentrale Komponente des Clients. Sie steuert den Zugriff auf die Schnittstellen nach außen und die Konfiguration des gesamten Clients. Durch die Trennung von Oberfläche und Funktionalität (mehr dazu unter "6.2.1.4 Der Dokumenteneditor" auf der Seite 61) geht die Steuerung immer durch das Interface *IPDClient* und die implementierende Komponente `PDClient` spricht im Anschluß daran *LocalIO*, *INetCom* oder *IPDCProperties* an.

6.2.1.2 Die Netzwerkschicht

Die Netzwerkschicht des Clients (Komponentendiagramm 6.3 auf der folgenden Seite) baut sehr stark auf der des Servers auf. Das Subsystem "Encryption" kann zum Beispiel von Server und Client benutzt werden. Es wird daher hier nicht noch einmal erläutert, sondern nur auf den Abschnitt "5.2.1.5 Die Netzwerkkomponenten" auf der Seite 44 verwiesen.

`NetCom` holt sich seine Konfigurationsdaten (Server, Portnummer, SSL, etc.) über das Properties-Interface *IPDCProperties* und stellt eine entsprechende Ver-

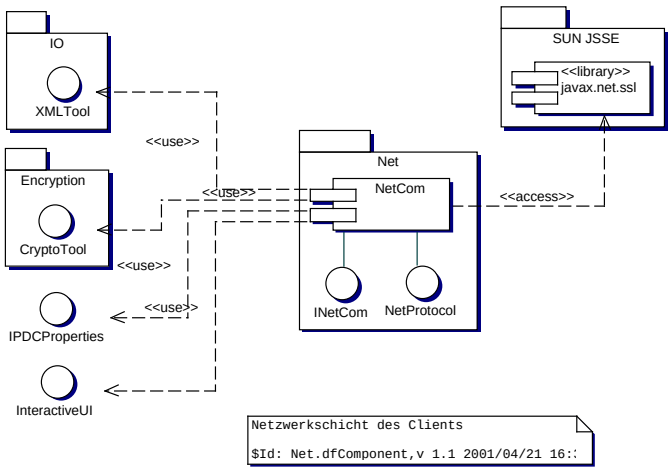

Abbildung 6.3: Die Netzwerkschicht des Clients

bindung zum Server her. Wie beim Server gilt auch hier, dass entweder SSL oder die parametrisierbare Verschlüsselung benutzt werden soll. Außerdem muss die Komponente den Client-Teil des Netzwerkprotokolls implementieren. Dazu kann sie das *XMLTool* nutzen, das für die (De-)Serialisierung der Protokoll-Kommandos verantwortlich ist.

Da es sich bei dem Client um ein Offline-Tool handelt, braucht die Netzwerkschicht nur aktiv zu werden, wenn sie auch wirklich Daten mit dem Server austauschen soll.

6.2.1.3 Die lokale Ein- und Ausgabe

Das Komponentendiagramm 6.4 auf der anderen Seite zeigt die Komponenten, die mit der lokalen Datenein- und -ausgabe zu tun haben.

Dazu gehört nicht nur die `FileIO`, die für das lokale Laden und Speichern von Dateien verantwortlich ist, sondern auch das *XMLTool* in der Implementierung unter Zuhilfenahme der Castor-XML-Library und ein "Command-Line-Interface" `CLI`, das eine minimalistische Bedienungsoberfläche zur Verfügung stellen soll.

`CastorImpl` benötigt die Pfade der XML-Mapping-Dateien und `FileIO` holt sich aus den Properties verschiedene Systemproperties, zum Beispiel das aktuelle Verzeichnis und den Dateinamen-Separator.

CryptoTool soll für die Verschlüsselung der lokal gespeicherten Dateien benutzt werden. Dafür ist jedoch eine komplexe Nutzerverwaltung mit PKI not-

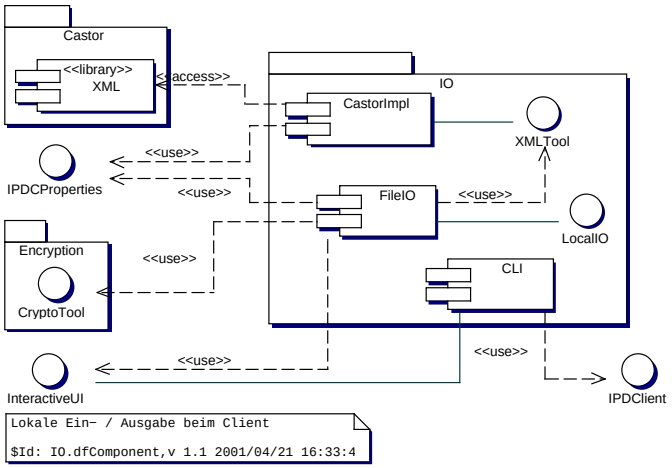

Abbildung 6.4: Die lokale Ein- / Ausgabeschicht des Clients

wendig, weshalb diese Funktionalität nicht im Prototypen implementiert wird.

Das `CLI` stellt ein Downloadtool ohne Editorfunktionalität auf Kommandozeilenebene zur Verfügung.

6.2.1.4 Der Dokumenteneditor

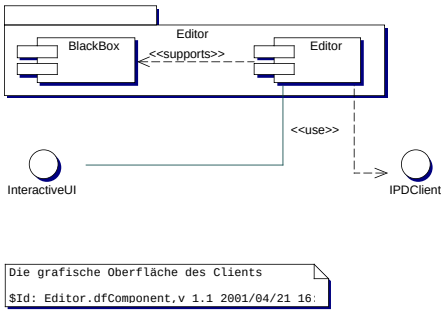

Abbildung 6.5: Die grafische Oberfläche des Clients

Im Diagramm 6.5 sind die Komponenten der grafischen Bedienungsoberfläche dargestellt. Die Oberfläche vereint die Funktionalität eines Downloadtools

und eines Editors miteinander. Der Editor unterstützt die von Castor aus dem XML-Schema für BlackBox-Dokumente generierten Klassen. Diese Funktionalität ist so speziell nur für den Prototypen des Clients relevant. An ihre Stelle soll eine allgemeinere Schnittstelle treten, die es der Oberfläche ermöglicht, dynamisch – je nach Dokumententyp – die entsprechenden Editorklassen zu laden.

Die Bedienungsoberflächen `CLI` und `Editor` müssen in eigenen Threads laufen. Die Oberflächen sind nur lose an den Client zu koppeln, die Interfaces *InteractiveUI* und *IPDCLient* stellen die einzigen Berührungspunkte zwischen den beiden Teilen des Clients dar. Das Komponentendiagramm 6.6 stellt noch einmal die gedachten Kommunikationswege dar. Die Oberflächen kommunizieren über das Interface *IPDClient* mit dem Client und die Client-Komponenten kommunizieren über *InteractiveUI* mit der Oberfläche.

Diese Trennung und die Auslagerung in einen eigenen Thread ist wichtig für die Kommunikation mit dem Benutzer. Wenn also zum Beispiel ein Download stattfindet, darf die Oberfläche nicht blockiert sein, sondern soll dem Benutzer stets über den aktuellen Stand des Downloads informieren.

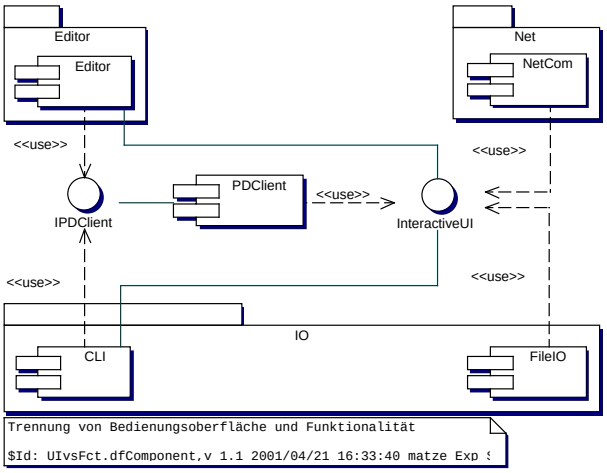

Abbildung 6.6: Lose Kopplung zwischen UI und Client

6.2.2 Implementierung

Wie eingangs schon erwähnt, wurde die Implementierung der ersten Iteration um Features erweitert, die erst in der zweiten Iteration hinzukommen sollten. Durch das aufwändige Fehlersuchen nach der Implementierung der ersten Iteration des Servers, blieb keine Zeit mehr, um noch eine vollständige zweite Iteration

durchzuführen. Deshalb wurde beschlossen, dass der Client jetzt schon seine grafische Oberfläche erhält, in der zumindest im Ansatz zu erkennen ist, welche Funktionalität diese einmal haben soll.

6.2.2.1 Die StartUp-Klassen

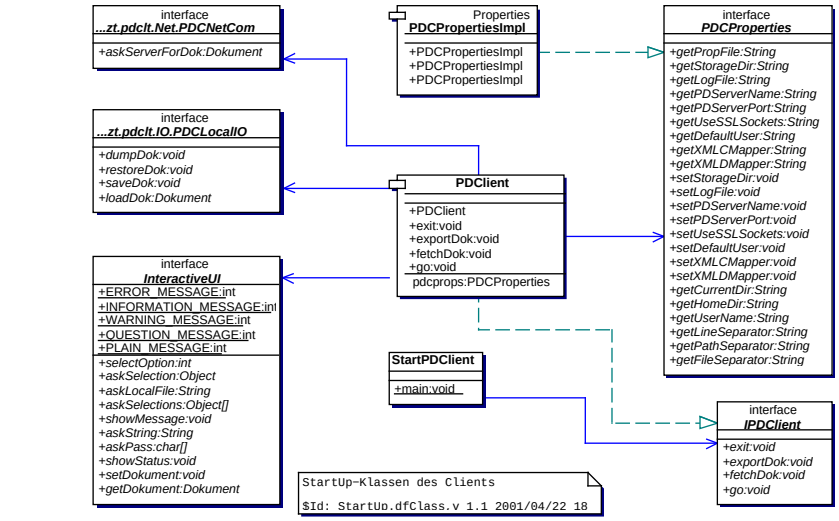

Abbildung 6.7: Die StartUp-Klassen des Clients

Das Klassendiagramm 6.7 zeigt die beim Start des Clients aktiven Klassen. `StartPDClient` enthält eine Singleton-Implementierung für `PDClient`, um sicherzustellen, dass innerhalb einer JVM nur ein Client läuft.

Nachdem `PDClient` instanziiert wurde, wird `PDCPropertiesImpl` erstellt. `PDClient` reicht an dieses Objekt die Kommandozeilenargumente weiter. Die Instanz konfiguriert sich dann aus einer Propertiesdatei und den Kommandozeilenargumenten. Über das Interface *PDCProperties* wird die Konfiguration den restlichen Komponenten des Clients zur Verfügung gestellt.

Danach wird die grafische Oberfläche gestartet. Sobald diese instanziiert wurde, wird der `main()`-Thread schlafen gelegt. Er wird erst wieder aktiv, wenn die Oberfläche Aktionen, wie zum Beispiel `fetchDok()`, am Client auslöst.

6.2.2.2 Die Netzwerkschicht

Hat die Oberfläche die Methode `fetchDok()` im Interface *IPDClient* ausgelöst, instanziiert der Client `NetCom` und beauftragt das Objekt mittels `askServer-`

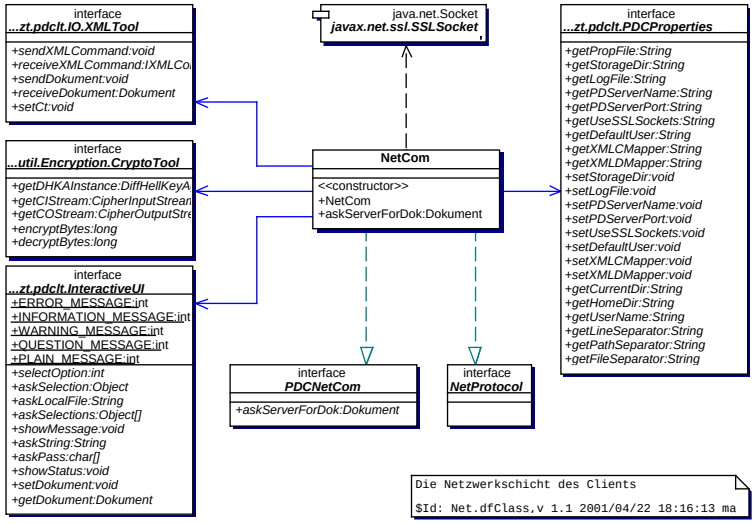

Abbildung 6.8: Die Netzwerk-Klassen des Clients

`ForDok()` aus dem Interface *PDCNetCom* ein Dokument vom Server zu laden. Das Klassendiagramm 6.8 zeigt die Umgebung dieser Klasse.

Sollte `NetCom` Informationen vom Nutzer benötigen (Login, Passwort oder auch Auswahl welches Dokument geladen werden soll), erfragt es die Informationen mittels der, vom Interface *InteractiveUI* zur Verfügung gestellten, Methoden.

Je nach Konfiguration benutzt `NetCom` entweder SSL-Sockets oder "normale" Sockets mit der parametrisierbaren Verschlüsselung aus dem "Encryption"-package (siehe "5.2.2.5 Netzwerk- und Verschlüsselungsklassen" auf der Seite 51).

`NetCom` implementiert die Client-Seite des Netzwerk-Protokolls und nutzt die Methoden aus *XMLTool*, um die XML-Kommandos zu (de-)serialisieren und über das Netzwerk auszutauschen.

Ist ein Dokument vom Server gesendet worden, wird es an `PDClient` als Rückgabewert aus `askServerForDok()` übergeben.

6.2.2.3 Die lokale Ein- und Ausgabe

Die im Diagramm 6.9 auf der anderen Seite dargestellten Klassen implementieren zwei verschiedene Funktionalitäten. Zum einen wird die Ein- und Ausgabe zwischen Client und dem lokalen Filesystem in *PDCLocalIO* definiert und zum

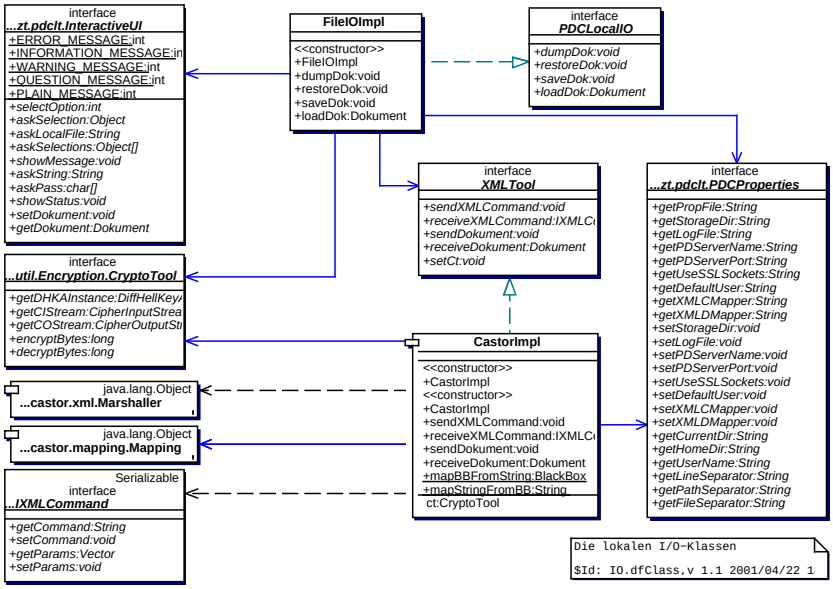

Abbildung 6.9: Die Klassen für die lokale Ein- und Ausgabe

anderen wird die Funktionalität der Castor-XML-Library in `CastorImpl` zu einzelnen Methoden zusammengeführt.

Eigentlich sah an dieser Stelle die erste Iteration noch eine Kommandozeilen-Oberfläche vor (siehe `CLI` im Komponentendiagramm 6.4 auf der Seite 61), die ein einfaches Downloadtool zur Verfügung stellen sollte. Aus den einleitend genannten Gründen, wurde die Implementierung dieser Komponente zu Gunsten der grafischen Oberfläche in die folgende Iteration verschoben.

Wenn die Oberfläche die Methode `exportDok()` im Client aufruft, weist der Client via `saveDok()` in *PDCLocalIO* die implementierende Klasse an, das Dokument in das lokale Filesystem zu serialisieren. `FileIO`, als implementierende Klasse, erfragt dann vom Nutzer über die Methoden in *InteractiveUI* einen Dateinamen und serialisiert das aktuelle BlackBox-Dokument als XML-Dokument in die entsprechende Datei.

`dumpDok()` in *PDCLocalIO* wird benutzt, um den gesamten vom Server erhaltenen Dokumentencontainer (siehe Abbildung 5.2 auf der Seite 40) an eine konfigurierbare Position im Filesystem (Methode `getStorageDir()` im Interface *PDCProperties*) als XML-Dokument zu speichern. Dies soll im weiteren Verlauf als Backup und "Quicklaunch" dienen. "Quicklaunch" soll beim Starten des Clients automatisch das zuletzt editierte Dokument laden.

6.2.2.4 Der Dokumenteneditor

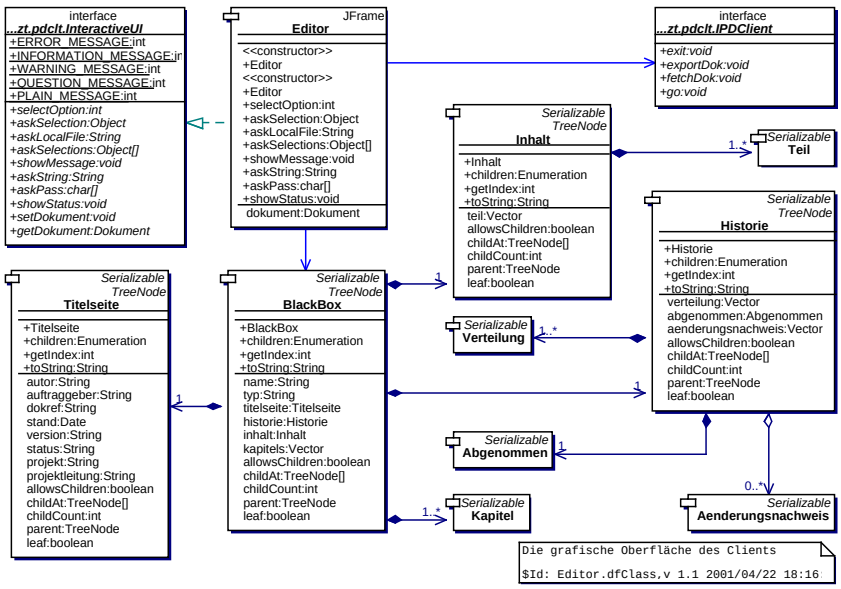

Abbildung 6.10: Die Klassen der grafischen Oberfläche

Diagramm 6.10 zeigt die Klassen der grafischen Oberfläche. `Editor` erweitert dabei `javax.swing.JFrame`. Eine Instanz dieser Klasse erstellt und layoutet bei der Initialisierung die Oberfläche. Nach der Instanziierung befindet sich das Objekt im "Event-Dispatching Thread" von Swing, so dass von `PDClient` aus keine Operationen mehr direkt an der Oberfläche ausgeführt werden dürfen (siehe auch Walrath und Campione 1999). Dieser läuft nämlich weiterhin im `main()`-Thread. Aktionen des Nutzers, die geeignet sind externe Schnittstellen zu gebrauchen (zum Beispiel "Load New..." zum Laden eines Dokuments vom Server), leiten diese Anforderung an den `main()`-Thread weiter und blockieren gegebenenfalls Eingaben in die Oberfläche.

Dieses Zwei-Thread-System unterstützte hervorragend die Entwicklung auf eine nur lose angekoppelte Oberfläche hin. In der Oberfläche wird jegliche Interaktion mit dem Nutzer durchgeführt. Ebenso wird nur in der Oberfläche der Inhalt des Dokuments bearbeitet. Sobald Funktionalitäten zum Datenaustausch (Netzwerk oder lokale Ein- / Ausgabe) gefordert sind, wird an den Clientkern (`PDClient`) delegiert.

Die Klasse `BlackBox` und die mit ihr zusammenhängenden Klassen implementieren (bisher nur zum Teil) das *javax.swing.tree.TreeModel*-Interface, um

das XML-Dokument Swing-konform in einem `javax.swing.JTree` verarbeiten zu können (Scheb 2000). Das Klassenmodell zur Repäsentation eines BlackBox-Dokuments ist noch nicht vollständig. Der größte Teil, unterhalb der Klasse `Kapitel`, ist noch nicht modelliert (vgl. auch XML-Schema C.8 bis C.10 auf den Seiten 93–95).

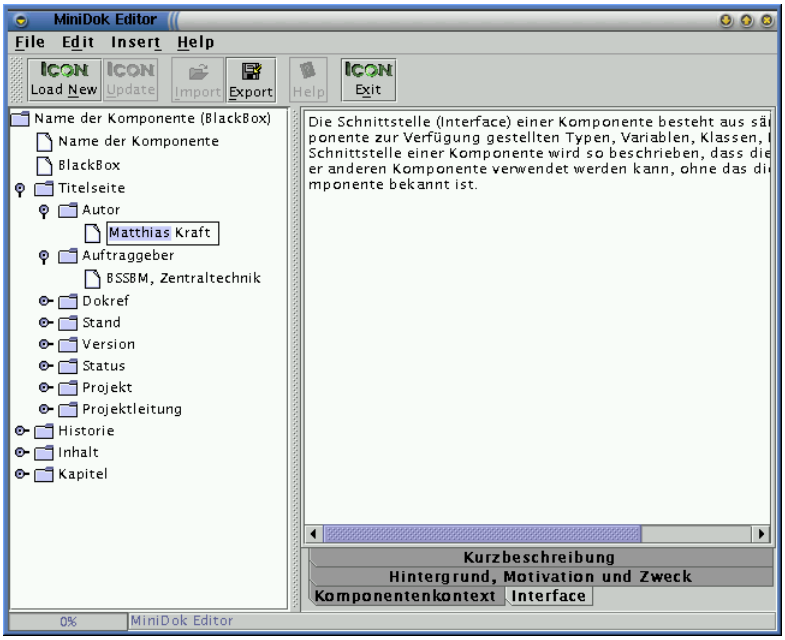

Abbildung 6.11: GUI mit Dokument

Die Implementierung geht jedoch soweit, dass nach dem Laden eines BlackBox-Dokuments das im Screenshot 6.11 dargestellte Abbild entsteht. Im linken Teil ist das geladene BlackBox-Dokument als Baumstruktur sichtbar. Die einzeiligen Informationen der Titelseite sind direkt im Baum editierbar (allerdings noch nicht persistent). Die drei anderen Teile des Dokuments sollen im Baum nur navigierende Funktion haben und im rechten Teil entsprechende Editoren darstellen. Zur Zeit wird im rechten Teil nur der Inhalt der vier BlackBox-Kapitel, untergliedert nach den Kapiteln, dargestellt.

Die Methoden des Interface *InteractiveUI*, die von den Client-Komponenten aufgerufen werden, wenn diese Eingaben vom Nutzer benötigen, erzeugen in der grafischen Oberfläche Standard-Dialoge aus `javax.swing.JOptionPane`. In Abbildung 6.12 auf der folgenden Seite ist ein Dialog vom Typ `showInputDialog()` dargestellt, der die vom Server gesendete Dokumentenliste zur Auswahl stellt.

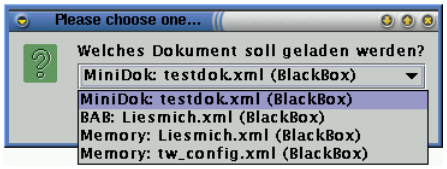

Abbildung 6.12: Dokumentenauswahl

6.2.3 Test

Auch beim Client wurden bisher nur Smoketests, d. h. Tests zur Überprüfung der Funktionalität, durchgeführt. Schwierigkeiten bereitete beim Client vor allem die Übernahme des generierten Sourcecodes aus JBuilder. Der Sourcecode musste erst für die Weiterentwickelung restrukturiert und überarbeitet werden.

Eine weitere Schwierigkeit stellte die Nebenläufigkeit von Client und grafischer Oberfläche dar. Es waren mehrere Versuche nötig, die richtige Synchronisation zwischen den beiden Threads zu finden.

Die Implementierung der restlichen Komponenten verlief ohne Probleme. Sie wiesen im Test auch nur kleinere Fehler auf, die schnell beseitigt werden konnten.

7 Zusammenfassung und Ausblick

Eine kurze Auswertung des Projektverlaufs und ein Ausblick auf noch zu integrierende Funktionalitäten, die während der Entwicklung nachgefragt wurden, beschließen diese Diplomarbeit.

7.1 Auswertung

Der folgende Abschnitt stellt den geplanten Projektverlauf der tatsächlichen Durchführung gegenüber. Anschließend wird noch einmal zusammenfassend das Ergebnis dieser Arbeit betrachtet.

7.1.1 Einhaltung des Projektplans

Der erste Teil des Projektplans – die theoretische Ausarbeitung – wurde termingerecht erfüllt. Die darauf folgende Konstruktionsphase 1 – die Erstellung des Prototypen – verlief bis zu den Tests am Server auch ohne Verzögerung. Wie jedoch in "5.2.3 Test" auf der Seite 53 dokumentiert, mussten die Tests und die Fehlerbereinigung ausgedehnt werden. Dadurch verzögerten sich die nachfolgenden Abschnitte um mehr als 2 Wochen. Da diese Verzögerung im Rahmen der Projektplanung nicht mehr aufzufangen war, wurde beschlossen, die erste Iteration um Elemente aus der zweiten zu ergänzen.

Der Client wurde um Teile der grafischen Oberfläche und die XML-(De-)Serialisierung erweitert. Da die Tests und die Fehlerbereinigung am Client relativ problemlos verlief, wurde danach mit etwa einmonatiger Verspätung ein erweiterter Prototyp der Software fertiggestellt.

Im Anschluss wurde die Dokumentation der ersten Iteration des Servers und des Clients in die Diplomarbeit eingefügt. Die Fertigstellung der Diplomarbeit erfolgte dann wieder termingerecht.

7.1.2 Stand der entwickelten Software

Das Resultat dieser Diplomarbeit ist eine Client- / Serverumgebung im Prototypstadium. Das grundlegende Design ist festgelegt. Eine Implementierung, die die gedachten Strukturen und die gewünschte Funktionalität aufzeigt, ist vorhanden. Gebrauchsfähig ist der Client als Downloadtool, d. h. in die Versionsverwaltung eingecheckte Dokumente, die in der Datenbank registriert sind, können vom Client bereits angefordert und vom Server ausgeliefert werden. Der Client ist dann in der Lage, diese Dokumente lokal zu speichern.

Alle darüber hinausgehenden Implementierungen können noch nicht produktiv genutzt werden. Sie sind als richtungsweisend für die weitere Entwicklung anzusehen. Dazu gehören hauptsächlich die grafische Oberfläche des Clients und die Persistenzschicht zur Datenbank am Server.

7.2 Weiterentwicklung

Vor der weiteren Entwicklung der Software steht zu allererst ein Review des Designs und der Implementierung. Anschließend kann anhand der dabei gewonnenen Erkenntnisse ein Refactoring durchgeführt und die zweite Iteration begonnen werden.

In der zweiten Iteration sollte vor allem die Vervollständigung der Editorfunktionalität im Client Priorität haben. Für den Editor muss das Klassenmodell der BlackBox-Dokumente vervollständigt werden. Außerdem wird ein Editor-Kit für die Bearbeitung der Kapitel benötigt. Ferner muss der XML-Import, inklusive der Validierung, und der Upload zum Server implementiert werden. Am Server muss dann schrittweise die Vervollständigung der CVS-Kommandos erfolgen – soweit dies sinnvoll ist. Auf jeden Fall werden noch die Kommandos “commit”, “status”, “tag” und “update” Einzug in die Komponente halten.

Da die zweite Iteration ein schon einsatzfähiges Produkt ergeben soll, sollten vor Beginn der Tests die Lizenzbedingungen festgelegt werden. Dabei ist auch darauf zu achten, dass diese nicht die Nutzung von GPL-basierten Komponenten unmöglich machen. Zur Zeit sind keine GPL-basierten Komponenten in die Software eingebunden. Daher kann an dieser Stelle die “Virusfunktion” der GPL noch nicht greifen. Die Applikation nutzt aber Softwareprodukte, die die GPL als Lizenzmodell haben: CVS und MySQL. Die Castor-Library hingegen nutzt eine BSD-ähnliche Lizenz. Daher kann sie ohne Probleme in die Software eingebunden werden.

Für die späteren Entwicklungsstufen sind die folgenden nach Wichtigkeit geordneten Punkte zu berücksichtigen:

- Kompression auf dem Übertragungsweg mittels des `java.util.zip`-packages, um die Down- und Uploadzeiten zu verkürzen,

- nach Tests, weitere Performanzverbesserungen,
- Einführung einer PKI,
- Integration der Verschlüsselung für die lokal gespeicherten Daten am Client,
- Entwicklung einer Präsentationsumgebung auf XSL-Basis für die Dokumente und
- Entwicklung einer Oberfläche für die Integration des Clients in TogetherJ.

A BlackBox-Dokumentationsrichtlinie

Die folgenden vier Printouts repräsentieren die aktuelle, seit dem 01.11.2000 für alle internen Entwicklungsprojekte gültige, BlackBox-Dokumentationsrichtlinie.

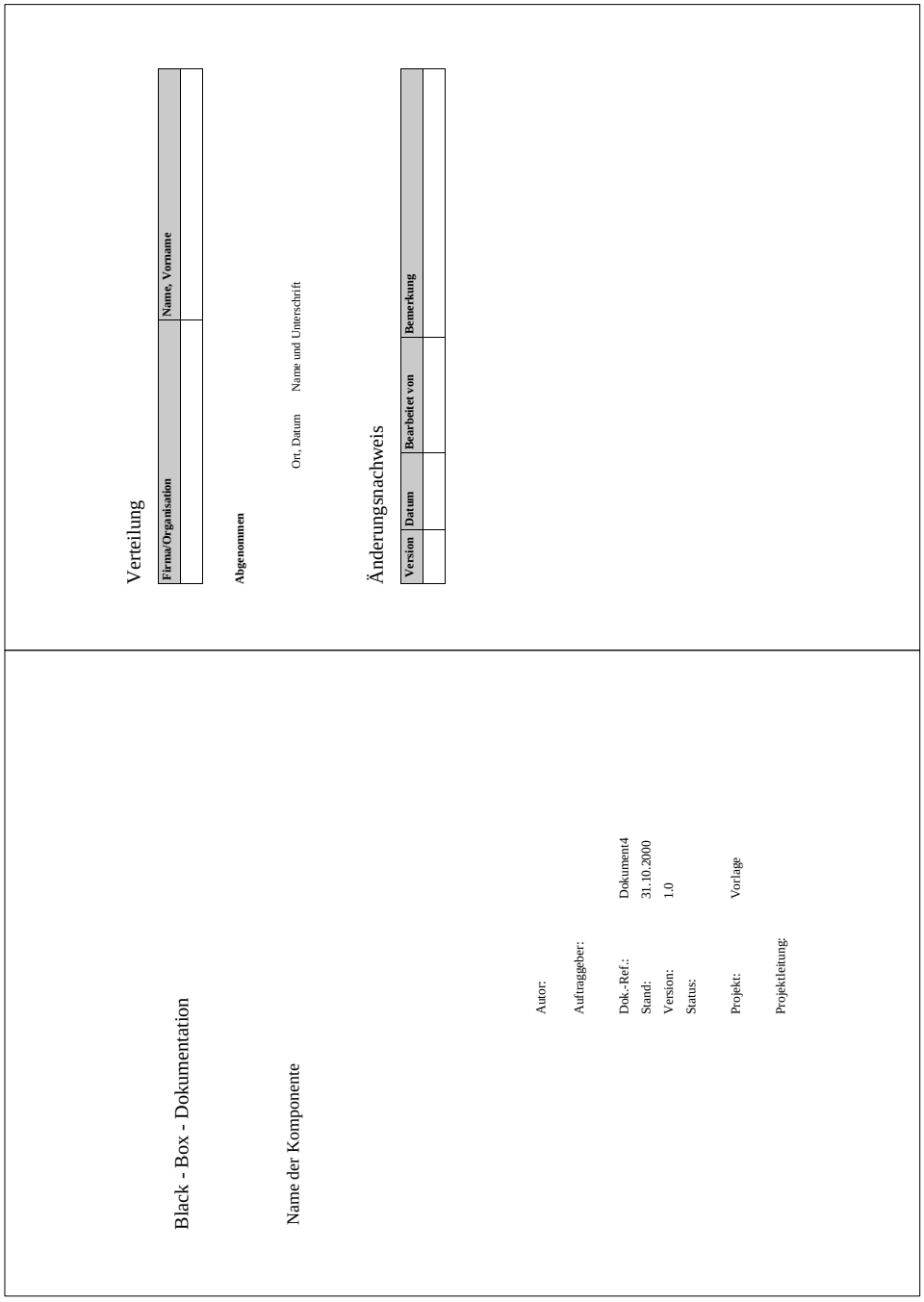

Abbildung A.1: BlackBox-Dokumentationsrichtlinie (1/2)

Inhalt

1 Kurzbeschreibung

In der Kurzbeschreibung ist die Funktionalität der Komponente in zwei bis drei Sätzen zu beschreiben.

Abbildung A.2: BlackBox-Dokumentationsrichtlinie (3/4)

2 Hintergrund, Motivation und Zweck

Im Gegensatz zur Kurzbeschreibung, die nur kurz die Funktionalität beschreibt, soll an dieser Stelle Informationen über den Hintergrund, die Motivation und den Zweck der Komponente enthalten sein.

Weiterhin können in diesem Abschnitt Begriffserklärungen gegeben werden, die die Einordnung der Komponente erleichtern.

Im Abschnitt Hintergrund wird beschrieben, wo die Komponente in einer Architektur steht.

Die Motivation beschreibt warum es notwendig ist die entsprechende Funktionalität in eine eigenständige Komponente abzulegen.

Der Zweck beschreibt die Funktionalität und was damit erreicht werden soll. Hier sollten keine Informationen über die Verwendung stehen, da dafür ein separater Abschnitt vorgesehen ist.

3 Komponentenkontext

3.1 Verwendung der Komponente

Dieser Abschnitt dient der allgemeinen Beschreibung über die Verwendung der Komponente. Es ist erklärt, wie die Funktionalität der Komponente genutzt werden kann. An Hand von kurzen Beispielsequenzen ist die Verwendung der Komponente aufzuzeigen. Wie die Komponenten – Interfacemethoden aufgerufen werden und unter welche Bedingungen das erfolgen kann.

3.2 Vorbedingungen

In den Vorbedingungen sollen alle Informationen stehen, die beschreiben welche anderen Komponenten und Systembestandteile notwendig sind, damit ein Funktionieren der Komponente sichergestellt ist. Dabei sind die Informationen in folgende Abschnitte zu unterteilen:

- Systemumgebung
- Direkte Abhängigkeit zu anderen Komponenten
- Indirekte Abhängigkeiten zu anderen Komponenten

3.3 Integration der Komponente

Soll die Komponente in anderen Komponenten benutzt werden, so ist darzustellen wie die Komponente eingebunden werden kann.

Abbildung A.3: BlackBox-Dokumentationsrichtlinie (5/6)

4 Interface

Die Schnittstelle (Interface) einer Komponente besteht aus sämtlichen von der Komponente zur Verfügung gestellten Typen, Variablen, Klassen, Funktionen etc.. Die Schnittstelle einer Komponente wird so beschrieben, dass die Komponente von einer anderen Komponente verwendet werden kann, ohne das die Realisierung der Komponente bekannt ist.

4.1 Interfacebeschreibung

In dieser Beschreibung werden alle Methoden der Komponente beschrieben. Die Methoden sind auf drei verschiedene Arten zu klassifizieren. Sie unterteilen sich in Feed - Methoden, Produce – Methoden und in Deliver – Methoden.

Die Feed – Methoden versetzen eine Komponente in einen definierten Anfangszustand.

Die Produce – Methoden werden durch den implementierten Algorithmus Informationen verarbeiten.

Die Deliver – Methoden werden verarbeitete Informationen liefern. Die Deliver – Methoden können auch eine Komponente nach der Verarbeitung in einen definierten Zustand versetzen.

Prinzipiell wird jede Methode durch Name, Parameter, Rückgabewert, Exception, Verhalten beschrieben. Dieser Abschnitt dient der allgemeinen Beschreibung über die Verwendung der Komponente. Es ist erklärt, wie die Funktionalität der Komponente genutzt werden kann. An Hand von kurzen Beispielsequenzen ist die Verwendung der Komponente aufzuzeigen.

4.2 Datenstrukturen

Erfolgt aus der Komponente ein Informationsaustausch mit anderen Komponenten so ist dieser zu beschreiben.

Dieses betrifft zum Beispiel die Verwendung von Datenbanken in einer Komponente. Die entsprechenden Tabellen sind in einem Ausschnitt aus einem ER - Diagramm zu beschreiben.

Werden in der Komponente Informationen über Dateien ausgetauscht, so sind die entsprechenden Dateiformate zu beschreiben.

4.3 Interfaceprotokoll

In diesem Abschnitt wird das Protokoll beschrieben wie die Interface – Methoden der Komponente genutzt werden sollen. Das bedeutet in welcher Abfolge diese Methoden aufgerufen werden sollen um ein bestimmtes Ergebnis zu erreichen. Dieses kann durch ein Zustands – Diagramm beschrieben werden.

Es sollte auch darauf geachtet werden das auch Zustände beschrieben werden wenn das Interface – Protokoll nicht eingehalten wird.

Insbesondere sind natürlich die Fehler – Zustände zu beschreiben.

Abbildung A.4: BlackBox-Dokumentationsrichtlinie (7/8)

B Ausgewählte Paragraphen des BDSG

Die hier aufgelisteten Paragraphen des BDSG werden in "2.3 Anforderungen an die Dokumentation" auf der Seite 6 genannt. Sie sind in die Arbeit aufgenommen worden, damit später der Inhalt dieser Paragraphen nachvollzogen werden kann. Zur Zeit werden die Datenschutzgesetze und -richtlinien überarbeitet, um sie an eine EU-Richtlinie zum Datenschutz anzupassen (Gerling 1999).

Aus dem Bundesdatenschutzgesetz (BDSG) vom 20. Dezember 1990 (BGBl. I, S.2954), zuletzt geändert durch Art.2, Abs.5 des Begleitgesetzes zum Telekommunikationsgesetz (BegleitG) vom 17.Dezember 1997 (BGBl. I S. 3108).

Erster Abschnitt: Allgemeine Bestimmungen

§ 9 [Technische und organisatorische Maßnahmen]

Öffentliche und nicht-öffentliche Stellen, die selbst oder im Auftrag personenbezogene Daten verarbeiten, haben die technischen und organisatorischen Maßnahmen zu treffen, die erforderlich sind, um die Ausführung der Vorschriften dieses Gesetzes, insbesondere die in der Anlage zu diesem Gesetz genannten Anforderungen, zu gewährleisten. Erforderlich sind Maßnahmen nur, wenn ihr Aufwand in einem angemessenen Verhältnis zu dem angestrebten Schutzzweck steht.

Anlage (zu § 9 Satz 1)

Werden personenbezogene Daten automatisiert verarbeitet, sind Maßnahmen zu treffen, die je nach der Art der zu schützenden personenbezogenen Daten geeignet sind,

1. Unbefugten den Zugang zu Datenverarbeitungsanlagen, mit denen personenbezogene Daten verarbeitet werden, zu verwehren (Zugangskontrolle),

2. zu verhindern, daß Datenträger unbefugt gelesen, kopiert, verändert oder entfernt werden können (Datenträgerkontrolle),
3. die unbefugte Eingabe in den Speicher sowie die unbefugte Kenntnisnahme, Veränderung oder Löschung gespeicherter personenbezogener Daten zu verhindern (Speicherkontrolle),
4. zu verhindern, daß Datenverarbeitungssysteme mit Hilfe von Einrichtungen zur Datenübertragung von Unbefugten genutzt werden können (Benutzerkontrolle),
5. zu gewährleisten, daß die zur Benutzung eines Datenverarbeitungssystems Berechtigten ausschließlich auf die ihrer Zugriffsberechtigung unterliegenden Daten zugreifen können (Zugriffskontrolle),
6. zu gewährleisten, daß überprüft und festgestellt werden kann, an welche Stellen personenbezogene Daten durch Einrichtungen zur Datenübertragung übermittelt werden können (Übermittlungskontrolle),
7. zu gewährleisten, daß nachträglich überprüft und festgestellt werden kann, welche personenbezogenen Daten zu welcher Zeit von wem in Datenverarbeitungssysteme eingegeben worden sind (Eingabekontrolle),
8. zu gewährleisten, daß personenbezogene Daten, die im Auftrag verarbeitet werden, nur entsprechend den Weisungen des Auftraggebers verarbeitet werden können (Auftragskontrolle),
9. zu verhindern, daß bei der Übertragung personenbezogener Daten sowie beim Transport von Datenträgern die Daten unbefugt gelesen, kopiert, verändert oder gelöscht werden können (Transportkontrolle),
10. die innerbehördliche oder innerbetriebliche Organisation so zu gestalten, daß sie den besonderen Anforderungen des Datenschutzes gerecht wird (Organisationskontrolle).

Dritter Abschnitt: Datenverarbeitung nicht-öffentlicher Stellen und öffentlich-rechtlicher Wettbewerbsunternehmen; Erster Unterabschnitt: Rechtsgrundlagen der Datenverarbeitung

§ 27 [Anwendungsbereich]

(1) Die Vorschriften dieses Abschnittes finden Anwendung, soweit personenbezogene Daten in oder aus Dateien geschäftsmäßig oder für berufliche oder gewerbliche Zwecke verarbeitet oder genutzt werden durch

1. nicht-öffentliche Stellen,
2. (a) öffentliche Stellen des Bundes, soweit sie als öffentlich-rechtliche Unternehmen am Wettbewerb teilnehmen,
   (b) öffentliche Stellen der Länder, soweit sie als öffentlich-rechtliche Unternehmen am Wettbewerb teilnehmen, Bundesrecht ausführen und der Datenschutz nicht durch Landesgesetz geregelt ist.

In den Fällen der Nummer 2 Buchstabe a gelten anstelle des § 38 die §§ 18, 21 und 24, 25 und 26.

(2) Die Vorschriften dieses Abschnittes gelten nicht für die Verarbeitung und Nutzung personenbezogener Daten in Akten, soweit es sich nicht um personenbezogene Daten handelt, die offensichtlich aus einer Datei entnommen worden sind.

§ 28 [Datenspeicherung, -übermittlung und -nutzung für eigene Zwecke]

(1) Das Speichern, Verändern oder Übermitteln personenbezogener Daten oder ihre Nutzung als Mittel für die Erfüllung eigener Geschäftszwecke ist zulässig

1. im Rahmen der Zweckbestimmung eines Vertragsverhältnisses oder vertragsähnlichen Vertrauensverhältnisses mit dem Betroffenen,
2. soweit es zur Wahrung berechtigter Interessen der speichernden Stelle erforderlich ist und kein Grund zu der Annahme besteht, daß das schutzwürdige Interesse des Betroffenen an dem Ausschluß der Verarbeitung oder Nutzung überwiegt,
3. wenn die Daten aus allgemein zugänglichen Quellen entnommen werden können oder die speichernde Stelle sie veröffentlichen dürfte, es sei denn, daß das schutzwürdige Interesse des Betroffenen an dem Ausschluß der Verarbeitung oder Nutzung offensichtlich überwiegt,
4. wenn es im Interesse der speichernden Stelle zur Durchführung wissenschaftlicher Forschung erforderlich ist, das wissenschaftliche Interesse an der Durchführung des Forschungsvorhabens das Interesse des Betroffenen an dem Ausschluß der Zweckänderung erheblich überwiegt und der Zweck der Forschung auf andere Weise nicht oder nur mit unverhältnismäßigem Aufwand erreicht werden kann.

Die Daten müssen nach Treu und Glauben und auf rechtmäßige Weise erhoben werden.

(2) Die Übermittlung oder Nutzung ist auch zulässig

1. (a) soweit es zur Wahrung berechtigter Interessen eines Dritten oder öffentlicher Interessen erforderlich ist oder
   (b) wenn es sich um listenmäßig oder sonst zusammengefaßte Daten über Angehörige einer Personengruppe handelt, die sich auf
   - eine Angabe über die Zugehörigkeit des Betroffenen zu dieser Personengruppe,
   - Berufs-, Branchen- oder Geschäftsbezeichnung,
   - Namen,
   - Titel,
   - akademische Grade,
   - Anschrift,
   - Geburtsjahr

   beschränken und kein Grund zu der Annahme besteht, daß der Betroffene ein schutzwürdiges Interesse an dem Ausschluß der Übermittlung hat. In den Fällen des Buchstabens b kann im allgemeinen davon ausgegangen werden, daß dieses Interesse besteht, wenn im Rahmen der Zweckbestimmung eines Vertragsverhältnisses oder vertragsähnlichen Vertrauensverhältnisses gespeicherte Daten übermittelt werden sollen, die sich
   - auf gesundheitliche Verhältnisse,
   - auf strafbare Handlungen,
   - auf Ordnungswidrigkeiten,
   - auf religiöse oder politische Anschauungen sowie
   - bei Übermittlung durch den Arbeitgeber auf arbeitsrechtliche Rechtsverhältnisse

   beziehen, oder
2. wenn es im Interesse einer Forschungseinrichtung zur Durchführung wissenschaftlicher Forschung erforderlich ist, das wissenschaftliche Interesse an der Durchführung des Forschungsvorhabens das Interesse des Betroffenen an dem Ausschluß der Zweckänderung erheblich überwiegt und der Zweck der Forschung auf andere Weise nicht oder nur mit unverhältnismäßigem Aufwand erreicht werden kann.

(3) Widerspricht der Betroffene bei der speichernden Stelle der Nutzung oder Übermittlung seiner Daten für Zwecke der Werbung oder der Markt- oder Meinungsforschung, ist eine Nutzung oder Übermittlung für diese Zwecke unzulässig. Widerspricht der Betroffene

beim Empfänger der nach Absatz 2 übermittelten Daten der Verarbeitung oder Nutzung für Zwecke der Werbung oder der Markt- oder Meinungsforschung, hat dieser die Daten für diese Zwecke zu sperren.

(4) Der Empfänger darf die übermittelten Daten für den Zweck verarbeiten oder nutzen, zu dessen Erfüllung sie ihm übermittelt werden. Eine Verarbeitung oder Nutzung für andere Zwecke ist nur unter den Voraussetzungen der Absätze 1 und 2 zulässig. Die übermittelnde Stelle hat den Empfänger darauf hinzuweisen.

§ 29 [Geschäftsmäßige Datenspeicherung zum Zwecke der Übermittlung]

(1) Das geschäftsmäßige Speichern oder Verändern personenbezogener Daten zum Zwecke der Übermittlung ist zulässig, wenn

1. kein Grund zu der Annahme besteht, daß der Betroffene ein schutzwürdiges Interesse an dem Ausschluß der Speicherung oder Veränderung hat, oder
2. die Daten aus allgemein zugänglichen Quellen entnommen werden können oder die speichernde Stelle sie veröffentlichen dürfte, es sei denn, daß das schutzwürdige Interesse des Betroffenen an dem Ausschluß der Speicherung oder Veränderung offensichtlich überwiegt.

§ 28 Abs.1 Satz 2 ist anzuwenden.

(2) Die Übermittlung ist zulässig, wenn

1. (a) der Empfänger ein berechtigtes Interesse an ihrer Kenntnis glaubhaft dargelegt hat oder
   (b) es sich um listenmäßig oder sonst zusammengefaßte Daten nach § 28 Abs.2 Nr.1 Buchstabe b handelt, die für Zwecke der Werbung oder der Markt- oder Meinungsforschung übermittelt werden sollen, und
2. kein Grund zu der Annahme besteht, daß der Betroffene ein schutzwürdiges Interesse an dem Ausschluß der Übermittlung hat.

§ 28 Abs.2 Nr.1 Satz 2 gilt entsprechend. Bei der Übermittlung nach Nummer 1 Buchstabe a sind die Gründe für das Vorliegen eines berechtigten Interesses und die Art und Weise ihrer glaubhaften Darlegung von der übermittelnden Stelle aufzuzeichnen. Bei der Übermittlung im automatisierten Abrufverfahren obliegt die Aufzeichnungspflicht dem Empfänger.

(3) Für die Verarbeitung oder Nutzung der übermittelten Daten gilt § 28 Abs.3 und 4.

C Diagramme

Auf den folgenden Seiten befinden sich u. a. Gesamtansichten der Komponenten- und Klassendiagramme von Server und Client.

- Abbildung C.1 auf der folgenden Seite zeigt ein Komponentendiagramm des Servers.
- Die Klassendiagramme C.2 bis C.4 auf den Seiten 87–89 zeigen die Gesamtübersicht der Server-Klassen.
- Abbildung C.5 auf der Seite 90 zeigt ein Komponentendiagramm des Clients.
- Die Klassendiagramme C.6 bis C.7 auf den Seiten 91–92 zeigen die Gesamtübersicht der Client-Klassen.
- Die Abbildungen C.8 bis C.10 auf den Seiten 93–95 zeigen eine grafische Repräsentation des XML-Schemas für BlackBox-Dokumente.

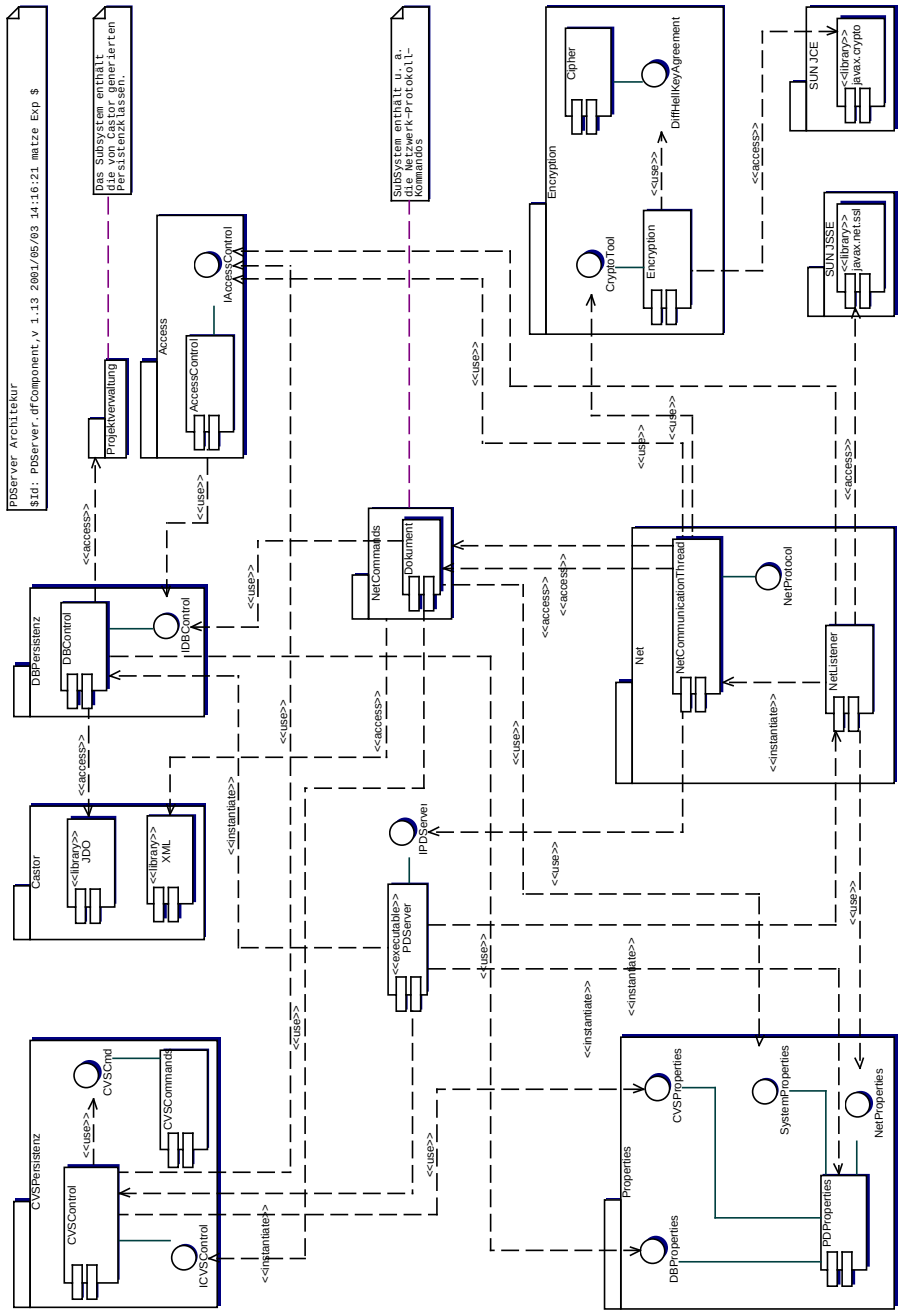

Abbildung C.1: Server Architekturübersicht

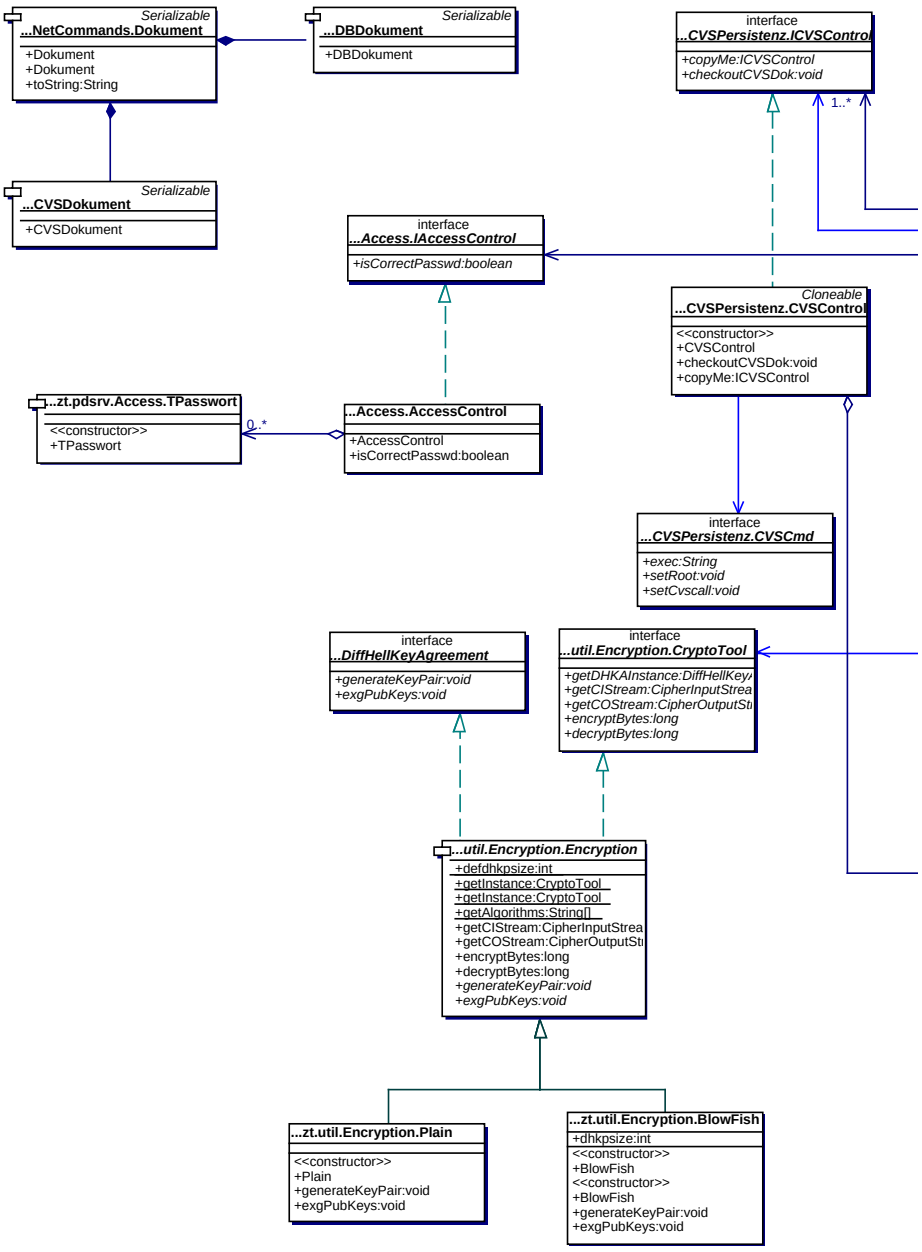

Abbildung C.2: Klassendiagramm des Servers (Teil 1)

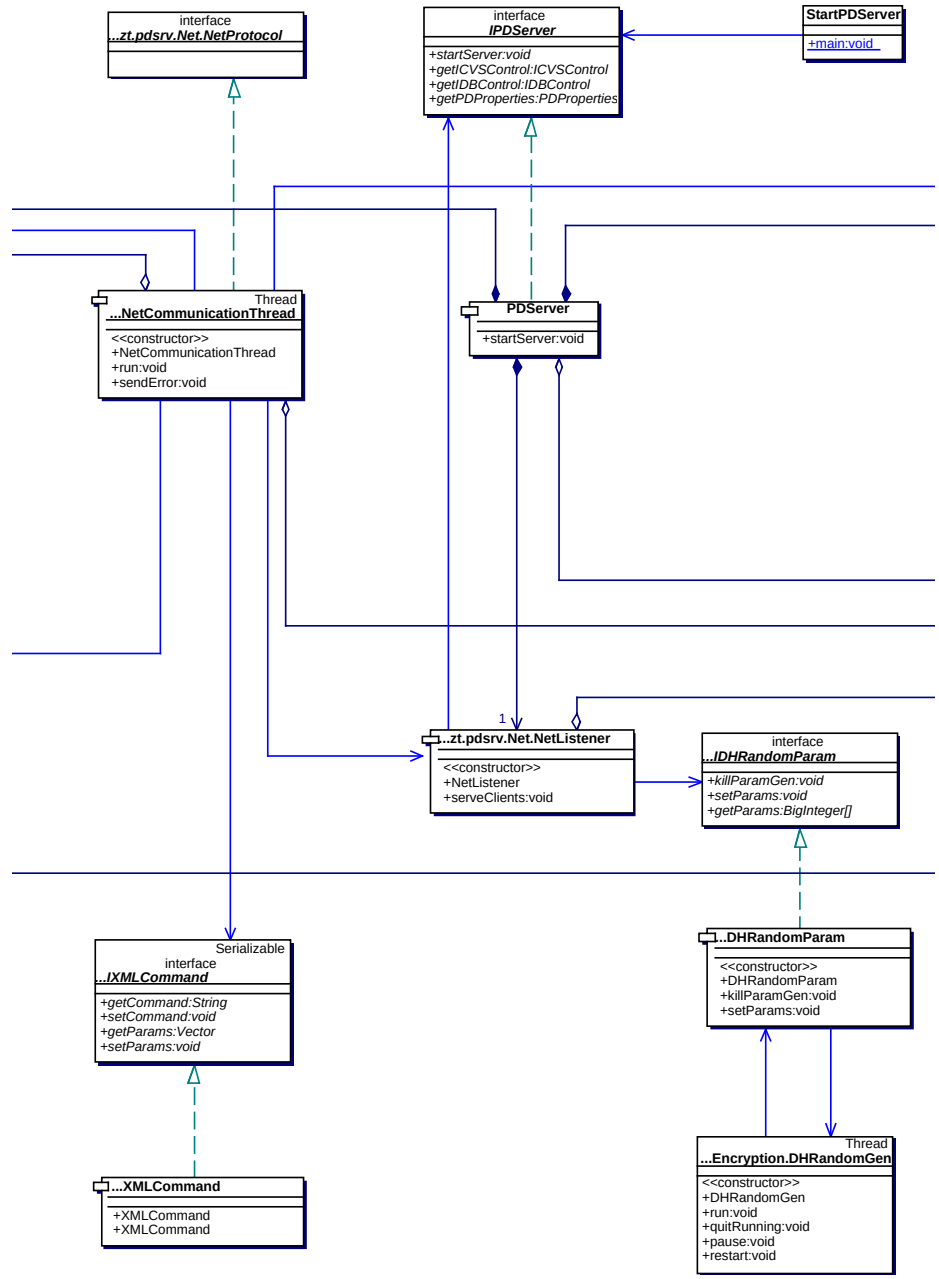

Abbildung C.3: Klassendiagramm des Servers (Teil 2)

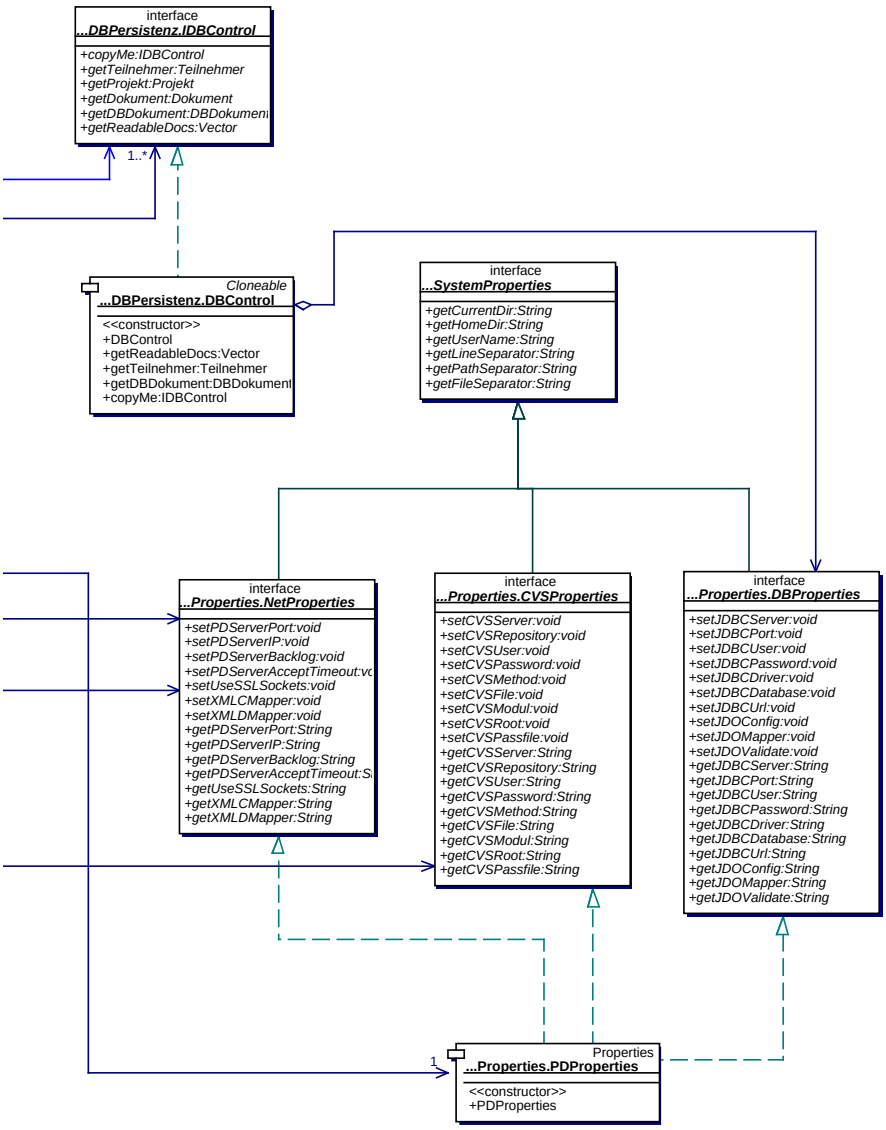

Abbildung C.4: Klassendiagramm des Servers (Teil 3)

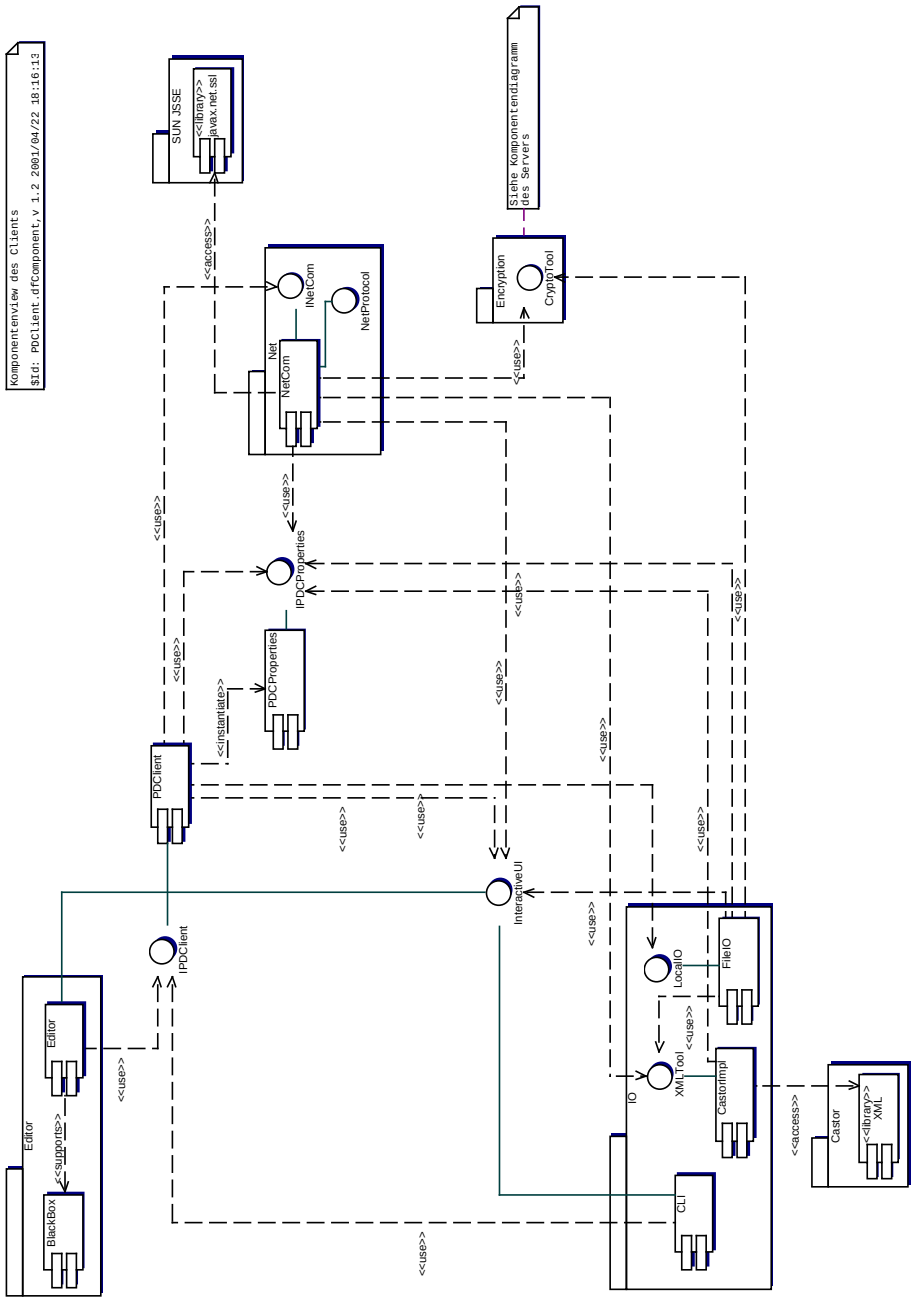

Abbildung C.5: Client Architekturübersicht

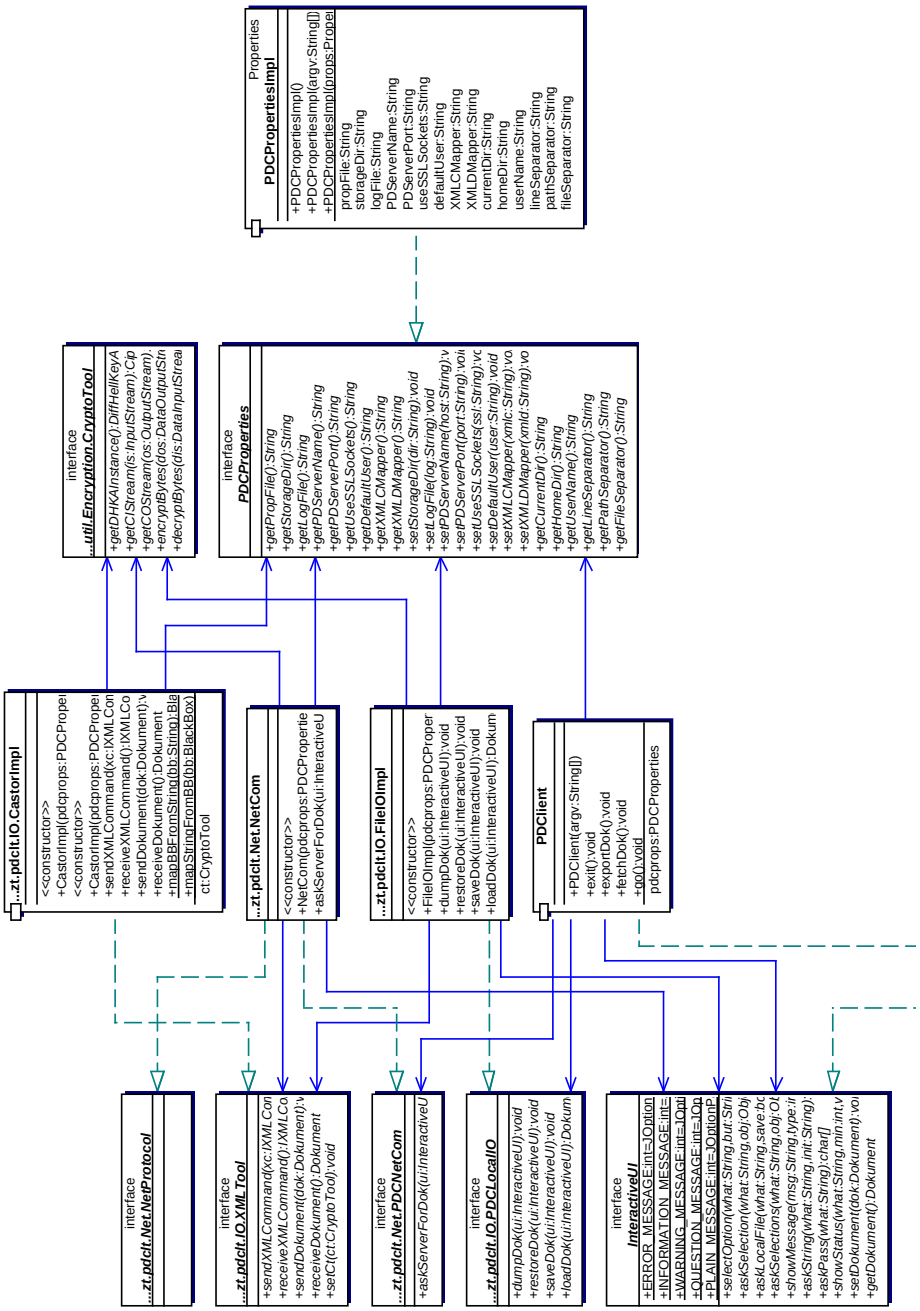

Abbildung C.6: Klassendiagramm des Client (Teil 1)

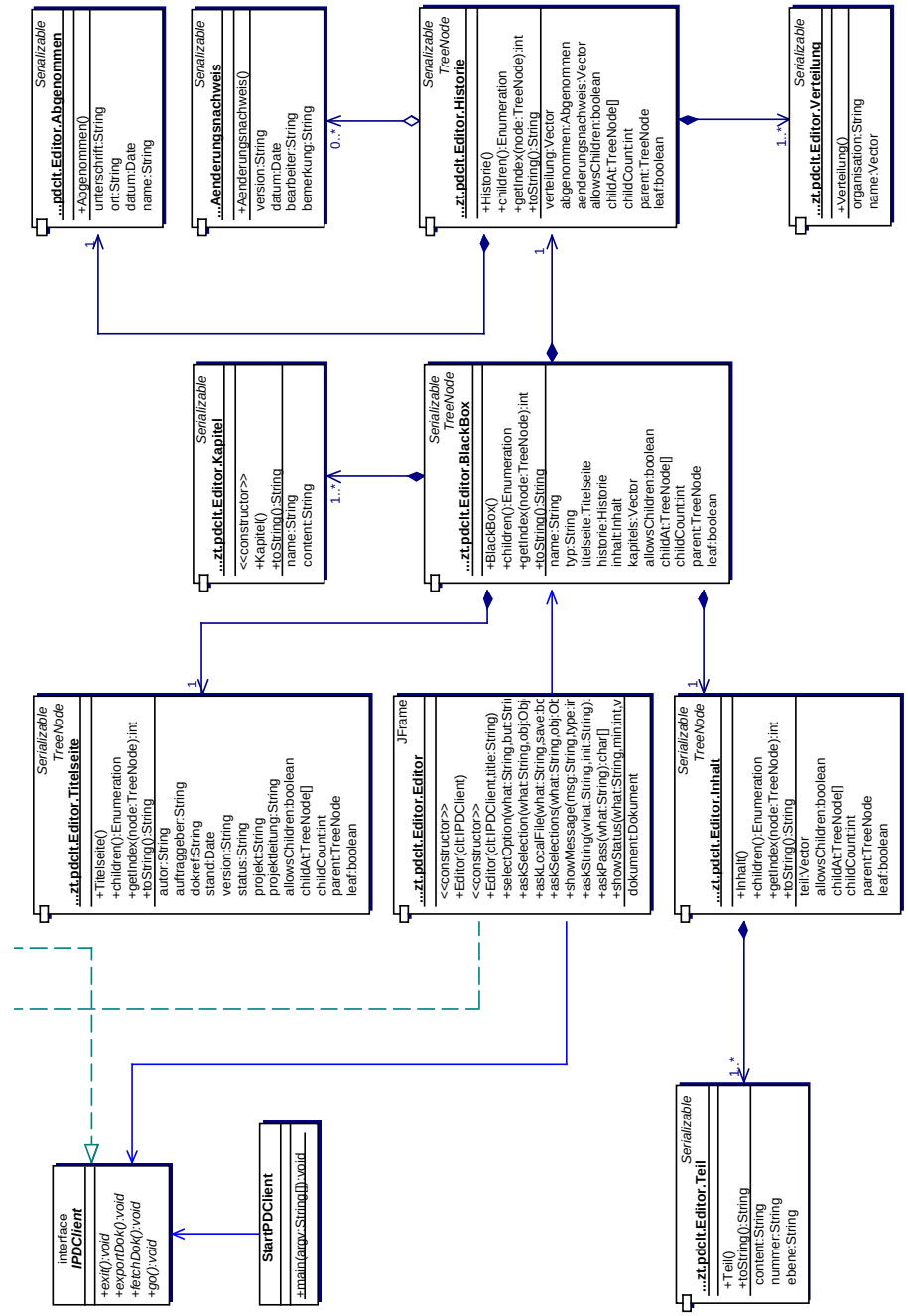

Abbildung C.7: Klassendiagramm des Client (Teil 2)

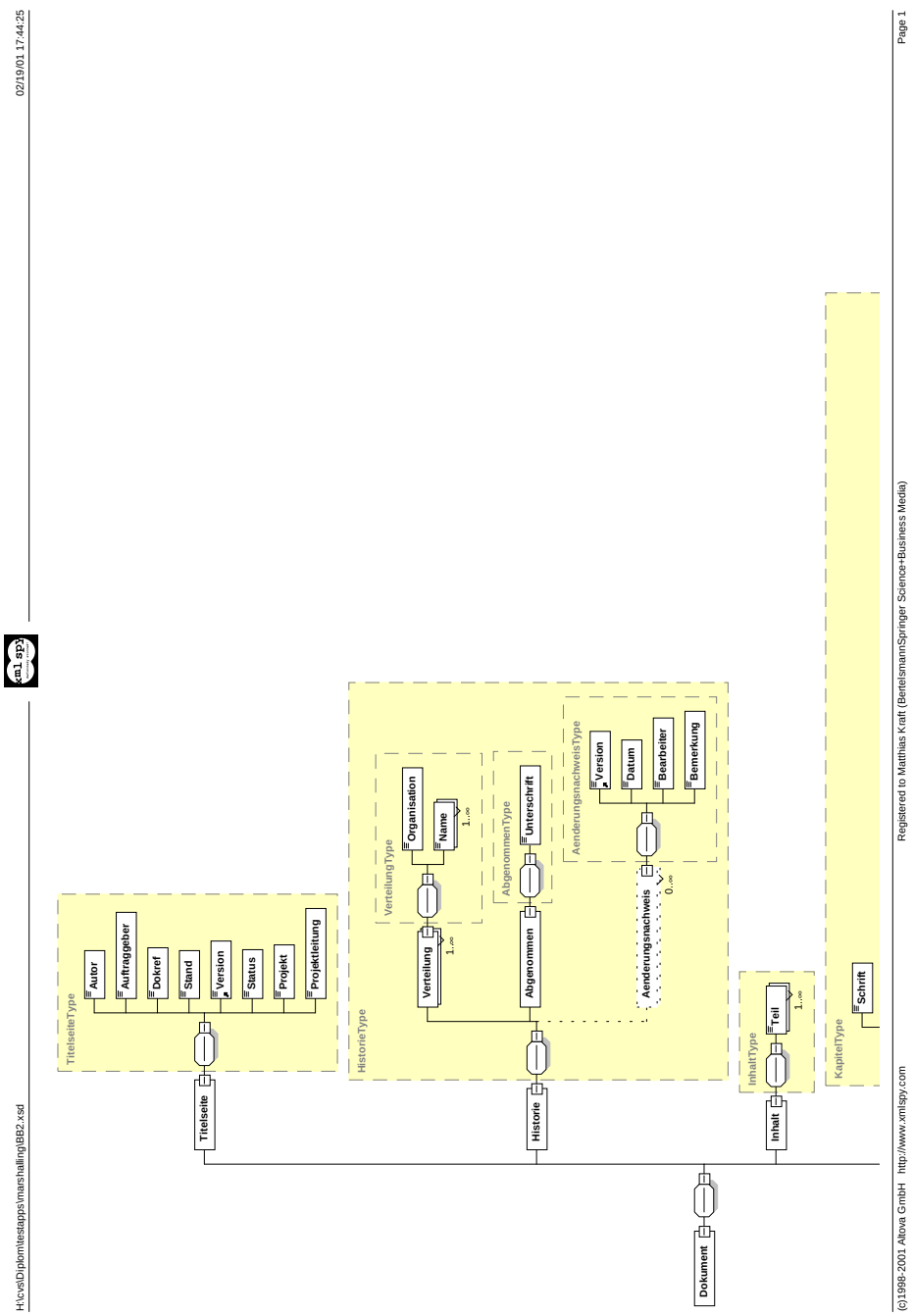

Abbildung C.8: Repräsentation des XML-Schemas zur Validierung von BlackBox-Dokumenten (Teil 1)

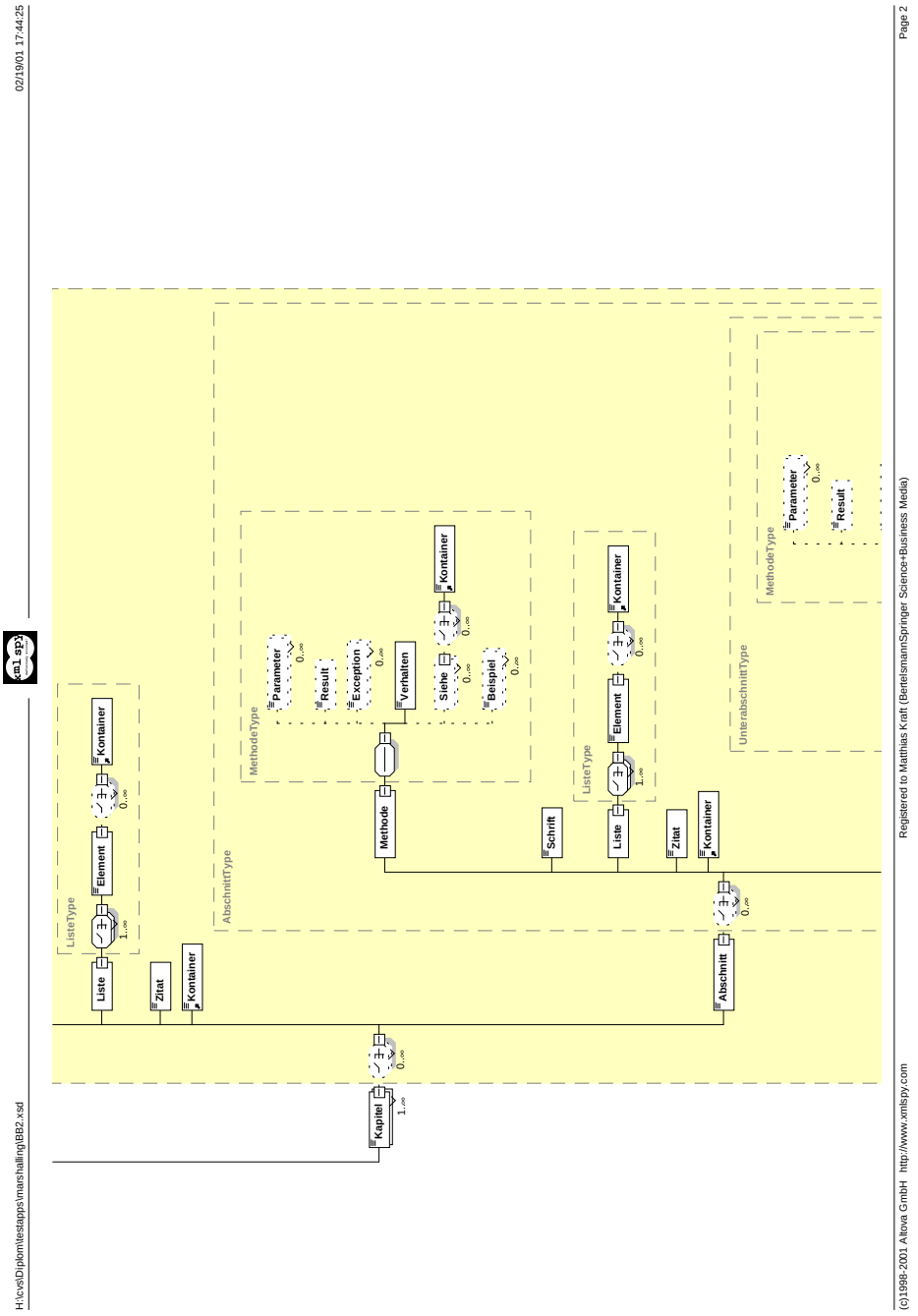

Abbildung C.9: BlackBox XML-Schema (Teil 2)

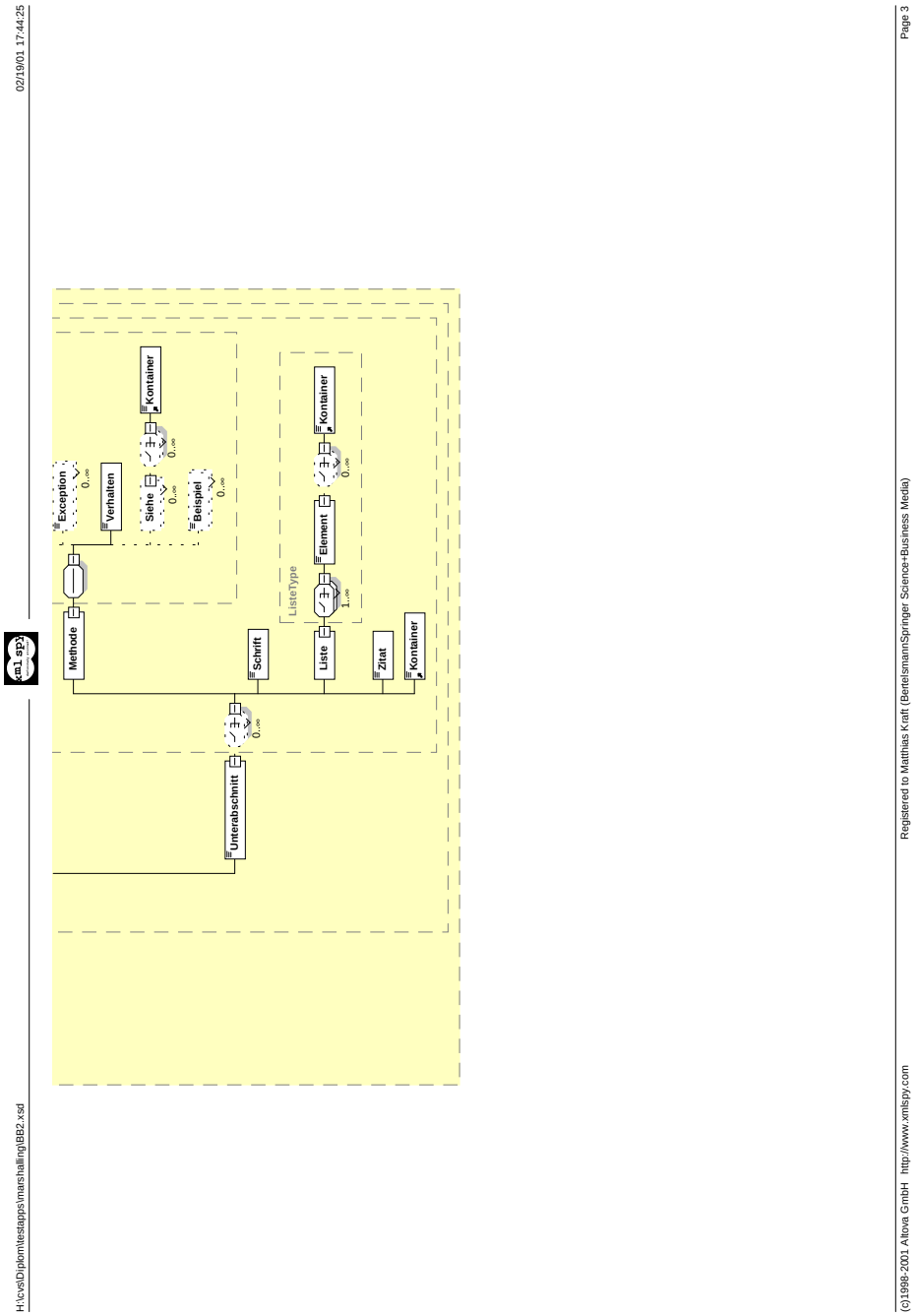

Abbildung C.10: BlackBox XML-Schema (Teil 3)

Abbildungsverzeichnis

Tabellenverzeichnis

Quellenverzeichnis

1 Ayer und Patrinostro 1992 Ayer, Steve ; Patrinostro, Frank: *Documenting the Software Development Process*. New York : The McGraw-Hill Companies, Inc., 1992 (McGraw-Hill systems design & implementation series). – Informationen die über die Quellenangabe in David (1996) hinausgehen sind von der Webseite der Library Of Congress http://catalog.loc.gov/. – ISBN 0070026041

2 Berg 1999 Berg, Clifford J.: *Advanced Java 2 development for enterprise applications*. 2nd edition. Prentice Hall PTR, 1999 (Sun Microsystems Press Java series). – ISBN 0–13–084875–1

3 Bray u. a. 2000 Bray, Tim ; Paoli, Jean ; Sperberg-McQueen, C. M. ; Maler, Eve: *Extensible Markup Language (XML) 1.0*. Second Edition. W3C: W3C XML Working Group (Veranst.), Oktober 2000. – URL http://www.w3.org/TR/2000/REC-xml-20001006. – Zugriffsdatum: 2001-01-16. – Copyright © 2000 W3C®(MIT, INRIA, Keio), All Rights Reserved. W3C liability, trademark, document use, and software licensing rules apply.

4 Couch 1999 Couch, Justin: *Java 2 networking*. The McGraw-Hill Companies, Inc., 1999 (McGraw-Hill Java masters). – ISBN 0–07–134813–1

5 David 1996 David, Eberhard: *Zur Relevanz der Dokumentation und Kommunikation in der betrieblichen Softwareentwicklung und -wartung*, Universität - Gesamthochschule - Paderborn, Dissertation, 1996

6 Eckstein 1999 Eckstein, Robert ; Mui, Linda (Hrsg.): *XML Pocket Reference*. 101 Morris Street, Sebastopol, CA 95472 : O'Reilly & Associates, Inc., 1999. – Siehe auch Bray u. a. (2000). – ISBN 1–56592–709–5

7 Fabricius 2000 Fabricius, Stefan: *Software Dokumentation*. 2000. – Internes Papier bei BertelsmannSpringer zur Kommentierung von Sourcecode.

8 Flanagan 1999 Flanagan, David: *Java in a Nutshell*. 3rd Edition. 101 Morris Street, Sebastopol, CA 95472 : O'Reilly & Associates, Inc., November 1999. – URL http://www.oreilly.com/catalog/javanut3/chapter/ch04.html. – Zugriffsdatum: 2001-01-19. – Kapitel 4 "The Java Platform" ist auch online verfügbar. – ISBN 1–56592–487–8

9 Fogel 1999 FOGEL, Karl F.: *Open Source Development with CVS*. Scottsdale, AZ : The Coriolis Group, LLC, 1999 (Coriolis OpenPress series). – URL http://cvsbook.red-bean.com/. – Zugriffsdatum: 2001-01-10. – ISBN 1576104907

10 Fowler und Scott 2000 FOWLER, Martin ; SCOTT, Kendall: *UML konzentriert*. 2. Auflage. Addison-Wesley Verlag, 2000. – ISBN 3–8273–1617–0

11 Gamma u. a. 1996 GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Entwurfsmuster: Elemente wiederverwendbarerer objektorientierter Software*. Addison Wesley Verlag, 1996 (Professionelle Softwareentwicklung). – ISBN 3–89319–950–0

12 Gerling 1999 GERLING, Dr. Rainer W.: Nachholbedarf – EG-Richtlinien zum Datenschutz. In: *c't magazin für computer technik* (1999), März, Nr. 6, S. 316 ff.. – ISSN 0724–8679

13 Großwendt 2000 GROSSWENDT, Volkmar: Ausgezeichnet – XML - eine Einführung. In: *XML Magazin* (2000), Januar, Nr. 1, S. 79–84. – Das XML Magazin ist eine ständige Beilage des Java Magazin's.

14 Harold 1999 HAROLD, Elliotte R.: *Java I/O*. 1st Edition. 101 Morris Street, Sebastopol, CA 95472 : O'Reilly & Associates, Inc., March 1999. – URL http://www.oreilly.com/catalog/javaio/chapter/ch05.html. – Zugriffsdatum: 2001-01-19. – Kapitel 5 "Network Streams" ist auch online verfügbar. – ISBN 1–56592–485–1

15 Heitig 1984 HEITIG, Waldemar: Angemessenheit bei der Dokumentation von DV-Anwendungen. In: HEILMANN, Wolfgang (Hrsg.) ; REUSCH, Günter (Hrsg.): *Datensicherheit und Datenschutz*. Wiesbaden : Forkel, 1984, S. 147–168. – ISBN 3–7719–6287–0

16 Holubek 1999 HOLUBEK, Andreas: EXtravagant – XML und XSL in der Praxis anwenden. In: *Java Magazin* (1999), März, Nr. 3, S. 12–14

17 Knudsen 1998 KNUDSEN, Jonathan ; LOUKIDES, Mike (Hrsg.): *Java(TM) Cryptography*. First Edition. 101 Morris Street, Sebastopol, CA 95472 : O'Reilly & Associates, Inc., 1998. – ISBN 1–56592–402–9

18 Macos 2000a MACOS, Dr. D.: *Black-Box-Dokumentation*. November 2000. – Internes Papier bei BertelsmannSpringer zur Dokumentation von Schnittstellen.

19 Macos 2000b MACOS, Dr. D.: *White-Box-Dokumentation*. Oktober 2000. – Internes Papier bei BertelsmannSpringer zur Dokumentation von Komponenten.

20 McLaughlin 2000 McLaughlin, Brett ; Loukides, Mike (Hrsg.): *Java(TM) and XML*. First Edition. 101 Morris Street, Sebastopol, CA 95472 : O'Reilly & Associates, Inc., 2000. – ISBN 0–596–00016–2

21 Meyer 1993 Meyer, Dr. Hanns-Martin: Softwarearchitekturen für verteilte Verarbeitung. In: Hansen, Wolf-Rüdiger (Hrsg.): *Client-Server-Architektur*. 1. Auflage. Bonn; Paris; Reading, Mass. : Addison-Wesley, 1993, S. 69–116. – ISBN 3–89319–611–0

22 Monson-Haefel 2000 Monson-Haefel, Richard ; Loukides, Mike (Hrsg.): *Enterprise JavaBeans(TM)*. Second Edition. 101 Morris Street, Sebastopol, CA 95472 : O'Reilly & Associates, Inc., 2000. – ISBN 1–56592–869–5

23 Oesterreich 1998 Oesterreich, Bernd: *Objektorientierte Softwareentwicklung*. 4., aktualisierte Auflage. München, Wien : Oldenbourg, 1998. – URL http://www.oose.de/. – Zugriffsdatum: 2000-10-31. – ISBN 3–486–24787–5

24 Reichel 2000 Reichel, Volker: Top in Form – Formatieren von Dokumenten mit XSL und FOP. In: *XML Magazin* (2000), Oktober, Nr. 10, S. 93–99. – Das XML Magazin ist eine ständige Beilage des Java Magazin's.

25 Rivest 1992 Rivest, Ronald L.: *The MD5 Message-Digest Algorithm*. RFC 1321. NE43-324, 545 Technology Square, Cambridge, MA 02139-1986: Massachusetts Institute of Technology, Laboratory for Computer Science (Veranst.), April 1992. – URL http://www.ietf.org/rfc/rfc1321.txt. – Zugriffsdatum: 2001-04-11. – Offizielle MD5 Spezifikation.

26 RSA Laboratories 1993 RSA Laboratories: PKCS #3: Diffie-Hellman Key-Agreement Standard / RSA Data Security, Inc. 100 Marine Parkway, Redwood City, CA 94065 USA, November 1993. – Forschungsbericht. – URL ftp://ftp.rsasecurity.com/pub/pkcs/ps/pkcs-3.ps. – Zugriffsdatum: 2001-01-31. Version 1.4; Copyright © 1991-1993 RSA Data Security, Inc. Public-Key Cryptography Standards (PKCS).

27 Scheb 2000 Scheb, Alexander: XML goes Swing – XML-Daten innerhalb einer Java-Anwendung verarbeiten. In: *XML Magazin* (2000), März, Nr. 3, S. 87–92. – Das XML Magazin ist eine ständige Beilage des Java Magazin's.

28 Schneier 1994 Schneier, Bruce: Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish). In: *Cambridge Security Workshop Proceedings (Dec. 1993)*, Springer-Verlag, 1994 (Fast Software Encryption), S. 191–204. – URL http://www.counterpane.com/blowfish.html. – Zugriffsdatum: 2001-04-19

29 Schulz 1999 SCHULZ, Dr. B. *BertelsmannSpringer Science+Business Media*. 1999. – URL http://www.bertelsmannspringer.de/u_profil/. – Zugriffsdatum: 2000-12-06

30 Sun Microsystems, Inc. 2000 Sun Microsystems, Inc. (Veranst.): *JavaTM Secure Socket Extension (JSSE) 1.0.2 – API User's Guide*. 2000. – URL http://java.sun.com/products/jsse/doc/guide/API_users_guide.html. – Zugriffsdatum: 2001-01-05

31 Walrath und Campione 1999 WALRATH, Kathy ; CAMPIONE, Mary: *The JFC Swing Tutorial: A Guide to Constructing GUIs*. Reading, Ma. : Addison-Wesley, Juli 1999 (The Java series). – ISBN 0–201–43321–4

32 Web:BauNetz URL http://www.baunetz.de/. – Zugriffsdatum: 2001-05-04. – Website der Firma BauNetz GmbH & Co. KG.

33 Web:BSSBM URL http://www.bertelsmannspringer.de/. – Zugriffsdatum: 2001-05-04. – Website der Firma BertelsmannSpringer Science+Business Media.

34 Web:Castor URL http://castor.exolab.org/. – Zugriffsdatum: 2001-01-05. – Website für die Java-Bibliothek ≫Castor≪.

35 Web:Cryptix URL http://www.cryptix.org/. – Zugriffsdatum: 2001-01-05. – Website für die Java-Bibliothek ≫Cryptix≪.

36 Web:CVS URL http://www.cvshome.org/. – Zugriffsdatum: 2001-01-10. – Website für das Versionierungssystem ≫CVS≪.

37 Web:FOP URL http://xml.apache.org/fop/. – Zugriffsdatum: 2001-02-03. – Homepage des FOP-Projekts.

38 Web:Forte4J URL http://www.sun.com/forte/ffj/. – Zugriffsdatum: 2001-01-04. – Website für das Tool ≫Forte for Java≪.

39 Web:GPL URL http://www.gnu.org/copyleft/gpl.html. – Zugriffsdatum: 2001-04-17. – Website zur GNU General Public License.

40 Web:JBuilder URL http://www.inprise.com/jbuilder/. – Zugriffsdatum: 2001-01-04. – Website für das Tool ≫JBuilder≪.

41 Web:JCE URL http://java.sun.com/products/jce/. – Zugriffsdatum: 2001-01-05. – Website für die Java Cryptography Extension.

42 Web:JCVS URL http://www.trustice.com/java/jcvs/. – Zugriffsdatum: 2001-04-17. – Website das Tool ≫jCVS II≪.

43 Web:JDBC URL http://java.sun.com/products/jdbc/. – Zugriffsdatum: 2001-05-09. – Website für die Java DataBase Connectivity-API.

44 Web:JSSE URL http://java.sun.com/products/jsse/. – Zugriffsdatum: 2001-01-05. – Website für die Java Secure Socket Extension.

45 Web:MySQL URL http://www.mysql.com/. – Zugriffsdatum: 2001-04-29. – Website für das relationale DBMS ≫MySQL≪.

46 Web:Q214828 URL http://support.microsoft.com/support/kb/articles/q214/8/28.asp. – Zugriffsdatum: 2001-05-07. – Microsofts Knowledge-Base Artikel zu verwendeten JVM-Versionen im Internet-Explorer.

47 Web:RUP URL http://www.rational.com/products/rup/. – Zugriffsdatum: 2001-05-09. – Die Webseite zum Rational Unified Process.

48 Web:W3C URL http://www.w3.org/. – Zugriffsdatum: 2001-01-10. – Website des World Wide Web Konsortiums.

49 Web:Xalan URL http://xml.apache.org/xalan-j/. – Zugriffsdatum: 2001-02-03. – Homepage des Xalan-Projekts.

50 Yeong u. a. 1995 Yeong, Wengyik ; Howes, Tim ; Kille, Steve: *Lightweight Directory Access Protocol*. RFC 1777. IETF Secretariat, c/o Corporation for National Research Initiatives, 1895 Preston White Drive, Suite 100, Reston, VA 20191-5434, USA: Network Working Group (Veranst.), März 1995. – URL http://www.ietf.org/rfc/rfc1777.txt. – Zugriffsdatum: 2001-05-09. – Offizielle LDAP Spezifikation.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, 11. Mai 2001

Matthias Kraft