Universität Kaiserslautern
Fachbereich Elektrotechnik
Lehrstuhl für Digitale Systeme
Prof. Dr.-Ing. S. Wendt

Diplomarbeit

Integration des Produkts Astoria in ein bestehendes Dokumentenmanagementsystem über eine CORBA-Schnittstelle

Oliver Schmidt
August 1998

Betreuer: Prof. Dr.-Ing. S. Wendt

Bearbeiter: Oliver Schmidt
Flurstr. 11
55758 Vollmersbach

Erklärung:

Hiermit erkläre ich, die vorliegende Diplomarbeit unter Verwendung der angegebenen Quellen und ohne fremde Hilfe angefertigt zu haben.

Walldorf, den 31. August 1998

Oliver Schmidt

Danksagung

Ich bedanke mich herzlich bei allen, die auf unterschiedlichste Art am Entstehungsprozeß dieser Arbeit teilgenommen haben und ohne die es diese Arbeit in der Form nicht geben würde.

An erster Stelle möchte ich Herrn Prof. Dr.-Ing. Siegfried Wendt danken, der als Leiter des Software Engineering Labors der Firma SAP AG diese Diplomarbeit betreute. Das von ihm im Studium vermittelte Weltbild hat meine persönliche Sicht stark geprägt und mir ermöglicht, diese Arbeit anzufertigen. Dabei förderte er stets die Freude an Erkenntnissen, die dazu beitragen, im ständig wachsenden Wald der Informationstechnik nicht die Orientierung zu verlieren. Sein mir entgegengebrachtes Vertrauen und der mir zugestandene Entscheidungsfreiraum waren außerordentlich groß und Basis einer sehr angenehmen Arbeitsatmosphäre.

Bedanken möchte ich mich auch bei Helga Liefkes und Bernhard Donner von Chrystal Software für die kostenlose Bereitstellung der Produkte Astoria 3.0, Astoria SDK und Bridges to Astoria. Bei Rückfragen erlebte ich beide als außergewöhnlich hilfsbereit.

Mein Dank gilt allen Mitarbeitern des Software Engineering Labors der Firma SAP: Christian Brand, Markus Cherdron, Frank Gales, Bernhard Gröne, Eberhard Iglhaut, Jürgen Langhauser, Johannes Otto, Stefan Steißlinger und vor allem Andreas Knöpfel. Ihnen verdanke ich viele wertvolle Anregungen und Ideen. Ihre stete Bereitschaft zu Diskussionen, die zur Klärung vieler Sachverhalte führte, war mir sehr hilfreich.

Oliver Schmidt

Inhaltsverzeichnis

1. Einleitung

1.1 Anmerkung zum Titel der Arbeit

Der Titel der vorliegenden Arbeit lautet „Integration des Produkts Astoria in ein bestehendes Dokumentenmanagementsystem über eine CORBA-Schnittstelle“. Dieser Titel mußte beim Anmelden der Diplomarbeit am 1. März 1998 angegeben werden. Allerdings wird diese Diplomarbeit diesem Titel nicht gerecht. Das hat folgende Gründe:

(a) Nach Anmeldung der Arbeit ist eine neue Version von Astoria erschienen. Da bei einer Kaufentscheidung zu Gunsten von Astoria, nur die aktuelle Version von Interesse ist, wurde die neue Version des Produkts evaluiert. Bei diesem Versionsübergang von Astoria 2.0 auf Astoria 3.0 wurde die Funktionalität derart erweitert, daß die beschreibenden Dokumente, auf die diese Arbeit aufsetzt, ihren Umfang verdoppelt haben.

(b) Das bestehende Dokumentenmanagementsystem, das als Laborprototyp realisiert ist, sollte um Funktionalitäten erweitert werden, die Astoria abdeckt. Die Änderungen an der CORBA-Schnittstelle wären so umfangreich gewesen, daß es nicht möglich gewesen wäre, das bestehende System zu ändern. Das Erarbeiten von Konzepten, die diese Änderungen betreffen, hätte den Rahmen einer Diplomarbeit gesprengt. Daher war eine Integration von Astoria nicht möglich.

Ein angemessener Titel dieser Arbeit ist „Evaluation des Dokumentenmanagementsystem Astoria 3.0“.

1.2 Kontext dieser Arbeit

Im Software Engineering Labor der Firma SAP AG in Walldorf wird im Rahmen des TimeLess-Projekts an einem Konzept zur Optimierung der Informationslogistik[1] in einem Softwareunternehmen gearbeitet. Hinter dem Projektnamen TimeLess verbirgt sich das in Bild 1 dargestellte Ackronym. Die Mitarbeiter einer Softwarefirma wie der SAP AG produzieren vornehmlich Information. Diese Information dient zum größten Teil zur Beschreibung von programmierten Systemen. Denn es handelt sich dabei um die Quelltexte, die von den einzelnen Entwicklern erstellt werden. Zusätzlich werden von den Entwicklern kleine Dokumente wie Skizzen oder Texte erstellt, die Anforderungen an das zu programmierende System beschreiben und als Kommunikationsgrundlage zwischen den Entwicklern dienen. In den Quelltexten und diesen kleinen Dokumenten ist ein Teil des Firmenwissens gespeichert. Der größte Teil des Wissens lagert allerdings in den Köpfen der Entwickler.

1. Unter Informationslogistik ist die Organisation des Flusses und der Lagerung von Information zu verstehen.

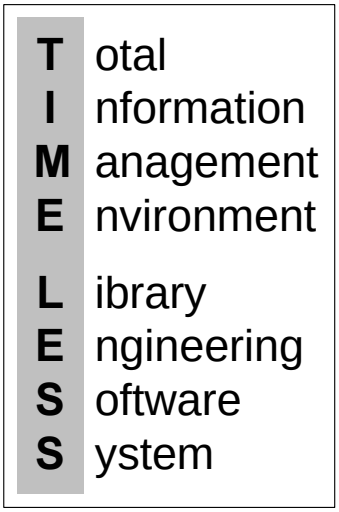


Bild 1: TimeLess-Ackronym

Dies hat aus Unternehmenssicht den Nachteil, daß jeder einzelne Entwickler unersetzlich ist, da jeder Spezialist auf seinem Gebiet ist und kein anderer dieses Gebiet kennt. TimeLess steht daher für einen Arbeitsstil, der es jedem einzelnen Entwickler leicht macht, Dokumente abzugeben. Die abgegeben Dokumente werden in einen Softwaresystem archiviert. Dieses Softwaresystem wird ebenfalls mit dem Namen TimeLess bezeichnet. Die Dokumente, die in TimeLess archiviert werden, sind dann der personenunabhängige Teil des Firmenwissen, der je nach Berechtigung von jedem Entwickler zugänglich ist.

Primär sollen vom Entwickler keine umfangreichen Abschlußberichte abgegeben werden, sondern die bereits erwähnten Kleinstdokumente. Damit ist es möglich, daß ohne größere Änderungen im Arbeitsstil des Entwicklers Dokumente abgegeben werden. Ein Vorgesetzter Komponiert aus den Kleinstdokumenten größere Dokumente. Komponieren eines größeren Dokuments kann man sich wie folgt vorstellen: Der Vorgesetzte erstellt einen Bericht, in dem er einen Text schreibt, der die Ergebnisse aus Interviews mit den Entwicklern enthält. Wenn dem Vorgesetzten bekannt ist, daß ein interviewter Mitarbeiter ein Kommentargrafik erstellt hat, kann er diese in den Bericht einfügen. Es soll auch möglich sein, daß der Vorgesetzte einen Platzhalter für diese Grafik einfügen kann, wenn diese Grafik noch nicht erstellt worden ist. Der zuständige Mitarbeiter erhält dann den Auftrag diese Grafik abzugeben.

Damit hat der Vorgesetzte zu überwachen, ob seine Mitarbeiter regelmäßig Dokumente abgegen. Auf der anderen Seite kann der Vorgesetzte für den Inhalt der komponierten Dokumente verantwortlich gemacht werden.

Ein Softwaresystem, das alle diese Möglichkeiten unterstützt, ist jedoch wertlos, wenn diese Strukturen nicht durch die Firmenpolitik eingeführt werden.

Archivierung von Information heißt auch, daß diese so abgelegt wird, daß sie leicht wieder gefunden werden kann. In TimeLess soll die Ablage so strukturiert sein, wie man sie in der realen Welt vorfindet. Dokumente, die in TimeLess abgelegt werden, befinden sich ausschließlich in Ordnern, die auf einem Regalbrett stehen, das zu einem Regal gehört, das in einem Saal steht usw. Die komplette Ablagestruktur ist in [Brand_97] beschrieben. Diese Ablagestruktur unterscheidet sich von den Ablagestrukturen bekannter Dateisysteme nicht nur darin, daß es unterschiedliche Containertypen[1] gibt, sondern auch dadurch, daß mit jedem Containertyp eine gewisse Kapazität verbunden ist. D.h. in einem Ordner in TimeLess können sich nicht beliebig viele Dokumente befinden. Genauso paßt nur eine endlich Zahl von Ordnern auf ein Regalbrett. Der Containertyp soll gleichzeitig ein Maß für die Informationsmenge sein, die in einem Container zu finden ist. Außerdem ist festgelegt, daß beispielsweise nur ein Regalbrett nur Ordner enthalten kann und nur in einem Regal enthalten sein kann, damit die Analogie zur realen Welt beibehalten wird.

Auch für die Organisation der Ablagestruktur soll es Verantwortliche geben. Diese werden als Bibliothekare bezeichnet. Wenn ein Dokument an TimeLess abgegeben wird, landet dieses bei einem Bibliothekar im Eingangskorb. Dieser trifft letztendlich die Entscheidung, wo das Dokument abgelegt wird. Der Bibliothekar kann das Dokument in dem Bereich ablegen, für den er zuständig ist. Es steht dem Bibliothekar aber frei, das Dokument nicht in seinem Zuständigkeitsbereich abzulegen.

Eine zweite Möglichkeit Dokumente zu finden ist die Volltextsuche. Nach Erfahrungen des Autors liefert eine Volltextsuche selten die gewünschten Ergebnisse. Entweder findet man das gewünschte Dokument nicht oder es werden so viele Dokumente gefunden, die für die gestellte Anfrage in Betracht gezogen werden können. Trotzdem soll im TimeLess-System eine Volltextsuche integriert werden. Diese kann dazu dienen, das Dokument innerhalb eines gewissen Bereichs wie einem Ordner oder einem Regal zu finden. Durch die Einschränkung auf einen kleinen Bereich, soll die Anzahl der gefundenen Dokumente gering gehalten werden. Außerdem wird davon ausgegangen, daß die Benutzer des Systems sich in der Ablagestruktur zurecht finden, daß die Volltextsuche im Idealfall überhaupt nicht benötigt wird.

Dokumente, die in TimeLess abgelegt werden, können nicht mehr geändert werden. ein Benutzer des Systems kann sich eine Kopie von einem Dokument machen, die er dann editieren kann. Die editierte Fassung des Dokuments kann dann wieder über den Bibliothekar ins System eingebracht werden. Dabei wird die alte Version des Dokuments nicht gelöscht. Dann würde das System nicht mehr dokumentieren können, wie die alte Version ausgesehen hat. Es ist auch möglich, daß das neue Dokument keine Beziehung mehr zur alten Version hat, weil man von diesem Dokument nur ein Bild benötigt hat. Daher soll TimeLess über keine automatische Versionierung verfügen. Allerdings soll, wenn eine Vorgänger-Nachfolgerversions-Beziehung zwischen zwei Dokumenten besteht, dieser Versionsübergang kommentiert werden. Dazu siehe auch [Langhauser_97].

1. Die Objekte der Ablagestruktur, die andere Objekte enthalten können, werden als Container bezeichnet.

Da es kein käufliches System gibt [IAO_97], das alle Anforderungen erfüllt, hat eine Gruppe des Software Engineering Labors damit begonnen, Prototypen zu bauen, die die obigen Anforderungen teilweise erfüllen. Der Entwicklungsprozeß und die Architektur dieser Prototypen ist in [Cherdron_98] beschrieben. Als einzige käuflich erwerbbare Komponente wird ein relationales Datenbanksystem eingesetzt. In diesen Prototypen gibt es zwei CORBA-Schnittstellen. Eine dieser Schnittstellen liefert eine Objektsicht auf das relationale Datenbanksystem. Die zweite lieferte eine dokumentenorientierte Sicht. In der Hoffnung, daß ein Dokumentenmanagementsystem an seiner Programmierschnittstelle ähnliche Begriffe anbietet wie die zuletzt genannte Schnittstelle, wird im Rahmen dieser Arbeit die Programmierschnittstelle eines käuflichen Dokumentenmanangementsystems analysiert. Ursprünglich war auch die Anpassung des betrachteten Systems an die vorhandene CORBA-Schnittstelle geplant. Warum dies nicht geschehen ist, wird in Kapitel 1.1 begründet.

1.3 Aufgabenstellung

Augangspunkt dieser Arbeit ist die Überlegung, daß in TimeLess Dokumente verwaltet werden. Dies ist auch die Aufgabe eines Dokumentenmanagementsystems. In der vorliegenden Arbeit wird ein solches System analysiert. Bei diesem System handelt es sich um das Dokumentenmanagementsystem Astoria 3.0 der Firma Chrystal Software. Die Wahl auf dieses System fiel, da es sich durch folgende Features auszeichnet:

(a) Hierarchische Ablagestruktur mit unterschiedlichen Containertypen

(b) Volltextsuche

(c) Ablage von strukturierten Dokumenten[1]

(d) Programmierschnittstelle

Wenn man die ersten drei Punkte betrachtet, dann decken sich diese mit den Anforderungen, die an das TimeLess-Softwaresystem gestellt werden. Die Programmierschnittstelle bietet die Möglichkeit das System um zusätzliche Funktionalität zu erweitern. Außerdem kann eine Anpassung an die bestehenden Schnittstellen des TimeLess-Softwaresystems erfolgen.

Allerdings reichen die Aussagen über das Vorhandensein eines Konzepts bei Weitem nicht aus, um festzustellen, ob die Konzepte von Astoria und TimeLess sich ähneln. Auch Marktstudien wie beispielsweise [IAO_98] können nur dazu dienen, Systeme auszuschließen, die nicht näher untersucht werden müssen.

In der vorliegenden Arbeit werden die Konzepte von Astoria dargestellt und mit den Konzepten von TimeLess verglichen. Bevor diese Konzepte vorgestellt werden, sollen zunächst die Vostellungen, die in TimeLess mit strukturierten Dokumenten verbunden sind, vorgestellt werden, Anschließend werden die in Astoria verwendeten Standards zur Strukturierung von Dokumenten näher beleuchtet. Abschließend erfolgt eine Bewertung, ob Astoria für den Einsatz im TimeLess-Projekt geeignet ist.

1. Astoria kennt die innere Struktur von SGML- und XML-Dokumenten.

2. Strukturierte Dokumente in TimeLess

2.1 Konzept

Im TimeLess-Softwaresystem sollen viele Kleinstdokumente abgelegt werden. Ein Kleinstdokument kann z.B. eine Klassenbeschreibung oder eine Zeichnung sein. Um zu vermeiden, daß die Dokumente zusammenhangslos in der Ablage liegen, sollen diese Kleinstdokumente in größere Dokumente eingebettet werden. Ein auf diese Weise komponiertes Dokument verfügt über sogenannte Dummies (Bild 2), die man sich als Platzhalter vorstellen kann, das jeweils mit einem Kleinstdokumenten beklebt werden kann. Ein Kleinstdokument, das auf diese Art und Weise eingeklebt werden kann, wird auch als Baustein bezeichnet.

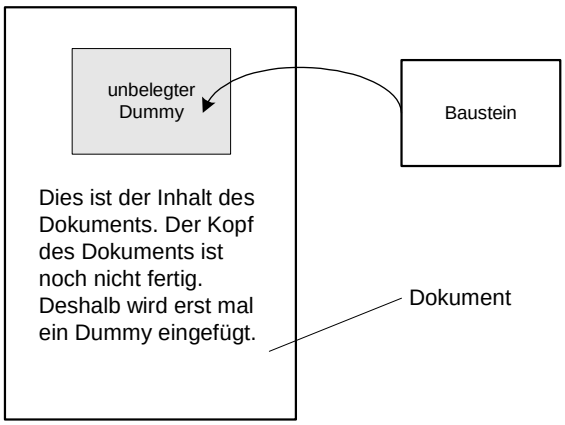


Bild 2: Dokument mit Dummy

Nun besteht ein Dokument in TimeLess nicht nur aus Dummies. Wenn man ein Dokument erstellt, möchte man nicht die Vorstellung besitzen, daß man einen Baustein macht, der später in ein Dokument eingefügt wird. Während des Erstellens soll man nur mit dem Dummy-Konzept konfrontiert werden, wenn man genau weiß, daß an einer Stelle des Dokuments noch etwas eingefügt werden soll, was man aber gerade nicht zur Hand hat oder was noch nicht existiert. Man könnte sich vorstellen, daß ein Dokument erstellt werden soll, in dem ein Firmenlogo enthalten ist. Der Autor dieses Dokuments kann dann einen Dummy an der entsprechenden Stelle plazieren und muß deshalb nicht sofort auf die Suche nach dem Logo gehen.

Außerdem gibt es Dokumente, die als Vorlage für andere Dokumente dienen. Diese kann man sich wie ein Formular vorstellen. Die freien Felder werden dabei durch Dummies realisiert. Das Formular wird ausgefüllt, in dem die Dummies belegt werden.

Ein so entstandenes Dokument kann wieder als Baustein verwendet werden. Ein Dummy muß nicht unbedingt sein Layout erhalten. Beispielsweise kann ein Dummy mit einem Baustein belegt werden, der über mehrere Seiten geht. Eine ausführliche Beschreibung des Konzepts findet man in [Langhauser_97].

2.2 Probleme bei der Realisierung

Bausteine werden nicht alle mit dem selben Tool erstellt. Eine Zeichnung wird mit einem anderen Tool erstellt als beispielsweise ein Quelltext. Dabei hat jedes Tool ein eigenes Format, in dem das Erstellte abgespeichert wird. Um ein Dokument mit Dummies, die unterschiedliche Inhalte besitzen, erzeugen zu können, muß es ein Tool geben, das alle diese Formate kennt. Das Problem dabei ist, daß es potentiell unendlich viele Formate geben kann, wobei sich meistens die Formate auch noch von Version zu Version eines Produkts ändern.

Bild 3 zeigt, daß ein strukturiertes TimeLess-Dokument neben dem eigentlichen Inhalt des Dokuments, der aus Texten und Bildern bestehen kann, noch die Bausteine und die Dummybelegung beinhalten muß. Mit sonstiger Inhalt des Dokuments sind auch die Positionen der Dummies innerhalb des Dokuments gemeint. Wie dieser Inhalt abgelegt wird, muß dem Viewer natürlich bekannt sein. Aus den Informationen aus dem strukturieren TimeLess-Dokument und den Darstellungsinformationen, die zur Interpretation der Bausteindaten benötigt werden, kann der Viewer dann eine Darstellung des Dokuments generieren.

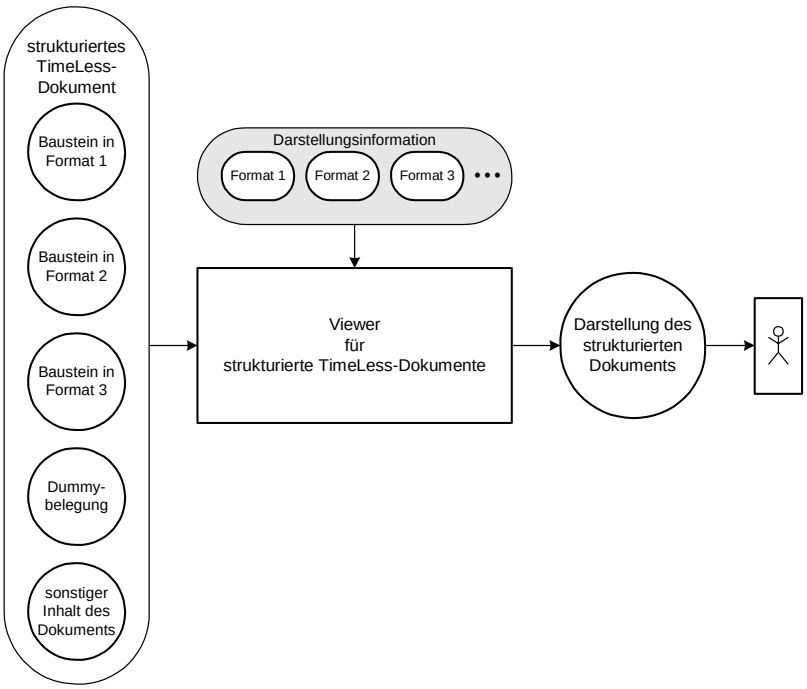


Bild 3: Anzeigen von strukturierten Dokumenten

Die Darstellungsinformation wird aber meistens nicht von den einzelnen Herstellern veröffentlicht. Diese Informationen sind folglich nur dem Hersteller bekannt. Zwar bietet fast jeder Hersteller ein Austauschformat an, dessen Interpretationsvorschriften frei verfügbar sind. Dann müßte ein Viewer mit diesen Austauschformaten arbeiten. Für das TimeLess-Projekt würde dies das Bauen eines Viewers bedeuten, der eine gewisse Menge von Austauschformaten beherrscht. Bei Neuerscheinung eines Formats muß ein neuer Viewer erstellt werden, der bei allen Nutzern des Systems installiert werden muß. Außerdem muß ein Autor dann das Dokument in zwei Versionen abgeben. Eine Version ist das Dokument im Austauschformat und die andere im Originalformat. Das Originalformat ist notwendig, da in den Austauschformaten häufig keine Information über spezielle Features des Erstellungstools vorhanden sind. Beispielsweise könnte das Austauschformat keine Nummerierungsinformation für Querverweise enthalten. Damit ist ein Querverweis im Austauschformat nicht vom sonstigen Text des Dokuments zu unterscheiden. Wenn dieser Baustein als Grundlage für ein neues Dokument dienen soll, dann müßte der Autor alle Querverweise von Hand ändern, die sein Textverarbeitungssystem sonst automatisch nachgeführt hätte.

Die obige Lösung hat das Problem, daß jeder Benutzer beim Erscheinen eines neuen Formats, einen neuen Viewer installieren muß. Wenn man den Viewer so baut, daß es das strukturierte Dokument in einem weit verbreiteten Standardformat abspeichert, dann benötigt jeder Benutzer nur einen Standardviewer (Bild 4). Dann müßte nur noch an einer zentralen Stelle des Systems Änderungen vorgenommen werden, wenn sich ein Format ändert. Allerdings gibt es bei dieser Lösung dann drei Formate, die in der TimeLess-Datenbank abgespeichert werden müssen, nämlich das Originalformat, das Austauschformat und das Standardformat.

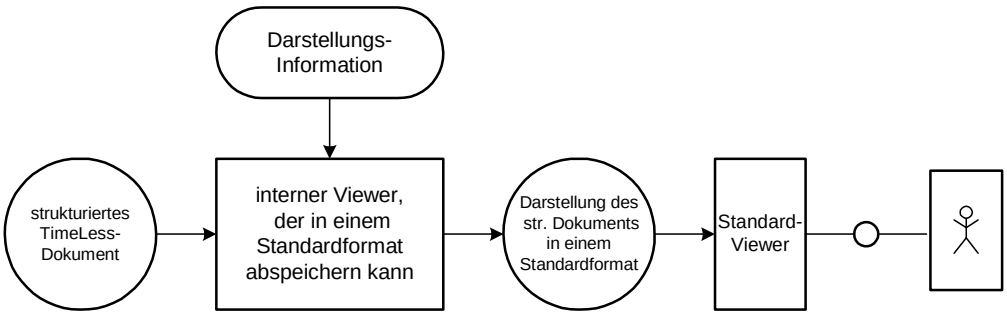


Bild 4: Darstellung eines strukturierten Dokuments in einem Standardformat

Ein Benutzer des Systems will aber nicht nur Dokumente anzeigen, sondern auch welche erstellen. Da in TimeLess die Dokumente, die in das System eingebracht worden sind, nicht mehr geändert werden, kann ein Dokument aus TimeLess nur als Basis für ein neues Dokument dienen. Wenn der Benutzer sich ein strukturiertes Dokument aus TimeLess kopiert, kann es sein, daß dieses aus Bausteinen besteht, die wiederum aus Bausteinen zusammengesetzt sind. Wie bereits erwähnt kann jeder Baustein mit einem anderen Tool erstellt worden sein, wobei jedes Tool ein anderes Format zum Abspeichern nutzt. Nun möchte man als Benutzer aber nicht mit vielen unterschiedlichen Tools an einzelnen Teilen des Dokuments arbeiten, da auf diese Art und Weise die Übersicht über das Gesamtdokument verloren geht.

Eine Lösung für dieses Problem ist in Bild 5 angegeben, die zur Zeit aber mangels geeigneter Standards nicht realisierbar ist. Im Vergleich zu Bild 3 ist der Speicher für strukturierte Dokumente anders partitioniert. Alle Bausteine sind in Bild 5 zum Inhalt des Dokuments zumsammengefaßt worden. Dagegen wird nun ein Speicher für das Gerüst des Dokuments gezeigt. Im Dokument gibt es Bilder und Texte, die nicht Baustein sind. Die Trennung zwischen den Bausteinen und den Nichtbausteinen soll für diese Realisierung nicht mehr gültig sein. Damit kann man sich das Gerüst des Dokuments wie eine Liste mit Dummyplazierungen vorstellen. Die Dummybelegung realisiert die Belegungsrelation. Darin wird sich auch gemerkt, ob dies ein Baustein ist, wie ihn auch der Benutzer erlebt, oder ob der Benutzer diesen Baustein als festen Bestandteil des Dokuments erlebt und diesen somit gar nicht mehr ansehen kann, daß es sich dabei um einen Baustein handelt.

Im strukturierten Dokument muß zusätzlich Rollenbeschreibungen von Formatakteuren abgelegt werden. Ein Formatakteur kann die Funktionalität von einfachen Anzeigefähigkeiten bis zur kompletten Textverarbeitung abdecken, wobei dieser jeweils für ein einziges Format zuständig ist. Ein Träger für diesen Mechanismus muß aus den Daten, die im strukturierten Dokument enthalten sind, das Dokument aufbauen, in dem er die zuständigen Akteure aktiviert. Im Aktionsfeld dieser Akteure gibt es dann eine Repräsentation des gesamten Dokuments. Auf dieses Aktionsfeld kann der Benutzer zugreifen und das Dokument modifizieren. Dabei ist dann jeder Formatakteuer für die Änderungen, die in seinem Aktionsfeld erfolgt sind, zuständig. Damit ist gewährleistet, daß jeder einzelne Baustein im korrekten Format abgespeichert wird. Genormt werden müßte lediglich die Schnittstelle[1] zwischen Formatakteur und Trägersystem. Dann müßte jeder Hersteller für sein Editionstool einen Beschreibung anbieten, mit der ein Formatakteur erzeugt werden kann.

Wenn man sich Formatakteur B betrachtet, fällt auf, daß es sozusagen für das Anzeigen der Seite zuständig ist. Dieser Akteur wird über denselben Mechanismus erzeugt und verwaltet wie die anderen Formatakteure auch. Damit kann man garantieren, daß dieses Dokument auch wieder als Baustein für ein weiteres Dokument verwendet werden kann. Denn dieser Akteur kann genauso eingebettet werden, wie die anderen Akteure. Außerdem ist man nicht an ein Format gebunden, mit dem man eine Seite darstellt.

Das Erzeugen eines Viewing-Formats ist ebenfalls kein Problem. Es gibt eine vollständige Darstellung des Dokuments, auf die ein Transformierer zugreifen kann und aus der er ein Viewing-Format generieren kann. Das Viewing-Format bringt Performanzvorteile. Denn wenn bei jeden Betrachten des Dokuments alle Formatakteure transportiert und aktiviert werden müssen, ist der Resourcenverbrauch dieses Systems sehr groß. Im Viewing-Format muß nämlich nur das Dokument selber transportiert werden, das keine Akteursbeschreibungen enthält. Der verarbeitenden Akteur (Viewer) ist schon beim Benutzer installiert.

1. Eine solche Schnittstelle wird in [DOM_98] festgelegt. DOM steht für Document Object Modell. Darin beschrieben sind eine Vielzahl von Schnittstellen. Diese Schnittstellen sollen Objekte einer Programmiersprache besitzen. Damit kann ein Dokument in einer Objektwelt dargestellt werden.
Allerdings gibt es noch keine Produkte, die diese Norm unterstützen, da es noch herstellerabhängige Auslegungen dieser Norm gibt und damit der Normungsprozeß noch gar nicht abgeschlossen ist.

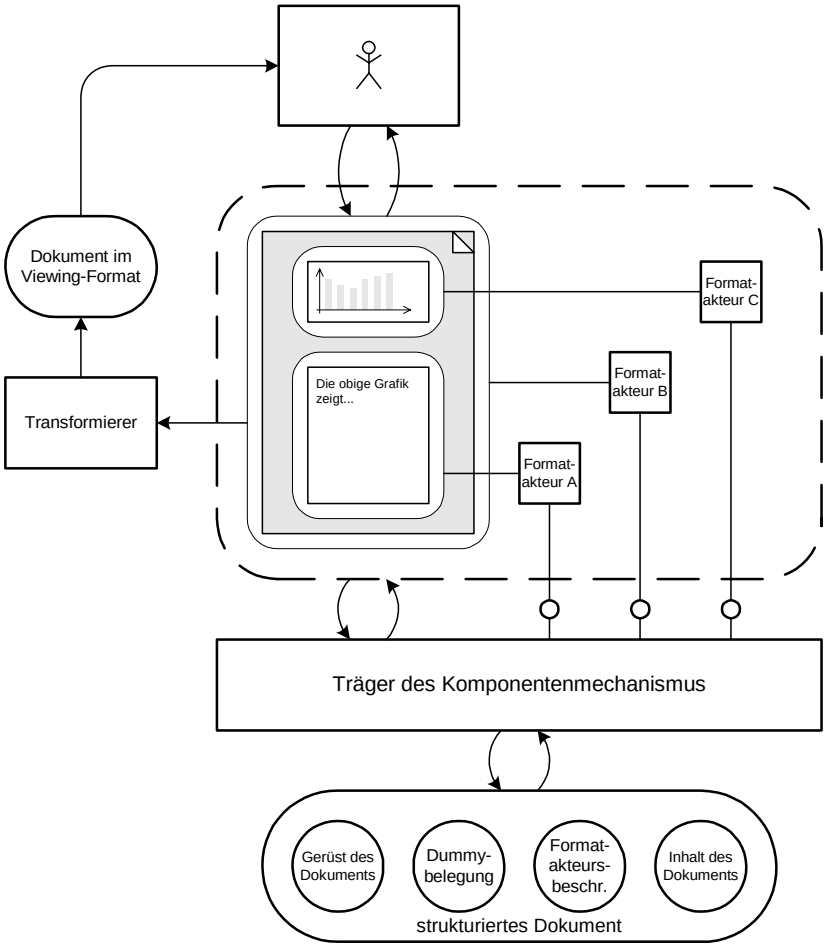


Bild 5: Konzept zur Realisierung von strukturierten Dokumenten

Bei der vorgestellten Lösung handelt es sich um eine Vision. Zur Zeit ist noch kein geeignetes Trägersystem vorhanden.

3. Markupsprachen

Wie im vorangegangenen Kapitel beschrieben gibt es in TimeLess ein Konzept zur Strukturierung von Dokumenten. Bei Dokumentenmanagementsystemen wie Astoria werden ebenfalls strukturierte Dokumente unterstützt. Um die Funktionalität von Astoria verstehen zu können, wird zunächst der benutzte Standard erklärt. Es handelt sich dabei um die Markupsprache SGML[1]. Da man sich noch nicht auf einen Strukturierungsstandard für Dokumente in TimeLess festgelegt hat, soll in diesem Kapitel auch eine Einordnung der Markupsprachen HTML[2] und XML[3] erfolgen. Dabei geht es eher um die prinzipiellen Konzepte als um eine vollständige Beschreibung. Für detailliertere Informationen sei auf [AborText_95] und [Sperberg-McQueen] verwiesen. Diese Texte bilden auch die Basis für die Aussagen dieses Kapitels.

3.1 Was ist Markup ?

Ein Markup-Dokument enthält nicht nur seinen Inhalt in Form von Text, sondern auch zusätzliche Informationen über Layout und Struktur des Dokuments. Jede Information, die nicht Inhalt des Dokuments ist, wird dabei als *Markup* bezeichnet. Der Begriff Markup stammt aus der Sprache der Setzer, in der die Anweisungen an den Setzer als Markup bezeichnet wurden. In den betrachteten Markupsprachen äußert sich Markup in Form sogenannter *Tags*.

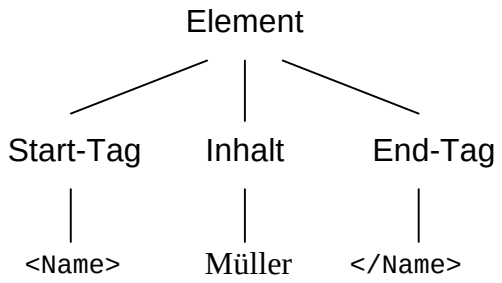


Bild 6: Aufbau eines Elements

Dieser markierte Bereich wird auch als *Element* bezeichnet. Das Element in Bild 6 enthält also den Text „Müller“ und ist vom Typ „Name“. Dieses Element wird durch ein Starttag und ein Endtag gekennzeichnet.

Ein Dokument besteht ausschließlich aus solchen Elementen. Es gibt auch die Möglichkeit, daß ein Element in einem anderen Enthalten ist, wie das folgende Beispiel zeigt.

1. SGML: Standard Generalized Markup Language
2. HTML: HyperText Markup Language
3. XML: eXtensible Markup Language

```
<Kapitel>
  <Überschrift>Herr<Name>Müller</Name>lernt SGML </Überschrift>
  <Absatz>...</Absatz>
</Kapitel>
```

Hier ist auch zu erkennen, daß es unterschiedliche Markuptypen gibt. Denn das Tag „Kapitel“ markiert die Struktur des Dokuments, wohingegen das Tag „Name“ den Inhalt des Dokuments markiert. Mit dem Überschrifts-Tag kann man beispielsweise eine Anweisung an einen Darstellungsakteur verbinden, die Überschrift „fett“ darzustellen. Allerdings hat dieses Tag auch den Charakter des Strukturmarkups, da hiermit angegeben wird wie ein Kapitel aufgebaut ist.

Wenn ein Tag immer auf die selbe Art und Weise interpretiert wird, dann spricht man auch von *prozeduralem Markup*. Die Textverarbeitungsprogramm oder der Editor, die die Markupinformation erzeugt hat, ist auch das einzige Programm, das diese Information lesen kann. Beim Parsen des Dokuments wird immer der selbe Prozeß gestartet, wenn dieses Tag erkannt wird. Dies ist bei den meisten Textverarbeitungssystemen der Fall. Es besteht aber auch die Möglichkeit, daß ein Dokument durch verschiedene Akteure auf unterschiedliche Art interpretiert werden. In diesem Fall spricht man von *deskriptiven Markup*. Als Beispiel (Bild 7) für deskriptives Markup kann man sich einen Layouterzeuger vorstellen, der mit einem Überschriftstag die Schrift und deren Größe zum Darstellen verbindet. Dabei wird das Namens-Tag ignoriert und der Text des Namens-Elements im Überschriftslayout dargestellt. Ein Akteur, der alle Namen, die in einem Dokument vorkommen, in eine Liste schreibt, extrahiert dahingegen nur den Text, der in den Namenselementen steht.

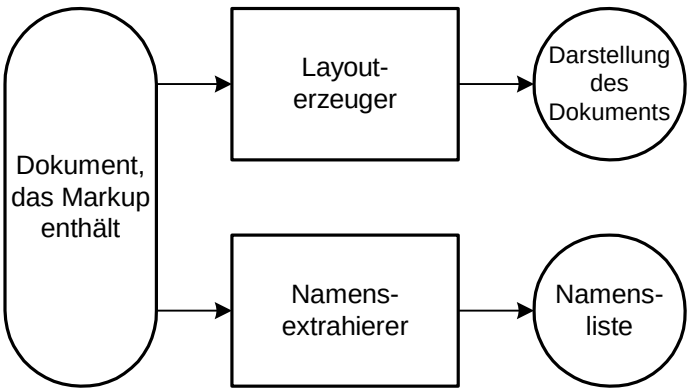


Bild 7: Unterschiedliche Interpretation durch verschiedene Akteure

3.2 SGML (Standard Generalized Markup Language)

3.2.1 Einsatzgebiet von SGML

SGML ist seit 1986 ISO-Standard. Eingesetzt wird SGML hauptsächlich bei der technischen Dokumentation und im Verlagswesen. Von der umfangreichen Spezifikation wird ein Kernbereich genutzt, wobei Funktionalitäten[1] spezifiziert worden sind, die selten genutzt werden, aber aufwendig zu implementieren sind. Dies ist der Hauptkrititkpunkt an SGML. Denn Werkzeuge, die den SGML-Standard unterstützen sind aus diesem Grund sehr teuer, weshalb SGML auch nur im professionellen Bereich von Großfirmen eingesetzt wird.

3.2.2 SGML technisch gesehen

SGML ist eine *Metasprache*. Eine Metasprache dient zum Beschreiben einer Sprache [WENDT_91]. In der Metasprache wird die Grammatik, die auch die Terminale enthält, der neu zu beschreibenden Sprache definiert. Die Grammatik in SGML heißt *DTD (Document Type Definition)*. Mit Hilfe der DTD kann der Typ eines Dokuments geprüft werden. Das zu prüfende Dokument wird dabei als *SGML-Dokument* bezeichnet. Dies kann zu Verwirrungen führen, da ein Dokument meistens nach seinem Typ benannt wird, z.B. WORD-Dokument oder Visio-Dokument. Wenn ein Dokument konform zu einer Rechnungs-DTD ist, dann spricht man trotzdem von einem SGML-Dokument und nicht von einer Rechnung oder einem Rechnungs-Dokument. In der vorliegenden Arbeit wird dieser Sprachgebrauch beibehalten und Dokumente, deren Grammatik durch eine DTD beschrieben werden, als SGML-Dokument bezeichnet.

Eine DTD wird meistens nicht vom Autor der SGML-Dokumente erstellt, sondern von einem sogenannten *Anwendungsentwickler*[2] (Bild 8). Als Anwendungsentwickler wird in der SGML-Welt derjenige bezeichnet, der die DTD und sonstige Vereinbarungsdokumente, die zum Betrieb eines SGML-Systems benötigt werden, erstellt. Der Autor muß natürlich die DTD kennen, damit er korrekte Dokumente erstellen kann. Zur Erstellung des Dokuments ist die DTD nicht unbedingt nötig. Mit der DTD wird für den Typprüfer festgelegt, auf welche Sprache das vorliegende SGML-Dokument geprüft wird.

Eine Textverarbeitung, die SGML-Dokumente erstellen kann, bietet mehr Komfort als nur die Typprüfung eines Dokuments. Bild 9 zeigt eine SGML-Textverarbeitung. Der Transformierer kann mittels der DTD eine interne Darstellung des SGML-Dokuments erzeugen. Das SGML-Dokument liegt häufig als Datei vor, wobei dies dann die Transportcodierung ist. Dagegen kann

1. Beispielsweise ist es in SGML zulässig, daß Elemente nicht mit einem Start- oder End-Tag gekennzeichnet sein müssen. Dieses erfordert einen komplexeren Parsing-Algorithmus.
2. Es gibt auch den Begriff der SGML-Anwendung. Zu einer SGML-Anwendung gehört nicht nur die DTD. Dieser Begriff wird später erklärt.

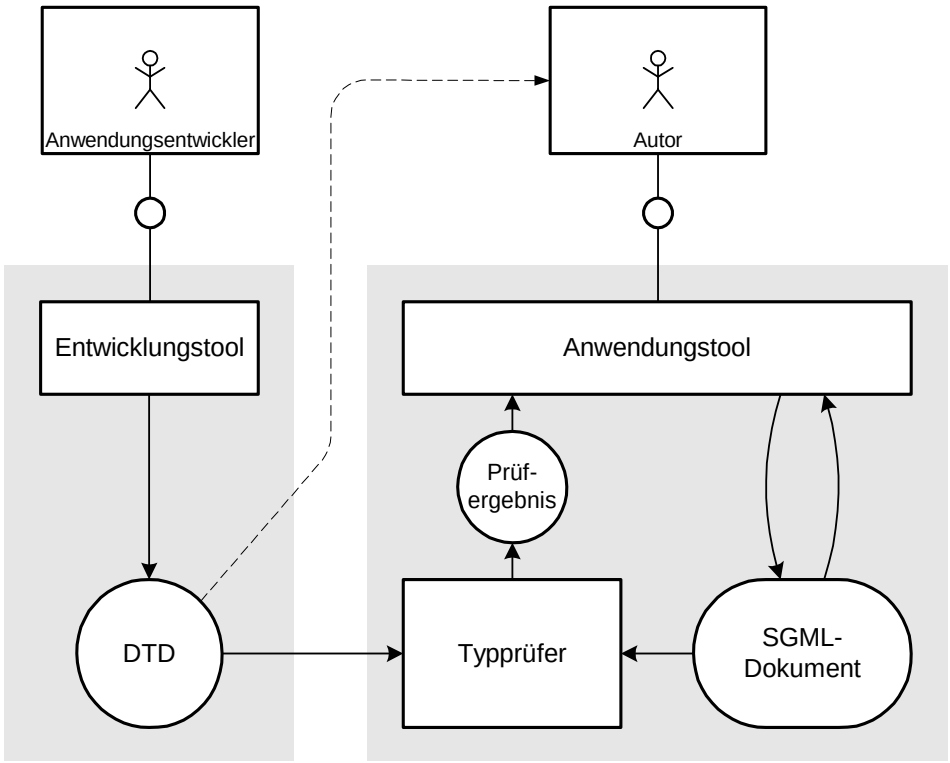


Bild 8: Typprüfung eines SGML-Dokuments

man die interne Darstellung als Arbeitscodierung bezeichnen. Änderungen am Dokument nimmt ein Benutzer des Textverarbeitungssystems an der internen Darstellung vor. Aus der internen Darstellung kann der Transformierer wieder die Transportcodierung des Dokument erzeugen, wenn der Benutzer das Dokument abspeichern möchte. Die Bedienung des Systems erfolgt über den Interaktionsakteur. Damit es dem Benutzer leicht gemacht wird, Dokumente zu erstellen, die konform zu einer durch die DTD beschriebenen Grammatik sind, hat der Interaktionsakteur auch Zugriff auf die DTD. Beim Editieren des Dokuments kann dem Benutzer beispielsweise eine Liste angezeigt werden, die die aktuell erlaubten Elemente beinhaltet.

Bisher wurden noch keine Aussagen darüber gemacht, in welcher Form dem Benutzer des Systems das Dokument präsentiert wird. Betrachtet man den Ausgabeakteur des Systems, so stellt man fest, daß dieser ein internes Layoutrepertoire besitzt, das z.B. bestimmte Schriftarten umfaßt. Dieses Layoutrepertoire ist nicht durch SGML festgelegt, sondern ist durch den Hersteller des Textverarbeitungssystem definiert. Nun muß eine Relation zwischen den Tags und diesem herstellerspezifischen Layoutrepertoire bestehen, damit der Ausgabeakteur das Dokument ausgeben kann. Die Relation wird durch die Layoutvereinbarungen beschrieben. Der Beschreibungsstandard für die Layoutvereinbarungen heißt DSSSL[1]. Die Layoutvereinbarung ist praktische eine Liste mit allen Tags und dem Hinweis, wie jedes einzelne Tag darzustellen ist.

1. DSSSL:Document Style Semantics and Specification Language
   Standard seit 1995

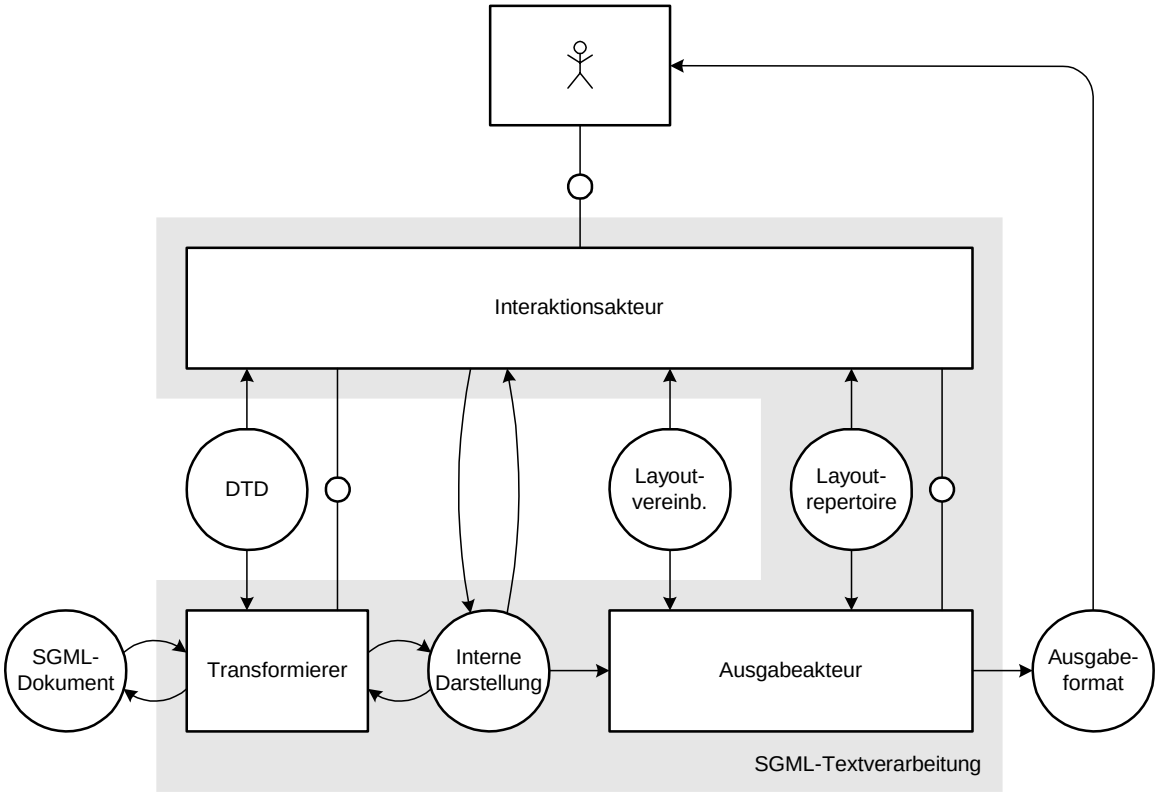


Bild 9: SGML-Textverarbeitung

Mit Ausgabeformat ist in Bild 9 ein ausgedrucktes Dokument oder eine Datei gemeint, die in einem Format abgespeichert wurde, das ein eindeutiges Layout besitzt. Wenn es sich bei der Textverarbeitung um ein WYSIWYG[1]-System handelt, muß der Interaktionsakteur ebenfalls Zugriff auf das Layoutrepertoire und den Layoutvereinbarungen haben. Um den das Dokument auf dem Bildschirm darstellen zu können, besitzt der Interaktionsakteur ebenfalls einen Ausgabeakteur, der in dem obigen Bild allerdings nicht dargestellt ist.

Ein weiterer Begriff, der immer im Zusammenhang mit SGML auftritt ist die *SGML-Applikation* oder *SGML-Anwendung*. Darunter versteht man die Vereinigung aus allen Vereinbarungen, die durch DTD und Layoutvereinbarung getroffen wurden. Sowohl die Layoutvereinbarung wie auch die DTD können ausgetauscht werden. Damit ist bei SGML das Eingabeformat und das Ausgabeformat von außen her beeinflußbar. Das selbe Dokument kann auf unterschiedliche Art ausgegeben, in dem lediglich die Layoutvereinbarungen geändert werden. Änderungen der DTD ziehen auch meistens eine Änderung der Layoutvereinbarungen nach sich, da in der DTD andere Tags definiert sind, die in der Layoutvereinbarung nicht vorkommen.

Ein SGML-Dokument liegt immer im Textformat vor. Auch die Buchstaben können auf unterschiedliche Art codiert sein. Das Codierungsverfahren steht in der sogenannten *SGML-Declaration*. Wie Bilder in SGML verwaltet werden, wird in Kapitel 3.2.4 geklärt.

1. WYSIWYG: What You See Is What You Get. Bezeichnung für Textverarbeitungssysteme, deren Bildschirmrepräsentation dem Aussehen des ausgedruckten Dokuments entspricht.

3.2.3 Beispiel

Um die bisherigen Aussagen zu verdeutlichen, soll an einem konkreten Beispiel gezeigt werden, wie eine SGML-Datei aussieht. Dazu wird das Beispiel aus [Sperberg-McQueen] verwendet.

```
<anthologie>
  <gedicht>
    <überschrift>The SICK ROSE
    <strophe>
        <zeile>O Rose thou art sick.
        <zeile>The invisible worm,
        <zeile>That flies in the night
        <zeile>In the howling storm:
    <strophe>
        <zeile>Has found out thy bed
        <zeile>Of crimson joy:
        <zeile>And his dark secretlove
        <zeile>Does thy life destroy.
  </gedicht>
  <!-- Hier können weitere Gedichte folgen -->
</anthologie>
```

In diesem Beispiel ist eine Anthologie[1] mit entsprechendem Markup versehen. Diese Anthologie besteht aus einem Gedicht, das zwei Strophen mit jeweils vier Zeilen besitzt. Es fällt auf, daß in diesem Beispiel Elemente vorkommen, die nur ein Anfangstag aber kein Endtag besitzen. Durch die DTD ist festgelegt, daß die Grenzen eines Elements bis zum nächsten zulässigen Elements gehen. So enthält das Element „erste Strophe“ alle vier Zeilen. Innerhalb eines Gedichts kann immer nur eine Strophe auf die nächste folgen. Wie die Relation zwischen den Tags aussieht, ist durch die DTD festgelegt:

```
<!ELEMENT anthologie      - - (gedicht+)>
<!ELEMENT gedicht         - - (überschrift?, strophe+)>
<!ELEMENT überschrift     - O (#PCDATA) >
<!ELEMENT strophe         - O (zeile+) >
<!ELEMENT zeile           O O (#PCDATA) >
```

Eine Zeile der DTD definiert jeweils einen Elementtyp. Nach dem Schlüsselwort Element folgt der Name des Elementtyps. Danach wird angegeben, ob ein Anfangstag und ein Endtag benötigt werden. Ein „-“ bedeutet, daß dieses Tag zwingend vorhanden sein muß. Das „o“ (optional) gibt an, daß dieses Tag auch weggelassen werden kann. Als letztes folgen dann die Grammatikregeln[2]. Durch das Schlüsselwort #PCDATA wird angegeben, daß es dieses Element nur noch durch Buchstaben (**P**arsed **C**haracter Data) ersetzt werden kann. Dieses Element kann dann keine weiteren Element mehr beinhalten.

1. Anthologie: Sammlung literarischer Texte
2. Symbole der Grammatik
   - (,) Gruppe von aufeinanderfolgenden Elementen
   - ? 0 oder 1
   - + mindestens eins

Wenn diese DTD zu Grunde liegt, um zu testen, ob unsere Anthologie eine korrekte Anthologie ist, dann muß dies bejaht werden. Die DTD gibt an, daß eine Anthologie aus mindestens einem Gedicht bestehen muß, das eine Überschrift haben kann und mindestens eine Strophe hat. Dabei sollte eine Strophe aus mindestens einer Zeile bestehen. Sowohl die Strophe als auch die Überschrift setzen sich aus Buchstaben zusammen. Nun fällt bei den Zeilen auf, daß es weder ein Anfangs- noch ein Endtag geben muß. Die Zuordnung, was als Zeile interpretiert wird, muß vom Typprüfer gemacht werden. Dies kann über Sonderzeichen, wie „CarriageReturn“ oder „LineFeed“ geschehen.

3.2.4 Behandlung von Grafiken in SGML

Ein SGML-Dokument liegt als reine Textdatei vor. Grafiken werden als sogenannte *Entities* eingefügt. Im Folgenden werden die Sprachkonstrukte vorgestellt, die zum Einfügen von Grafiken benötigt werden.

Entitäten werden im SGML-Dokument definiert. Enitäten können als reine Textersetzung definiert werden.

```
<!ENTITY anrede „Sehr geehrte Damen und Herren !“>
```

Im Text werden diese durch ein vorangestelltes „&“ gekennzeichnet.

```
<Brief>
  ...
  Betr...
  &anrede
  Innerhalb dieses Briefes gibt es kein Markup...
</Brief>
```

Als weitere Möglichkeit kann eine Entität auch eine Datei enthalten, die kein SGML-Dokument ist. Diese Datei könnte beispielsweise eine Landkarte beeinhalten. Die komplette Beschreibung der Entitäts-Definition folgt weiter unten.

```
<!ENTITY landkarte SYSTEM „deutschland.bmp“ NDATA bild>
```

Diese Entität wird nicht durch ein vorangestelltes „&“ in das Dokument eingefügt. Zum Einfügen wird der Attribut-Mechanismus benutzt. Einem Element in SGML kann man Attribute zuordnen. Attribute bestehen immer aus einem Namen und einem Wert. Der Wertebereich des Attributs wird, genau wie der Name, in der DTD definiert.

```
<bericht status=draft>....</bericht>
```

Im vorliegenden Beispiel hat das Berichtselement das Attribut „status“, das mit dem Wert „draft“ belegt ist. Dabei stammt die Belegung „draft“ aus dem in der DTD definierten Wertebereich für dieses Attribut. Es gibt einige Attribute, mit denen man eine feste Interpretation ver-

bindet, wie beispielsweise das ID-Attribut. Damit wird immer auf eine Entität verwiesen. Der Wert des ID-Attributs wird immer mit einem Entitäts-Namen belegt. Welche Attribute ein Element besitzt, hängt immer vom Elementtyp ab.

Zum Einfügen der Grafik-Entität kann man sich vorstellen, daß nicht nur eine ID als Attribut zugelassen ist, sondern auch Attribute, die die Position der Grafik zu lassen.

```
<graph id=landkarte x_postion=0 y_position=0>
```

Nun soll die Beschreibung der obigen Entitäts-Definition vervollständigt werden. Die Definition enthält das Schlüsselwort „SYSTEM“. Hiermit wird angegeben, daß das nun folgende eine systemspezifische Definition ist. Hier bedeutet dies, daß „deutschland.bmp“ als Dateiname zu interpretieren ist. Nun könnte man ja an dieser Stelle eine SGML-Datei einfügen, die das System wieder ohne weitere Vereinbarungen interpretieren kann. Ein Bild, wie im vorliegenden Fall, ist aber nicht mit den SGML-Regeln zu interpretieren, was durch das Schlüsselwort NDATA (**N**on Sgml-**DATA**) angedeutet wird. Also muß diese Datei auf eine andere Art und Weise interpretiert werden. Wie das geschieht gibt der letzte Teil dieser Entity-Definition an. „Bild“ ist der Name einer sogenannten Notation, die einen Hinweis auf die Interpretationsvorschrift gibt. Als Beispiel könnte man die Notation für „bild“ folgendermaßen festlegen:

```
<!NOTATION bild SYSTEM bmp>
```

Hiermit wird definiert, daß für ein Bild der Bmp-Filter des Transformierers genutzt werden soll (Bild 10). In diesem Bild kann man auch erkennen, daß ein SGML-Dokument nicht nur aus

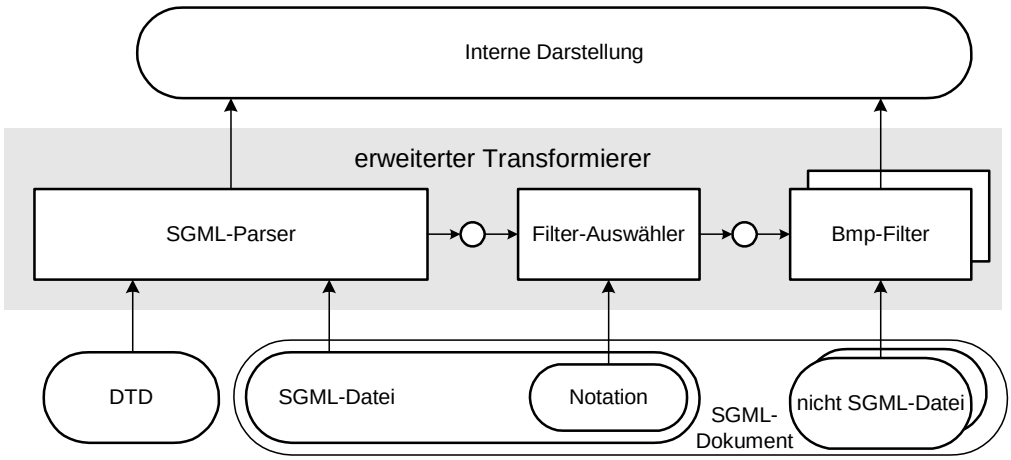


Bild 10: Erweiterter Transformierer

einer Textdatei besteht. Grafiken werden in getrennten Dateien abgespeichert. Welche Formate durch den erweiterten Transformierer verarbeitet werden können, hängt davon ab, welche Filter enthalten sind. Da in der Regel diese Filter fest in das komplette Textverarbeitungssystem eingebaut sind, gibt es auch keine Möglichkeit dieses Repertoire zu erweitern.

SGML bildet mit diesen Mechanismen einen sehr allgemein gehaltenen Standard. Die Entwicklung von DTDs und Layoutvereinbarungen (SGML-Applikation) ist aufwendig. Das Problem besteht meistens darin, daß eine DTD mit einer gewissen Strenge die Struktur festlegt, die ein entsprechendes Dokument besitzen soll. Auf der anderen Seite sollte sie aber auch nicht zu restriktiv sein, damit nicht irgendwelche Fälle ausgeschlossen werden, die im weiteren Nutzen der DTD gewünscht werden können. Denn sollte man sich durch die DTD zu sehr eingeschränkt haben, muß bei geänderten Anforderungen die DTD geändert werden.

3.3 HTML (HyperText Markup language)

3.3.1 Einsatzgebiet von HTML

HTML ist die bekannteste der hier vorgestellten Markupsprachen. Eine erste Version entstand im Jahre 1992. Während des Erstellens der Diplomarbeit ist die Version 4.0 aktuell. Der Standard wird durch das WWW-Konsortium überwacht, wobei es zur Zeit durch wettbewerbsbedingte Gegebenheiten zwei Hersteller gibt, die zum Teil auch proprietäre Lösungen in den Standard einbringen. HTML hat eine intensive Anwendung gefunden, da die im Internet dargestellte Seiten fast ausschließlich in HTML formuliert sind.

3.3.2 HTML technisch gesehen

HTML ist keine Metasprache wie SGML. Vielmehr ist die Sprache HTML eine SGML-Applikation. Dazu gehört eine genormte DTD, nämlich die HTML-DTD. Durch die Festlegung auf eine DTD ist auch klar, daß es ein festes Tag-Repertoire gibt, das nicht erweitert werden kann. Wenn alle Tags bekannt sind, dann kann auch festgelegt werden, wie die einzelnen Tags darzustellen sind. Bei HTML-verarbeitenden Anwendungen wie Web-Browser oder HTML-fähigen Textverarbeitungen sind DTD und Layoutvereinbarung nicht mehr sichtbar, da sie fest in die entsprechende Anwendung eingebaut sind. Ursprünglich sollte HTML zum Strukturmarkup genutzt werden. Mittlerweile steht allerdings das Layoutmarkup im Vordergrund.

Bei der HTML-Spezifikation handelt es sich nicht nur um eine reine Layoutspezifikation. Bei den im Internet beschriebenen Seiten handelt es sich meistens um HTML-Dokumente[1]. Auf diesen Seiten gibt es Elemente, die nicht nur passiven Charakter besitzen. Dazu gehören beispielsweise die folgenden Elemente:

1. Hier hat sich der Begriff HTML-Dokument eingebürgert, obwohl man von einem SGML-Dokument sprechen könnte.

- Hyperlinks
- Interactive Maps (Fläche mit Hyperlinkcharakter)
- Forms (Felder, in die Text eingetippt werden kann)

Daß durch die Interaktion mit dem Element Prozesse angestoßen werden können soll am Beispiel eines Hyperlinks gezeigt werden. Ein Hyperlink wird in HTML wie folgt erzeugt:

```
<a href=“http:\\www.sap.com\“>
```

Durch das Attribut „href“ des Ankerelements, ist vereinbart, daß der Wert des Attributs als Text dargestellt wird. Dieser Text ist anklickbar. Wenn dieser Text angeklickt wird, werden die durch die angegebenen URL[1] identifizierten Daten geladen. Außerdem ist an der Farbe des Textes (rot oder blau) zu erkennen, daß der sich der Abwickler merkt, ob man diese Seite schon besucht habt. Diese Vereinbarung ist nicht in der DTD festgelegt, sondern in [W3C_98].

Die neueste Version von HTML bietet auch die Möglichkeit über sogenannte Cascading Style Sheets (CSS) Layoutdefinitionen auszutauschen. Diese Nachbesserungen sind aber typisch für HTML, denn der ganze Standard wurde immer dann weiterentwickelt, wenn zum einen neue Features benötigt wurden oder eine neue Version eines Web-Browsers auf den Markt kam.

3.4 XML (eXtensible Markup Language)

3.4.1 Einsatzgebiet von XML

XML ist wegen Kritikpunkten an HTML und SGML entstanden. Bei HTML stört vor allem das fest vorgegebene Tag-Repertoire. An dieser Stelle wünscht man sich mehr Flexibilität. SGML bietet die Möglichkeit neue Tags zu definieren. Allerdings ist durch die umfangreiche Spezifikation von SGML ein sehr großer Funktionsumfang abzudecken, der in den meisten Fällen überhaupt nicht gebraucht wird. Dieser Kritikpunkt stammt hauptsächlich aus Bereichen der Softwareindustrie, die SGML-Werkzeuge anbieten. Außerdem soll das Veröffentlichen von Dokumenten im Internet mit XML einfacher sein als mit SGML, da in der XML-Spezifikationen beispielsweise Sprachkonstrukte, die Links beschreiben, vorhanden sind. Der SGML-Standard hat für Links keine Norm.

Durch diese Kritikpunkte wurde ein Gremium in der WWW-Organisation ins Leben gerufen, die sich seit Mitte 1997 mit dem Erstellen eines neuen Standard, nämlich XML, beschäftigen. Ende Februar 1998 gab es dann die erste Version der Spezifikation. Da der Standard noch rela-

1. URL: Unified Resource Locater. Dabei handelt es sich um eine Adresse im Internet.

tiv neu ist, ist auch die Anzahl der Werkzeuge, die XML können, noch nicht sehr groß. Die technischen Betrachtungen im nächsten Kapitel zeigen aber, daß SGML-Werkzeuge ohne Probleme XML verarbeiten können.

3.4.2 XML technisch gesehen

XML ist wieder eine Metasprache. Bei der Normung hat man sich stark an SGML orientiert. Letztendlich ist XML eine Teilmenge von SGML. Es gibt die Möglichkeit DTDs zu erstellen. Wobei auch hier das Problem auftaucht, wie man die potentiell unendlich vielen Tags darstellt. Zur Layoutfestlegung befindet sich ein DSSSL-ähnlicher Standard in der Normungsphase. Dieser neue Standard heißt XSL[1].

Im Vergleich mit SGML sind einige Sprachkonstrukte nicht mehr zugelassen. Im Gegensatz zu SGML müssen alle Elemente durch ein Anfangs- und End-Tag gekennzeichnet sein. Damit wird natürlich die Entwicklung eines Parsers für XML-Dokumente stark vereinfacht. Durch die Erfahrungen mit Linking-Techniken aus HTML wurde an dieser Stelle schon ein Teil von Schlüsselwörtern festgelegt, die bestimmte Typen von Links normen. Genauere Informationen über XML findet man auf den Internet-Seite des WWW-Konsortiums oder in [Light_97].

3.5 Vergleich der Sprachen SGML, HTML und XML

Durch die große Verbreitung im Internet ist HTML der weitverbreitetste Standard. Da immer neue Anforderungen an die Technologien im Internet gestellt werden, hat sich herausgestellt, daß der HTML-Standard auf Dauer zu starke Einschränkungen bietet. Allerdings sind es nicht nur Anforderungen, die sich auf die Anwendung im Internet beziehen. Der Wille zu einem gemeinsamen Format, daß zwischen beliebigen Werkzeugen ausgetauscht werden kann, trägt auch dazu bei, daß nach einem geeigneten Standard gesucht wird. Denn zur Zeit benutzt z.B. jeder Textverarbeitungsherrsteller sein eigenes Format, wobei eine Konvertierung von einem Format in ein anderes selten ohne Problem funktioniert. Zwar gibt es Standards wie Postscript[2], die aber nur zum Anzeigen oder Ausdrucken des Dokuments geeignet sind. Eine Weiterverarbeitung eines Dokuments im Postscript-Format ist auch wieder an die Werkzeuge eines einzigen Herstellers gebunden. Außerdem bezieht sich das Beispiel Postscript hauptsächlich auf Layoutbedürfnisse.

Eine weitergehende Vision ist, daß ein Dokument von unterschiedlichen Werkzeugen genutzt werden kann. Eine technische Zeichnung sollte sowohl von einem CAD-Werkzeug bearbeitet werden können, wie auch von einer Lagerverwaltungssoftware, die Informationen aus der durch Markup gekennzeichneten Liste verarbeiten kann.

1. XSL: eXtensible Style Sheet
2. Postscript: Genormte prozedurale Seitenbeschreibsprache der Firma Adobe.

SGML bietet alle diese Möglichkeiten. Allerdings steht der Implementierungsaufwand für die SGML-Fähigkeit des Systems in keinen Verhältnis zum eigentlichen Knowhow des entsprechenden Softwarehauses. Daher wurde XML geschaffen.

Mit der Frage, in wie weit man ein Dokument aus einem Standard in ein Dokument eines anderen Standards konvertieren kann, ist auch eine weitere Frage verknüpft. Man muß sich fragen, welche Anforderungen an dieses Dokument nach der Konvertierung gestellt werden. Außerdem gibt es bei SGML und XML nur Dokumente, die innerhalb einer Applikation erstellt worden sind. Die Frage nach der Konvertierungfähigkeit heißt daher, können Dokumente die innerhalb einer SGML- oder XML-Applikation in eine andere Applikation konvertiert werden?

SGML nach XML

Wenn man sich in der SGML-Applikation auf die Sprachkonstrukte beschränkt hat, die auch XML nutzt, ist eine Konvertierung kein Problem.

SGML nach HTML

HTML ist eine SGML-Applikation. Trotzdem kommt der Fall vor, daß eine SGML-Applikation, die nicht HTML ist, in HTML gewandelt werden soll. Dies ist genau dann der Fall, wenn HTML als Ausgabeformat genutzt werden soll. Da in HTML mit jedem Tag ein bestimmtes Layout verbunden ist, werden die Tags aus der Quell-SGML-Applikation nach Layoutgesichtspunkten angepaßt.

XML nach SGML

Da XML eine Teilmenge von SGML ist, ist eine Konvertierung ohne weiteres möglich.

XML nach HTML

Bei der Konvertierung von XML nach HTML wird HTML ebenfalls als Ausgabeformat genutzt. Es gelten die gleichen Aussagen wie bei der Wandlung von SGML nach HTML.

HTML nach SGML

HTML ist eine SGML-Applikation. Wenn HTML in eine andere SGML-Applikation gewandelt werden soll, hängt es von der Grammatik der Ziel-SGML-Applikation. Normaler Weise sollte eine Wandlung aber kein Problem sein.

HTML nach XML

Diese Wandlung kann Probleme bereiten, da HTML mittels Sprachkonstrukten von SGML beschrieben wird, die in XML nicht vorhanden sind. Ansonsten gelten die gleichen Aussagen, die bei der Wandlung von HTML nach SGML gemacht wurden.

3.6 Einsatz im TimeLess-Projekt

In Bild 5 wurde gezeigt, welche Informationen in einem strukturierten Dokumente abgespeichert werden. Mit XML oder SGML können Gerüst des Dokuments, Dummybelegung und Inhalt, sofern es sich um Text handelt, unterstützt werden. Wenn ein Baustein in einem Format vorliegt, das dem Viewer nicht bekannt ist, gibt es aber auch keine Möglichkeit, diesen Baustein darzustellen. Dieses Problem wird nicht durch eine der Markupsprachen gelöst. Dazu müssen weitere Technologien genutzt werden, wie z.B. OLE[1].

HTML ist als Viewing-Format geeignet, da es ein weitverbreiteter Standard ist. Da der Zugriff auf TimeLess über einen Web-Browser geschehen soll, ist damit auch sichergestellt, daß ein Viewer auf dem Rechner des TimeLess-Nutzers installiert ist.

1. OLE: Object Linking & Embedding. Technologie von Microsoft

4. Astoria-Überblick

In diesem Kapitel soll der Aufbau des Systems gezeigt werden. Damit ist die Lokalisierung der Programmierschnittstelle möglich. Der letzte Teil dieses Kapitels gibt einen Überblick über die Programmierschnittstelle und darin immer wieder kehrende Prinzipien.

4.1 Der Aufbau von Astoria

Astoria ist eine Client-Server-Applikation. Auf der Serverseite werden Dokumente gelagert. Neben den Dokumenten sind auf dem Server Verwaltungsdaten wie Benutzer und deren Rechtslage abgelegt. Mit der Applikation auf der Clientseite erfolgt der Zugriff auf die Dokumente. Diese Applikation dient zur Interaktion mit dem Benutzer (Bild 11).

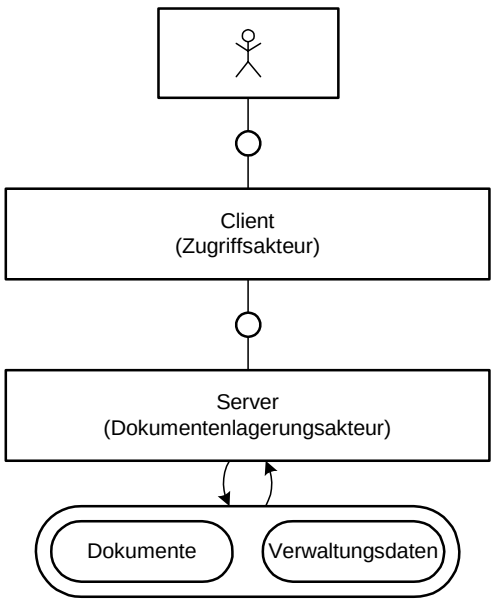


Bild 11: Client und Server bei Astoria

Die Daten des Systems werden auf der Serverseite in einem objektorientierten Datenbanksystem abgelegt. Von diesem Datenbanksystem werden an der Zugriffsschnittstelle keine Begriffe angeboten, die sich auf Dokumente beziehen. Daher muß es eine Anpassungsschicht innerhalb des Systems geben, die diese Begriffsanpassung leistet. In Astoria wird diese Anpassungsschicht als *Document-Management-Engine (DME)* bezeichnet. Die Document-Management-Engine gefindet sich in Bild 12 komplett auf der Clientseite. Ob es auch einen Teil dieser Anpassungsschicht auf dem Server gibt, war nicht festzustellen. Da sich die API[1] auf der Clientseite befindet, ist für die Konzeptbetrachtungen nicht von Interesse, wie diese Schicht reali-

1. API: Application Programming Interface

siert ist. Ein Programmierer, der Programme schreibt, die auf der API aufsetzen, erlebt die Trennung von Client und Server nämlich nicht. Er sieht vom kompletten System nur die Programmierschnittstelle.

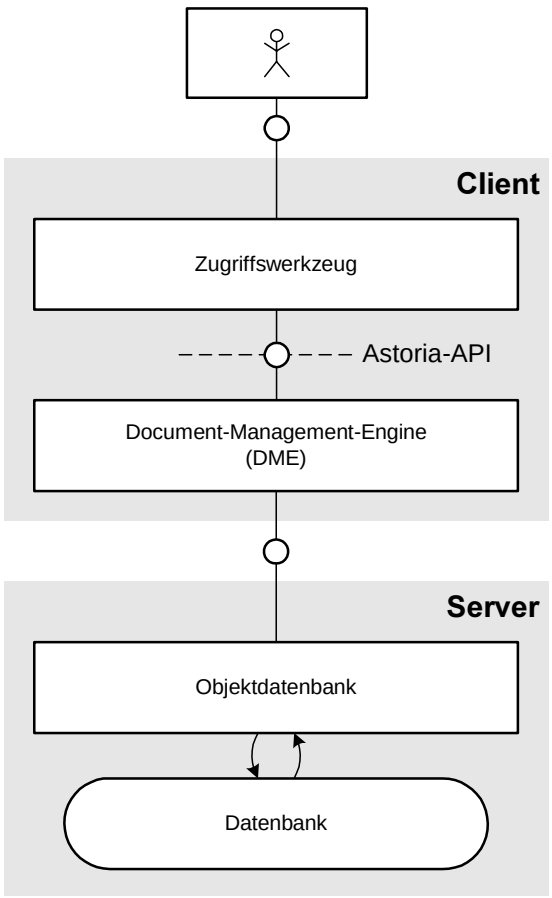


Bild 12: Realisierung von Client und Server

Ein Editor, der auf dem Clientrechner installiert ist, kennt die Ablage von Astoria nicht. Dieser Editor kann nur Dokumente bearbeiten, die in Form einer Datei im Dateisystem des Clientrechners abgelegt sind. Ein Dokument, das in Astoria abgelegt ist, muß daher zum Editieren aus der Datenbank exportiert werden und im Dateisystem des Clients gespeichert werden. Dabei können zwei Fälle unterschieden werden:

(a) Der Benutzer erhält eine Kopie des Dokuments. Damit ist das Originaldokument immer noch in der Datenbank vorhanden. Wenn die Kopie des Dokuments wieder in Astoria abgelegt wird, dann handelt es sich dabei um ein anderes Dokument, das mit dem zuvor „exportierten“ nichts mehr zu tun hat.

(b) Der Benutzer erhält das Original des Dokuments. Auf der Serverseite wird dann vermerkt, daß kein anderer Benutzer dieses Dokument ändern darf. Der Export einer Kopie ist trotzdem für andere Benutzer erlaubt. Der Export des Originals wird auch als *Checkout* bezeichnet. Wenn der Benutzer, der das Original geändert hat, dieses System wieder ins System einbringt, dann spricht man vom *Checkin*. Während des

Editierens darf der Benutzer den Dateinamen und den Ablageort im Dateisystem nicht ändern, da Astoria sonst keine Identifizierungsmöglichkeit für dieses Dokument besitzt.

Wenn der Editor ein Dokument abspeichert, dann speichert er dieses im lokalen Dateisystem. Damit der Benutzer nicht einen zweiten Schritt machen muß, um das Dokument nach dem Abspeichern im Dateisystem in Astoria abzulegen, gibt es von Chrystal Sofware ein Produkt namens *Bridges to Astoria*. Dieses Produkt ist für unterschiedliche Editoren erhältlich, die eine Programmierschnittstelle besitzen. Bridges to Astoria setzt sowohl auf der Programmierschnittstelle von Astoria wie auch auf der Programmierschnittstelle des Editors auf (Bild 13). Im Editor kann der Benutzer dann direkt Zugriff auf Astoria erlangen, ohne daß er über ein zweites Werkzeug auf Astoria zugreifen muß, da statt einer Fileselectbox, die auf das Dateisystem zugreift, eine vergleichbare Sicht auf Astoria erscheint.

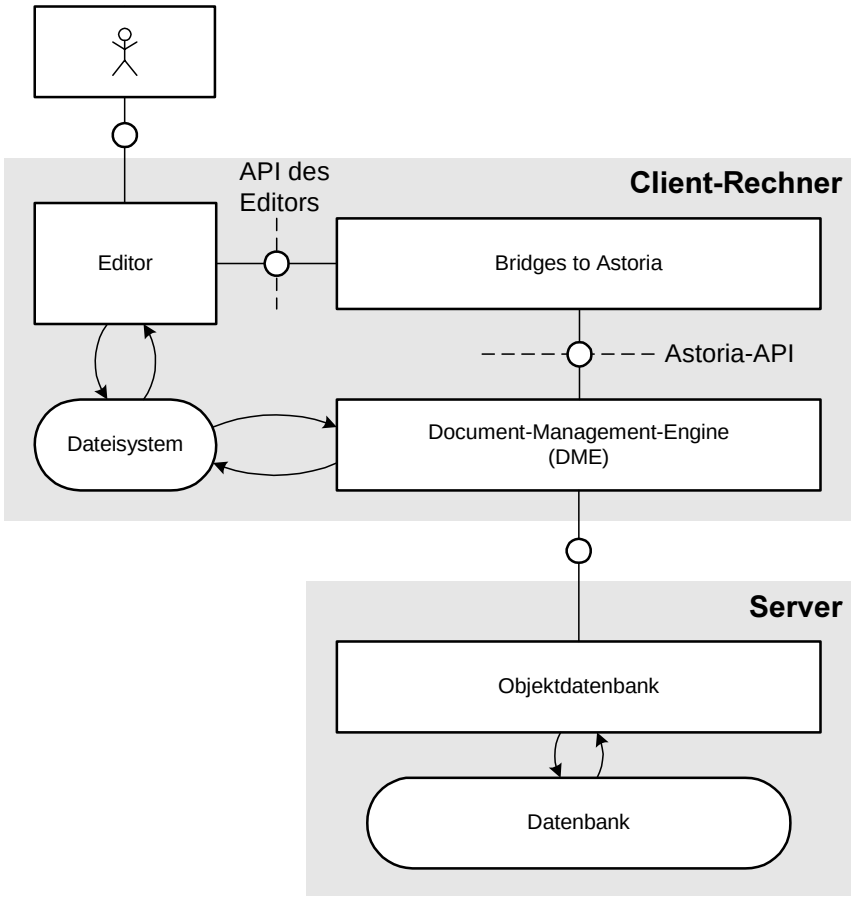


Bild 13: Bridges to Astoria

Im Lieferumfang von Astoria befindet sich auch ein Navigationstools, das als *Navigator* bezeichnet wird. Dieses Produkt benutzt ebenfalls die vorgebene Programmierschnittstelle. Dies äußert sich darin, daß Verwaltungseinheiten und Operation bei API und Zugriffstools in den meisten Fällen identisch sind.

Bisher wurde nur angesprochen, daß Astoria Dateien identifizieren kann. Nämlich genau dann, wenn es darum geht, diese zu importieren. Es gibt aber auch die umgekehrte Möglichkeit, daß Objekte in der Astoria-Datenbank durch Dateien identifiziert werden können. Astoria kann eine Datei erzeugen, die dann als *externe Referenz* bezeichnet wird. Der Inhalt einer solchen Datei stell eine Referenz auf ein Objekt in der Datenbank dar (Bild 14).

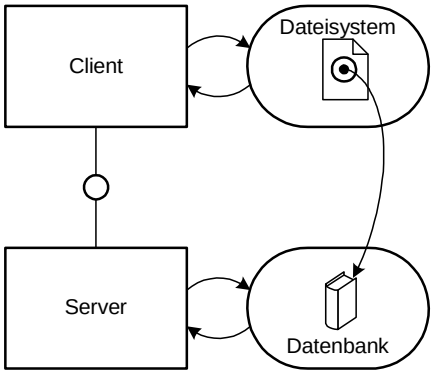


Bild 14: Externe Referenzen

4.2 Aufbau der API

Die API wird durch eine Klassenbibliothek realisiert, wobei die zu Grunde liegende Programmiersprache C++ ist. Diese Programmierschnittstelle ist in [Astoria_SDK_98] derart beschrieben, daß nach einer Einleitung, die den Umgang mit der Entwicklungsumgebung und Grundprinzipien der API erklärt, eine Aufzählung aller vorkommenden Klassen erfolgt. Diese Arbeit stellt ebenfalls eine Erklärung der Grundprinzipien des Systems dar. Die Darstellung der Konzepte unterscheidet sich allerdings von der in [Astoria_SDK_98]. Außerdem werden Konzepte herausgestellt, die erst durch das Studium der Klassenbeschreibung und Benutzerhandbüchern wie [Astoria_Help_98] gefunden wurden, die aber nicht in der Einleitung erwähnt werden.

4.2.1 Aufbau der Klassenbibliothek

Da die Klassenbibliothek aus einer Vielzahl von Klassen besteht, soll in diesem Kapitel alle Klassen in drei Kategorien eingeordnet werden:

(a) Klassen für Dienstakteure, die zum Betrieb des Systems notwendig sind. Dazu gehört z.B. ein Transaktionsdienstakteur.

(b) Klassen, die Verwaltungseinheiten aus der Dokumentenwelt beschreiben. Diese beschreiben beispielsweise Dokumente oder dienen zum Aufbau der Struktur in der Ablage.

(c) Klassen, die Funktionalität nutzen, die in den Klassen (a) und (b) beschrieben worden ist.

Alle Aufrufe der API sind einschrittig. D.h. wenn eine Methode aufgerufen wird, dann wurde entweder die Operation, die in der Methode beschrieben wird, erfolgreich ausgeführt, oder die Operation wurde nicht ausgeführt. Manchmal kann es vorkommen, daß eine Operation aus Benutzersicht einschrittig ist, von der Realisierung aber mehrschrittig ist. Der Programmier muß deshalb mehrere Anweisungen über die API an das Astoria-System geben. Dazu gibt es die Möglichkeit eine Transaktionsklammer um alle Operationsschritte zu setzen. Vor dem ersten Operationsschritt wird dem Transaktionsdienstakteur mitgeteilt, daß nun eine mehrschrittige Operation beginnt. Nach dem letzten Operationsschritt wird das Ende der mehrschrittigen Operation mitgeteilt. Wenn einer der Operationsschritte nicht ausführbar ist, kann man dem Transatkionsdienstakteur mitteilen, daß die gesamte Operation fehlgeschlagen ist. Durch diese Anweisung werden alle Operationsschritte wieder rückgängig gemacht.

Weitere Dienste stellt der Sitzungsverwalter[1] zur Verfügung. Mittels des Sitzungsverwalters meldet man sich beim Server an und ab. Außerdem liefert dieser Einstiegspunkte in die Ablagestruktur. Dieser Akteur stellt praktisch die Brücke zwischen der Dokumentenwelt und der Realisierung dar, da sowohl die Begriffe wie Ablageort der Datenbank im Dateisystem als auch Benutzer oder Wurzelcontainer der Ablage abgefragt werden können.

Zu den Klassen, die zu Punkt (a) gehören, befinden sich auch Klassen die Import- und Export-Dienste für Dokumente beschreiben. Diese werden in Kapitel 5.4.4.2 ausführlich besprochen.

In den Klassen, die unter Punkt (b) verstanden werden, werden Begriffe beschrieben, die auch der Bentutzer erlebt. Diesen Klassen ist Kapitel 5 gewidmet.

Die Klassen unter Punkt (c) sind zum Betrieb des Systems nicht unbedingt notwendig. Objekte, die durch einen Teil dieser Klassen beschrieben wird, übernehmen die Kommunikation zum mitgelieferten Navigationswerkzeug. Damit können beispielsweise eigene Menüs in das Navigationswerkzeug eingebracht werden. Eine andere Klasse beschreibt Objekte, die Dialogfenster bereitstellen. Diese Dialogfenster dienen zur Abfrage von Parametern, die beim Aufruf einer Methode angegeben werden müssen. Wenn man beispielsweise ein Dokumente ins System einbringen will, dann muß man dem System mitteilen an welcher Stelle der Ablage dieses Dokument abgelegt werden soll. Wenn die entsprechende Methode eines solchen Objekts aufgerufen wird, wird automatisch ein Dialogfenster geöffnet, das Felder für alle benötigten Daten besitzt.

Die Klassen, die unter Punkt (c) zusammengefaßt worden sind, werden nicht weiter besprochen, da weder das mitgelieferte Navigationswerkzeug benutzt wird, noch die Dialogfenster benötigt werden.

1. Der Sitzungsverwalter wird durch die Klasse XD_Database beschrieben.

Bild 15 zeigt die erwähnten Schnittstellen. Der dunkel graue Bereich zeigt alle Schnittstellen, die zum Betrieb des Systems notwendig sind. Daher wurden diese Schnittstellen auch als Kern-API bezeichnet. Die Schnittstelle zu den Objekten in der Datenbank soll symbolisieren, daß die C++-Applikation zu jedem Objekt in der Datenbank einen Kanal hat. Es geht in dieser Darstellung um die Existenz der Schnittstelle, die die Document-Management-Engine besitzt.

Als Beispiel für ein Objekt, das durch die Klassen, die in Unterpunkt (c) klassifiziert werden, wird in Bild 15 ein Dialogerzeuger gezeigt. Dieser hat einen Speicher für die Parameter, die der Benutzer übergibt. Durch die C++-Applikation wird das Aufbauen eines Dialogfensters angestoßen. Der Dialogerzeuger baut dann das entsprechende Dialogfenster auf. Die eingegebenen Parameter werden beim Aufruf einer Methode der Kern-API übergeben. Nach dem Methodenaufruf hat die C++-Applikation wieder den Abwickler.

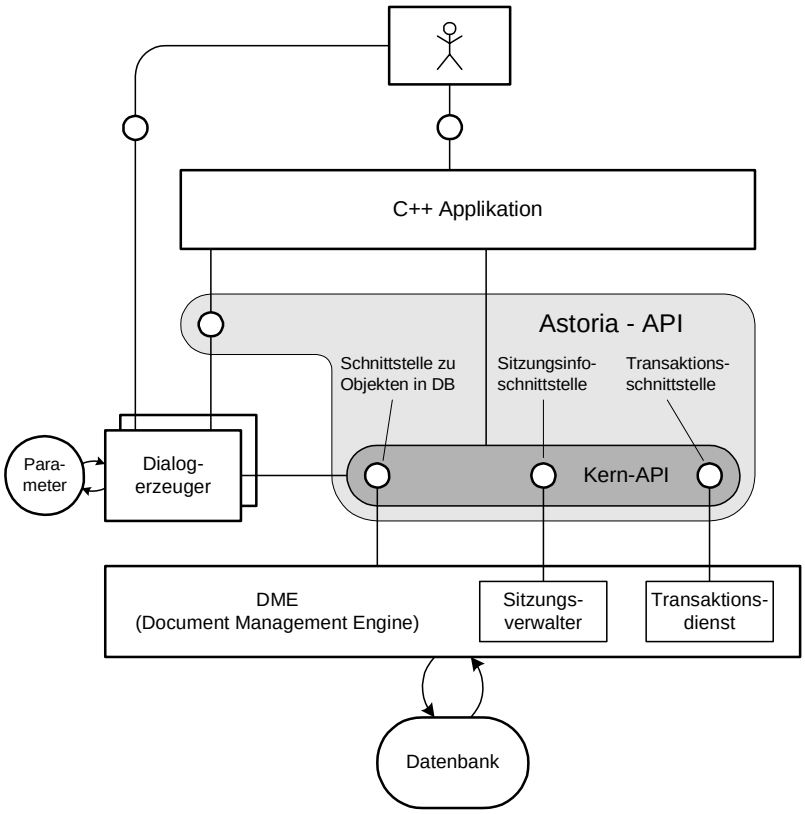


Bild 15: Schnittstellen der API

4.2.2 Der Klassenbaum

Zunächst soll eine Einführung in einige Prinzipien der objektorientierten Programmierung gegeben werden. Danach wird auf diese Prinzipien im Astoria-Klassenbaum eingegangen.

4.2.2.1 Kurze Einführung in die Objektorientierung

Unter einer Klasse versteht man in der Objektorientierung eine Art Bauplan für Objekte. In der Klasse ist beschrieben welche Speicher ein Objekt besitzt. Diese Speicher werden auch als *Attribute* bezeichnet. Außerdem wird ein Akteursanteil des Objekts beschrieben. Zum einen wird beschrieben, welche Aktionen dieser Akteur ausführen kann, zum anderen wird der Prozeß selber beschrieben, der beim Ausführen einer Aktion gestartet werden soll. Was hier mit Aktion bezeichnet wird, ist als Prozedur realisiert. In der Objektorientierung werden diese Prozeduren als *Methoden* bezeichnet.

Ein Attribut kann innerhalb eines Objekts als globale Variable gesehen werden. Alle Methoden, die zu diesem Objekt gehören, können auf dieses Attribut zugreifen. Ein anderes Objekt soll den Wert eines Attributs nur über Methodenaufruf erfahren können. Diese Aussagen sollen an einem Beispiel verdeutlicht werden:

Betrachtet wird eine Klasse namens Zähler. Ein Objekt der Zählerklasse soll einen Speicher für den Zählerstand besitzen (Bild 16). Dazu wird als Wertebereich des Speichers der Bereich der natürlichen Zahlen angegeben. Außerdem sollen folgende Aktionen ausgeführt werden können: „Erhöhen des Zähler um 1“, „Setzen des Zählers auf 0“ und „Nennen des aktuellen Zählerstands“. Nun soll es ein Zählerobjekt A geben. Wenn ein anderes Objekt B dieses Zählerobjekt A kennt, kann diese Anweisungen geben. Das Anweisungsrepertoire ist auf die drei Anweisungen „Erhöhe den Zähler um 1“, „Setze den Zähler auf 0“ und „Nenne mir den aktuellen Zählerstand“ beschränkt. Objekt B kann den Zählerstand nicht direkt ändern, in dem es zu Objekt A sagt, „Zählerstand des Objekts A ist jetzt 23“. Eine Anweisung dieses Objekt kann nur durch den Auftragsverteiler angenommen werden. Dieser gibt den Auftrag an den entsprechenden Akteur weiter.

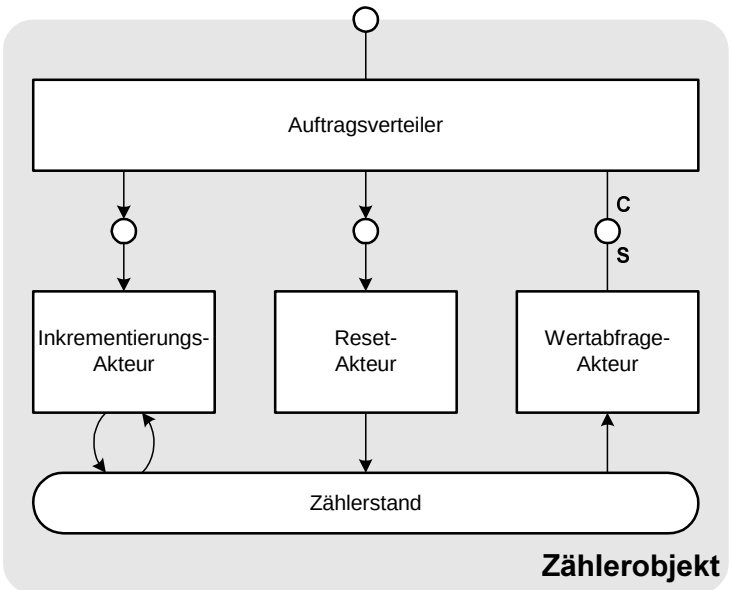


Bild 16: Zählerobjekt

Im obigen Beispiel wurden Beziehungen gezeigt, die zwischen Objekten bestehen. Objekt B kennt Objekt A. Es gibt aber nicht nur Beziehungen zwischen Objekten sondern auch zwischen Klassen. Hier soll lediglich die Vererbungsbeziehung betrachtet werden. In dieser Beziehung treten Klassen in zwei Rollen auf. Die vererbende Klasse wird als *Oberklasse* bezeichnet und die erbende Klasse als *Unterklasse*. Unter *Vererbung* wird verstanden, daß die Unterklasse alle zur Vererbung freigegebenen Attribute und Methoden der Oberklasse besitzt. Eine Unterklasse der obigen Zählerklasse besitzt die Beschreibung des Speichers für den Zählerstand und die Beschreibungen, wie die drei Aktionen auszuführen sind. In dieser Unterklasse kann z.B. eine weitere Methode zum Erniedrigen des Zählers enthalten sein (Bild 17). Damit hat ein Objekt der Unterklasse eine größere Funktionalität als ein Objekt der Oberklasse. Diese Funktionserweiterung wird auch als *Aggregation* bezeichnet.

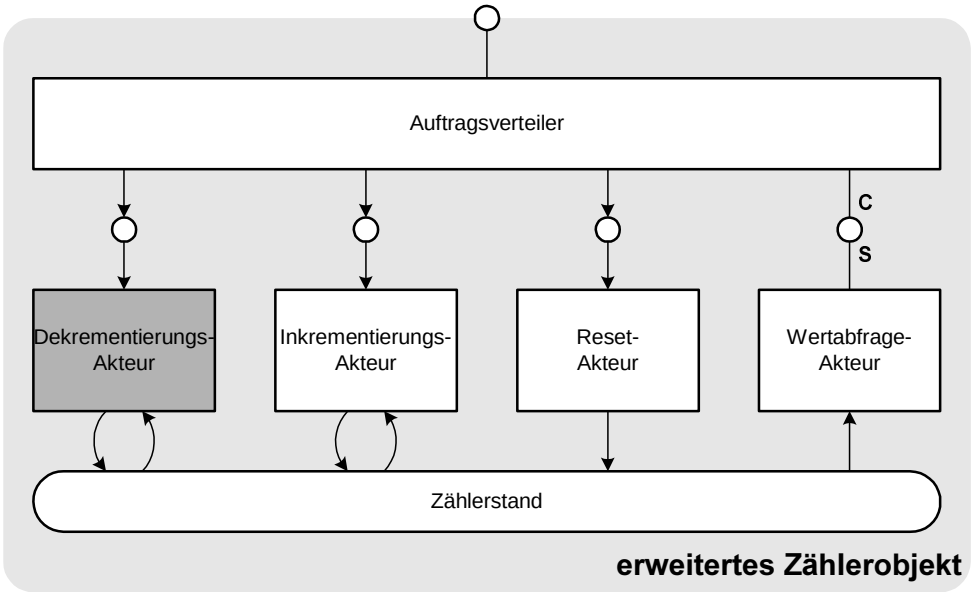


Bild 17: erweitertes Zählerobjekt

Je nachdem unter welchem Aspekt man ein Objekte betrachtet, ist es zweckmäßig das Objekt als Akteur darzustellen oder das Objekt als Speicher. Die Darstellung in Bild 16 und 17 wurde daher so gewählt, daß der Rahmen nicht eckig, aber auch nicht rund gezeichnet wurde.

Die bisherigen Beispiele sind sehr nahe an der Implementierung orientiert. Dagegen ist in der Astoria-Klassenbibliothek die Implementierung nicht zu sehen. Aufgezählt werden lediglich die Methoden, die eine Klasse besitzt. Die vorkommenden Speicher werden durch Methoden belegt und ausgelesen. Ein deutliches Zeichen für die Existenz eines internen Speichers sind Methoden die mit „set“ oder „get“ beginnen. Wenn es beispielsweise eine get_color-Methode gibt, dann hat dieses Objekt auch einen Speicher namens „color“.

Der Wertebereich dieses Speichers wird den Typ der Übergabeparameters oder des Rückgabewerts festgelegt. Nun kann es drei Arten von Wertebereichen geben:

(a) Der Wertebereich wird durch einen Standardtyp der verwendeten Programmiersprache festgelegt.

(b) Der Wertebereich ist ein Aufzählungstyp. Dieser Aufzählungstyp wird durch die Klassenbibliothek zur Verfügung gestellt.

(c) Der Wertebereich ist vom Typ Referenz auf Objekte einer bestimmten Klasse. Die zulässigen Klassen werden durch die Klassenbibliothek zur Verfügung gestellt.

Wenn man nun einen Aspekt des Systems betrachtet, sind immer mehrere Objekte unterschiedlicher Klassen beteiligt. Zur Erklärung eines solchen Aspektes sind Speicher vom Wertebereich (c) von Interesse. Durch diese Speicher werden nämlich die Relationen zwischen den einzelnen Objekte aufgebaut. Dies soll einem ausführlichen Beispiel erklärt werden.

4.2.2.2 Beispielberechtigungswesen

Ähnliche Konzepte wie in diesem Beispiel werden auch im Astoria-Berechtigungswesen verwendet. Dieses Beispiel ist allerdings stark vereinfacht.

In der Beschreibung zur Astoria-Klassenbibliothek ([Astoria_SDK_98]) werden zwei Beschreibungen angeboten. Die erste ist der Klassenbaum und als zweites werden die einzelnen Klassen beschrieben. Für das Berechtigungsbeispiel werden diese Beschreibungen in vereinfachter Form übernommen.

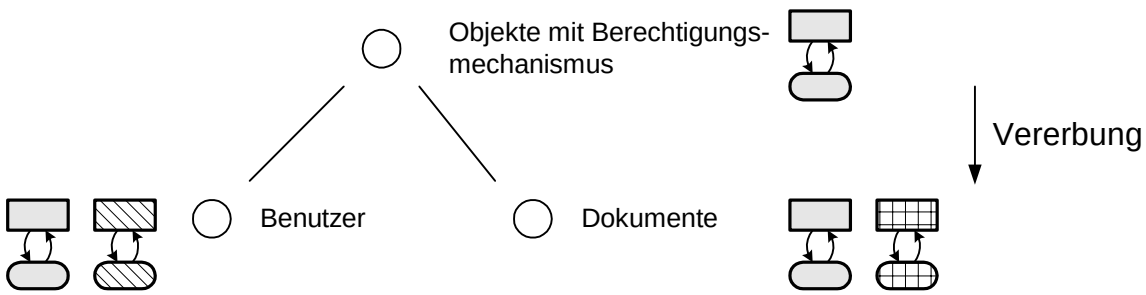


Bild 18: Klassenbaum des Berechtigungsbeispiels

Bild 18 zeigt drei Klassen, wobei die Klasse der Objekte mit Berechtigungsmechanismus die Oberklasse der Klassen Benutzer und Dokumente sind. Die Vererbung ist dadurch angedeutet, daß die Objekte der Unterklassen jeweils den grauen Akteur und den grauen Speicher besitzen, die auch in der Oberklasse vorhanden sind.

Eine Beschreibung der einzelnen Klassen könnte wie folgt aussehen:

Klasse: Benutzer

Methoden: Namenändern, Paßwortändern, Paßwortauslesen,
Aufzählen aller schreiberlaubter Objekte,
Aufzählen aller leseerlaubter Objekte

Da bei dieser Klassenbeschreibung keine Attribute angegeben sind, muß man aus den Methodennamen auf die Attribute schließen.

Gedachte Attribute:
Namen, Paßwort
Liste aller schreiberlaubter Objekte,
Liste aller leseerlaubter Objekte

Die Methoden werden Bild 18 durch den schraffierten Akteur symbolisiert. Die Attribute werden durch den schraffierten Speicher dargestellt.

Klasse: Dokumente

Methoden: Lesen, Schreiben[1]

Gedachte Attribute:
Inhalt des Dokuments

Klasse: Objekte mit Berechtigungsmechanismus

Methoden: Auflisten aller schreiberlaubter Benutzer,
Auflisten aller leseerlaubter Benutzer,
aktuellem Benutzer Schreiberlaubnis geben,
aktuellem Benutzer Schreiberlaubnis entziehen,
aktuellem Benutzer Leseerlaubnis geben,
aktuellem Benutzer Leseerlaubnis entziehen,
Prüfen, ob der aktuelle Benutzer Schreiberlaubnis hat,
Prüfen, ob der aktuelle Benutzer Leseerlaubnis hat

Gedachte Attribute:
Liste aller schreiberlaubten Benutzer,
Liste aller leseerlaubten Benutzer

Dies sind alle Informationen, die auf der Ebene der Programmiersprache gegeben sind. Die Erklärung des Berechtigungswesen ist mit diesen Informationen nicht vollständig möglich. Auch lassen sich die gegebenen Informationen wesentlich kompakter darstellen.

1. Die Bedeutung von „lesen“ und „schreiben“ soll nicht im Mittelpunkt stehen. Es geht hier lediglich darum zu zeigen, daß diese Methoden geschützt werden können. Außerdem wird eine Klasse, die Dokumente repräsentiert mehr als nur zwei Methoden besitzen.

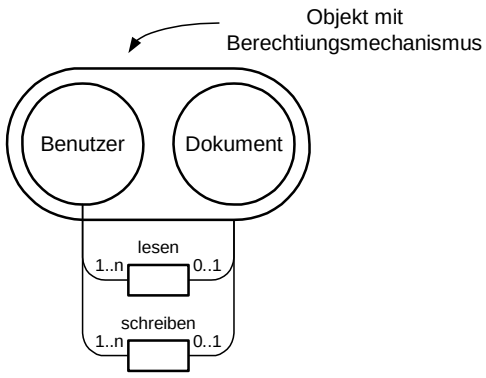


Bild 19: kompakte Darstellung der Relationen zwischen den Objekten, die am Berechtigungswesen teilnehmen

Das Entity-Relationship-Diagramm in Bild 19 zeigt welche Beziehungen durch die Listen der Referenzen gemeint sind. Jeder Benutzer hat je eine Liste von den Objekten, auf die er lesend oder schreibend zugreifen darf. Umgekehrt hat jedes Berechtigungsobjekt eine Liste der Benutzer, lesend oder schreibend auf dieses Objekt zugreifen darf. Durch diesen Mechanismus können Dokumente und Benutzer[1] geschützt werden.

Wie am Ende von Kapitel 4.2.2.1 angekündigt läßt sich lediglich mit den Referenzen die Vergabe von Rechten erklären. Attribute, die keine Referenzen auf andere Objekte darstellen, werden dann zur Erklärung benötigt, wenn man die Semantik des zu schützenden erklären will.

Bislang ist noch nicht geklärt, ob die Prüfung der Berechtigung durch das System gemacht wird oder ob der Programmierer vor jeden Aufruf einer zu schützenden Methode diese Prüfung selbst programmieren muß. Im hier vorliegenden Fall soll das System, die Berechtigungsprüfung machen. Dies ist in Bild 20 so dargestellt, daß ein Auftrag, der über den Interaktionsakteur dem System mitgeteilt wird, nur an den Berechtigungsprüfer geschickt werden kann. Der Berechtigungsprüfer gibt diesen Auftrag nur an den Schreib-/Leseakteur weiter, wenn der aktuelle Benutzer auch in der entsprechenden Lese- oder Schreibliste eingetragen ist.

Durch die unterschiedlichen Füllmuster in Bild 20 soll der Bezug zu Bild 18 dargestellt werden. Der Interkationsakteur stellt die Applikation dar, die ein Programmierer erstellt. Dieser Interkationsakteur kann beispielsweise eine Login-Prozedur enthalten, mit der der Benutzer authentifiziert wird. Um Überprüfen zu können, ob der Benutzer das richtige Paßwort eingegeben hat, kann der Interaktionsakteur die Attribute aller Benutzer lesen. Wenn das Paßwort korrekt war, wird dieser Benutzer als der aktuelle Benutzer gesetzt. Hier muß es natürlich die Möglichkeit geben, eine Liste aller Benutzer abzufragen. Außerdem gibt es in den oben beschriebenen Klas-

1. Lesen und schreiben kann man sich beim Benutzer so vorstellen, daß ein anderer Benutzer das Paßwort oder den Namen abfragen oder ändern darf. Dabei handelt es sich um Administrator-Rechte, die aber über den gleichen Mechanismus geschützt werden.

sen keine Methode, mit der der aktuelle Benutzer gesetzt werden kann. Diese Aufgaben kann man einem in Bild 20 nicht gezeigten Sitzungsverwalter zuordnen. Alle weiteren Methoden sind in den obigen Klassen beschrieben.

Ein Teil der Attribute der Objekte mit Berechtigungsmechanismus-Klasse und der Benutzer-Klasse sind in den Schreib- und Lese-Listen zusammengefaßt. Mit Hilfe dieser Listen kann der Berechtigungsprüfer prüfen, ob ein Auftrag des aktuellen Benutzers zulässig ist. Ob diese Prüfung erfolgt, ist nicht aus der Klassenbeschreibung zu ersehen. Allerdings ist diese Prüffunktionalität genau das, was das Berechtigungswesen ausmacht. Zum Testen solcher Funktionalitäten wurden Testprototypen angefertigt, die aber nicht in dieser Arbeit beschrieben sind.

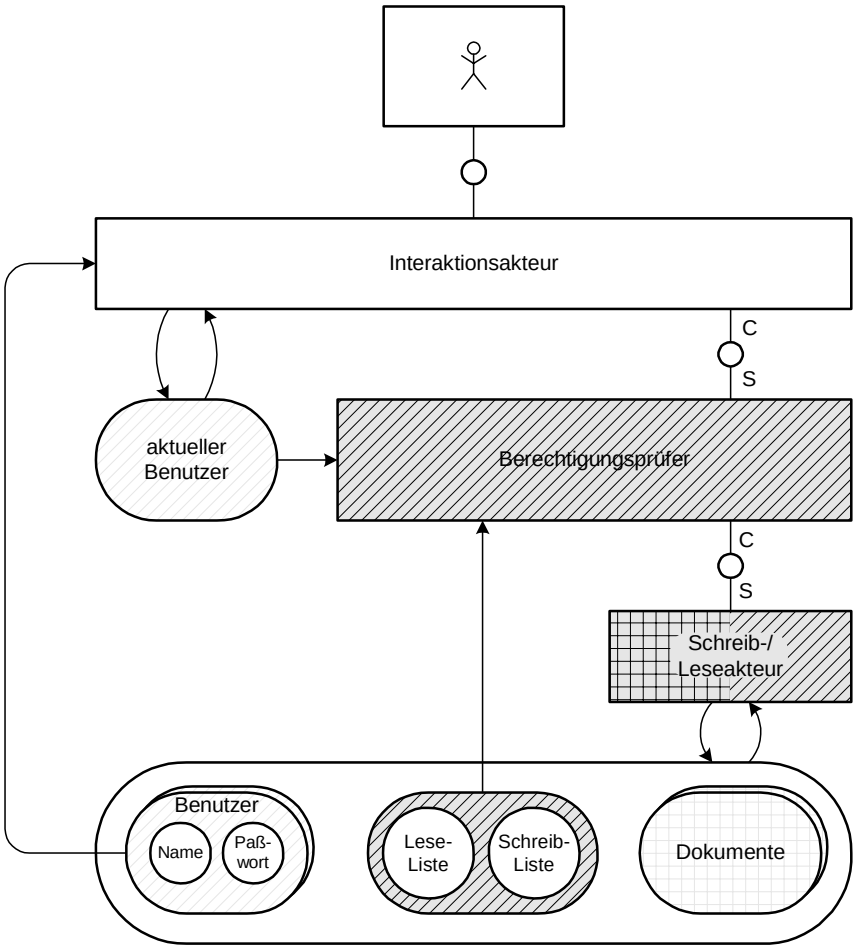


Bild 20: Aufbaubild zum Berechtigungswesen

Da die zu schützenden Methodenaufrufe mit Schreib- und Lesezugriffen sehr allgemein gehalten sind, kann man sich leicht vorstellen, daß jegliche Art von Änderungen an den Benutzerdaten, an der Schreib- oder Leseliste und an den Dokumenten über einen Schreibauftrag geschieht. Deshalb ist der Schreib-/Leseakteur mit allen vorkommenden Füllmustern gefüllt. Wenn vom Benutzer Berechtigungsdaten aus den Schreib- oder Leselisten gefragt werden, sollen diese Anfrage auch über den Schreib-/Leseakteur geschehen.

Mit Sicherheit ist nicht immer eine 1-zu-1-Zuordnung zwischen den Methoden und Akteuren bzw. den Attributen und Speichern gegeben. Allerdings beschreiben die in Bild 19 und 20 gezeigten Modelle die Mechanismen wesentlich deutlicher als die reine Klassenbeschreibung. Die Klassenbeschreibung enthält auch nicht alle Angaben, die zur Erklärung eines Mechanismus notwendig sind. In [Bungert_98] werden daher auch Implementierungsmodelle und Anschauungsmodelle unterschieden. Die Modelle, die in dieser Arbeit angeben werden sind Anschauungsmodelle.

Eine anschauliche Erklärung ist nach Meinung des Autor mittels Aufbaubildern wie in Bild 20 gezeigt am einfachsten möglich, da sie das Denken in Verschaltung wiederspiegeln ([Gales_98]). Leider konnte kein Aufbaubild erarbeitet werden, in dem das ganze System beschrieben wird. Die betrachteten Funktionalitäten von Astoria greifen stark ineinander. Da im oben gezeigten Fall, der nur einen einzigen Funktionalitätsaspekt beschreibt, Akteure und Speicher so zusammen gefaßt werden, daß es keine eindeutige Abbildung auf die Klassenbeschreibung gibt, ist in dieser Darstellung ein gewisses Maß an Wilkür vorhanden. Es soll in dieser Arbeit aber nicht erraten werden, wie das Astoria-System aufgebaut sein könnte, sondern die Prinzipien erklärt werden. Daher werden dort, wo es dem Autor angemessen erscheint Aufbaubilder gezeigt. Dafür werden bei anderen Aspekten Entity-Relationship-Diagramme gezeigt. Damit werden die Beziehungen zwischen den einzelnen Objekten gezeigt, wie dies auch in Bild 19 der Fall war.

4.2.2.3 Mehrfachverbung

Ein Klassenbaum ist streng hierarchisch aufgebaut. Damit kann man Objekte unter einem ganz bestimmten Aspekt klassifizieren. Als Beispiel kann man sich eine biologische Klassifizierung vorstellen, bei der es von einer Klasse Mensch zwei Unterklassen Mann und Frau gibt. Wenn man nun eine Klasse für alle Mitarbeiter einer Firma einführen möchte, dann ergibt sich folgendes Problem. Da in dieser Firma sowohl Männer als auch Frauen beschäftig sind, wird aus diesem Baum ein Netz (Bild 21).

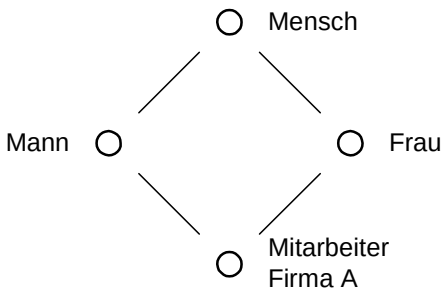


Bild 21: Klassennetz

Wenn man die strenge Hierarchie beibehalten will, dann bleibt nur die Möglichkeit zwei neue Unterklassen einzuführen: Eine Unterklasse der männlichen Mitarbeiter und eine Unterklasse der weiblichen Mitarbeiter (Bild 22).

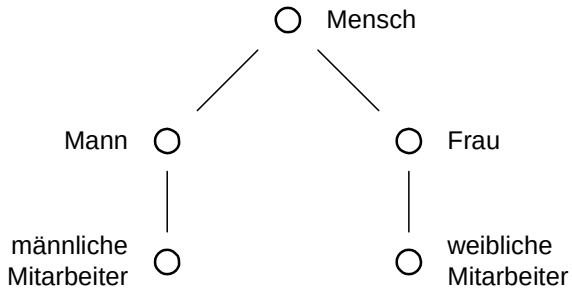


Bild 22: Klassenbaum

Die beiden neuen Unterklassen besitzen beide die gleichen Methoden und Attribute. In einigen objektorientierten Programmiersprachen ist es möglich, daß durch die sogenannte *Mehrfachvererbung* ein Objekt durch mehrere Klassen beschrieben wird. D.h. dieses Objekt hat die Attribute und die Methoden von beiden Klassen. Wenn es diese Möglichkeit nicht gibt, kann es auch zwei Klassen geben, die bei denen ein Teil der Methoden identisch ist. Das folgende Beispiel soll zeigen, welche Verständnisprobleme dabei auftreten können.

Bild 23 zeigt einen Klassenbaum. Die Klasse A ist die Wurzel des gezeigten Klassenbaums. Von den drei Unterklassen B, C und D hat die Klasse D noch eine Unterklasse E. Die Klassen B und E sollen eine Beziehung, die keine Verebungsbeziehung darstellt, zu Objekten der Klasse F besitzen. Diese Beziehung soll immer von den Objekte der Klasse B oder E hergestellt werden.

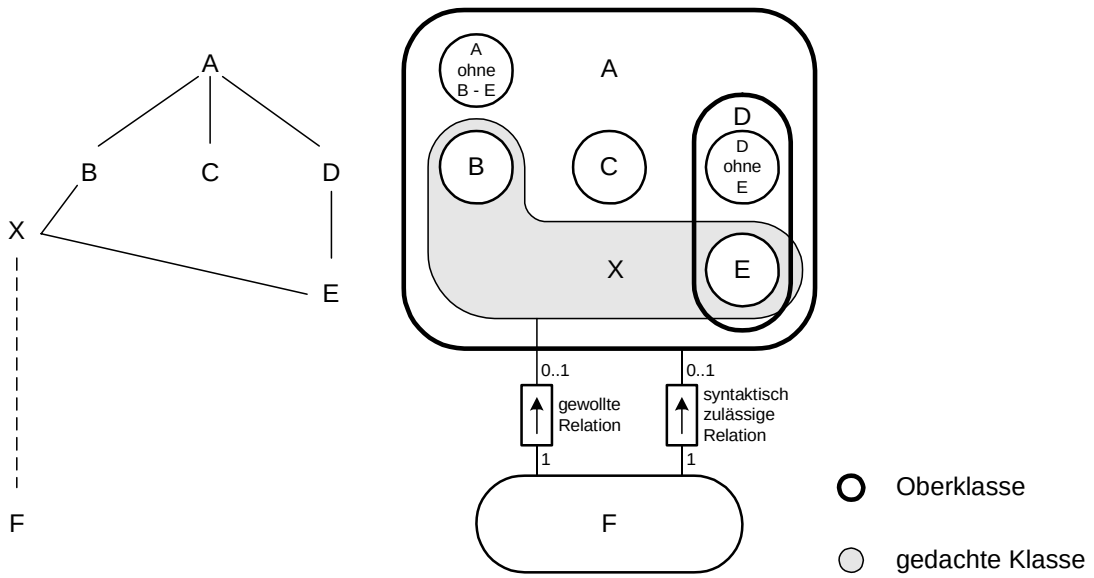


Bild 23: Beispielklassenbaum

Ein Objekt aus der Klasse F soll man fragen können, zu welchem Objekt der Klasse B oder E es gehört. Bei dieser Frage erhält man als Antwort eine Referenz auf ein Objekt. Da diese Objekte aus unterschiedlichen Klassen sind, muß der Rückgabewert eine Referenz auf Objekte aus der nächsten gemeinsamen Oberklasse sein. In diesem Fall ist diese Oberklasse die Klasse A. Man kann sich vorstellen, daß in der Oberklasse A eine Methode beschrieben wird, mit der man den Typ des Objekts abfragen kann. Damit kann man diesem Objekt wieder seinen wahren Typ zuordnen.

Allerdings hat die Klasse A vier Unterklassen. Damit ist es formal möglich, daß ein Objekt der Klasse F auch eine Beziehung zu einem Objekt der Klasse C oder D hat. Dies kann zwar nicht vorkommen, da Objekte aus diesen Klassen gar keine Methoden besitzen, die diese Relation aufbauen können. Es wäre aber anschaulicher gewesen, wenn man die Klasse X zur Verfügung gehabt hätte als Typ für den Rückgabewert. Denn Objekte aus dieser Klasse sind genau diejenigen, zu denen F-Objekte eine Relation besitzen können.

In dieser Arbeit werden die gewollten Relationen gezeigt, da die syntaktisch möglichen Relationen eher zur Verwirrung und nicht zum Verständnis des Systems führen.

4.2.3 Der Callback-Mechanismus von Astoria

Der Callback-Mechanismus wird in der Astoria-API sehr häufig verwendet. Deshalb soll die Verwendungsweise an dieser Stelle erklärt werden.

Bei diesem Mechanismus werden Unterprogramme, sogenannte Callback-Routinen, vom Astoria-System aufgerufen. Der Anstoß für den Aufruf eines Unterprogramms geht dagegen bei allen anderen Unterprogrammen vom Benutzer aus. Die Callback-Routine kann mit einer Interrupt-Routine verglichen werden. Durch einen API-Aufruf wird die Vereinbarung getroffen, welche Routine aufzurufen ist, ähnlich wie Adressen in der Interrupt Vektor-Tabelle[1] eines Prozessors eingetragen werden. Bei Astoria können Aktionen die vom Benutzer gestartet werden das Astoria-System dazu veranlassen, daß es eine Callback-Routine aufrufen.

Callbacks werden in Astoria immer dann benutzt, wenn eine Liste vom Objekten zurückgeben wird. Beispielsweise kann man sich vorstellen, daß durch einen API-Aufruf eine Liste aller Benutzer zurückgegeben wird. Statt einem Datentyp für einen Rückgabewert zu wählen, der eine Liste beschreibt, wird für jeden Benutzer diesselbe Callback-Routine aufgerufen. Die Callback-Routine erhält als Übergabeparameter den gefundenen Benutzer. Je nachdem aus welchem Grund alle Benutzer aufgezählt werden sollen, kann man sich unterschiedliche Callback-Routinen vorstellen. Beispielsweise kann eine Callback-Routine dazu genutzt werden alle Benutzer in eine Liste einzutragen. Eine andere Anwendung wäre das Suchen eines Benutzers. Der Rückgabewert der Callback-Routine kann dann dazu genutzt werden, die Aufzählung aller Benutzer abzubrechen.

1. In der Interrupt Vektor-Tabelle stehen die Startadressen der Interrupt-Routinen. Darin wird eine Zuordnung zwischen den Interrupts und den aufzurufenden Rouinen getroffen. Wenn ein Interrupt auftritt, wird die Routine aufgerufen, die in der Interrupt Vektor-Tabelle für diesen Interrupt eingetragen ist.

Die Übergabeparameter und der Rückgabewert hängen von der API-Routine ab, die den Callback veranlaßt.

Bild 24 zeigt, was nach einem API-Aufruf geschieht, wenn eine Callback-Routine aufgerufen werden soll. Nach dem Aufruf eine Methode der API, werden Aktionen ausgeführt, mit denen die Daten aus der Datenbank geholt werden. Wenn diese Daten vorliegen, wird die Callback-Routine aufgerufen, wobei die geholten Daten an die Callback-Routine übergeben werden. In der Callback-Routine werden dann diese Daten weiterverarbeitet. Nach dem Rücksprung aus der Callback-Routine wertet die API-Routine den Rückgabewert der Callback-Routine aus. Wenn es noch weitere Daten gibt, die aus der Datenbank geholt werden sollen und die Callback-Routine nicht durch ihren Rückgabewert signalisiert, daß das Datenholen abgebrochen werden soll, wird der nächste Datensatz geholt und die Callback-Routine erneut aufgerufen. Sollte das Datenholen abgebrochen werden, wird von der API-Routine zurück ins Anwenderprogramm gesprungen. Je nach Rückgabewert der API-Routine kann das Anwenderprogramm entscheiden, ob dieser Aufruf erfolgreich war. Bei Bedarf wird eine Fehlerbehandlung erfolgen.

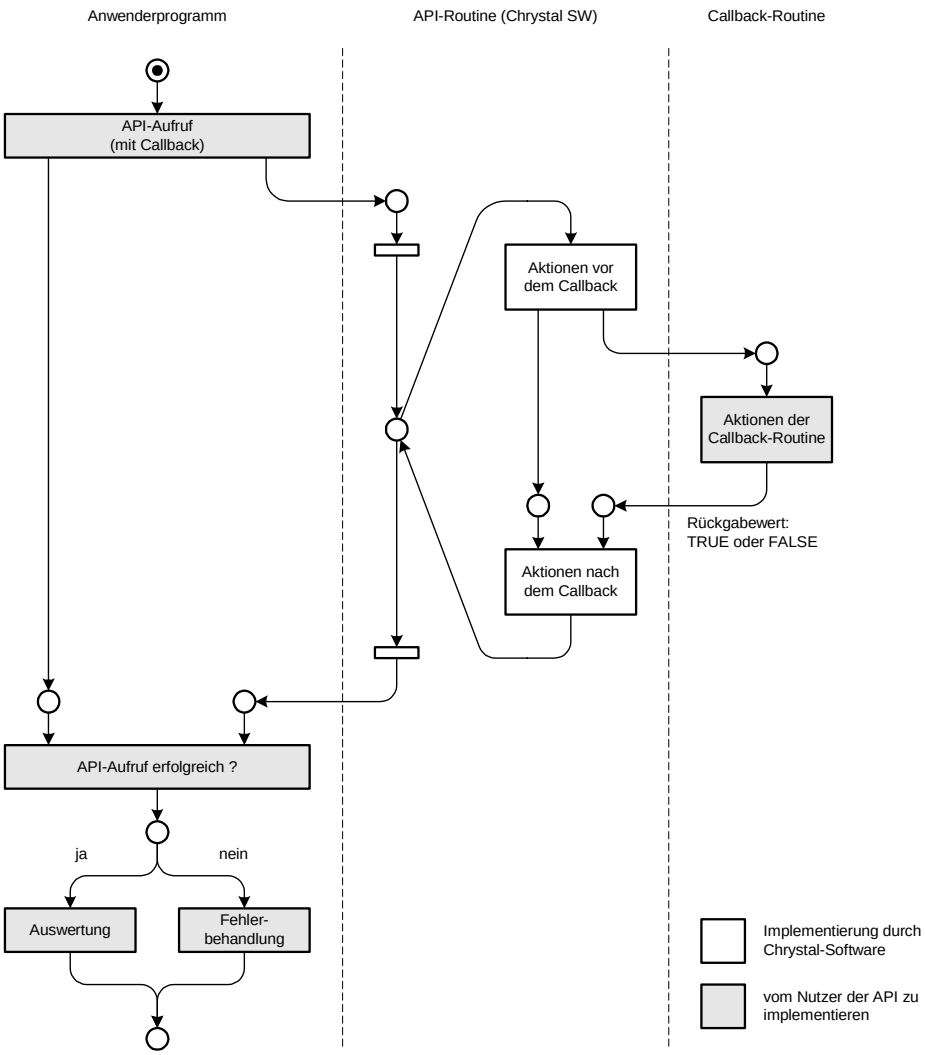


Bild 24: Callbacks in Astoria

5. Konzepte von Astoria

In diesem Kapitel soll die Erklärung der Funktionalität nach dem Separation of Concerns-Prinzip erfolgen. Dabei werden nacheinander einzelne Concerns wie Ablagestruktur oder Berechtigungswesen vorgestellt. Für jeden dieser Concerns soll zuerst das Konzept erklärt werden. Danach wird die Realisierung durch die Astoria-API beschrieben. Dabei werden, soweit es sich um Objekte handelt, die in der API-Dokumentation beschrieben werden, die entsprechenden Klassennamen[1] verwendet. Ein Klassenbaum für die Objekte, die in der Datenbank abgelegt werden können, befindet sich im Anhang.

5.1 Zugriff auf Dokumente

Wenn man auf ein Dokument Zugriff hat, dann kann man auf dieses Dokument zeigen. Die anderen Identifikationsmöglichkeiten sind Nennen und Umschreiben [Wendt_97]. Dabei wird der Ausdruck „Zugriff auf ein Dokument haben“ so verstanden, daß ein Benutzer Operationen auf diesem Dokument ausführen kann. Der Unterschied zwischen Zeigen und Zugreifen besteht durch die Operationen die tatsächlich ausgeführt wird. Ein Zugriffsschutz verhindert das Ausführen von Operationen auf Dokumenten, auf die man zeigen kann. Für dieses Kapitel soll das Zeigen auf Dokumente betrachtet werden.

Um auf Dokumente zeigen zu können, gibt es bei Astoria mehrere Möglichkeiten:

(a) Suchen

(b) Navigation

(c) externe Referenzen

Bei der Suche gibt man eine Umschreibung des Dokuments an. Beispielsweise kann man nach allen Dokumenten suchen, die das Wort „Volltextsuche“ enthalten. Als Rückmeldung auf einen Suchauftrag erhält man eine Liste der Dokumenten, die das Wort „Volltextsuche“ enthalten, um bei diesem Beispiel zu bleiben. Auf das Dokument in der Liste kann man nun zeigen. Wenn man den Namen des Dokuments als Suchkriterium angibt, handelt es sich dabei um eine Nennung.

Die Navigationsstruktur besteht aus Containern, die wieder Container oder Dokumente enthalten können. Ausgehend von einem Wurzelcontainer, der in keinem anderen Container mehr enthalten ist, kann man sich zu einem Ort innerhalb der Ablage hin navigieren. Dieser Ort ist durch Angabe aller Container von der Wurzel an, eindeutig bestimmt. Man spricht auch vom *Pfad*, wenn man alle Container von der Wurzel bis zum „aktuellen“ Container angibt. Der Pfad ist Teil der Umschreibung eines Dokuments. Durch den Containernamen wird der Name einer Klasse von Dingen angegeben, die in diesem Container enthalten sind. Dabei ist der Name der

1. Die Namen aller Objekte der Astoria-Klassenbibliothek beginnen mit „XD_“. Deshalb werden die Namen ohne „XD_“ benutzt.

Container oder Dokumente gemeint, die in diesem Container enthalten sind. Wenn man einen Container mit Namen „Menschen“ besitzt, heißt das nicht, daß Container und Dokumente zur Klasse der Menschen gehören, sondern es können beispielsweise zwei Container darin enthalten sein, die Dokumente über Männer und Frauen enthalten und folglich auch so benannt werden. Männer und Frauen gehören zur Klasse der Menschen.

Auf die Container muß man beim Navigieren ebenfalls immer zeigen können. Wenn man navigiert, dann zeigt man auf einen Container und gibt dem System die Anweisung den Inhalt des Containers darzustellen, auf den man gerade zeigt. Ist in diesem Container ein Dokument enthalten, kann man auf dieses zeigen und dem System Aufträge geben, in denen man angibt, was mit diesem Dokument gemacht werden soll.

Die dritte Möglichkeit, Zugriff auf ein Dokument zu erhalten, ist der Zugriff über externe Referenzen. Dabei handelt es sich um ein Zeigen auf das Dokument, wenn man dies aus Benutzersicht betrachtet. Durch Zeigen auf eine Datei zeigt der Benutzer auf ein Dokument.

5.2 Die Ablagestruktur

5.2.1 Prinzip der Ablage

Astoria bietet eine hierarchische Ablagestruktur. Dokumente werden in Containern abgelegt. Dabei gibt es zwei Arten von Containern, nämlich Folder und Cabinet. Ein Cabinet wird als Wurzel bezeichnet. Im Filesystem des Servers findet man jeweils eine Datei, die genau diesem Cabinet entspricht (Bild 25). Diese Datei muß in einem vereinbarten Verzeichnis[1] des Dateisystems stehen. Für den Benutzer sind alle Cabinets innerhalb dieses Verzeichnisses zugänglich. Genau genommen ist also dieses Verzeichnis die Wurzel der Ablagestruktur, wobei der Begriff Verzeichnis nicht an der Programmierschnittstelle angeboten wird.

In einem Cabinet können Dokumente unterschiedlicher Typen, aber auch Folder enthalten sein. Es ist nicht möglich, daß ein Cabinet in einem Folder liegt. Ein Folder kann, genau wie ein Cabinet, Dokumente und Folder enthalten. Damit ist es möglich eine Ablagestruktur mit beliebig vielen Folderstufen aufzubauen. Festgelegt ist nur, daß die Wurzel ein Cabinet ist. Die Tiefe der Ablagehierarchie oder die Anzahl der Dokumente ist nur aus technischen Gründen (kein Speicherplatz mehr vorhanden) beschränkt.

1. Diese Vereinbarungen stehen in einer Datei namens „Astoria.ini“.

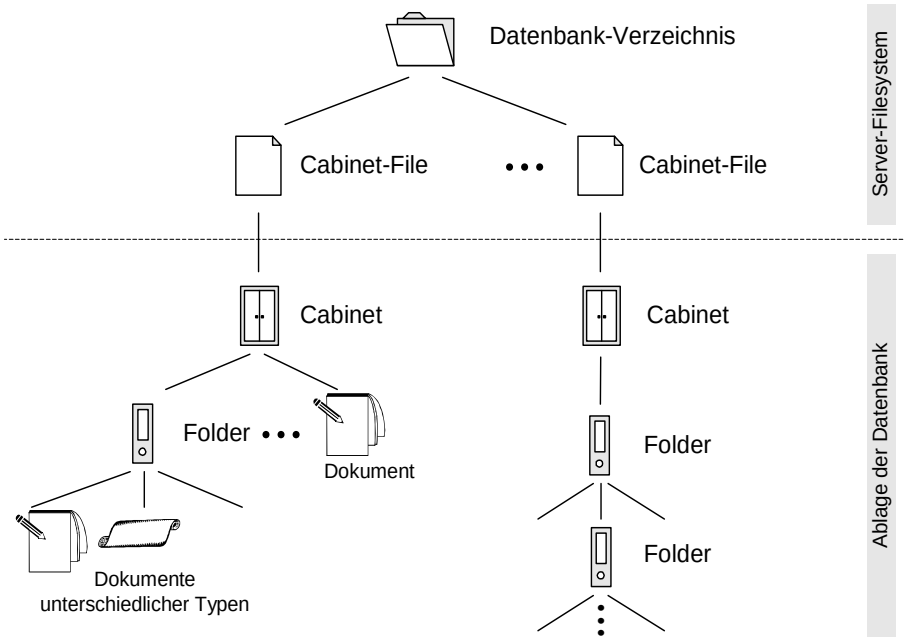


Bild 25: Ablagestruktur exemplarisch

5.2.2 Realisierung der Ablage

In Bild 26 sind die Objekte dargestellt, die die API zum Aufbau der Ablage anbietet. Im Entity-Relationship-Diagramm werden immer Mengen von Individuen dargestellt. Diese Individuen sind Objekte. Daher sind die Entitäten als die Menge aller Objekte zu verstehen, die durch eine-Klasse beschrieben werden.

Zunächst sollen die Klassen der Objekte beschrieben werden, durch die der Aufbau der Ablagehierarchie beschrieben werden. Danach werden die einzelnen Klassen erklärt, die die stellvertretenden Objekte für die Dokumente beschreiben, die in Astoria abgelegt werden können. In diesem Zusammenhang wird alles als Dokument bezeichnet, was in Form einer Datei abgespeichert kann und in Astoria abgelegt werden kann.

Ein Cabinet wird als Wurzelcontainer gesehen. Im Wesen eines Containers liegt, daß er andere Objekte beinhalten kann. Allerdings wird in der Cabinet-Klasse[1] nicht beschrieben, wie dieser Enthaltenseinsmechanismus realisiert wird. Dazu bedient man sich der Cabinetstub-Klasse. Ein Objekt dieser Klasse kann eine 1-zu-1-Beziehung zum Cabinet pflegen und Objekte vom Typ Folderitem enthalten. Die Methoden und Attribute zur Realisierung der Enthaltenseinsrelation werden von der Klasse Folder geerbt. Daher heißt die Klasse der enthaltenen Objekte Folderitem. An dieser Stelle findet man auch den Grund, warum ein Folder wieder einen Folder enthalten kann. Die Folderitem-Klasse ist nämlich eine Oberklasse von Folder.

1. Die Cabinet-Klasse dient zum Erzeugen eines Cabinets. Dieses äußert sich darin, daß auf dem Server eine neue Datei für dieses Cabinet erzeugt wird.

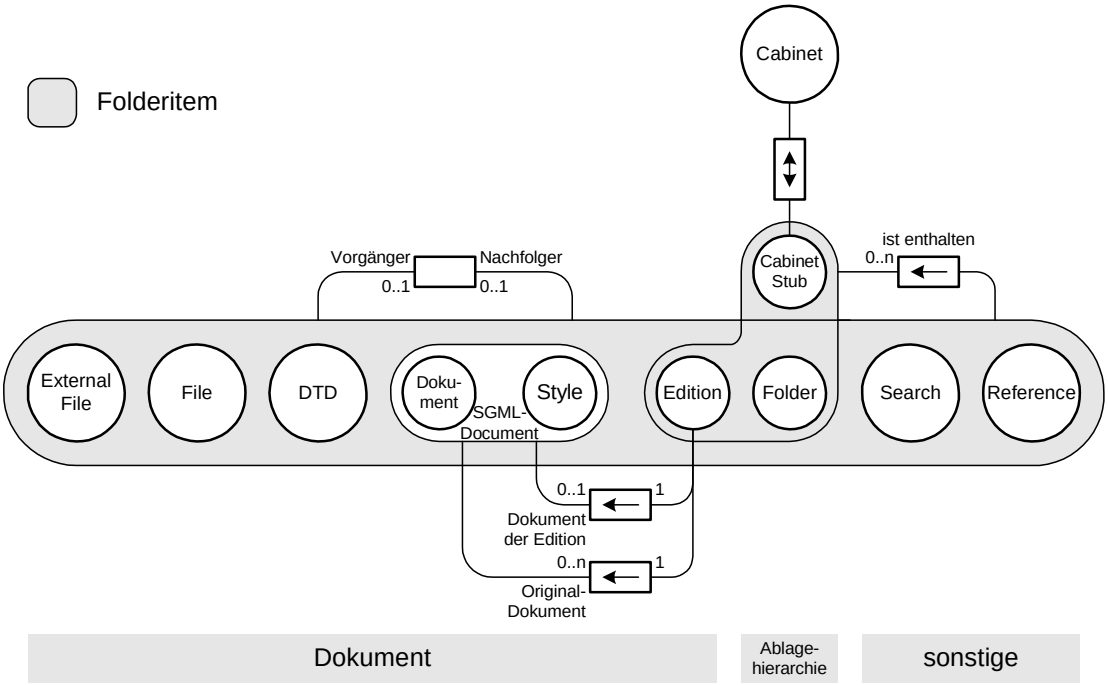


Bild 26: Objekte zur Realisierung der Ablagestruktur

Da Cabinetstub eine Unterklasse von Folder ist, müßte auch ein Cabinetstub in einem Folder enthalten sein können. Syntaktisch ist dies durchaus möglich. Allerdings ist in keinem der beteiligten Objekte eine Methode vorgesehen, mit der man ein Cabinetstub-Objekt unter ein Folder-Objekt hängen kann. Aus diesem Grund kommt dieser Fall auch nicht vor.

Die im Container enthaltenen Objekte haben eine Ordnungsbeziehung zueinander. Beim Erzeugen eines Folderitems muß immer angegeben werden nach oder vor welchem anderen Folderitem das neu erzeugte Objekte liegen soll. Als Alternative dazu kann auch der Folder als Bezugsobjekt angegeben werden, in dem man angibt, daß das neu erzeugte Objekt das erste oder das letzte in diesem Folder ist.

Damit ist der Implementierungsteil der Ablagestruktur beschrieben. Als nächstes folgt die Beschreibung der einzelnen Objekttypen, die in der Ablage zu finden sind.

Eine Referenz auf ein Dokument, die auf ein Dokument zeigt, das nicht in Astoria enthalten ist wird als *external File* bezeichnet. Dieses Dokumente liegt in Form einer Datei auf einem Rechner, auf den ein Client Zugriff hat. Da jeder Client ein anders organisiertes Dateisystem hat, kann es sein, daß diese Datei nur von einem Rechner aus zugänglich ist.

Wenn ein Dokument in Astoria abgelegt wird, ohne daß das Dokumentenmanagementsystem die innere Struktur dieses Dokuments kennt, wird dieses als *File* bezeichnet. In der Datenbanksprache wird ein solches Objekt auch als BLOB[1] bezeichnet.

1. BLOB: Binary Large OBject

Die SGML-Fähigkeit von Astoria sieht man an den Dokumententypen, die abgelegt werden können. Im Gegensatz zum File ist dem Dokumentenmanagementsystem bekannt, wie die innere Struktur eines Dokuments von Typ *DTD* aussieht. Astoria kann nämlich mittels DTD den Typ eines SGML-Dokuments feststellen. Dazu muß dem System mitgeteilt werden, welches abgelegte Dokument als DTD in Frage kommt.

Folglich gibt es auch einen Dokumententyp für SGML-Dokumente. Auch hier ist dem System bekannt, daß dieses Dokument eine vom System verarbeitbare Struktur besitzt. Ein *Style*-Dokument entspricht den Layoutvereinbarungen, die in Kapitel 3.2.2 erwähnt worden sind. Die Struktur dieses Dokuments entspricht dem SGML-Standard. Daher ist die Style-Klasse auch als Unterklasse von SGML-Dokument realisiert.

Von einem SGML-Dokument kann eine *Edition* erstellt werden. Ein Edition-Objekt ist ein Schnappschuß, des aktuellen Dokuments. Man kann sich eine Edition wie ein Mappe vorstellen, die alle Bestandteile dieses Dokuments enthält, zu denen Bilder, DTD oder Style-Dokumente gehören, aber auch das Dokument selbst. Deshalb muß ein Editionsobjekt die Enthaltenseinsrelation realisieren können, wie sie auch von den Foldern genutzt wird. Ein Editions-Objekt kann zusätzlich zu einem Folder-Objekt eine Relationen zu SGML-Dokumenten pflegen. Zum einen gibt es eine Relation zum Originaldokument, von dem die Edition aus erstellt worden ist. Dieses Originaldokument kann an einem beliebigen Ort der Ablage abgelegt sein. Die andere Relation besteht zwischen der Edition und dem SGML-Dokument, das in der Editionsmappe enthalten ist.

Astoria bietet die Möglichkeit zur Volltextsuche. Normaler Weise muß zum Ausführen einer Volltextsuche ein Anfrageausdruck formuliert werden, der das gesuchte Dokument umschreibt. Wenn man diesen Ausdruck nicht bei jeder Suche neu formulieren will, kann dieser Ausdruck in der Ablagestruktur abgelegt werden. Das dafür zuständige Objekt ist vom Typ *Search*. Dieses Objekt ist nicht nur der Stellvertreter für den Suchausdruck, sondern auch der Akteur, der den Auftrag zum Suchen annimmt.

Die Klasse der *Reference*-Objekte besitzt einen Speicher, der eine Referenz auf ein anderes Objekt hat, das sich in der Datenbank befindet. Hier tritt beispielsweise das Problem, das in Kapitel 4.2.2.3 mit Bild 23 dargestellt wird. Es können nämlich nur Referenzen auf Objekte angelegt werden, auf die ein Benutzer per Volltextsuche oder Navigation Zugriff hat. Beispielsweise enthält die Datenbank Objekte, die stellvertretend für den Benutzer stehen. Diese sind aber weder durch Volltextsuche, noch durch Navigation zu erreichen. Deshalb wird mit Sicherheit nie eine Referenz auf ein Benutzerobjekt durch ein solches Referenzobjekt identifiziert werden. Syntaktisch wäre es aber zulässig. Gewollt sind hier Referenzen auf Folderitems und auf versionierbare Objekte, die im nächsten Kapitel beschrieben werden.

Daß Folder eine Oberklasse von CabinetStub und Edition ist, sieht man daran, daß in der Folderklasse nur beschrieben wird, wie diese Enthaltenseins-Relation gepflegt wird. CabinetStub-Objekte können zusätzlich eine Relation zum Cabinet pflegen. Bei den Edition-Objekten ist die zusätzliche Relation zu den SGMLDocument-Objekten in der Klassenbeschreibung vorhanden.

5.3 Versionierung von Dokumenten

5.3.1 Prinzip der Versionierung

In Astoria können nur Dokumente versioniert werden. D.h. es gibt keine Versionen von Suchausdrücke, Folder, Referenzen etc. Wenn man ein Dokument über Navigation in der Ablagestruktur oder Volltextsuche[1] gefunden hat, dann hat man automatisch die aktuelle Version des Dokuments. An diesem Dokument hängen dann alle Versionen wie an einer Kette aufgereiht. Für jede Version des Dokuments steht eine Karteikarte zur Verfügung, die zwei Felder enthält. In das eine Feld kann man den Namen der Version eintragen. Das zweite Feld kann einen längeren Text aufnehmen, in dem beispielsweise der Versionsübergang begründet wird.

Eine neue Version wird immer beim Importieren oder beim Checkin eines Dokuments angelegt. Auf Wunsch kann allerdings auch die alte Version überschrieben werden.

5.3.2 Realisierung des Versionierungsmechanismus

In Kapitel 5.2.2 wurden Objekte beschrieben, die man in der Ablagehierarchie findet. Eine Versionierung von Objekten wird bei Astoria nur bei Dokumenten unterstützt. Damit kommen als versionierbare Objekte nur noch Objekte der Klassen File, external File, SGML-Document und DTD in Frage.

Da die Klassen der oben beschriebenen Objekte keinen Versionierungsmechanismus besitzen, hat jedes Objekt dieser Klasse eine 1-zu-1-Beziehung zu einem Objekt aus der Klasse der versionierbaren Objekte. Diese Zuordnung ist in der Tabelle von Bild 27 dargestellt. Die Versionierung funktioniert über die Vorfahre-Nachfahre-Relation der versionierten Objekte. Dabei besteht diese Relation immer zwischen Objekten des gleichen Typs. D.h. ein FNode kann nur immer nur einen FNode als Vorgänger haben. Es ist nicht möglich, daß ein FNode-Objekt als Nachfolger ein DTDRoot-Objekt hat.

Wenn es von einem SGML-Dokument mehrere Versionen gibt, dann besteht eine Relation zwischen dem SGMLDocument-Objekt und dem neuesten RNode-Objekt. Ältere Versionen des Dokuments können über die Vorfahre-Nachfahre-Relation des RNode-Objekts erreicht werden.

Bild 27 zeigt, daß an jedem versionierbaren Objekt eine Revisionstamp hängen kann. Die Revisionstamp realisiert die Karteikarte für Namen und Beschreibung der Version. Die Funktionalität zum Anlegen und Ändern des Namens und der Beschreibung wird in der NamedObject-Klasse beschrieben, die eine Oberklasse sowohl für Revisionstamp als auch für die Folderitems

1. Bei der Volltextsuche werden zwar auch die älteren Versionen durchsucht. Allerdings erhält man als Ergebnis wieder die neue Version, auch wenn sich der Suchbegriff nur in einer älteren Version befindet.

Dokument	versionierbares Objekt
external File	external FNode
File	FNode
DTD	DTDRoot
SGML-Document	RNode

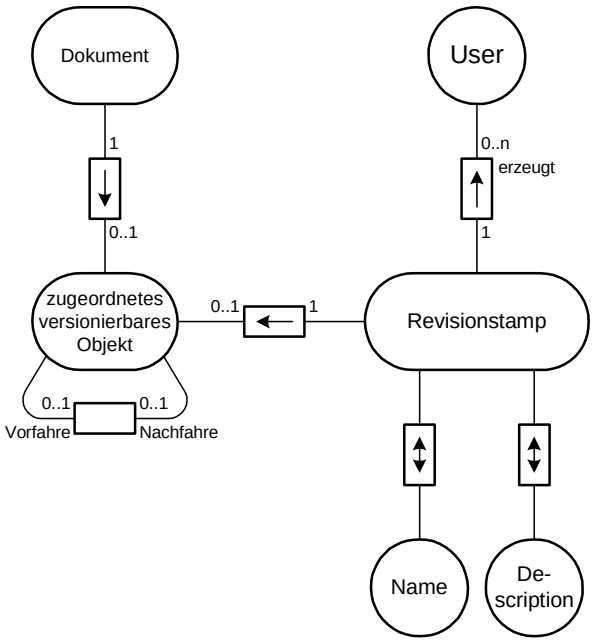


Bild 27: Versionierung in Astoria

darstellt. Von daher hat auch jedes Folderitem einen Namen und eine Beschreibung. Ein Revisionstamp-Objekt besitzt zusätzlich die Möglichkeit die Relation zu einem User-Objekt zu pflegen.

In der API-Dokumentation steht, daß das User-Objekt für den Benutzer steht, der die Revisionstamp erzeugt hat. Der Benutzer, der die Revisionstamp erzeugt, ist aber auch derjenige, der die neue Version des Dokuments erzeugt hat. Daher ist wohl der Benutzer gemeint, der die neue Version erzeugt hat.

Versionierbare Objekte unterliegen nicht dem Berechtigungswesen. Da ein Zugriff nur über die Objekte aus der Klasse der Dokumente (Folderitems) erhält, reicht es aus, daß die Berechtigung für die zuletzt genannten Objekte geprüft wird.

5.4 SGML-Funktionalität

5.4.1 SGML-Dokumente

5.4.1.1 SGML-Dokumente aus Sicht des Benutzers

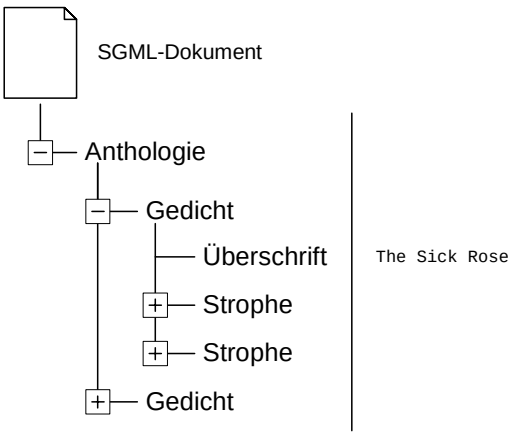


Bild 28: Navigation in einem SGML-Dokument[a]

a.Die Darstellung soll zeigen, daß die Struktur des Dokuments sichtbar ist. Es gibt kein Tool, das genau diese Darstellung benutzt.

Astoria kennt die innere Struktur von SGML-Dokumenten. Wenn man über Navigation oder Volltextsuche ein Dokument gefunden hat, dann hat man das ganze Dokument identifiziert und kann beispielsweise dieses Dokument exportieren. Bei SGML-Dokumenten kann man sogar in der Struktur des Dokuments navigieren (Bild 28). Hier wird das Beispiel aus dem SGML-Kapitel wieder aufgegriffen. Die Elemente des Dokuments stellen die Knoten im Baum dar. Im Beispiel besteht die Anthologie aus zwei Gedichten. Wenn ein Benutzer nur das zweite Gedicht exportieren möchte, dann ist dies auch möglich.

Hat man einen Knoten im Baum erreicht, der ein Blatt darstellt, wird auch dessen Wert angezeigt. Das Bild zeigt den Text des Überschrifts-Elements, nämlich „The Sick Rose“.

5.4.1.2 Interne Darstellung eines SGML-Dokuments

Im Kapitel über Versionierung wurde bereits gezeigt, daß die Objekte der Folderitem-Klasse nicht den Inhalt des Dokuments repräsentieren. So ist ein Objekt der SGMLDocument-Klasse nur der Identifikator für ein ganzes Dokument, wobei nur die aktuelle Version direkt referenziert wird.

Um den Inhalt des Dokuments zu repräsentieren gibt es mehrere Klassen, die alle Unterklassen der versionierten Objekte sind. Objekte der RNode- und SNode-Klasse bieten die Mechanismen zum Aufbau des Ableitungsbaums. Der Text selbst steht in Objekten der CNode-Klasse. Es gibt noch eine Klasse, in der die Text für Kommentare enthalten sind, diese heißt InfoNode.

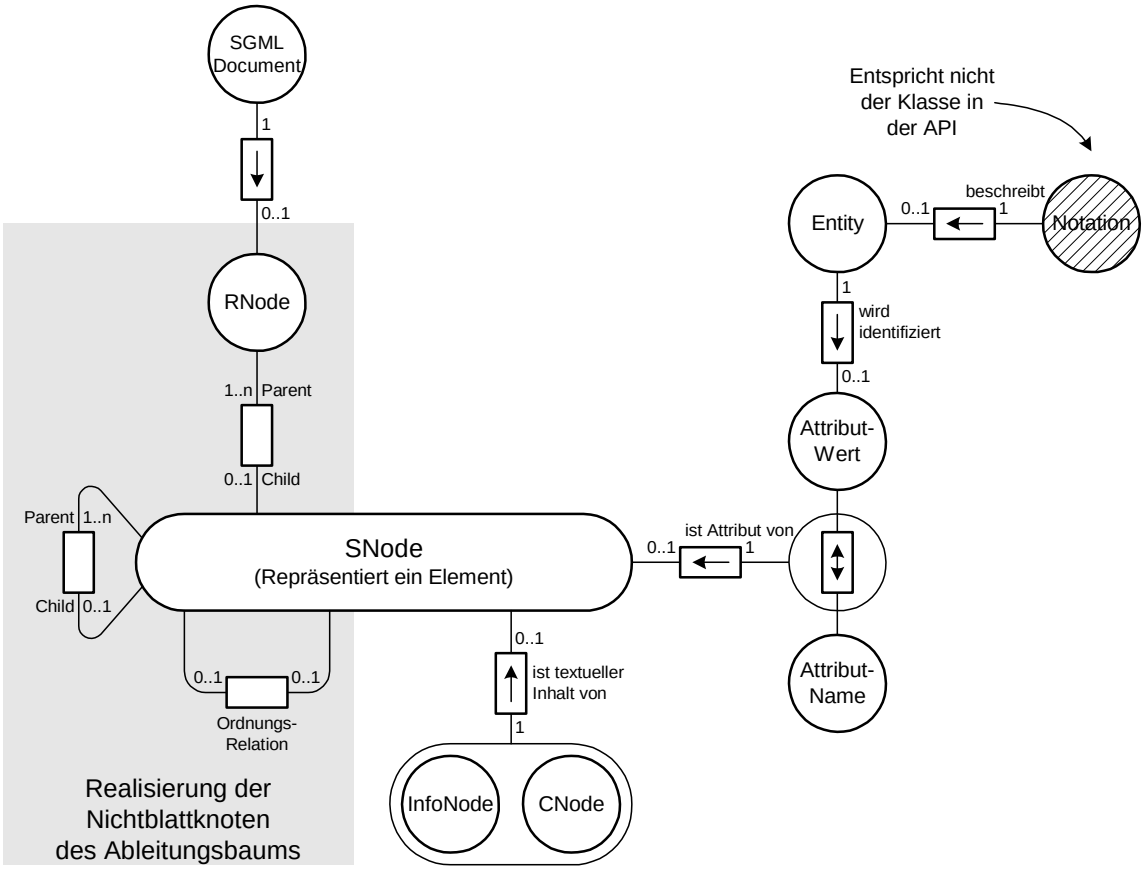


Bild 29: Interne Darstellung eines SGML-Dokuments

In Bild 29 stellt ein RNode die Wurzel des Dokuments dar. An diesem Knoten können beliebig viele SNodes als Kinder hängen. Der Name eines SNodes ist identisch mit den Namen eines Tags aus dem SGML-Dokument. Es gibt zwei Arten von Beziehungen, die SNode-Objekte zueinander haben können. Die eine ist eine Hierarchiebeziehung wie sie zwischen Anthologie und Gedicht im obigen Beispiel vorkommt. Was in der API-Dokumentation als Parent-Child-Beziehung bezeichnet wird, stellt eine Enthaltenseinbeziehung dar. Diese Relation reicht aber noch nicht aus, da auch die Reihefolge, in der die Unterknoten an einem Knoten hängen von Bedeutung ist. In Bild 28 gibt es drei Unterknoten des Knotens Gedicht, nämlich Überschrift und zweimal Strophe. Die Information, daß zuerst das Überschriftselement, dann das Element für die erste Strophe und schließlich das Element für die letzte Strophe in diesem Dokument vorkommen, muß sich ebenfalls gemerkt werden. Dazu gibt es die Ordnungsrelation, die durch die Oberklasse von SNode realisiert wird.

Da SNode und RNode bezüglich des Baumaufbaus ähnliche Eigenschaften besitzen, hat man für beide eine gemeinsame Oberklasse namens ListNode implementiert. Ein RNode-Objekt muß die Relation zum SGMLDocument-Objekt pflegen können, was für ein SNode-Objekt nicht vorgesehen ist. Dagegen kann ein RNode-Objekt keine Attribute und keine Texte verwalten, was alleine Aufgabe der SNode-Objekte ist.

Ein SNode-Objekt kann eine Liste von Attributen[1] verwalten. Ein Attribut besteht immer aus einem Namen und einem Wert. Setzen und Abfragen von Namen und Werten ist über die Methoden der SNode-Klasse realisiert.

Manche Werte können auch IDs von Entities sein. Beispielsweise könnte ein Grafikelement ein Attribut namens „ID“ besitzen. Das ID-Attribut hat den Wert „Landkarte“. Damit ist eine Referenz auf ein Entität gemeint, die bei Astoria in Form von Objekten der Entity-Klasse vorkommen. Ein Entity-Objekt kann auch eine Notation besitzen. Entity-Objekte können ein Objekt referenzieren, das sich in der Datenbank befindet. Referenziert werden dabei File- und FNode-Objekte, sowie deren artverwandte Objekte, die stellvertretend für Dokumente stehen, derern Struktur Astoria nicht bekannt ist.

Wenn der Wert eines Attributs keine ID für eine Entität ist, hat dieser Wert für Astoria auch keine Interpretation. Attribute mit Entitäts-ID werden in der API als „Entity-Attributs“ bezeichnet.

Die Relation zwischen SNode und der Gruppe aus CNode und InfoNode, soll so verstanden werden, daß es entweder eine 1-zu-1-Beziehung zwischen SNode und CNode besteht oder diese Beziehung zwischen SNode und InfoNode existiert. Objekte der Klasse InfoNode enthalten einen Kommentartext. Wenn in diesem der dazugehörige SNode eine Relation zu einem InfoNode-Objekt hat, dann stellt das SNode-Objekt einen Kommentar im SGML-Dokument dar. Dagegen ist bei der Relation zu einem CNode-Objekt ein SNode als Element zu sehen. Im SNode-Objekt ist der Name des Tags gespeichert und das CNode-Objekt beinhaltet den Text des Elements.

Ein SGML-Dokument besitzt einen Deklarationsteil, in dem Notations und Entities deklariert werden. Dieser Deklarations-Teil wird durch die Relation mit Objekten aus der DTDRoot-Klasse hergestellt (Bild 31). Das DTDRoot-Objekt ist die Wurzel für den Deklarationsteil. Dieser Teil wird als *DeclarationStub* bezeichnet. Interessanter Weise existiert in diesem Bild zwischen Notations und Entities keine Relation. Denn eine Notation beschreibt die Darstellung einer oder mehrerer Entities. Folglich besteht eine gedankliche Relation. Das Entity-Objekt ist Stellvertreter für eine Entität, die im Dokument vorkommt. Wenn man das Dokument layoutgerecht darstellen will, muß man wissen, wie man diese Entität darstellt, was über die Notation angegeben wird. Astoria muß allerdings zur Verwaltung der internen Dokumentenstruktur

1. Es gibt auch eine Klasse namens Attribute in der Astoria-API. Diese Klasse beschreibt allerdings Objekte, die für die interne Darstellung der DTD benötigt werden. Werte und Namen von Attribute werden von einem SNode-Objekt nicht als Referenz auf ein Attribute-Objekt zurückgegeben sondern als Zeichenketten (Strings).

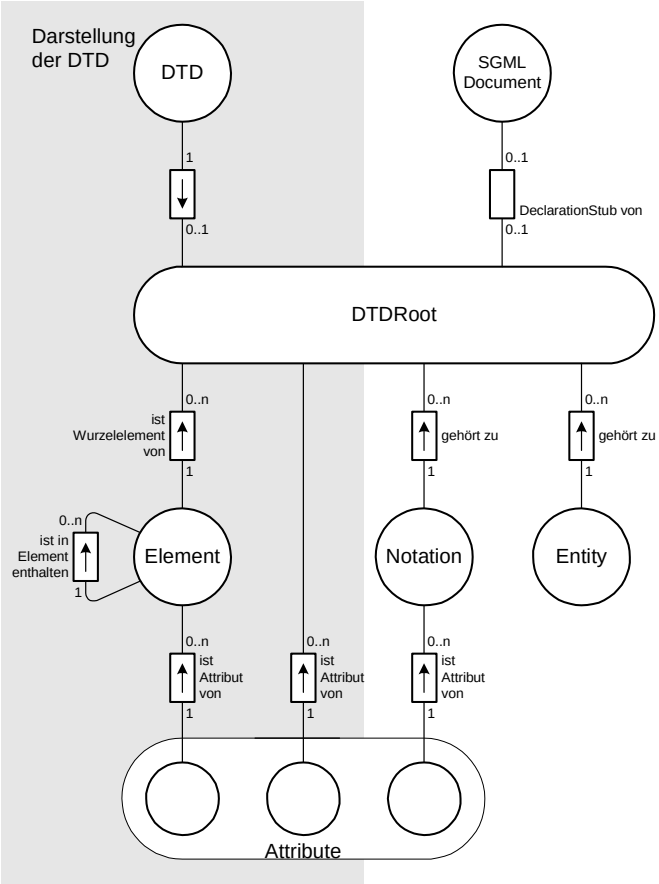


Bild 30: Interne Darstellung von DTD und Deklarationsteil eines SGML-Dokuments

keine layoutgerechte Darstellung erzeugen. Ein externer Dokumentenviewer muß diese Relation erzeugen können, denn sonst kann das Dokument nicht dargestellt werden. Astoria benötigt diese Relation also nicht unbedingt.

Im Zustand von Entity- und Notation-Objekten ist jeweils der Text gespeichert, der in einem SGML-Dokument steht. Dieser Text wird beim Export der Dokuments in die Datei geschrieben, die beim Export erzeugt wird.

Eine Notation kann Attribute besitzen, die beispielsweise beschreiben, in welcher Qualität ein Bild darzustellen ist. Attribute werden hier, im Gegensatz zu den SNode-Objekten, durch eine Relation zu den Attribute-Objekten realisiert. Ein Attribut-Objekt besitzt einen Speicher für seinen Namen und für seinen Wert. Bei der Erzeugung eines Attributs können auch Defaultwerte vorgegeben werden.

5.4.2 Ablage der DTD in Astoria

5.4.2.1 Die DTD aus Benutzersicht

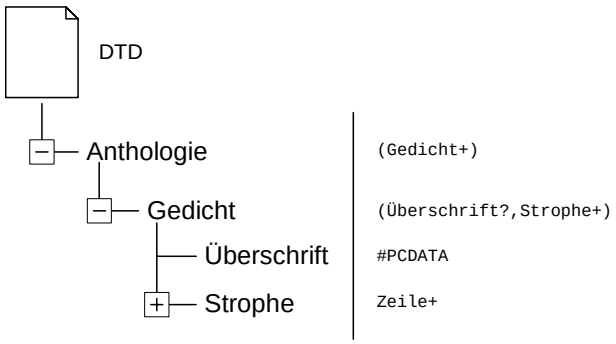


Bild 31: Navigation in der DTD[a]

a.Die Darstellung soll zeigen, daß die Struktur des DTD sichtbar ist. Es gibt kein Tool, das genau diese Darstellung benutzt

Eine DTD liegt genau wie ein SGML-Dokument in der Ablage. Wenn man die DTD im Zugriff hat, kann man in dieser ebenfalls navigieren. Bild 31 zeigt die Strukturierung der DTD. Die Darstellung zeigt die Enthaltenseinsrelationen zwischen den einzelnen Elementtypen. Regeln, die angeben, ob ein Element nur einfach oder mehrfach vorkommen sind auf der rechten Seite der Darstellung zu finden.

5.4.2.2 Interne Darstellung einer DTD

Identifiziert wird die DTD über ein Objekt der Klasse DTD. Dieses Objekt hat eine Referenz auf ein Objekt der DTDRoot-Klasse. Wie der Name schon sagt, ist dieses Objekt die Wurzel für die interne Baumdarstellung der DTD. Die DTDRoot-Klasse ist mit der RNode-Klasse vergleichbar, die die Wurzel für die interne Dokumentendartstellung ist.

Ein Objekt der Element-Klasse ist Stellvertreter für eine Typdefinition. Relationen hat ein solches Objekt nur zu anderen Element-Objekten, die in diesem Element enthalten sind und zu den Attribute-Objekten, die die Attribute beschreiben, die ein solches Element besitzen darf. Information darüber, ob Anfangs- oder End-Tags benötigt werden, wird beim Erzeugen eines solchen Element-Objekts im Zustandsspeicher des Objekts gespeichert. Die Regeln, die in Bild 31 auf der rechten Seite zu sehen sind, werden ebenfalls als Text im Zustandsspeicher abgelegt.

5.4.3 Relationen zwischen DTD und SGML-Dokument

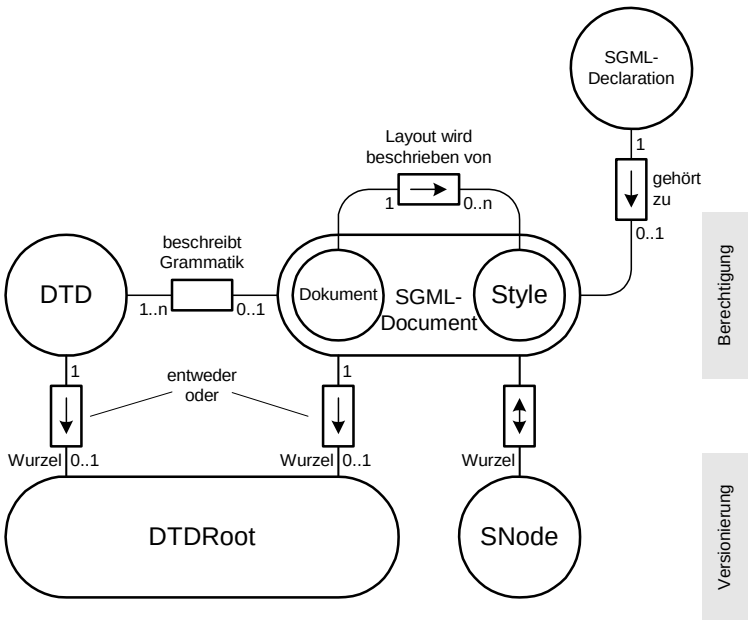


Bild 32: Relationen zwischen den SGML-Dokumenten

Bild 32 zeigt, daß in Astoria bekannt ist, mit welcher Grammatik (DTD) ein Dokument beschrieben wird. Astoria kann auch prüfen, ob das Dokument dieser DTD entspricht. Es ist zwar möglich, Dokumente in Astoria abzulegen, die zu keiner DTD konform sind. Dies führt aber zu Fehlermeldungen bei der Typprüfung.

Außerdem gibt es eine Relation zwischen dem Dokument und seinem Style-Dokument. Wie auch bei der DTD können mehrere Dokumente zu einem Style-Dokument gehören. Zu jedem SGMLDocument-Objekt kann ein SGMLDeclaration-Objekt gehören. Dieser Objekt trägt z.B.Information darüber, wie die Zeichen kodiert sind.

Die Rollenbezeichnung Wurzel bei DTDRoot und SNode sollen andeuten, daß es sich bei diesen Objekten um die Wurzeln des Inhaltsbaums handelt. Diese Objekte sind nicht die Wurzel von DTD oder SGML-Dokumenten. Ein DTDRoot-Objekt ist entweder die Wurzel des Inhalts einer DTD oder die Wurzel eines DeclarationStubs des SGML-Dokuments. Jedes SGML-Dokument besitzt als Wurzel für seinen Inhalt eine Relation zu einem SNode-Objekt.

In Bild 32 ist nochmals angedeutet, daß DTD- und SGMLDocument-Objekte dem Berechtigungswesen unterliegen und nicht versioniert werden können. Bei DTDRoot- und SNode-Objekten ist es genau umgekehrt.

5.4.4 Import und Export von SGML-Dokumenten

5.4.4.1 Workunits

Die vorangegangenen Kapitel zeigen, daß in Astoria Dokumente aus einer Vielzahl von Objekten bestehen können. Da gibt es zum einen den Baum mit Objekten, aus dem die aktuelle Version des Dokuments besteht und zum anderen mehrere Bäume, die zu älteren Versionen gehören. Ein solcher Baum ist immer durch das Wurzelobjekt (DTDRoot, RNode, FNode oder ExternalFNode) identifiziert. Versionierte Objekte können in Astoria aber auch in sogenannten *Workunits* gruppiert werden. Workunits verwalten eine beliebige Liste von versionierten Objekten. Meistens wird jedoch ein Baum oder ein Teilbaum in einer Workunit zusammengefaßt. Wenn man ein Workunit-Objekt als Akteur sieht, dann ist dieser Akteur der Ansprechpartner, wenn man die durch die Workunit identifizierten Objekte des Dokuments ein- oder auschecken will.

5.4.4.2 Import- und Export-Akteur

Die Objekte, die die interner Darstellung des Dokuments repräsentieren können sowohl als Speicher als auch als Akteur gesehen werden. Der Teil des Zustandsspeichers des Objekt, der z.B. Textpassagen des SGML-Dokuments enthält, kann als Speicher gesehen werden. Außerdem gibt es Speicher, die die Referenzen zu anderen Objekten gleicher Aufgabenstellung enthalten. Durch diese Referenz ist die Struktur des Dokuments beschrieben. Ein Objekt der Klasse SGMLDocument kann ebenfalls als Speicher gesehen werden. Darin gespeichert sind Ablageort und Name des Dokument. Allerdings muß man einen Teil des Objekts auch wie einen Akteur sehen. Denn zu diesem Objekt kann man sagen, daß es das mit ihm identifizierte Dokument exportieren kann.

Seltsamer wird es allerdings, wenn man sich fragt, wie man ein Dokument importiert. Dann sagt man zu einem SGMLDocument-Objekt: „Importiere Dich !“ Wenn man dieses Objekt als Stellvertreter für ein Dokument in der Ablage sieht, dann gibt man eine Anweisung an ein Objekt, daß noch gar nicht im System vorhanden ist. Ein angemessenere Sicht ist, daß das SGMLDocument-Objekt zum einen die Datenstruktur ist, in der der Inhalt gepeichert des Dokuments gespeichert wird und zum anderen der Akteur ist, dem ich den Auftrag geben kann, das Dokument zu importieren. Nach erfolgreichem Import identifiziert eine Referenz auf dieses Objekt das Dokument in der Ablage.

Die oben beschriebene Vorgehensweise beim Im- und Export von Dokumenten ist in der betrachtetenden Version 3.0 von Astoria noch zulässig. In späteren Versionen soll es zwar die beteiligten Objekte noch geben. Diese sollen aber nicht mehr diesen Akteurs-Charakter besitzen. D.h. in den Folgeversionen werden die Methoden aus der Klassenbeschreibung verschwinden, die sich auf Im- und Export beziehen. Statt die Methoden an die Objekte zu binden, hat man sogenannte *Interfaces* eingeführt. Diese Interfaces beschreiben die Schnittstelle zu Akteuren, die für Import und Export genutzt werden. Die Datenobjekte dienen dann nur noch als Referenzen.

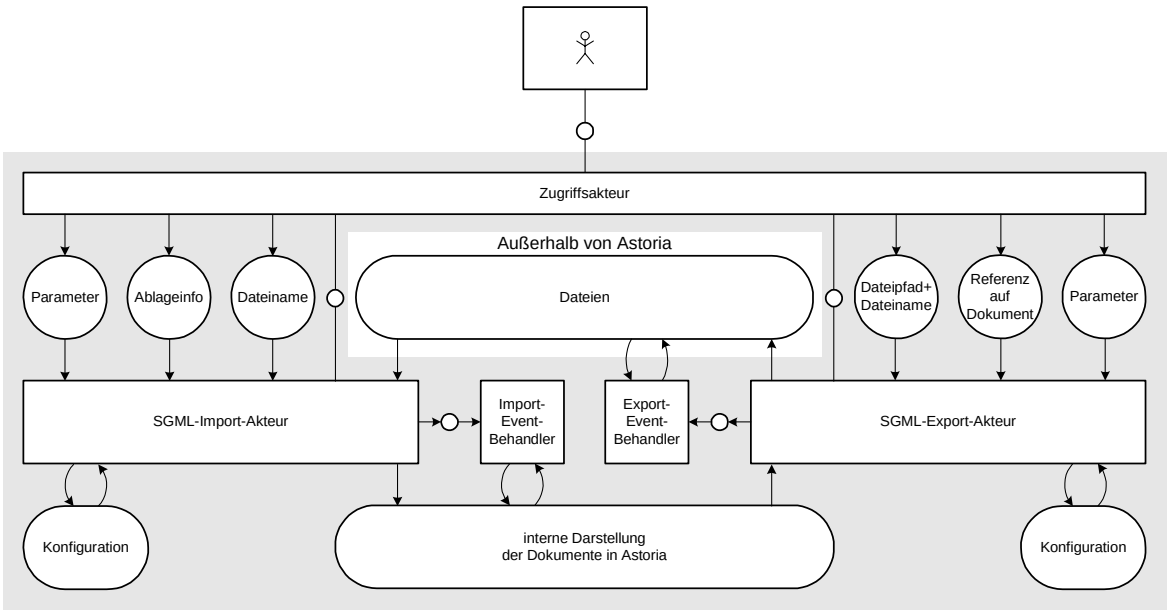


Bild 33: Import- und Export-Akteure in Astoria

Der Zugriffsakteur in Bild 33 greift genau auf diese Interfaces zu. Dieses Bild soll an Hand des Importakteurs erklärt werden. Für den Exportakteur ändert sich lediglich die Transportrichtung des Dokuments.

Der Importakteur kann konfiguriert werden. Über den Zugriffsakteur werden Parameter angegeben, die sich der Import im Konfigurationsspeicher merkt. Beispielsweise kann Astoria auch XML-Dateien einlesen. Wenn die zu importierende Datei ein XML-Dokument beinhaltet, dann kann der Import-Akteur so konfiguriert werden, daß er weiß, daß diese Datei einen anderen Header hat als ein SGML-Dokument. Damit der Importakteur weiß, welches Datei er zu importieren hat, muß ihm beim Import-Auftrag den Dateiname angeben. Ablageort und Name des Dokuments befinden sich im Bild in dem Ablageinfo-Speicher.

Wird nun eine Datei importiert, dient der Importakteur als Parser. Die zu importierende Datei wird gelesen und daraus die interne Darstellung des Dokuments generiert. Diese interne Darstellung besteht aus den in den vorangegangenen Kapiteln beschriebenen Objekten. Wenn der Parser einen gültigen Sprachkonstrukt erkannt hat, wirft dieser ein Event, dessen Datenteil Informationen über die Art des Sprachkonstrukts enthält. Der Eventbehandler kann die interne Darstellung modifizieren. Dieser Mechanismus ist sehr vielseitig nutzbar, dar der Eventbehandler in der gleichen Programmiersprachenwelt beschrieben wird, wie eine Applikation[1], die die API nutzt. Als Aufgaben des Eventbehandlers kann man sich beispielsweise eine Filterfunktion denken, die gezielt Informationen aus dem Dokument herrausfiltert, die nicht in Astoria abgelegt werden sollen. Eine andere Anwendung ist das Vermerken des Importdatums oder des Ablageorts innerhalb von Astoria im Dokument.

1. Der Eventbehandler wird durch eine C++-Unterroutine realisiert. In den Konfigurationsdaten kann man die Adresse dieser Unterroutine in den Konfigurationsdaten ablegen. Soll dieser Mechanismus nicht genutzt werden, dann gibt man keine Adresse einer solchen Unterroutine an.

Bild 33 zeigt durch die Symmetrie, daß Import und Export auf ähnliche Weise funktionieren. Wenn ein Dokument exportiert werden soll, zeigt man auf das zu exportierende Dokument und gibt als Zielort den Pfad im Dateisystem und den Dateinamen an. Der Exportvorgang kann ebenfalls durch die Konfigurationsdaten beeinflußt werden. Beispielsweise kann der Exportakteur Dokumente im SGML- oder im XML-Format abspeichern. Die Information, was der Exportakteur in die Datei zu schreiben hat, erhält dieser, in dem er den Zustand einer Gruppe von Objekten, die z.B. zu einer Workunit gruppiert worden sind, betrachtet.

Beim Exportakteur kann der Eventbehandler die Datei modifizieren, in die Astoria das zu exportierende Dokument schreibt. Anwendungen dafür wären beispielsweise das Einbringen eines Exportdatums in die Datei.

5.5 Berechtigungswesen

5.5.1 Prinzip des Berechtigungswesens

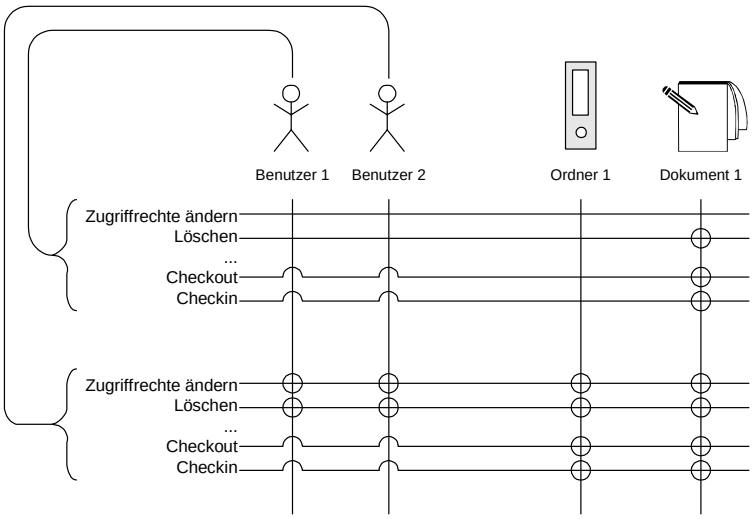


Bild 34: Vergabe von Methodenausführungsrechten an einzelne Benutzer

Astoria bietet vom Konzept her ein sehr universelles Berechtigungswesen. Geschützt wird das Ausführen von Methoden. Dabei kann das Methodenausführungsrecht für jede Methode und für jedes Objekt, das diesen Methodenschutzmechanismus implementiert hat, getrennt vergeben werden. Vergeben wird dieses Recht an Benutzer oder an Gruppen von Benutzern. Bild 34 zeigt ein Beispiel dafür, wie diese Rechte vergeben sein können. In diesem Bild sind keine Benutzergruppen enthalten.

Da es keinen Sinn macht, einen Benutzer auschecken zu wollen, kann dieses Recht auch nicht geschützt werden. Eine Auflistung, welche Methoden von welchem Objekt geschützt werden können finden man in [Astoria_Help_98, Seite 17]. Eine Applikation kann alle Rechte verändern, vorrausgesetzt der aktuelle Benutzer hat die entsprechenden Privilegien.

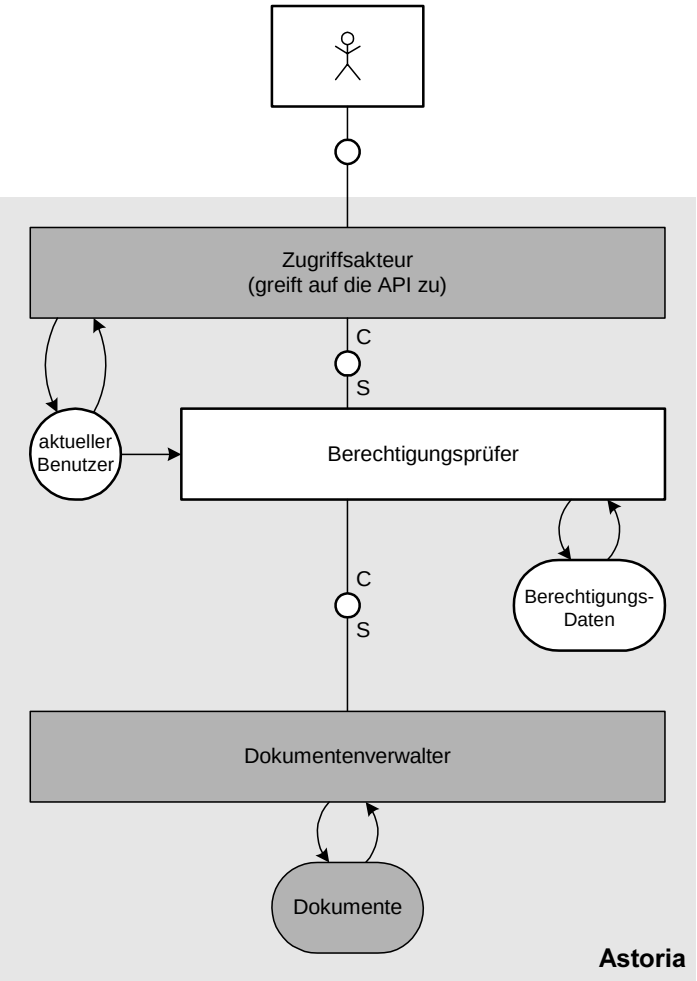


Bild 35: Berechtigungsprüfer in Astoria

In Bild 35 wird die Applikation, die mittels der API erstellt worden ist, als Zugriffsakteur bezeichnet. Von diesem Zugriffsakteur aus, kann kein Auftrag ausgeführt werden, der nicht durch den Berechtigungsprüfer vorher geprüft wird. Allerdings kann mittels der API der aktuelle Benutzer gesetzt werden. Das setzen des aktuellen Benutzers wird nicht durch das Berechtigungswesen geschützt. Es ist also die Aufgabe des Entwicklers, der den Zugriffsakteur entwickelt, daß eine Paßwortabfrage[1] erfolgt.

1. Unter Windows NT wird vom mitgelieferten Navigationstool keine Paßwortabfrage angeboten. Als Benutzer wird der Name des NT-Benutzers gesetzt und man ist automatisch als dieser Benutzer eingeloggt.

Das Beispiel in Bild 34 zeigt, daß bei einer sehr geringen Anzahl von Benutzern und Objekten, viele Einträge in die gezeigte Berechtigungsmatrix erfolgen müssen. Ein Teil der Privilegien wird durch den Administrator[1] vergeben. Dazu gehören Rechte, die angeben, ob ein Benutzer sich den Inhalt eines Ordners anschauen darf. Damit der Administrator nicht für jeden neu angelegten Benutzer diese Rechte vergeben kann, gibt es Benutzergruppen. Wie in Bild 36 dargestellt können Gruppen wieder Mitglied in einer Gruppe sein.

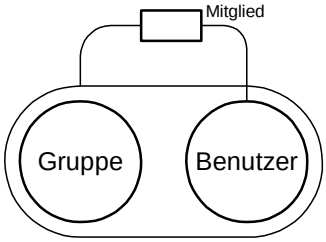


Bild 36: Gruppierung von Benutzern

Außerdem gibt es die Möglichkeit Rechte zu vererben. Das heißt zwischen Objekten gibt es eine Vorfahre-Nachfahre-Beziehung. Der Vorfahre gibt an welche Rechte vererbt werden. Erben heißt in diesem Fall, daß der Nachfahre die Rechte besitzt, die der Vorfahre als vererbbar festlegt. Bei Astoria findet man diese Beziehung zwischen den Containern in der Ablagestruktur. Wenn ein Container I einen anderen Container II enthält, dann ist Container I der Vorfahre von Container II.

In Bild 37 sollen alle Rechte automatisch vererbt werden. Ordner I soll die Wurzel eines Teilbaums in der Ablagestruktur sein. Auf Objekte in diesem Ordner dürfen die Methoden A,B,C und D ausgeführt werden. Ordner II filtert das Ausführungsrecht für Methode B und C aus. Deshalb können nur noch die Methoden A und D ausgeführt werden. Bei diesem Vererbungsmechanismus werden die Rechte herausgefiltert unabhängig davon, um welche Benutzer oder Gruppen es sich handelt.

Wenn man sich vorstellt, daß mit Methode D die Privilegien geändert werden können, dann ist es denkbar, daß Benutzer 1 trotzdem das Recht zum ausführen von Methode C an Gruppe 1 vergibt.

Bei Ordner III wurde das Ausführungsrecht von Methode A sowohl für Benutzer 1 als auch für Gruppe 1 herausgefiltert. Es gibt auch die Möglichkeit, daß Rechte einem Benutzer oder einer Gruppe genommen werden. Im Beispiel hat Benutzer 1 der Gruppe 1 das Recht Methode B auszuführen genommen.

1. Ein Benutzer, der Mitglied in der von Astoria fest angelegten Administratoren-Gruppe ist, hat alle Rechte auf alle Objekte. Es gibt keine Möglichkeit dessen Rechte einzuschränken.

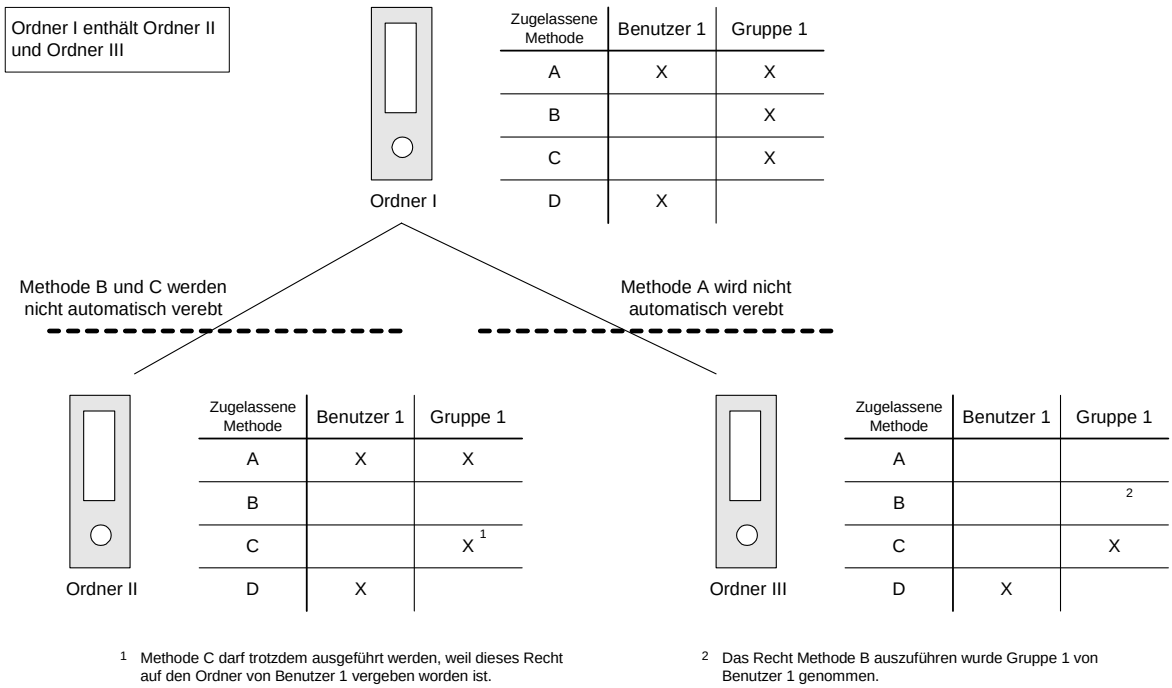


Bild 37: Vererbung von Rechten in der Ablagestruktur

5.5.2 Realisierung des Berechtigungswesen

Die in den vorangegangenen Beispielen geszeigten Relationen werden durch Referenzen realisiert. In diesem Kapitel soll lediglich gezeigt werden, wie die Klassen der API heißen. Außerdem wird eine kompakte Darstellung aller Relationen gegeben.

Die Klasse NamedControlledObject in Bild 38 ist die Oberklasse aller Objekte, die am Berechtigungswesen teilnehmen. Von jedem dieser Objekte gibt es eine Teilmenge von Methoden, die geschützt werden können. Diese Methoden sind die Capabilities des Objekts. Für jede dieser Capabilities gibt es eine CapabilityGroup. Die GroupItem-Objekte, die sich als Mitglied bei einem CapabilityGroup-Objekt angemeldet haben, besitzen die Erlaubnis die entsprechende Methode auszuführen. Damit sich nicht jeder in allen Gruppen anmelden kann, ist die Anmelde-Methode ebenfalls über diesen Mechanismus geschützt.

Ein Benutzer (User) hat automatisch alle Rechte der Gruppen (Group), in denen er Mitglied ist. Eine Gruppe vererbt quasi alle ihre Rechte an seine Mitglieder. Mitglied werden in einer Gruppe (Group) kann man sich so vorstellen, daß man dann automatisch auch Mitglied in allen CapabilityGroups wird, in der die Group Mitglied ist.

Die Vererbungsrelation, die zwischen FolderItems besteht ist in Bild 38 nicht dargestellt. Es ist lediglich angedeutet, daß Capabilities vererbt werden können. Wenn ein Capability vererbt wird, werden die erbenden Objekte Mitglied in den entsprechenden CapabilityGroups.

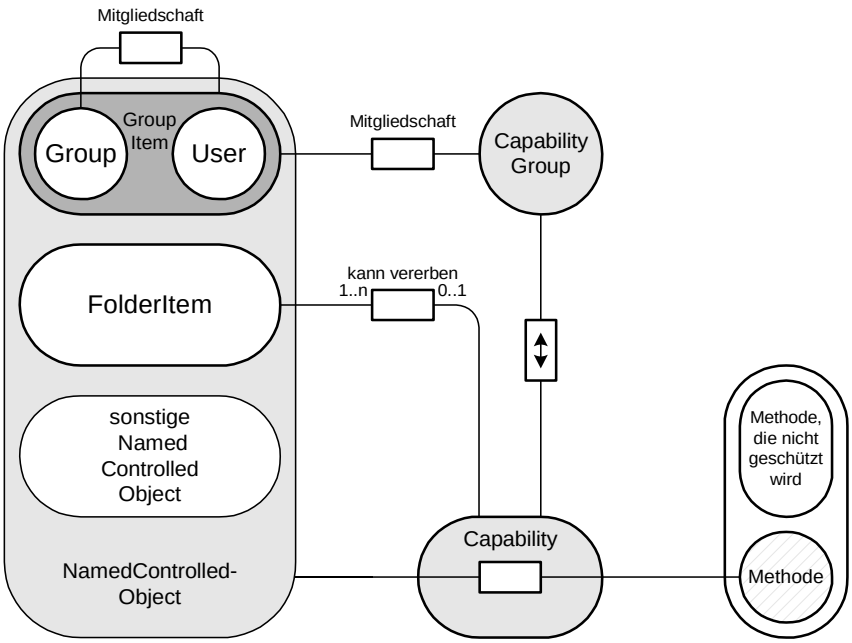


Bild 38: Klassen, die am Berechtigungswesen beteiligt sind

5.6 Attribute, die vom Benutzer definiert werden

5.6.1 Konzept der Benutzerattribute

Ein Dokument besitzt Attribute, wie Name oder Erstellungsdatum. Diese Attribute sind von Astoria fest vorgegeben. Damit man nicht auf die fest vorgegebenen Attribute beschränkt ist, besteht die Möglichkeit als Benutzer Attribute zu definieren und diese an ein Dokument zu heften. Prinzipiell können diese Attribute an alle Objekte geheftet werden, die am Berechtigungswesen teilnehmen oder die versioniert werden können. Die mitgelieferten Tools lassen die Definition von solchen Attributen nur mit den Administrationstools zu. Dort wird Name und Wertebereich des Attributs festgelegt. Es gibt vier Arten von Wertebereichen:

(a) Wertebereich Kalenderdatum

(b) Wertebereich Text

(c) Wertebereich natürliche Zahlen

(d) Aufzählungswertebereich, mit benutzerdefinierten Werten

Bei der Definition eines Attributs mit Wertebereich (a), (b)oder (c) werden die Werte durch das Trägersystem festgelegt. Wertebereich (d) wird definiert, in dem im Administrationstool festgelegt werden, welche Werte zu diesem Wertebereich gehören. Es ist aber nicht zwingend erfor-

derlich, daß diese Attribute in einen Administrationstool definiert werden. Ein selbst programmiertes Tool, das die API nutzt, kann ebenfalls die Methoden zum Definieren von Attributen nutzen.

Wenn der aktuelle Benutzer das Privileg besitzt, benutzerdefinierte Attribute anheften zu dürfen, kann dies mit Hilfe des Navigationstools geschehen. Der Benutzer sieht das Repertoire von definierten Attributen und kann diese an Objekte in der Ablagestruktur vergeben. Daher muß dies in einem Tool erfolgen, mit dem man auch die Objekte identifizieren kann, die ein zusätzliches Attribut erhalten sollen.

5.6.2 Realisierung des Benutzerattributekonzepts

Zur Definition von benutzerdefnierten Attributen gibt es für jeden Attributtyp eine Klasse. Diese sind Unterklassen der CustomAttributeDefinition-Klasse, die durch die Partition in Bild 39 dargestellt sind. Ein Objekt dieser Klasse repräsentiert einen benutzerdefinierten Typ. Das Anlegen eines solchen Typs soll am Beispiel eines Aufzählungstyp erklärt werden:

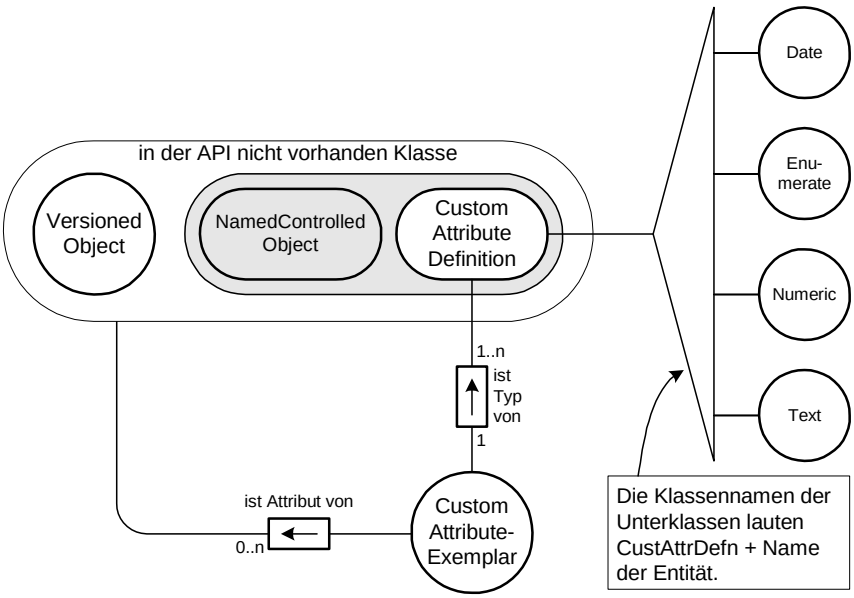


Bild 39: Realisierung von benutzerdefinierten Attributen

Zuerst wird ein Objekt der Klasse CustAttrDefnEnumerate instantiiert. Dieses Objekt ist dann der Akteur, dem der Auftrag gegeben werden kann: „Lege einen neuen Attributstyp namens Dokumentenstatus an, dessen Wertebereich {draft, release} ist.“ In der weiteren Betrachtung ist es sinnvoll dieses Objekt als Speicher für die Typbeschreibung des neuen Dokumentenstatustyps zu sehen.

Wenn an Objekte der Klassen[1] VersionedObject oder NamedControlledObject ein Attribut vom Typ Dokumentenstatus geheftet werden soll, dann geschieht dies über Methodenaufrufe des zu attributierenden Objekts. Der Typ des Attributs wird durch eine Referenz auf die Dokumentenstatus-Beschreibung angegeben, die im Beispiel als Objekt der Klasse CustAttrDefnEnumerate realisiert ist. Zusätzlich muß ein Wert übergeben werden, mit dem das Attribut belegt wird. Damit dieser Wert gespeichert werden kann, muß ein Attributexemplar erzeugt werden, das den Speicher für diesen Wert enthält.

Bild 39 zeigt, daß ein benutzerdefiniertes Attribut wieder an ein benutzerdefiniertes Attribut geheftet werden kann. Dieser Fall wird durch das mitgelieferte Navigationstool nicht abgedeckt. Allerdings ist dies durch API-Aufrufe zu erreichen, da die CustomAttributeDefinition-Klasse eine Unterklasse von NamedControlledObject ist. Damit kann der Zugriff auf diese Attribute durch das Berechtigungswesen geschützt werden.

5.7 Suchmaschine

5.7.1 Funktionsweise der Suchmaschine

Die Suchmaschine in Astoria kann ausschließlich zum Suchen von Dokumenten und Bestandteilen eines Dokuments (Elemente) genutzt werden. Dabei ist es nicht möglich nach einem Ordner oder anderen in der Ablage vorkommenden Objekten wie z.B. Suchausdrücke, zu suchen. Von der Suchmaschine können Dokumente durchsucht werden, die nicht in einem Standardtextformat vorliegen. Um Dokumente lesen zu können, die von WORD oder Interleaf erzeugt wurden, muß die Suchmaschine wissen, wie diese Formate zu interpretieren sind.

Bei der Suche wird ein Suchausdruck angegeben, der eine Umschreibung des Dokuments[2] darstellt. Die Anfrage an die Suchmaschine kann als Auftrag verstanden werden: „Nenne mir alle Dokumente oder Elemente, auf die die Umschreibung durch den Suchausdruck paßt !“ Die Anfrage liefert eine Menge von Objekten zurück. Diese Menge ist eine Teilmenge der Menge aller Dokumente eines bestimmten Typs[3]. Entweder sind alle Typen zugelassen oder nur ein einzelner Typ, wie z.B. DTD-Dokumente. Kombinationen, in denen man zwei Typen als Suchergebnis zuläßt, sind nicht möglich. Alle Dokumente, auf die die Umschreibung durch den Suchausdruck paßt, der aktuelle Benutzer aber keine Zugriffsrecht hat, werden nicht angezeigt.

1. An dieser Stelle tritt der in Kapitel 4.2.2.3 geschilderte Fall auf. Es gibt zwei Klassen, die in unterschiedlichen Ästen des Klassenbaums liegen, die die gleiche Funktionalität anbieten. Dies erkennt man daran, daß die Klassen NamedControlledObject und VersionedObject bezüglich der benutzerdefinierten Attribute exakt die gleichen Methoden anbieten.
2. Meistens wird nach Dokumenten gesucht. Wenn im Folgenden von Dokument gesprochen wird, schließt die auch den Fall mit ein, daß nur nach einem Teil des Dokuments gesucht wird.
3. Als Typen zugelassen sind strukturierte Dokumente, Style-Dokumente, DTDs, Editionen oder Elemente.

Zur Formulierung des Suchausdrucks gibt es ein fest vorgegebenes Repertoire an Schlüsselwörter. Im Folgenden wird eine Grammatik angegeben, mit der ein solcher Suchausdruck erzeugt wird. Es werden die Begriffe verwendet wie sich auch in der Dokumentation von Astoria benutzt werden.

Terminalrepertoire = { Dokumentenname, Dokumenteninhalt, Erzeugungsdatum, Autorname,..., ist gleich, ist ungleich, beinhaltet,..., UND, ODER,...}[1]

Superzeichenrepertoire = { S[2], O[3], T[4], N[5], V[6], W[7] }

Einziges Axiom: S

Ableitungsregeln:
(1) S ⇒ T
(2) T ⇒ NVW
(3) T ⇒ (TOT)

Terminalersetzung:
(4) N ⇒ Dokumentname | Dokumenteninhalt | ...
(5) V ⇒ ist gleich | ist ungleich | beinhaltet |...
(6) W ⇒ Wertebereich Zeichenketten
(7) W ⇒ Wertebereich Zahlen
(8) W ⇒ Wertebereich Daten
(9)[8] O ⇒ UND | ODER | enthält | ist Komponente von

Das einzige Axiom dieser Grammatik ist Suchausdruck (S). Ein Suchausdruck besteht aus einem Term. Dieser Term kann durch eine Sequenz von Nomen, Verb und Wert ersetzt werden. Wenn diese Sequenz von Superzeichen durch die entsprechenden Terminale ersetzt wird, ergibt sich ein Ausdruck der Art „Dokumentennamen ist gleich ’Einführung in Astoria’“. Mit Regel (3) kann eine Verkettung von Termen mittels Operatoren erzeugt werden. Als Beispiel könnte ein solcher Ausdruck wie folgt aussehen: „Dokumentennamen ist gleich ’Einführung in Astoria’ UND Dokumenteninhalt beinhaltet ’Suchmaschine’“.

Ein Suchausdruck kann in der Ablagestruktur abgelegt werden. Damit ist es möglich, die gleiche Anfrage an das System zu einem späteren System zu wiederholen.

1. Eine Aufzählung aller Such-Nomen, -Verben und -Operatoren befindet sich in [Astoria_SDK_98] und in [Astoria_Help_98].
2. Suchausdruck
3. Operator
4. Term
5. Nomen
6. Verb
7. Wert
8. Die letzten beiden Operatoren beziehen sich auf die Suchen von Elementen innerhalb der Dokumentenstruktur.

Da nicht immer die komplette Ablagestruktur durchsucht werden soll, kann man auch den Suchbereich einschränken. Die Einschränkung erfolgt nicht durch ein ein Formulierung eines Terms. Zu jedem Suchausdruck wird angegeben, in welchem Bereich gesucht wird. Es kann allerdings lediglich angegeben werden, in welchen Cabinets gesucht wird. Eine weitere Einschränkung auf Ordner innerhalb eines Cabinets ist nicht möglich.

5.7.2 Realisierung der Schnittstelle zur Suchmaschine

5.7.2.1 Realisierung des Suchausdrucks mit Hilfe der API

In der Astoria-API gibt es eine Reihe von Objekten, mit denen ein Suchausdruck aufgebaut werden kann. Ein Objekt der Klasse Term besitzt Attribute, in denen sich Nomen, Verb und Wert gemerkt werden können. Wenn der komplette Suchausdruck nur aus einem Term besteht, dann wird dieser auch als Rootterm[1] bezeichnet (Bild 40). Identifizierbar ist ein solcher Suchausdruck ein Objekt der Klasse TransientSearch oder Search. Diese beiden Klassen unterscheiden sich darin, daß ein Objekt der Klasse Search in der Ablagestruktur abgelegt werden kann. Daher muß diese Klasse von FolderItem erben. Beim Erzeugen eines TransientSearch-Objekts muß kein Ablageort angegeben werden. Jedes Objekt, egal welche dieser Search-Klassen das Objekt beschreiben, kann eine Relation zu FolderItem-Objekten pflegen. Damit wird der Suchbereich angegeben. Allerdings beschränkt sich in der aktuellen Version die Menge der Objekte auf Objekte der CabinetStub-Klasse. D.h. als Suchbereich kann nur ein oder mehrere Cabinets in Frage kommen.

Ein Suchausdruck kann aus einer Verkettung von mehreren Termen durch Operatoren bestehen. In diesem Fall wird der Suchausdruck ausgehend von einem RootOperator-Objekt aufgebaut. An diesem Objekt können wieder Terme oder Operatoren als Kinder hängen. Dabei gibt es auch die Möglichkeit, daß die Reihenfolge, in der die Terme oder Operatoren als Kinder an das Vater-Objekt gehängt werden, beachtet werden muß. Dies ist genau der Fall, wenn der Operator nicht kommutativ ist, wie beispielsweise der Enthält-Operator.

5.7.2.2 Ausführen einer Suche

Die Objekte der Klasse Search oder TransientSearch können auch als Akteur gesehen werden. Dieser Akteur erhält den Auftrag eine Suche auszuführen und liefert eine Liste mit den Suchergebnissen zurück. Das Objekt selbst dient dabei als Identifikator für den Suchausdruck.

5.7.2.3 Integration der Suchmaschine in Astoria

Bild 41 zeigt ein Modell, das eine mögliche Integration der Suchmaschine in Astoria zeigt. Zuerst soll das Einbringen eines Dokuments erklärt werden. Danach wird der Suchvorgang beschrieben, der durch den Benutzer angestoßen werden kann.

1. In der API gibt es keine Klasse namens Rootterm oder Rootoperator.

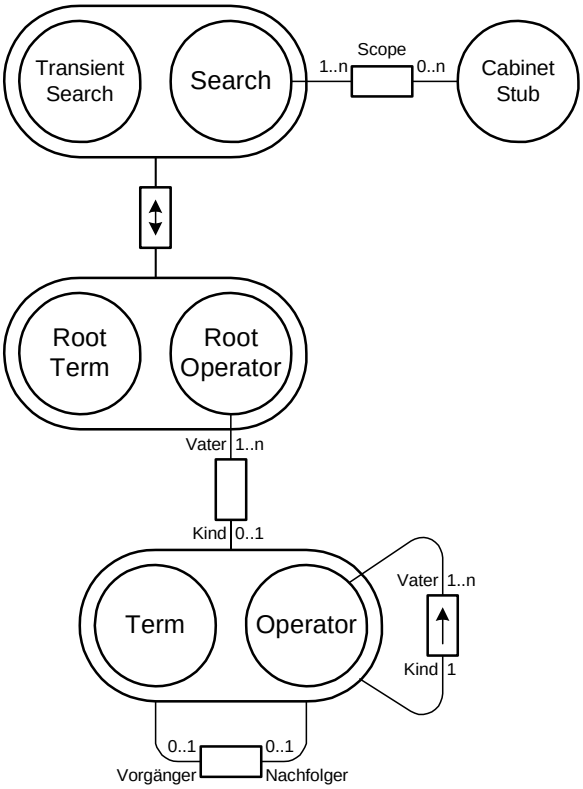


Bild 40: Objekte, die an der Suche beteiligt sind

Wenn ein Dokument in Astoria eingebracht wird, wird es automatisch von der Suchmaschine durchsucht. Dabei wird für jedes Wort, das nicht „der, die, das“ o.ä. ist, ein Eintrag in einer Liste angelegt. Dieser Eintrag besteht aus dem Schlüsselwort, das Cabinetfile, in dem das Dokument abgespeichert wird und eine Referenz, mit der das Objekt für den Dokumentenverwalter identifiziert wird. Auf weitere Ablageinformation hat die Suchmaschine keinen Zugriff. Diese Ablageinformation beschreibt, wo innerhalb der Ablagestruktur das Dokument abgelegt wird.

Dokumente können gesucht werden, in dem man einen Suchausdruck angibt. Außerdem muß der Benutzer angeben, in welchen Cabinets gesucht werden soll. Wenn die Suchmaschine den Auftrag erhält, eine Suche auszuführen, dann muß sie den Suchausdruck interpretieren. Mittels der Schlüsselwortliste können die in Frage kommenden Dokumente ermittelt werden. Man kann auch nach Texten suchen, die aus mehreren Wörtern bestehen. Da in dieser Schlüsselwortliste nicht alle Wortkombinationen gemerkt werden können, wird nach den einzelnen Wörtern gesucht, die in dem gesuchten Text vorkommen. Dokumente, die alle diese Wörter erhalten, werden über den Kanal der Suchmaschine zum Dokumentenvewalter angefordert. Diese Dokumente müssen dann wieder aufs neue durchsucht werden. Bevor die Liste der gefundenen Dokumente an den Benutzer geschickt werden kann, muß der Berechtigungsprüfer gefragt werden, auf welche Dokumente der aktuelle Benutzer Zugriff hat. Dokumente, auf die kein Zugriffsrecht besteht, werden aussortiert. Die Liste der verbleibenden Dokumente wird dann an den Benutzer zurückgegeben.

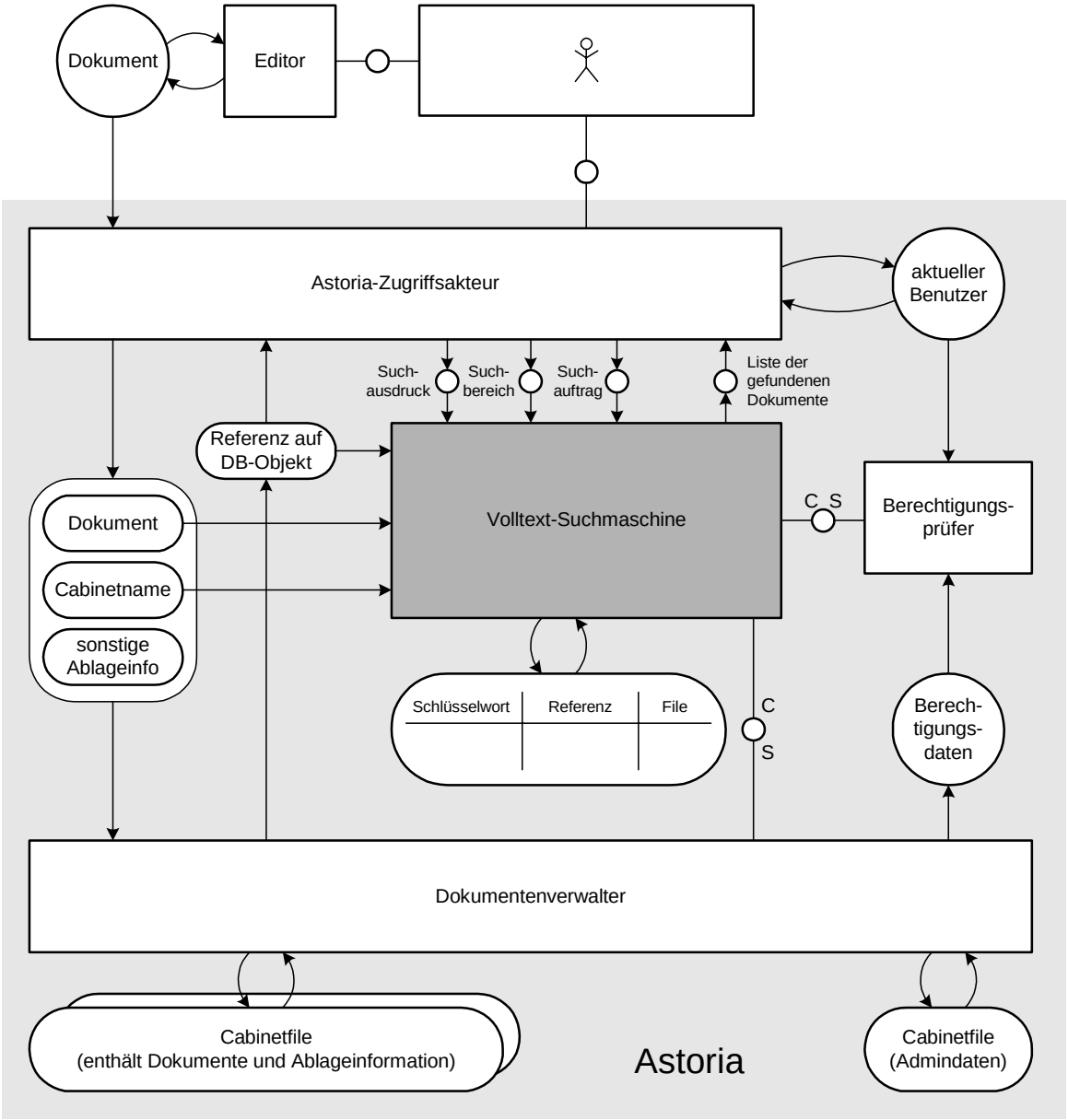


Bild 41: Integration der Suchmaschine

In Bild 41 wurden bewußt die Realisierung der Datenbank gezeigt, die ihre Daten in Cabinetfiles abspeichert. Die Suchmaschine kennt nur Begriffe aus der Filesystem-Welt. Daher kann das Suchen innerhalb der Ablagestruktur von Astoria nicht von der Suchmaschine angeboten werden. Die Ablagestruktur ist der Suchmaschine nicht bekannt.

5.8 Workflow-Möglichkeiten von Astoria

5.8.1 Der Workflow-Begriff

Mit dem Begriff Workflow verbindet man Geschäfts-Prozesse, an denen einzelne Personen in unterschiedlichen Rollen beteiligt sind. Bei diesem Prozeß durchläuft ein Dokument mehrere Stationen, an denen die Person entsprechend ihrer Rolle das Dokument bearbeitet. Wenn ein solcher Bearbeitungsschritt abgeschlossen ist, transportiert das System das Dokument zur nächsten zuständigen Station. Entscheidungen, die die Person während der Bearbeitung getroffen hat, können diesen Prozeß beeinflussen. Ein solcher Workflow-Prozeß soll am Beispiel einer Bestellung erklärt werden, die mit einem Rechensystem verarbeitet wird:

Ein Mitarbeiter einer Firma möchte etwas bestellen. Dazu legt er eine Bestellung im Rechner an. Damit beginnt der Workflow-Prozeß. Die Bestellung wird zum Vorgesetzten des Mitarbeiters transportiert, der diese Bestellung genehmigt, bevor sie in den Einkauf gelangt. Wenn der Vorgesetzte die Bestellung ablehnt, wird sie an den Mitarbeiter mit dem Hinweis, daß die Bestellung abgelehnt wurde, geschickt. Damit ist der Workflow-Prozeß beendet. Sollte die Bestellung genehmigt werden, ist diese nun im Einkauf. Auf diese Art und Weise durchläuft die Bestellung einzelne Stationen, bis sie zum Besteller mit dem Hinweis auf Auslieferung des Produkts zurückkehrt.

5.8.2 Workflow in Astoria

Astoria bietet die Möglichkeit, daß Aktionen[1], an denen Dokumente beteiligt sind, einen oder mehrere Prozesse starten können. Diese Prozesse werden bei Astoria durch C++-Methoden beschrieben. Realisiert werden diese C++-Routinen in Windows-DLLs[2].

In der Astoria-Konfigurationsdatei werden die DLLs registriert, womit die Existenz dieser DLLs in Astoria bekannt sind. In Astoria können Dokumente Gruppiert werden, die diese C++-Routine aufrufen, wenn beispielsweise ein Dokument ausgecheckt wird. Der Name der Gruppe muß identisch mit dem Eintrag in der Konfigurationsdatei sein.

Da der dieser Mechanismus in Astoria sehr stark mit der Plattform, nämlich Windows und der API verbunden ist, erfolgt eine genauere Erklärung im nächsten Kapitel.

1. Diese Aktionen sind: Check in, Check out, Wiederrufen von Check out, Abspeichern des Dokuments und Erzeugen einer Edition.
2. DLL: Dynamic Link Library. Mechanismus von Windows, der es ermöglicht Funktionsbibliotheken auszutauschen, ohne daß eine Applikation, die diese Bibliothek nutzt, neu compiliert werden muß. Diese Bibliothek liegt in Form einer Datei mit der Endung .dll vor.

5.8.3 Realisierung des Workflow-Mechanismus

Eine Gruppe von versionierten Objekten, die solche Prozedur aufrufen können, wird in Astoria als Link bezeichnet. Die versionierten Objekte, die diese Fähigkeiten besitzen, sind die Stellvertreter für Dokumente oder Bestandteile davon. Gezeigt werden diese in Bild 42.

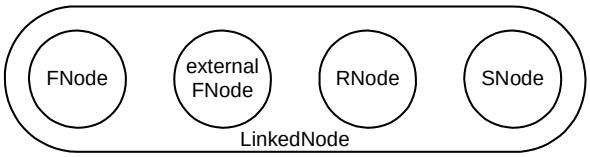


Bild 42: Objekte, die am Workflow teilnehmen können

Da eine Aktion, die ein solches Objekt betrifft, nicht nur eine einzelne Prozedur aufrufen kann, gibt es eine Klasse von Objekten, die die Relation zwischen LinkedNode-Objekten und Link-Objekten realisiert. Diese Objekte sind von der Klasse LinkItem.

Der Identifikator für die Prozedur, die in der DLL aufgerufen wird, ist nicht ein Objekt aus der Klasse Link. Dafür gibt es wieder eine eigene Klasse namens Trigger. Damit ist es möglich, daß unterschiedliche Objekt-Gruppen die gleiche Prozedur aufrufen.

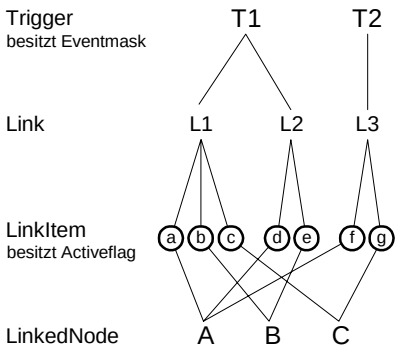


Bild 43: Gruppierung der Objekte

Bild 43 zeigt exemplarisch, wie eine Gruppierung aussehen kann. LinkedNode A ist in allen drei Links enthalten. Wenn das Dokument ausgecheckt wird, das Objekt A enthält, wird die Prozedur, die durch Trigger T1 identifiziert wird, zweimal aufgerufen (Gruppierung A-(a)-L1-T1 und A-(d)-L2-T1). Danach wird die T2-Prozedur aufgerufen. Wenn man verhindern will, daß T1 zweimal aufgerufen wird, gibt es die Möglichkeit, daß das Active-Flag von LinkItem-Objekt (a) auf passiv gesetzt wird. Damit wird Trigger T1 nur noch über die Verbindung LinkedNode A - LinkItem (d) - Link L2 zum Aufruf der Prozedur genutzt.

Für den Fall, daß die Trigger-Prozedur nur bei einer bestimmten Aktion aufgerufen werden soll, besitzt ein Trigger-Objekt eine Eventmask. Diese enthält für jede Aktion, die einen solchen Prozeduraufruf veranlassen kann, ein Bit. Wenn das Bit gesetzt ist, wird die Prozedur aufgerufen. Sollte das Bit nicht gesetzt sein, wird die Prozedur nicht aufgerufen.

Registrierung einer Trigger-Prozedur

Ein Trigger-Objekt besitzt das Attribut Name. Dieser Name muß identisch sein mit einem Eintrag in der Konfigurations-Datei von Astoria. Der Eintrag kann wie folgt aussehen:

```
[Trigger]
T1 = C:\Atoria\Trigger\Beispiel.dll
```

In diesem Fall muß das Trigger-Objekt den Namen „T1“ besitzen. Da in einer DLL-Datei eine ganze Funkionsbibliothek abgespeichert werden kann, muß ein Name der aufzurufenden Prozedur vereinbart ebenfalls werden. Diese Prozedur heißt in Astoria: XD_FireTrigger

Nun ist es dem Anwender freigestellt in der Konfigurationsdatei mehrere unterschiedliche Trigger-Objekte die gleiche Prozedur aufzurufen. Die könnte durch folgenden Eintrag realisiert werden:

```
[Trigger]
T1 = C:\Atoria\Trigger\Beispiel.dll
T2 = C:\Atoria\Trigger\Beispiel.dll
```

Wenn man Bild 43 betrachtet dann könnte es passieren, daß die gleiche Prozedur dreimal hintereinander aufgerufen wird, wenn eine Aktion bezüglich LinkedNode A einen Prozeduraufruf triggern kann. Dann wird die XD_FireTrigger-Prozedur zweimal durch T1 und einmal durch T2 aufgerufen.

6. Bewertung

Als Maßstab für die Bewertung von Astoria soll die Nutzbarkeit als Plattform für das TimeLess-Softwaresystem dienen. Dazu soll zunächst beschrieben werden, welches Konzept TimeLess bezüglich eines Concerns verfolgt. Danach wird eine Realisierungsmöglichkeit angegeben. Schließlich wird eine Bewertung gegeben.

Im letzten Teil dieses Kapitels werden alle Ergebnisse der Bewertung zusammengefaßt. Außerdem sollen Punkte angesprochen werden, die die Handhabung des System betreffen.

6.1 Die Ablagestruktur

6.1.1 Konzept der Ablage in TimeLess

Die Ablagestruktur in TimeLess ist an einer Bibliothek orientiert wie sie auch in der Realität vorkommt. Es gibt unterschiedliche Containertypen, die auch in einer Bibliothek beobachtet werden können. In TimeLess geht man davon aus, daß sich die Bibliothek auf einem Campus innerhalb eines Landes befindet. Damit ist ein Land der Wurzelcontainer.

Jeder Containertyp hat eine räumliche Ausdehnung. Die Größe eines Containers soll ein Maß für die Informationsmenge sein, die in diesem Container enthalten ist. Dabei ist festgelegt, daß nur Container eines bestimmten Typs in einem anderen enthalten sind. Diese Beziehung ist in Bild 44 dargestellt. Damit diese Ablagestruktur nicht durch den Benutzer geändert werden kann, wird dieses durch das TimeLess-Softwaresystem überwacht. Außerdem gibt es Prüfungen, mit denen die korrekte räumliche Anordnung gewährleistet wird. D.h. man kann beispielsweise nicht einen Saal innerhalb einer Etage dorthin plazieren, wo schon ein anderer Saal ist.

Mit der Nachbildung einer Welt, die der Benutzer des Systems kennt, möchte man zum einen die Einarbeitungszeit in das System klein halten. Zum anderen soll das unbewußte Sammeln von Information gefördert werden. D.h. wenn man in dieser Ablagehierarchie navigiert, kommt man durch vielen unterschiedlichen Orten vorbei. Die grafische Darstellung der Container soll es erleichtern, sich diese Orte zu merken. Wie gut dies funktioniert hängt allerdings von der grafischen Darstellung und dem Aufbau der Ablagestruktur ab. Dabei soll die Ablagestruktur so aufgebaut werden, daß eine Klassifizierung des Dokuments vorgenommen wird.

Da TimeLess als Archivsystem geplant wird, werden Dokumente in TimeLess grundsätzlich nicht geändert werden. Daher hat man die Vorstellung, daß ein Nutzer des Systems immer nur auf Kopien zugreift. Das Originaldokument ist vom Benutzer nicht zugänglich. Originale können mit den Druckplatten verglichen werden, die im Keller eines Verlags liegen. Diese werden weder verändert, noch verkauft.

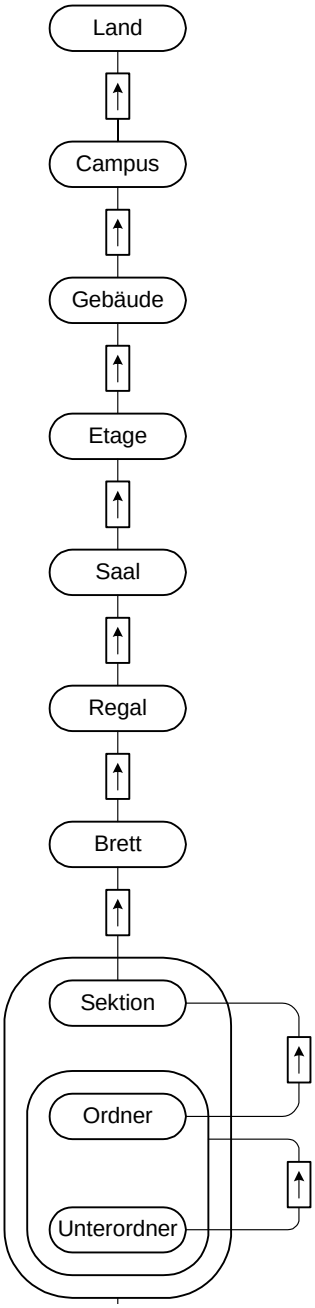


Bild 44: TimeLess-Ablagestruktur[a]

a.Dieses Bild wurd aus [Brand_97] übernommen

Wenn ein Benutzer ein Dokument in TimeLess einbringen will, muß er in der Rolle eines Autors beim System angemeldet sein. Der Autor kann sein Dokument bei einem Bibliothekar abgeben, der für einen Teilbereich der Ablage zuständig ist.

6.1.2 Realisierung der Ablagestruktur mittels Astoria

Dem TimeLess-Team ist es bewußt, daß je näher man zum Benutzer kommt, desto geringer ist die Chance, daß ein System diese Anforderungen abdeckt. Gerade die Benutzeroberfläche (siehe Bild 45) wird es in dieser Form nicht zu kaufen geben.

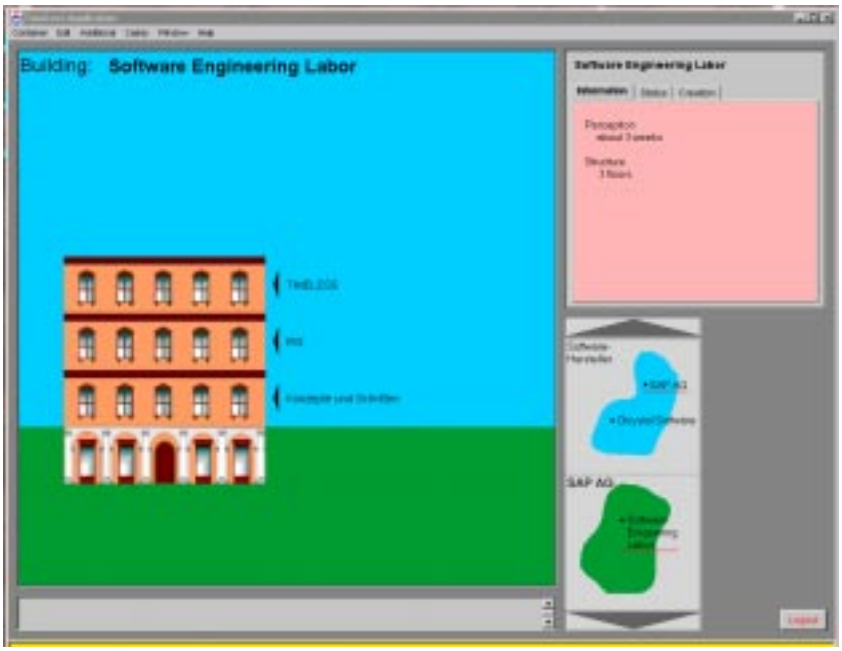


Bild 45: Benutzeroberfläche des TimeLess-Prototyps

Die Anforderungen an die Ablagestruktur sind Eng mit dieser grafischen Darstellung der Oberfläche verbunden. Genau wie die grafische Darstellung, ist es sehr unwahrscheinlich, daß ein Dokumentenmanagementsystem eine ähnliche Ablagestruktur besitzt. Wie eine Anpassung der beiden Konzepte erfolgen kann, soll im Folgenden erklärt werden.

Astoria bietet eine hierarchisch organisierte Ablage. Da es in Astoria nur zwei Containertypen gibt, muß die Typisierung von Containern durch das TimeLess-Team realisiert werden. Die Typinformation kann über ein benutzerdefiniertes Attribut an den Container geheftet werden. Im Zusammenhang mit der Ablagestruktur stellt das Anbieten von benutzerdefinierten Attributen einen sehr leistungsfähigen Mechanismus dar (Bild 46). Damit können sich nämlich auch die Informationen gemerkt werden, die die räumliche Position des Containers betreffen.

Die Interpretation der Daten, die in den benutzerdefinierten Attributen gemerkt werden, muß ebenfalls vom TimeLess-Team implementiert werden, da Astoria deren Bedeutung nicht kennt. Konkret heißt dies, daß das Überprüfen der Beziehungen zuneinander, die sich auf ihre räumliche Lage beziehen, zusätzlich implementiert werden müssen. Die Prüfungen werden auch als *logische Prüfungen* bezeichnet.

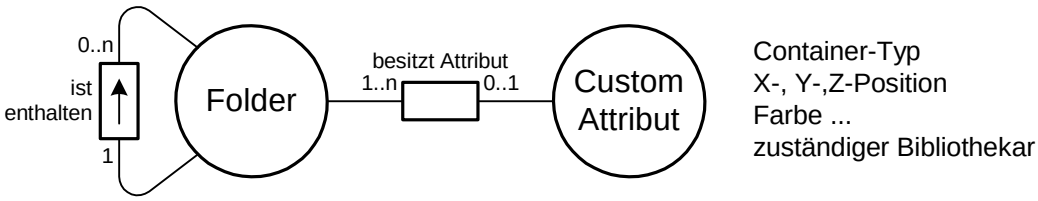


Bild 46: Realisierung der Ablagestruktur mittels Astoria

Wenn man beispielsweise ein Regal innerhalb eines Saals verschieben will, reicht es nicht aus, daß nur logische Prüfungen erfolgen. Es muß auch eine sogenannte *juristische Prüfung* erfolgen. Damit ist gemeint, daß geprüft werden muß, ob ein Benutzer auf Grund seiner Privilegien das Regal überhaupt verschieben darf. Da diese räumliche Verschiebung Astoria nicht bekannt ist, kann sie auch nicht vom Berechtigungswesen abgedeckt werden. Was geschützt werden kann ist das Ändern von benutzerdefinierten Attributen. Näheres dazu siehe Kapitel 6.4.2.

Die Realisierung des Zugriffs auf Kopien in der Ablage könnte wie folgt realisiert werden. Es gibt die Möglichkeit Referenzen auf Dokumente zu erzeugen und abzulegen. Dazu muß die Ablage in zwei Teile geteilt werden. Der Teil der Ablage, der die Dokumente enthält soll für den Benutzer nicht sichtbar sein. Im anderen für den Benutzer sichtbaren Teil der Ablagestruktur werden lediglich Referenzen abgelegt. Welchen Teil der Ablage der Benutzer sieht, kann durch die Anwendung bestimmt werden. Problematisch wird allerdings dann das Nutzen der Suchmaschine, da diese immer nur die Originaldokumente findet. Von diesem Originaldokument gibt es allerdings keine Möglichkeit auf die Referenzen zu schließen, die auf das Dokument zeigen. Daher bleibt hier nur die Möglichkeit, das gleiche Dokument an mehreren Stellen der Ablage abzulegen. Da Anfragen an das System möglich sein sollen, wie „wo steht denn das gleiche Dokument noch in der Ablage ?“ müssen die Relationen zwischen den einzelnen Dokumenten in der auf Astoria aufsetzenden Anwendung gepflegt werden. Dann bleibt allerdings die Frage, wie man diese Relationen in Astoria ablegt. Es wäre denkbar einen Containern einzurichten, der der Anwendung zur Ablage von Konfigurationsdaten dient. Die Dokumente, die in diesem Container enthalten sind, könnten zusätzliche Informationen, die nur die Anwendung interpretieren kann, enthalten.

6.1.3 Bewertung der Realisierungsmöglichkeiten bezüglich der Ablage

Die Containerhierarchie von TimeLess ist mit Astoria mit vertretbarem Aufwand zu realisieren. Im vorangegangenen Kapitel wurde eine Realisierungsmöglichkeit mittels der benutzerdefinierten Attribute gezeigt.

Die Konzepte bezüglich der Dokumentenablage unterscheiden sich in Astoria und TimeLess. In Astoria hat der Benutzer immer das Originaldokument im Zugriff. Dagegen hat der TimeLess-Benutzer immer nur Kopien des Dokument im Zugriff. Der Ablageort dieser Kopie ist von großer Bedeutung, da an diesem Ort auch Kopien von Dokumenten stehen können, die mit diesem Thema verwandt sind. Außerdem sollen dieses Kopien an mehreren Orten in der Ablage stehen können. Die Realisierung dieses Konzepts ist mit Astoria nur mit größerem Aufwand möglich.

6.2 Versionierung

6.2.1 Das Versionierungs-Konzept in TimeLess

Da Dokumente in TimeLess nicht geändert werden, muß keine automatische Versionierung erfolgen. Es soll jedoch möglich sein, daß ein Dokument bei einem Bibliothekaren abgeliefert mit dem Hinweis abgeliefert wird, daß es sich dabei um die Nachfolge-Version des Dokuments handelt. Außerdem muß dann eine Begründung abgegeben werden, wo die Unterschiede zwischen diesen Versionen liegen.

6.2.2 Realisierung des Versionierungskonzepts mittels Astoria

Das Versionierungskonzept von Astoria ist zur Realisierung des TimeLess-Versionierungskonzepts nicht geeignet. In Astoria wird ein Exemplar eines Dokuments in der Ablage identifiziert, das Stellvertreter für alle Versionen dieses Dokuments ist. In TimeLess sollen alle Versionen eines Dokuments sichtbar in der Ablage sein, wobei diese Versionen an unterschiedlichen Orten in der Ablage sein können. In Astoria liegen alle Versionen am selben Ort.

Da es in TimeLess aber diese Versionsbeziehung zwischen Dokumenten geben soll, muß diese Beziehung realisiert werden. Hier taucht wieder das Problem auf, daß Beziehungen zwischen Objekten der Ablagestruktur verwaltet werden müssen, die in der gewünschten Form nicht durch Astoria unterstützt werden.

6.2.3 Bewertung der Realisierungsmöglichkeiten bezüglich des Versionierungskonzepts

Das Versionierungskonzept von Astoria kann nicht für den Einsatz in TimeLess genutzt werden. Zur Realisierung dieses Konzepts ist eine komplette Neuimplementierung durch das TimeLess-Team notwendig.

6.3 Strukturierte Dokumente

Das Konzept der strukturierten Dokumente in TimeLess wurde in Kapitel 2 ausführlich besprochen. Eine Beurteilung des Konzepts in Astoria kann nicht gegeben werden, da von den TimeLess-Anforderungen nur die Benutzersicht spezifiziert ist. Es wurde noch keine Entscheidung getroffen, welche Standards zum Strukturieren von Dokumenten genutzt werden. Die Angebotene Funktionalität löst nicht alle Probleme, die mit der Realisierung der gestellten Anforderungen verbunden sind, was allerdings an den genutzten Standards und nicht am Konzept von Astoria liegt.

6.4 Berechtigungswesen

6.4.1 Das Berechtigungswesen von TimeLess

Für das Berechtigungskonzept von TimeLess gibt es zum Abgabezeitpunkt der Arbeit noch keine konkrete Spezifikation. Allerdings gibt es Vorstellungen, wie dieses Berechtigungskonzept aussehen könnte.

Ein und dieselbe Person kann unterschiedliche Rollen haben. Diese Rollen sind Autor, Leser und Bibliothekar. Mit jeder Rolle ist der Besitz einer Menge von Schlüsseln verbunden, die als Gruppe von Privilegien gesehen werden können. Neben der Gruppierung von Rechten, die an einen Schlüssel gebunden sind, gibt es einige Ideen, die nun vorgestellt werden sollen:

Üblicher Weise gibt es in einem Mehrbenutzersystem Benutzer, die als Administratoren bezeichnet werden. Diese besitzen alle Rechte. Im TimeLess-Softwaresystem soll es keinen Benutzer geben, der alle Privilegien besitzt. Jeder Benutzer soll einen privaten Bereich besitzen. Dieser Bereich wird in [Brand_97] als Büro bezeichnet. Wenn dem Mensch, der hinter der Verwaltungseinheit Benutzer steht, etwas zu stößt, wäre das Büro im System für immer verschlossen. Um dies zu verhindern, soll es möglich sein, daß eine Gruppe von Benutzern einen Einbruch in dieses Büro beantragen kann. Beispielsweise kann die komplette Abteilung in das Büro des Chefs einbrechen, wenn jeder Mitarbeiter damit einverstanden ist. D.h. daß die Menschen miteinander kommunizieren müssen, bevor ein solcher Einbruch erfolgen kann. Alle beteiligten Personen müssen dem Softwaresystem ihre Entscheidung mitteilen. Das Softwaresystem kann nach bestimmten Regeln entscheiden, ob diese Gruppe von Personen eine Einbruchsberechtigung besitzt. Erst dann wird der Einbruch zugelassen.

6.4.2 Realisierung des Berechtigungswesen mittels Astoria

In Astoria gibt es die Möglichkeit Benutzer in Gruppen zusammen zu fassen. Diese Gruppen können dann wieder Mitglied in einer anderen Gruppe sein. Mitgliedschaft in einer Gruppe bedeutet, daß das Mitglied alle Rechte der Gruppe besitzt. Damit ist die Gruppierung von Rech-

ten möglich. Wenn man eine Sorte Gruppen als Rechte sieht und ein die anderen Gruppen als Schlüssel, dann ist eine Gruppe, die als Rolle gesehen werden kann, Mitglied in einer Gruppe, die als Schlüssel gesehen werden kann. Diese Beziehung wird selten geändert. Dagegen muß die Mitgliedschaft des einzelnen Benutzers in einer Rollengruppe häufig geändert werden, da dieser immer nur in einer Rolle im System eingeloggt sein kann. Bei einem Wechsel der Rolle vom Leser zum Bibliothekar muß der Benutzer aus der Leser-Gruppe entfernt werden und Mitglied in der Bibliothekars-Gruppe werden.

Das Berechtigungswesen von Astoria bietet die Möglichkeit, die Rechte für einzelne Methodenaurfrufe zu vergeben. Damit ist das Berechtigungswesen sehr universell gehalten. Allerdings gibt es ein fest vorgeschriebenes Repertoire von Methoden für jedes Objekt, das geschützt werden soll. Wenn in der Applikation, die die API nutzt, eine Methode geschützt werden soll, die nicht durch dieses Repertoire vorgesehen ist, gibt es keine Möglichkeit diese Methode beim Berechtigungswesen anzumelden. Diese Problem soll an einem Beispiel verdeutlicht werden.

In Kapitel 6.1.2 wurde bereits angesprochen, daß es eine dreidimensionale Ablagewelt in TimeLess gibt, die Astoria nicht bekannt ist. Die Koordinaten des Containers im dreidimensionalen Raum sollten mittels benutzerdefinieren Attributen gemerkt werden. Als Operationen, mit denen die Koordinaten eines Containers geändert werden können, sind Drehen und Schieben vorstellbar. Über das Berechtigungswesen ist allerdings nur das Ändern der benutzerdefinierten Attribute (Koordinaten) schützbar. Wenn ein Benutzer nur das Recht hat das Regal zu drehen, aber nicht zu verschieben, dann kann das Astoria-Berechtigungswesen diese beiden Operationen nicht unterscheiden[1]. Durch je ein Benutzerattribut für Drehen und Schieben könnte zwar mittels einer Prüfung, ob dieses Attribut geändert werden darf, festgestellt werden, ob der aktuelle Benutzer die entsprechenden Privilegien besitzt. Allerdings müßte dann ein Teil der Berechtigungsprüfung in der selbst erstellten Applikation erfolgen.

In Astoria hat ein Administrator automatisch alle Rechte. Um zu gewährleisten, daß für jeden Benutzer ein privater Bereich vorhanden ist, darf es keinen Benutzer mit Administratorrechten geben. Die Realisierung des Abstimmverfahrens, bei dem entschieden wird, ob eingebrochen werden darf oder nicht, muß ebenfalls durch das TimeLess-Team realisiert werden.

6.4.3 Bewertung der Realisierungsmöglichkeiten bezüglich des Berechtigungswesens

Gruppierung von Rechten ist durch das Berechtigungswesen von Astoria sehr leicht abdeckbar. Mit dem angebotenen Gruppierungsmechanismus ist es auch möglich, Benutzerrollen zu realisieren. Problematisch ist, daß das Repertoire an vergebbaren Rechten nicht erweitert werden kann. Außerdem muß das Konzept des beantragten Einbruchs komplett neu implementiert werden.

1. Auch wenn dieser Fall konstruiert erscheinen mag, veranschaulicht er sehr deutlich die Einschränkungen, die durch das Berechtigungswesen von Astoria hingenommen werden müssen.

Das Berechtigungswesen von Astoria deckt einen Teil der Funktionalität ab, der den TimeLess-Anforderungen entspricht. Allerdings ist es notwendig, daß eine großer Teil an Funktionalität zusätzlich selbst gebaut werden muß.

6.5 Die Suchmaschine

6.5.1 Die Suchmaschine in TimeLess

Der Zugriff auf Dokumente soll in TimeLess erst in zweiter Linie durch eine Suchmaschine erfolgen. Die Ablagestruktur spielt in TimeLess eine zentrale Rolle. Durch eine gut organisierte Ablagestruktur soll es einfach sein, zu den Dokumenten hin zu navigieren. Beim Navigieren sollen sich Orte in der Ablage gemerkt werden. Vielleicht weiß man aber nur noch den Namen eines Regals und hat vergessen, wo es steht. Daher soll es in TimeLess möglich sein, per Suchmaschine diese Orte aufzufinden.

Da Suchmaschinen oft eine unüberschaubare Menge an Dokumenten finden, ist die Wahrscheinlichkeit sehr groß, daß viele Dokumente gefunden worden sind, die mit dem gewünschten Thema nichts zu tun haben. In TimeLess soll es daher möglich sein, den Suchbereich sehr stark einzuschränken. Beispielsweise könnte man sich vorstellen, daß man vor einem Regalbrett steht, auf dem die Dokumentation zu Astoria steht. Nun sucht man nach allen Dokumenten, die das Wort „Trigger“ enthalten. Damit ist sichergestellt, daß nur Dokumente gefunden werden, in denen der Begriff „Trigger“ mit dem Workflow-Mechanismus vom Astoria verbunden wird. Ohne die Einschränkung des Wertebereichs könnten beispielsweise auch Anleitungen zu Oszilloskophen gefunden werden. Dabei ist alleine durch den Ort, an dem sich der Benutzer befindet, ersichtlich, daß die Verwendung des Wortes „Triggers“ im Astoria-Sprachgebrauch gemeint war.

6.5.2 Realisierung der Suchmaschine mittels Astoria

Die Suchmaschine von Astoria liefert lediglich Dokumente als Suchergebnis. Damit ist es nicht möglich nach einem Container zu suchen. Außerdem sind die Möglichkeiten, den Suchbereich einzuschränken, auf Cabinets beschränkt. Ein Cabinet ist allerdings ein Wurzelcontainer, der vergleichbar mit einem Land in TimeLess ist. Alle Untercontainer müssen als Folder realisiert werden. Mittels Eigenimplementierung ist es aber möglich, alle Dokumente auszusortieren, die nicht in einem gewissen Bereich sind.

Es ist zwar möglich, die Suche nach Containern selbst zu bauen. Ein Vorteil, den eine Suchmaschine bietet, ist, daß komplexe Suchausdrücke angegeben werden können. Die Auswertung diese Ausdrücke müßte ebenfalls vom TimeLess-Team selbst implementiert werden.

6.5.3 Bewertung der Realisierungsmöglichkeiten bezüglich der Suchmaschine

Die Suchmöglichkeiten von Astoria decken einen kleinen Teil der TimeLess-Anforderungen ab. Das Fehlen einer Bereichseinschränkung, die sich auf die Ablage bezieht, macht eine Aufwendige Eigenimplementierung notwendig. Außerdem ist eine Suche nur nach Dokumenten möglich. Eine Suche nach Containern ist nicht möglich.

6.6 Das Workflow-Konzept

6.6.1 Das Workflow-Konzept in TimeLess

Bisher gibt es nur an einer Stelle eine konkrete Vorstellung, wo ein Workflow-Konzept benötigt wird. Dabei handelt sich um das Einbringen von Dokumenten in TimeLess. An diesem Workflow-Prozeß sind zwei Rollen beteiligt. Dabei handelt es sich um den Autor des einzubringenden Dokuments und den zuständigen Bibliothekar.

Der Autor soll sein Dokument beim Bibliothekar in den Eingangskorb legen, wobei noch eine Ablageempfehlung abgegeben sollte. Um das Erstellen der Ablageempfehlung zu vereinfachen, könnte man sich vorstellen, daß der Autor das Dokument selbst in der Ablagestruktur einsortiert. Dem Autor muß dabei bewußt sein, daß das Dokument nicht abgelegt wird, sondern beim Einsortieren lediglich ein Ablageantrag ausgefüllt wird. Dieser Antrag wird zusammen mit dem Dokument an den zuständigen Bibliothekaren geschickt. Sollte der Bibliothekar das Dokument an der vorgeschlagenen Stelle ablegen, dann wird der Autor benachrichtigt und der Workflow-Prozeß ist beendet. Der Bibliothekar kann auch die Ablage verweigern. Dann muß das Dokument wieder zum Autor zurück geschickt werden.

6.6.2 Realisierungsmöglichkeiten des Workflow-Konzepts

Im Prinzip muß es die Möglichkeit geben einen Bereich der Ablage zu überwachen. Wenn in diesen Bereich ein neues Dokument eingebracht wird, dann muß der Workflow-Prozeß gestartet werden.

Die Formulierung des Prozesses[1] ist in Astoria sehr universell, da dies in C++ geschieht. Allerdings ist das nach Meinung des Autors nicht die angemessene Sprachebene, um einen Workflow-Prozeß zu formulieren. Angemessen wäre eine Sprache, die die Begriffe des

1. Trigger-Prozeß in der Astoria-Terminologie

vorangegangen Kapitel nutzt. Es muß sich aber im Laufe des Projekts noch herrausstellen, wie oft es notwendig wird, einen solchen Prozeß zu beschreiben. Wenn dies nur zur Entwicklungszeit der Fall ist, dann profitiert man sogar durch die Universalität der Sprache.

Ein viel größeres Problem stellt die Tatsache dar, daß Astoria keinen Trigger-Prozeß auslösen kann, wenn ein Dokument in einen Ordner eingebracht wird. Die Anzahl der Aktionen, die einen solchen Prozeß starten können, ist in Astoria auf Dokumente beschränkt, die schon im System vorhanden sind. Denn dieses Dokument muß in einem Linkcluster[1] enthalten sein. In einem Linkcluster können aber nur vorhandene Dokumente enthalten sein.

Auch hier wäre eine Eigenimplementierung notwendig.

6.6.3 Bewertung der Realisierungsmöglichkeiten bezüglich des Workflow-Konzepts

Das Workflow-Konzept von Astoria läßt sich nicht mit den Anforderungen von TimeLess vereinbaren. Deshalb ist eine Eigenimplementierung durch das TimeLess-Team notwendig.

6.7 Zusammenfassung der Bewertung

Astoria ist keine geeignete Plattform für das TimeLess-Softwaresystem. Durch Abweichungen im Ablagekonzept zwischen TimeLess und Astoria sind manche TimeLess-Anforderungen nur mit erheblichen Implemtierungsaufwand zu realisieren. TimeLess stellt wesentlich höhere Anforderungen an die Ablage als Astoria. Die Ablage von Astoria läßt zwar die durch TimeLess geforderte Ablagehierarchie zu. Dafür ist der Dokumentenbegriff in TimeLess und Astoria aber verschieden. Im TimeLess-Projekt wird ein Dokument als etwas nicht mehr veränderbares angesehen. Wenn das Dokument geändert wird, dann dokumentiert dieses etwas anderes als vor der Änderung. Astoria zeigt sich vielmehr als Sperrverwalter von noch nicht fertig gestellten Dokumenten. Durch die zentrale Abalge ist es möglich, daß mehre Benutzer zur Fertigstellung des Dokuments beitragen. Astoria verwaltet, daß immer nur einer die Fertigstellung des Dokuments voran treibt, in dem ein Checkin-Checkout-Mechanismus angeboten wird. Dokumente, die in TimeLess abgelegt werden, sind bereits fertig.

Negativ ist aufgefallen, daß auf den ersten Blick sehr allgemein gehaltene Konzepte, wie beim Berechtigungswesen oder der Suchmaschine, durch die Realisierung so starke Einschränkungen haben, daß die TimeLess-Konzepte damit nur noch sehr mühsam zu realisieren sind.

1. Ein Linkcluster ist eine Gruppe von Komponenten eines Dokuments, die durch ein Objekt der Klasse Link identifiziert sind. Der Begriff Linkcluster wird in der Dokumentation zum Navigationstool verwendet.

Leider ist die Programmierschnittstelle nur auf der Clientseite vorhanden. Normaler Weise gehen Aktionen immer vom Benutzer aus. Da es sich bei Astoria aber um ein Mehrbenutzersystem handelt, wäre ein Kanal vom Server zum Client wünschenswert. Wenn ein Benutzer Änderungen veranlaßt, die alle Benutzer betreffen gibt es bei Astoria keinen Kanal, über den ein Benutzer mit den anderen kommunizieren kann.

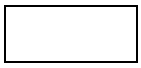
Astoria hat im Bezug auf Installation, Stabilität und Administrierbarkeit einen ausgezeichneten Eindruck macht. Mit Hilfe von [Astoria_Install_98] und [Astoria_Admin_98] war die Installation des Systems ohne Probleme möglich.

Auch der Umgang mit dem SDK[1] war einfach und problemlos. Die Programmierschnittstelle läßt erkennen, daß Konzepte der Objektorientierung konsequent umgesetzt worden sind. Die Einführung in die Konzepte in [Astoria_SDK_98] erleichtert das Verständnis.

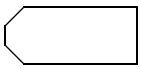
1. SDK: Software Development Kit

Anhang A: Astoria-Klassenbaum

Die Symbolik, die für die Darstellung in Bild 47 gewählt wurde, hat folgende Bedeutung:



Klasse, die Objekte beschreibt, die in der Datenbank abgelegt werden können.



Oberklasse von der kein Exemplar erzeugt wird.
Diese Klassen werden häufig als Datentyp für Referenzspeicher genutzt. In diesen Referenzspeichern können Referenzen auf Objekte aus allen Unterklassen abgespeichert werden.

Beispiel:

Man kann sich eine Abfrage vorstellen, die eine Referenz auf das erste Objekt liefert, daß in einem Ordner enthalten ist. Da in einem Ordner Objekte von unterschiedlichem Typ abgelegt werden können, kann als Typ nur die Oberklasse angegeben werden. Deshalb ist der Speicher für den Rückgabewert ein Speicher vom Typ Referenzspeicher für FolderItem-Objektreferenzen.

Object-Klasse:
Die Object-Klasse ist die Oberklasse aller Objekte, die in der Datenbank abgelegt werden. Darin beschrieben sind Methoden, mit denen man den Typ eines Objekts erfragen kann. Außerdem findet man in der Klassenbeschreibung Methoden zum Abspeichern und Laden von Objekten in und aus der Datenbank.

NamedObject-Klasse:
Jedes Objekt, das durch eine Unterklasse von NamedObject beschrieben wird, besitzt neben einem Namen eine Beschreibung. Die Methoden zum Ändern und Abfragen von Namen und Beschreibung werden in dieser Klasse beschrieben.

Alle sonstigen Klassen sind in der Arbeit beschrieben. Neben diesen Klassen gibt es Klassen, die Objekte beschreiben, die nicht in der Datenbank vorhanden sind. Diese findet man wie auch weitere Implementierungsdetails in [Astoria_SDK_98].

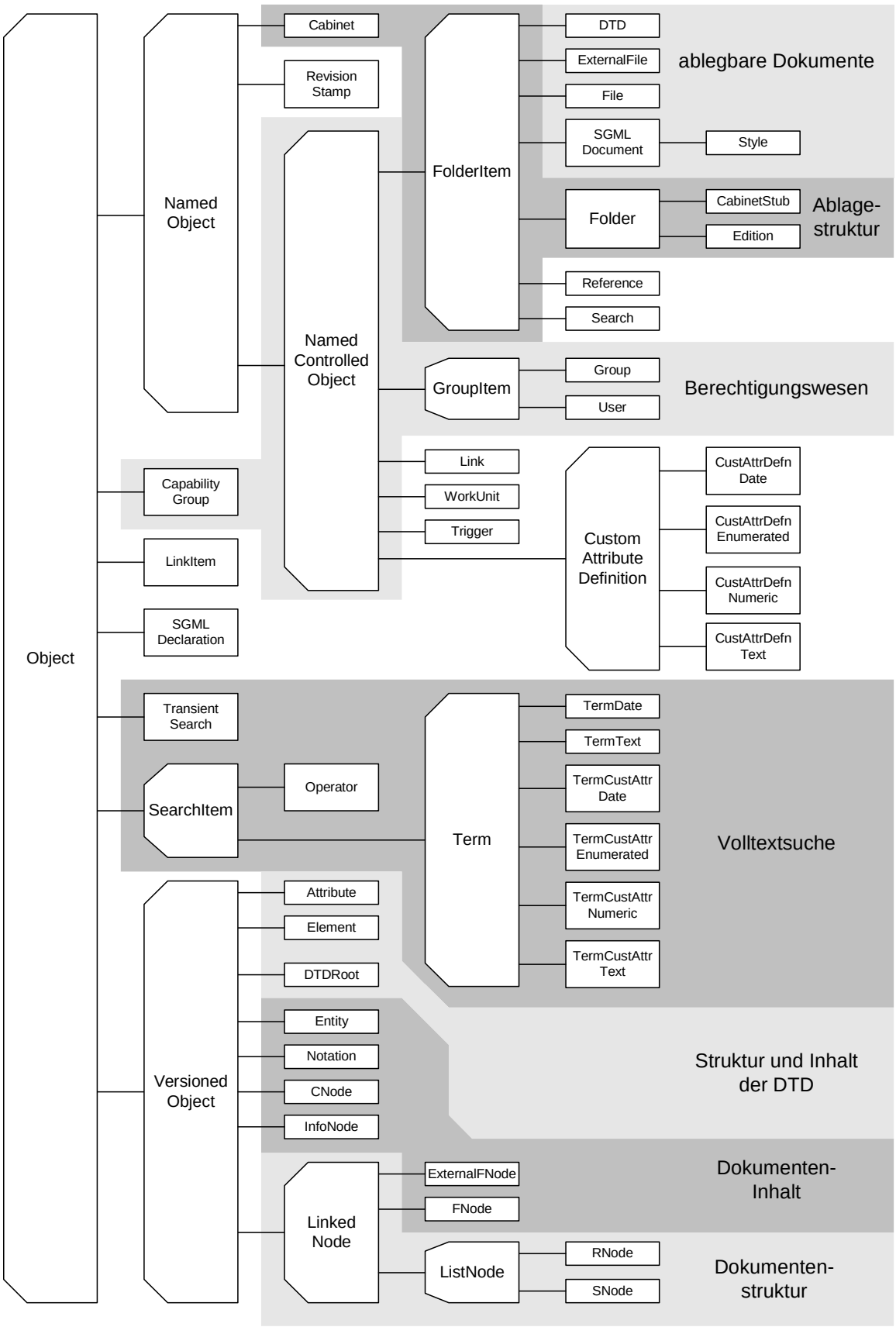


Bild 47: Astoria-Klassenbaum

Literaturverzeichnis

[AborText_95] AborText, Inc
Getting Started with SGML
http://www.cyberdynamic.com/htmfiles/arbor_toc.html

[Astoria_Help_98] Chrystal Software
Astoria Help
Version 3.0
Astoria-Dokumentation, März 1998

[Astoria_Install_98] Chrystal Software
Astoria Help
Version 3.0
Astoria-Dokumentation, März 1998

[Astoria_SDK_98] Chrystal Software
Astoria Installation Guide
Version 3.0
Astoria-Dokumentation, März 1998

[Brand_97] Christian Brand
Spezifikation und Entwurf der Navigationskomponente für ein "Corporate Memory" und Implementierung eines Demonstrationsprototyps
Diplomarbeit am Lehrstuhl für digitale Systeme, Universität Kaiserslautern, 1997

[Bungert_98] Andreas Bungert
Beschreibung programmierter Systeme mittels Hierarchien intuitiv verständlicher Modelle
Shaker Verlag

[Cherdron_97] Markus Cherdron
Konzept- und Prototypentwicklung einer Komponente zur zentralen Abwicklungssteuerung eines "Corporate Memory"-Systems
Diplomarbeit am Lehrstuhl für digitale Systeme, Universität Kaiserslautern, 1998

[DOM_98] World Wide Web Consortium
Level 1 Document Object Model Specification
http://www.w3.org/TR/1998/WD-DOM-19980720/

[Gales_98] Frank Gales
Ein Beitrag zur Begriffswelt der Schaffung programmierter Systeme
Shaker Verlag

[IAO_97] Anette Weisbecker, Christoph Altenhofen, Sven Bauer
Gutachten zur Konzeptions von "TIMELESS" aus Sicht des Dokumenten-Managements
Fraunhofer Institut Arbeitswirtschaft und Organisation, Oktober 1997

[IAO_98] Hans-Jörg Bullinger, Christoph Altenhofen, Mirjana Petrovic
Marktstudie Dokumenten- und Workflow-Management-Systeme
Fraunhofer Institut Arbeitswirtschaft und Organisation, 1998

[Langhauser_97] Jürgen Langhauser
Spezifikation und Entwurf des Schemas der Verwaltungsdaten eines “Corporate Memory” und Implementierung der Zugriffskomponente
Diplomarbeit am Lehrstuhl für digitale Systeme, Universität Kaiserslautern, 1997

[Light_97] Richard Light
Presenting XML
Sams.net Publishing, 1997

[Sperberg-McQueen] C. M. Sperberg-McQueen, Lou Burnard
A Gentle Introduction to SGML
http://www-tei.uic.edu/orgs/tei/sgml/teip3sg/index.html

[W3C_98] World Wide Web Consortium
HTML 4.0 Specification, Kapitel 12
http://www.w3.org/TR/REC-html40/struct/links.html

[Wendt_91] Siegfried Wendt
Nichtphysikalische Grundlagen der Informationstechnik
Springer, 1991

[Wendt_97] Siegfried Wendt
Zeigen, Nennen, Umschreiben - die drei Alternativen der Identifikation
Mosaikstein am Lehrstuhl für digitale Systeme, Universität Kaiserslautern, November 1997