Projektgruppe 269

AiR

Endbericht

Endbericht

PG269 - A.I.R.

Sommersemester 96

Die Mitglieder der Projektgruppe

14. August 1996

Lehrstuhl Software-Technologie
Fachbereich Informatik
Universität Dortmund

Inhaltsverzeichnis

Inhaltsverzeichnis

Kapitel 9

Einleitung

9.1 Projektgruppe

AIR - Eine Umgebung zur Verwaltung verknüpfter Informationen

Wintersemester 95/96 und Sommersemester 96

9.2 Veranstalter:

Stefan Dißmann, Informatik X, Uni-Dortmund

9.3 Teilnehmer:

- Harald Bender
- Ulrich Bielenstein
- Nicole Buchholz
- Phillip Ghadir
- Rudi Gross
- Lars Heete
- Matthias Heiduck
- Thomas Hötte
- Holger Isenberg
- Holger Karlisch
- Athanasios Papoylias
- Markus Schaffrin

9.4 Aufgabe:

Das World-Wide-Web ist das Medium im Internet, welches ideale Voraussetzungen bietet, auf bestimmte Art und Weise Informationen, egal welcher Art, zur Verfügung zu stellen. Dabei erfüllt dieses Medium sowohl Werbezwecke als auch den Zweck der gezielten Informationsrecherche.

Das World-Wide-Web (WWW) stellt eine in zunehmendem Maße in vielen Bereichen – insbesondere auch außerhalb der Informatik – genutzte Informationsquelle dar. Das WWW ermöglicht es Informationsanbietern, große Datenmengen in aktuellem Zustand mit kurzen Zugriffszeiten einer nahezu unbegrenzten Anzahl von Konsumenten vergleichsweise kostengünstig anzubieten. Angelehnt an die Standard Generalize Markup Language (SGML) stellt HTML eine Document Type Definition (DTD) von SGML dar.

Als Standard für die Darstellung von Informationen haben sich Dokumente etabliert, die in der Hypertext Markup Language (HTML) verfaßt sind. Der Betrieb entsprechender Server beim Anbieter bzw. Browser beim Konsumenten hat als Stand der Technik eine weite Verbreitung bis in den Bereich der privaten PC-Nutzung gefunden.

Während sich bei der technischen Abwicklung des Netzverkehrs lediglich lange Zugriffszeiten aufgrund der Überlastung des Netzes oder der Server einzelner Informationsanbieter als hinderlich herausstellen, treten insbesondere bei der Informationssuche und -nutzung durch Konsumenten immer wieder Unzulänglichkeiten und Mängel auf, deren Ursachen in einer ungeeigneten Strukturierung und einer schwer verständlichen Zusammenstellung der angebotenen Informationen liegen. Typische Beispiele für solche Mängel sind:

- veraltete und nicht mehr relevante Daten;
- redundante Angaben an verschiedenen Stellen;
- für das Medium WWW unzureichend aufbereitete Daten, d.h. zu große „Datenberge“ oder zu kleine „Datenhäppchen“;
- „zufällige“ Verbindungen zwischen verschiedenen Daten ohne erkennbares Strukturierungsprinzip;
- nicht erkennbare Verweise in graphischen Darstellungen und Bildern;
- unübersichtliche Einstiegsseiten oder fehlende Nachschlagregister.

In vielen Fällen ist daher das Finden von relevanten Informationen nur durch die Kenntnis zusätzlicher Hinweise oder durch viel Geduld möglich.

Versucht man, die Ursachen für diese Mängel in der Informationspräsentation zu erforschen, so stellt man in vielen Fällen fest, daß Informationen häufig nicht geeignet für die Präsentation im Medium „WWW“ aufbereitet und noch seltener anschließend sorgsam gepflegt und gewartet werden.

Im Gegensatz zum World-Wide-Web, dessen Benutzerschnittstelle so vereinfacht ist, daß es von jeder Anwenderschicht bedient werden kann, verlangt das Gestalten der einzelnen Informationsseiten in HTML sowohl einen großen Arbeitsaufwand, als auch ein Verständnis für Hypertext-Anwendungen.

Daraus resultieren große Inkonsistenzen bei Informationsangeboten, wie z.B.:

- schlechte Hypertextaufbereitung
- veraltete und nicht mehr relevante Daten

- redundante Angaben an verschiedenen Stellen.

An diesem Punkt setzt AIR an, um den Benutzer beim Aufbereiten von Informationen zu unterstützen und eine einheitliche Präsentation zu ermöglichen.

Die Aufgabe der Projektgruppe AIR war es, ein Software-System zu konzipieren, zu entwerfen und zu realisieren, welches bei einem Informationsanbieter die Aufbereitung, Pflege und Wartung solcher Informationen unterstützt, die auf einem WWW-Server als abgeschlossenes Informationsbündel angeboten werden sollen. Diese Unterstützung sollte derart erfolgen, daß aus mehreren unterschiedlichen Beschreibungen Dokumente erzeugt werden, die die durch diese Beschreibungen festgelegten Anforderungen zusammenfassen und in eine Darstellung umsetzen. Dabei wurden die folgenden Aspekte getrennt festgelegt: das Muster der gewünschten Gesamtstruktur, Muster für die Strukturen einzelner Seiten, Muster für die Layouts einzelner Seiten, die Erzeugung von Seiten als Instanzen verschiedener Muster und die Zuordnung von den gewünschten Dateninhalten zu Seiteninstanzen. Die Aufteilung der verschiedenen, die Darstellung eines Dokuments bestimmenden Faktoren ermöglichte eine einfache Aufbereitung von Informationen, das Festlegen und das Einhalten von vordefinierten, aber problembezogenen Standards, das konsistente Erweitern vorhandener Informationen und das redundanzfreie Zusammenstellen einer Menge von Informationen.

Die Umsetzung sollte durch das AIR Hypermedia-Produktionssystems vorgenommen werden, das aus den verschiedenen Beschreibungen die notwendigen Dateien im HTML-Format mit geeigneten Inhalten und Verweisen erzeugt. Neben der Fehleranalyse während des Transformationsvorgangs sollte das AIR Hypermedia-Produktionssystem statische Analysemöglichkeiten mit einer geeigneten Darstellung der Ergebnisse bereitstellen, die jederzeit die Überwachung der Aktualität der präsentierten Daten und der definierten Strukturen von Daten und Verbindungen ermöglichen. Während des Produktionsvorgangs wurde dann die Konformität der definierten Anforderungen und der erzeugten HTML-Dokumente erzwungen. Aus dieser Aufgabe ergab sich der Name des Systems: AIR = Advanced Information Rearrangement.

Während der Entwicklung des AIR Hypermedia-Produktionssystems sollten von der Projektgruppe verschiedene etablierte Beschreibungs- und Strukturierungskonzepte der Software-Technologie auf ihre Eignung zur Definition der Struktur und der Darstellung von Hypermedia-Systemen untersucht und gegebenenfalls angepaßt und einsetzt werden. Hierbei sollten insbesondere die Möglichkeiten objekt-orientierter Konzepte wie Einkapselung, Aggregation und Spezialisierung untersucht werden. Die Projektgruppe konnte bei diesen Arbeiten auf Zwischenresultate aktueller Forschungsarbeiten am Lehrstuhl Software-Technologie zurückgreifen.

Die Entwicklung von AIR erfolgte auf der Grundlage eines „klassischen“ Prozeßmodells mit den Phasen Analyse, Design und Realisierung, wobei eingeschobene Reviews und eine zweigeteilte Implementierungsphase den erfolgreichen Abschluß des Projekts sicherstellen sollte. In der Analysephase bot es sich an, die Projektgruppe zunächst eigene Ansichten und Erfahrungen im Umgang mit im WWW verfügbaren Informationen sammeln zu lassen, um so die Notwendigkeit der Projektgruppen-Aufgabe anhand von realen Beispielen zu demonstrieren.

Neben der Entwicklung des AIR Hypermedia-Produktionssystems sollte durch diese Projektgruppe Erfahrungen in zwei weiteren Bereichen gesammelt werden. Auf der Tagung „Software-Engineering im Unterricht der Hochschulen SEUH 95“ wurden von verschiedenen Vortragenden Organisationsformen für studentische Projektarbeit vorgestellt, die Alternativen zu den häufig praktizierten „demokratischen“ Gruppen darstellten. Durch eine striktere Zuordnung von Aufgaben zu einzelnen Studierenden soll deren persönliche Verantwortung für inhaltliche oder organisatorische Teilaufgaben des Projekts gestärkt und so die Motivation für die

Arbeit und die Identifikation mit der Arbeit verbessert werden. Da für jeden Aufgabenbereich ein kompetenter Fachmann festgelegt ist, sollen sich sowohl die Qualität als auch die Effizient der Gruppenarbeit verbessern.

9.5 Erweiterungsmöglichkeiten:

1. Entwicklung eines speziellen Editors zur Definition und Pflege von Informationen
2. Gestaltung einer Bibliothek von Definitionsmustern für Strukturen und Layouts
3. Erweiterung des Systems um einen kooperativen Ansatz für Mehr-Autoren-Fähigkeit

9.6 World Wide Web - Problemumfeld und -beschreibung

Eine der Aufgaben im Vorfeld der Projektgruppe (PG) war das „Stöbern im WWW“. Ziel war es, festzustellen, wie Informationen präsentiert werden und die Ergebnisse schriftlich festzuhalten. Außerdem sollte eine Aufgabenbeschreibung angefertigt werden, sowie mögliche Lösungsansätze formuliert werden. Die folgende Ausarbeitung stellt eine Zusammenfassung der Ergebnisse aller PG-Teilnehmer dar.

Das World Wide Web, kurz WWW, versucht die verschiedensten verteilten Informationsquellen unter einer Benutzeroberfläche zusammenzuführen.

Dies kann im einfachsten Fall schon das lokale Dateisystem des eigenen Rechners sein. Man benötigt somit zunächst einmal nichts weiter als einen sogenannten WWW-Browser, zum Beispiel Netscape. Dieser Browser arbeitet dann vergleichbar einem Dateimanager mit integriertem Dateibetrachter, der die Inhalte von Dateien der unterschiedlichsten Formate darstellen kann.

Jedoch liegt der eigentliche Reiz des WWW im Zugriff auf fast alle erdenklichen Internet-Dienste in einem Client-Programm gebündelt und um Multimedia- und Hypertext-Fähigkeiten erweitert. Hat man erst einmal einen Internet-Zugang, so braucht man nur noch einen beliebigen WWW-Server anwählen und kann dann auf die verschiedenen Informationsquellen zugreifen.

Es ist kaum noch ein Problem, Informationen in elektronischer Form zu bekommen, sondern eher, aus der Datenvielfalt das wirklich Interessante und Relevante zu extrahieren. Da keine zentrale Instanz das Informationsangebot überwacht oder einen umfassenden Katalog anbietet, bleibt dem Anwender nur die Verwendung eines der zahlreichen Suchwerkzeuge übrig, die allerdings auch nicht die gesamte Informationsvielfalt bewältigen.

Die gezielte Informationssuche stellt gerade für den Internet Neuling eine große Hürde dar. Aber auch der erfahrene WWW Kenner trifft oft auf die selben Probleme.

Es folgen typische Beispiele, die allen PG-Teilnehmern während des „Stöberns im WWW“ aufgefallen sind:

- unübersichtliche Einstiegsseiten
- ungenügende Informationsaufbereitung, Pflege und Wartung
- eigentlich der Information dienende Seiten werden mit unnötig großen Grafiken überladen
- fehlende oder unzureichende Strukturierung mit der Folge, daß Informationen entweder doppelt oder gar nicht vorhanden sind
- unsinnige Zusammenstellung von Dateninhalten
- nicht brauchbare Daten bei den Anbietern
- zuviel Werbung und zuwenig Inhalte

- schlechte Strukturierung großer Datenmengen (Seiten, deren Länge zu weit über den darstellbaren Bereich hinausgehen, sind oft unübersichtlich)
- Querverweise zu weiterführenden Informationen enden im Nirwana
- unsaubere Grafiken durch schlechtes Ausmaskieren
- Bildhintergründe sind teilweise farblich so schlecht auf die Schrift abgestimmt, daß kaum etwas zu lesen ist
- Seiten, die viele Bilder enthalten, benötigen teilweise sehr lange Ladezeiten
- es fehlt bei umfangreichen Bildern die Möglichkeit, genau die Bilder auszuwählen, die man tatsächlich benötigt

9.7 Aufgabenbeschreibung und Lösungsansätze

Bei der genauen Beschreibung der Aufgaben des AIR-Systems kam es zum Teil zu sehr unterschiedliche Vorstellungen der PG-Teilnehmer. Ebenso unterschiedlich waren die möglichen Lösungsansätze - jeder von uns hatte unterschiedliche Vorstellungen darüber, was die spätere Realisierung betraf. Im folgenden werden nicht alle Vorstellungen aufgeführt, stattdessen wird das Wesentliche aus den Hausaufgaben zusammengestellt.

Grundsätzliche Aufgabe des AIR-Systems ist das Erstellen, Warten und Pflegen eines Informationsbündels für as WWW. Die Eingabe soll primär als Text erfolgen, zusätzlich könnten Grafiken und Ton, sowie Tabellen zugelassen werden. Die Ausgabe findet statt als strukturierte Seiten in HTML (Hypertext Markup Language), wobei eine freie Wählbarkeit der Struktureinschränkungen durch den Benutzer zugelassen wird. Ein leichtes Einfügen neuer Seiten, sowie Warten alter Seiten muß möglich sein. Als Programmiersprachen kommen folgende objektorientierte Sprachen in Betracht: BETA, C++, oder JAVA. Des weiteren ist eine ausführliche Dokumentation des späteren Produktes wichtig. Diese Punkte wurden noch nahezu übereinstimmend von allen Teilnehmern genannt. Nun folgt eine Auflistung, wie die HTML-Seiten allgemein aussehen sollten, wie ihre Ergonomie sein könnte und welche Struktur sie im einzelnen haben dürften.

Allgemein ausgedrückt:

- Bilder sollten nicht mitgeliefert, sondern selektiert werden können
- vernünftiger Einsatz von Multimediamitteln, wenn es primär darum geht, zu informieren
- mehr Aufwand in die Beschreibung des kommenden spart sinnloses Laden (genauere Beschreibung der Links)
- automatische Inhaltsangaben-Generierung (eine Liste aller vorhandenen Seiten)
- alle Seiten ähnlich stylen um Beziehungen zum Gesamtdokument zu gewährleisten}

Ergonomie. sinnvoll gewählter Seitenhintergrund (guter Kontrast zur Schrift)

Seiten sollen nur Fenstergröße haben, falls Verzweigungen vorhanden

Struktur und Seiten. • keine Links (Verweise) auf andere, weit entfernte Seiten ohne Warnung oder Hinweis

- Informationen nur auf Seiten mit wenig Links
- Hierarchische Struktur (Baumstruktur), freies Wandern in dieser Struktur (forward (Sohn), back (Vater))

- Kapitelhierarchie?
- Zwang zur vorgegebenen Seitenstruktur, Layout jedes Seitentyps gleich, gleiche Positionierung

Indexseite. • Sammlung aller Informationen mit Verweis.

- verhindert, daß Daten nur von Leuten gefunden werden, die wissen, wo sie innerhalb des möglicherweise konfusen Ganzen stehen

Top oder auch Titelseite. • grobe Gliederung der vorhandenen Informationen

- Logo
- Seite darf nur einmal vorkommen

Verteilseite. • darf beliebig oft vorkommen

- Knoten mit mehreren Söhnen
- pro Verweis ein erklärender Text

Das Inhaltsverzeichnis wird automatisch erstellt, der Benutzer kann entscheiden, wie detailliert.

Weitere denkbare Verzeichnisse wären ein Stichwortverzeichnis und Bildverzeichnis.

9.8 Kurzeinführung in die HyperText Markup Language (HTML)

9.8.1 Einleitung

- Informationen über HTML aus dem WWW bzw. aus Newsgroups (oder aus sources)
- HTML wird seit 1990 zur Informationsaufbereitung im WWW benutzt
- HTML ist Document Type Definition (DTD) von SGML
- HTML-Dokumente sind SGML-Dokumente mit einer angepaßten Semantik für eine allgemeine Informationsdarstellung
- SGML --> HTML --> AIR (vgl. Tex --> Latex)

SGML = Standard Generalized Markup Language

9.8.2 Struktur und Syntax

- HTML-Elemente werden Tags genannt
- die meisten Elemente benötigen ein Start- und End-Tag (Ausnahmen z.B.: **<li>**, **
**)
- **<tag attributes>content</tag>** (nicht case-sensitive)
- Max. Länge der attributes 1024 Zeichen, laut SGML-Deklaration
- Kommentare: **<!-- Kommentar -->**, aber **<!alte Daten>**
- Text wird automatisch umgebrochen (vgl. mit TeX), CR und TABs werden ignoriert

9.8.3 Struktur eines HTML-Dokuments

```
<!DOCTYPE HTML PUBLIC „-//IETF//DTD HTML 2.0//EN">
<HTML>
<HEAD>
<TITLE>AIR-Homepage</TITLE>
</HEAD>
<BODY>
...
</BODY>
</HTML>
```

- HEAD: enthält den Titel und optional zusätzliche Funktionen, optional
- BASE: Angabe einer Basisadresse
- LINK: Link auf den Autor oder andere Infos (z.B. mirrors)
- META: Dokumentbezogene Informationen (z.B. über Inhalt, Autor, Datum)
- TITLE: der Titel erscheint in der Fensterleiste des Browsers und soll Indentifizierung der Seite ermöglichen (z.B. nicht Einführung, sondern Einführung in...)
- BODY: Inhalt, background-Attribut (Netscapism: Text-/Linkfarben ändern)

9.8.4 Befehlsübersicht

- Blockstruktur-Elemente: <p>,
 (optional kann <p> auch mit </p> beendet werden)
- Preformatted Text: <pre>...</pre>
- Headings: <hx>...</hx> für x=1-6 (nie um mehr als 1 springen, z.B. von h1 auf h3)
- Listen
  - unordered list: <ul>,
  - ordered list: <ol>,
  - directory list: <dir>, <li> (für kurze list items bis 20 Zeichen)
  - menu list: <menu>, <li> (kompaktere Formattierung)
  - definition list: <dl>, <dt>, <dd>
- Typographische Elemente: <i>, <b> (italic, bold)
- Horizontal rule: <hr> (Netscapism: width=x%, size=x)
- Image: **<img ALIGN=***top||middle||bottom(default)* **ALT="text" ISMAP SRC>**
- Hyperlinks: anchor
  - Link auf entfernte Server **...**
  - Lokaler Sprung **...**
    auf **...**
  - Mailto **...**
  - Formulare zur Eingabe von Daten
  - Sonderzeichendarstellung: z.B. **ü** = ü, oder **&#ascii-code;**

9.8.5 HTML 3.0 Erweiterungen

- zusätzliche Attribute: z.B. align=center, left, right
- Figure: Text kann Bilder umfließen

```
<fig src=...>
<caption>titel</caption>
<credit>Autor</credit>
</fig>
```

- Tabellen

```
<table border=x cellpadding=x cellspacing=x>
<caption>titel</caption>
<tr><th>1<th>2<th>3
<tr><td>A<td>I<td>R
</table>
```

- Darstellung von Formeln (starke Anlehnung an LaTeX) - Beispiel für Integral f(x) von a nach b über 1+x:

```
<math>
&int;_a_b^{f(x)<over>1+x}dx
</math>
```

9.9 Vorbereitende Literatur:

Bartsch, W.; K. Bergner; R. Hettler; B. Paech: Studenten Entwickeln Universelles Hochschulinformationssystem: Erfahrungen aus einem Softwaretechnik-Praktikum. Software Engineering im Unterricht der Hochschulen SEUH ´95, Teubner 1995, S. 129-141

Berners-Lee, T.: World Wide Web Initiative. http://info.cern.ch/hypertext/WWW/TheProjekt.html

Bogaschewsky, R.: Hypertext-/Hypermedia-Systeme - Ein Überblick. Informatik Spektrum 15 (3), 6/1992, S. 127-143

Booch, Grady: Object Oriented Analysis and Design with Applications. Benjamin/Cummings 1993

Conklin, J.: Hypertext: An Introduction and Survey. IEEE Computer 10 (9), 9/1987, S. 17-41

Connolly, D. (Ed.): Hypertext Markup Language (HTML). http//www.hal.com/users/connolly/html-spec/HTML_TOC.html

Gloor, P.A.: Hypermedia-Anwendungsentwicklung. Teubner 1990

Hornecker, Eva: Teamtraining und Präsentationstechniken für das Software-Engineering-Praktikum. Software Engineering im Unterricht der Hochschulen SEUH ´95, Teubner 1995, S. 69-81

Jacobson, Ivar: Object-Oriented Software Engineering. Addison-Wesley 1992

Knudsen, J. Lindskov; M. Löfgren; O. Lehrmann Madsen; B. Magnusson: Object-Oriented Environments - The Mjølner Approach. Prentice Hall 1993

Madsen, Ole Lehrmann; Birger Møller-Pedersen; Kristen Nygaard: Object-Oriented Programming in the BETA Programming Language. Addison-Wesley 1993

Mühlhäuser, M.: Hypermedia-Konzepte zur Verarbeitung multimedialer Information. Informatik Spektrum 14 (5), 10/1991,S. 281-290

Ockenfeld, M.; E. Wetzel: Grundlagen und Perspektiven der Multimedia-Techniken. Computer und Recht 6/1993, S. 385-389

Quibeldey-Cirkel, Klaus: Das Objekt-Paradigma in der Informatik, Teubner 1994

Rumbaugh, J; M. Blaha; W. Premerlani; F. Eddy; W. Lorenson: Object-Oriented Modeling And Design. Prentice Hall 1991

Schneider, Kurt: Auf der Suche nach maßgeschneiderten Unterrichtsformen – das angeleitete Praktikum. Software Engineering im Unterricht der Hochschulen SEUH ´93, Teubner 1993

Stein, Wolfgang: Objektorientierte Analysemethoden. BI Wissenschaftsverlag 1994

Tochtermann, Klaus: Ein Modell für Hypermedia. Shaker 1994

Weber-Wulff, Debora: Teambildung in Programmierung und Software-Engineering Kursen. Software Engineering im Unterricht der Hochschulen SEUH ´95, Teubner 1995, S. 82-89
-: Leipziger Empfehlungen zum Elektronischen Publizieren. NfD 45, 1994, S. 220-235

9.10 Realisierung:

Zeitplan für das Wintersemester 95/96:

Zeitraum	Projektphase	Aufgaben
1995		
28. KW		Kennenlernen, Planung der Seminarphase 1
29.-41. KW	Vorbereitung	Vorbereitung der Vorträge für die Seminarphase 1 (Hypertext, Hypermedia, HTML), experimentelles Erleben des WWW
42. KW	Organisation	Festlegen der Selbstorganisation der Gruppe, Aufgabenzuordnung und Präzisierung des Projektablaufplans
43. KW	Seminarphase 1	Vorträge und Diskussionen
44.-47. KW	Anforderungsanalyse	Erstellung eines ersten Anforderungskatalogs, verbale Beschreibung der Komponenten und ihrer Arbeitsweisen; Experimente mit HTML
48. KW	Review	Präsentation der Ergebnisse der Analyse
49.-50. KW	Anforderungsanalyse (Fortsetzung)	Überarbeitung der Ergebnisse und Erstellung einer objekt-orientierten Modellierung
51. KW	Reflexion	Gruppenorganisation
1996		
1. KW	Vorbereitung	Vorbereitung für die Seminarphase 2 (BETA)
2. KW	Seminarphase 2	Vorträge und Diskussionen
3.KW	Design	Ergänzung der Modellierung zum Design
4.-5. KW	Implementierung	Implementierung ausgewählter Teile des Designs zur Überprüfung der Tragfähigkeit des Konzepts
6. KW	Dokumentation	Aufbereitung der Entwicklungsergebnisse und Erstellung des Zwischenberichts

Zeitplan für das Sommersemester 1996:

Zeitraum	Projektphase	Aufgaben
16. KW	Review	Zwischenbericht und Implementierung

Zeitraum	Projektphase	Aufgaben
17.-21. KW	Implementierung	Überarbeitung und Fortsetzung der Implementierung
22. KW (Pfingsten)	Vorbereitung	Ausarbeitung von Positionsaussagen für die folgende Reflexion
23. KW	Reflexion	Diskussion der Eignung von BETA; Erstellung eines Positionspapiers
24.-25. KW	Anforderungsanalyse	Auswahl einer der möglichen Erweiterungen und entsprechende Analyse
26.-27. KW	Implementierung	Realisierung der Analyseergebnisse
28. KW	Dokumentation	Erstellung des Abschlußberichts

Kapitel 10

Konzeption des AIR-Systems

10.1 Begriffe im AIR-Kontext

Die Projektgruppe hat zwei Sprachen entwickelt, die dem einheitlichen Aufbau von thematisch verwandten Seiten dienen. Der einheitliche Aufbau wird durch das Trennen von Inhalt und Aufbereitung erzielt. Die Unterscheidung zwischen inhaltlicher und struktureller Beschreibung macht eine Sprache zur Beschreibung von *Daten*, sowie eine Sprache zur Beschreibung von *Typen* notwendig.

Im AIR-Kontext werden inhaltliche Beschreibungen von Seiten **Seitenobjekte** genannt. Die strukturellen Beschreibungen von WWW-Seiten heißen **Seitentypen**.

Seitenobjekte werden in der **Page Instance Language** beschrieben. Ein Seitenobjekt ist immer von einem bestimmten Seitentyp. Seitentypen werden in der **Page Description Language** beschrieben. Sie sind Muster für Seiten, die quasi nur noch ausgefüllt werden müssen.

Seitenobjekte können in sogenannten *Domains* zusammengefaßt werden. Diese bieten eine Hierarchisierungsmöglichkeit an, um große Mengen von Seitenobjekten überschaubar zu halten. Außerdem sind Domains sehr hilfreich, um automatisch Links auf "alle Seitenobjekte, die ein bestimmtes Kriterium erfüllen", zu generieren.

10.1.1 Beispiel für ein leeres Seitenobjekt

```
<object name="WonderAIR" type="titlepage">
       <author>
            </>
     <abstract>
            </>
         <Head>
            </>
        <Title>
            </>
         <Date>
            </>
    <Description> <!--optional -->
            </>
```

Technisch betrachtet, kommt noch eine weitere Sprache hinzu. Diese wird benötigt, da der strukturelle Aufbau von dem Erscheinungsbild einer WWW-Seite getrennt wird. Dies macht in solchen Fällen Sinn, wo Personen mit unterschiedlichen Berechtigungen oder Zielsetzungen auf die selben Seitenobjekte zugreifen müssen. Durch die Trennung von Struktur und Ansicht kann so ein Seitenobjekt für verschiedene Zwecke verschieden aufbereitet werden.

Die Sprache zur Beschreibung der Ansicht wird **Page View Language** genannt. Sie ist ein um das Vererbungskonzept erweitertes Tcl. Die Ansicht einer Seite wird *View* genannt.

Im weiteren Verlauf dieses Endberichts wird das Wort Hyper-Dokument auftauchen. Es bezeichnet eine Menge von WWW-Seiten, die von einer gemeinsamen Wurzel - der Startseite - aus erreichbar sind, zwischen denen ein inhaltlicher Zusammenhang besteht und die daher gemeinsam verwaltet werden.

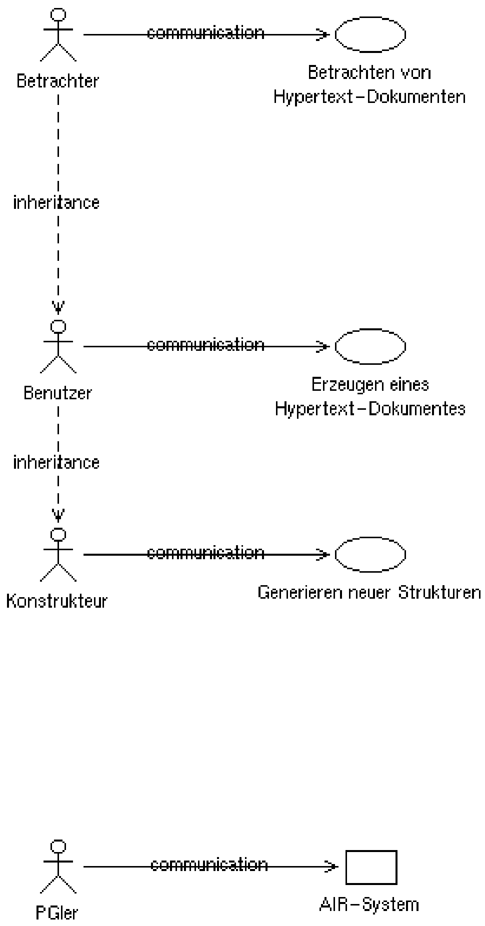
10.2 Das Programm/Produkt: AIR

Basierend auf den drei Sprachen **PDL**, **PVL** und **PIL** entwickelten wir das Software-System **AIR**, das Seitenobjekte verwalten kann. Aus diesen sollen auf Wunsch Hyper-Dokumente generiert werden können, wobei die Verknüpfungen (sog. Links) innerhalb der Seitenobjekte eines Hyper-Dokumentes automatisch von dem AIR-System angelegt werden. Die Konsistenz der Seitenobjekte, sowie die inhaltliche Aktualität können durch den Einsatz der AIRbase - der internen Datenbank - überwacht werden.

Bevor wir den Aufbau des AIR-Systems näher betrachten, ist es nötig, die Personengruppen eindeutig zu benennen, die mit dem AIR-System in Berührung kommen, da sonst Mißverständnisse aufkommen können.

Das AIR-System unterscheidet zwei Arten von Anwendern: den *Konstrukteur* und den *Benutzer*. Benutzer und Konstrukteure arbeiten direkt mit dem AIR-System. *Betrachter* kommen nur mit den Hyper-Dokumenten in Berührung, die ein Benutzer mit dem AIR-System generiert. Zu guter letzt gibt es noch die *Programmierer*, die wir an einigen Stellen erwähnen müssen. Diese implementieren das AIR-System.

Die aufgezählten Anwender des AIR-Systems werden in der unteren Abbildung noch einmal dargestellt. In der Abbildung wird deutlich, daß die Anwender in einer Vererbungsrelation stehen. An oberster Stelle steht hier der Betrachter, der keinerlei Rechte im AIR-System hat. An unterster Stelle (sozusagen der Urenkel des Betrachters) steht der Programmierer, der alle Fähigkeiten seiner Vorfahren in sich vereint.



Charakterisierung der AIR-Anwender

Nun folgen die Definitionen der AIR-Anwender.

Betrachter. Betrachter bezeichnet die Klasse aller Anwender, die sich HTML-Seiten (->Hyper-Dokumente) mit einem WWW-Browser anschauen.

Benutzer. Benutzer bezeichnet Anwender, die mit dem AIR-System ->Seitenobjekte anlegen und daraus dann ->Hyper-Dokumente erzeugen.

Konstrukteur. Konstrukteure erstellen aufgrund ergonomischer Kriterien, die in den Layout-Vorgaben herausgestellt wurden, ->Seitentypen sowie ->Views, die an Seitentypen gebunden sind.

Programmierer. Programmierer in diesem Projekt sind die Mitglieder der Projektgruppe.

10.3 Entwicklung des AIR-Systems

Die endgültige Zielsetzung der Projekt-Gruppe kristallisierte drei getrennte Arbeitsbereiche heraus. Die erste Gruppe AIRcontrol beschäftigte sich mit der Spezifikation der Sprachen **PDL**, **PVL** und **PIL** sowie der Ausarbeitung von Styleguides für Seitentypen. Die Gruppe AIRcraft sollte das Gesamtsystem entwerfen und beschreiben. Zu guter letzt sollte die Gruppe AIRbase die interne Datenbank bereitstellen. Sie sollte auf Anforderungen der anderen beiden Gruppen reagieren.

Die Entwürfe der Teilgruppen lieferten die Grundlagen für die Implemtierungsphase. Hier wurden die Grenzen der vorherigen Arbeitsgruppen verwischt. Die Gruppe AIRcontrol, die im Vorfeld die Grammatiken für die Sprachen **PVL**, **PDL** und **PIL** entworfen hatten, implementierten den Parser für die drei Sprachen. Die Gruppe AIRcraft, die ursprünglich den "HTML-Generator", schreiben sollte, hat diesen weitestgehend an die Gruppe AIRcontrol abgetreten. Lediglich die Koordination der Aktionen lagen in der Verantwortung von AIRcraft.

Das AIR-System besteht aus einer Reihe von cgi-Programmen (siehe Kapitel AIR-Show), über die autorisierte Benutzer oder Konstrukteure Aktionen ausführen können. Obwohl das AIR-System auch aus einer UNIX-Shell gesteuert werden könnte, macht der Einsatz erst wirklich Sinn, wenn die cgi-Programme aus einem WWW-Browser heraus aufgerufen werden. Einerseits besteht die Möglichkeit, mittels eines speziellen Views für Benutzer, lokal WWW-Seiten von Seitenobjekten zu erzeugen, die den Inhalt der Seitenobjekte in Formularen darstellen. So kann ein Benutzer diese Daten einfach ändern und das Seitenobjekt einfach aktualisieren.

Die andere Möglichkeit besteht darin, sich mit dem Browser auf der AIR-System Startseite einzuloggen. Dazu muß man sich entweder als Benutzer oder als Konstrukteur authorisieren. Die Startseite bietet einfache Links, die die einzelnen Funktionen des AIR-Systems aufrufen. Eine Beispiel-Startseite ist in der folgenden Abbildung zu sehen.

Startseite des AIR-Systems für Benutzer

Kapitel 11

AIRcraft: Gesamtsystem

In diesem Text wird die Konzeption des AIR-Gesamtsystems geschildert, verbunden mit den Gründen, die zu dieser speziellen Konzeption geführt haben. Anstatt AIR-Gesamtsystem wird im weiteren nur noch vereinfachend der Ausdruck AIR-System verwandt.

Bevor mit dem Design des AIR-Systems begonnen wurde, waren schon einige Grundlagen festgelegt. Diese Festlegungen führten zu der hier beschriebenen Konzeption, die nachfolgend erläutert werden.

a). Die Ziel-Vorgabe für das AIR-System leitet sich aus den Zielen der Projektgruppe **AIR** und den aus den Hausaufgaben resultierenden Lösungsansätzen (siehe Kapitel Einleitung) ab. Kurz zusammengefaßt lautet sie:

> Das AIR-System wandelt Informationen (Text, Grafik und Ton) standardisiert, in nach ergonomischen Maßstäben günstige, sowie unter Hypertext-Gesichtpunkten sinnvoll strukturierte, HTML-Darstellung um.

b). Es gibt verschiedene Gruppen von Anwendern des AIR-Systems, jede mit andersgearteten Ansprüchen an das AIR-System. Diese Gruppen sind **Betrachter**, **Benutzer** und **Konstrukteur**. Die Identifizierung und Festlegung dieser Gruppen ist entscheidend für die AIR-Konzeption. Die Definition sind im Abschnitt *Konzeption des AIR-Systems* zu finden.

c). Der Aufbau und die Gestaltung der Dokumente geschieht auf drei unterschiedlichen Ebenen. Erstens müssen die einzelnen Seitentypen definiert werden, also festgelegt werden, welche Komponenten zu einer bestimmten Seite gehören. Zweitens muß beschrieben werden, *wo und wie* die einzelnen Komponenten auf den Seiten erscheinen sollen und drittens, *was* dargestellt werden soll. Für die Beschreibung auf jeder Ebene steht im AIR-System eine Sprache zur Verfügung. Diese sind die **PDL** (Page Definition Language), die **PVL** (Page View Language) und die **PIL** (Page Instantiation Language).

Der *Betrachter* benutzt keine dieser Sprachen, er kommt lediglich mit dem Output des AIR-Systems, d.h. mit den HTML-Seiten in Kontakt.

Der *Benutzer* schreibt seine Informationen (Text, Grafik und Ton) in der Sprache **PIL**. Mit den beiden anderen Sprachen kommt er nicht in Berührung. In PIL werden die Inhalte beschrieben, die letztendlich auf den HTML-Seiten zu sehen sein sollen. Der Benutzer kann sich in der **PIL** einen Seitentyp auswählen und die Komponenten der Seite mit den Informationen füllen.

Welche Komponenten in einem Seitentyp vorkommen, das bestimmt der *Konstrukteur* mit der Sprache **PDL**. Für jeden Seitentypen wird bestimmt, welche Komponenten vorkommen *können* und welche Komponenten einen Inhalt bekommen *müssen*. Weiterhin ist es möglich, daß Seitentypen die Komponenten eines anderen Seitentypen erben, damit auf einfache Weise eine Ähnlichkeit zwischen allen Seiten, die zusammengehören, hergestellt werden kann.

In der **PVL** wird schließlich beschrieben, wie die Seiten eines jeden Seitentyps dargestellt werden sollen. In dieser Sprache legt der Konstrukteur fest, an welchen Stellen und auf welche Art und Weise die einzelnen Komponenten auf den HTML-Seiten erscheinen.

Diese drei Punkte bilden die Grundlage für die gesamte Konzeption. Die Arbeitsweise des AIR-Systems ist grundsätzlich die eines Compilers. Es übersetzt die vom Benutzer geschriebenen PIL-Dokumente nach HTML. Damit ist die erste Komponente des AIR-Systems der Compiler. Die Übersetzung des Compilers ist allerdings nicht nur von dem Inhalt des PIL-Dokumentes abhängig, sondern auch von den durch den Konstrukteur definierten und von dem Benutzer ausgewählten Seitentypen. Diese Informationen müssen dem Compiler als PDL- und PVL-Dokumente zur Verfügung stehen. Diesem Zweck dient die zweite Komponente des AIR-Systems: Die AIR-Database. Diese wird im folgenden nur noch **AIRbase** genannt.

Der Compiler holt sich beim Compilieren Informationen aus der Datenbank. Die Datenbank kann dem Compiler nur vernünftige Informationen liefern, wenn die abgespeicherten Informationen semantisch und syntaktisch korrekt sind. Es dürfen demnach keine nach semantischen oder syntaktischen Gesichtspunkten unkorrekten Dokumente in der Datenbank stehen. Um das zu gewährleisten, ist eine dritte Komponente nötig: Der AIRfilter. Der Name dieser Komponente leitet sich direkt aus ihrer Arbeitsweise ab. Unkorrekte Dokumente bleiben im Filter hängen, während die korrekten durch den Filter hindurch in die Datenbank rieseln. Normalerweise durchläuft ein Compiler verschiedene Schritte, von denen die Überprüfung der syntaktischen und semantischen Korrektheit der erste ist. Diese Funktionalität bietet jetzt der **AIRfilter**. Der Compiler verwendet nur Dokumente aus der **AIRbase**. Alle Dokumente aus der **AIRbase** mußten vorher durch den **AIRfilter** und sind somit syntaktisch und semantisch korrekt. Eine Überprüfung auf semantische und syntaktische Korrektheit im Compiler ist daher überflüssig und wird aus der Compiler-Komponente herausgenommen.

Dieses Vorgehen macht aber ein Filesystem nötig, d.h. eine Möglichkeit, auch unkorrekte Dokumente abzuspeichern, denn inkorrekte Dokumente kommen nicht in die AIRbase und es ist unrealistisch anzunehmen, es würden nur korrekte Dokumente geschrieben. Von diesem Filesystem aus muß man natürlich auch die Möglichkeit haben, Dokumente, die man für korrekt hält, an die AIRbase zu senden oder zu schauen, was sich alles schon an korrekten Dokumenten in der AIRbase befindet.

Es dürfen aber nicht alle Anwender auf dieselben Funktionen zugreifen (siehe b). Die Anwendergruppen haben fest definierte Rechte im AIR-System. Der *Konstrukteur* darf z.B. PVL-Dokumente schreiben oder verändern, der *Benutzer* jedoch nicht. Anschauen dürfen sie sich jedoch beide. Es wird also eine Kontrolloberfläche benötigt, die dem *Konstrukteur* zusätzliche Kommandos zur Verfügung stellt. Diese Kontrolloberfläche stellt die vierte und letzte Hauptkomponente des AIR-Systems dar und trägt den Namen **AIRcontrol**.

Kapitel 12

AIRControl

Die Gruppe AIRControl befaßt sich vornehmlich mit der Definition von Styleguides für Seitenlayouts sowie der Definition der drei Sprachen PDL, PVL und PIL.

12.1 Die Sprachen

Das AIR-System besteht aus drei Sprachen, die unterschiedliche Funktionen haben. Jede dieser drei Sprachen wird genauer in den folgenden Unterkapiteln betrachtet.

Mit der **Page Definition Language (PDL)** wird festgelegt, aus welchen Bestandteilen ein Seitentyp zusammengesetzt ist; sie definiert den semantischen Inhalt der später zu erzeugenden HTML-Seiten.

Die zweite Sprache **Page View Language (PVL)** ist in unserem System keine eigenständige Sprache, da sie in die **Page Definition Language (PDL)** eingebettet ist. Grundsätzlich ist sie unabhängig von der eigentlichen PDL, deshalb existiert auch ein eigenes Unterkapitel. Die PVL beschreibt das Layout, d.h. mit ihr wird das spätere Aussehen der HTML-Seiten festgelegt. Diese Sprache ermöglicht das Festlegen von einheitlichen Design für verschiedene Seiten.

Mit der **Page Instance Language (PIL)** wird ein konkretes Seitenobjekt beschrieben, d.h. durch die PIL wird ein ``leeres Formular‘‘ definiert, das anschließend vom *Benutzer* auszufüllen ist. Die PIL-Objekte werden direkt aus der PDL abgeleitet, ohne das PVL-Details berücksichtigt werden.

12.1.1 Fragestellungen

Die PDL und die PVL werden vom *Konstrukteur* benutzt, da dieser für die folgenden Fragestellungen verantwortlich ist:

- Welche Seitentypen braucht unser System?
- Welche Bestandteile hat ein Seitentyp?
- Welchen Typ (Text, Bild, etc.) hat ein bestimmter Bestandteil?
- Wie häufig darf ein Bestandteil auftreten?
- Wie wird dieser Seitentyp graphisch dargestellt?

- Wie wird ein Bestandteil graphisch dargestellt?

Die PIL wird dagegen vom *Benutzer* verwendet, der folgende Fragen zu beantworten hat:

- Welcher existierende Seitentyp ist für das gewünschte Objekt geeignet?
- Welcher Text/ welches Bild gehört zu welchem Bestandteil?
- Welche Verweise sollen im Seitenobjekt vorhanden sein?
- Wie häufig wird ein Bestandteil benötigt?

12.1.2 Grundbegriffe der Grammatikbeschreibungen

Syntax und Semantik der Regeln

A ::= B	A wird durch B definiert.
A ::| B | C | D	A wird entweder durch B oder C oder D definiert.
A ::? B	A wird durch B definiert oder entfällt.
A ::+ B	A wird als Liste von B definiert und enthält mindestens ein Element.
A ::* B	A wird als Liste von B definiert oder entfällt.

Sonstige Definitionen

`<CompMod>`
Einer Komponente wird einer der folgenden Type-Modifier zugewiesen:

? Komponente ist optional

+ Komponente ist mindestens einmal vorhanden (Liste)

* Komponente ist eine optionale Liste

`<TypeMod>`
Falls **REF** zusätzlich zum Datentyp angegeben wird, ist dies eine Referenz

`Schlüsselwörter`
stehen in einfachen Anführungszeichen

12.1.3 Die Page Definition Language (PDL)

Beschreibung

Der Konstrukteur beschreibt mit Hilfe der PDL seine Seitentypen. Jeder Typ erhält einen eindeutigen Namen, seine **ID**, der für die Ableitung weiterer Seitentypen benötigt wird. Eine Ableitung geschieht auf folgende Weise:

```
TYPE abzuleitender_TYP : vererbender_TYP
```

Im *DeclarationPart* werden die inhaltlichen und gestalterischen Bestandteile festgelegt.

Die inhaltlichen Bestandteile sind **ATTRIBUTE** und **COMPONENT**. Attribute sind einfache Strings, Komponenten können dagegen auch komplex aufgebaut sein. So können in einer Komponente andere Komponenten und Attribute enthalten sein. Dabei gibt es keine Schachtelungsgrenzen, aber zur Übersichtlichkeit sollte die Schachtelungstiefe nicht zu groß werden.

Über *CompMod* wird die Häufigkeit des Auftretens von Komponenten festgelegt (s.o.)

Durch die Angabe von **REF** wird eine Referenz bzw. ein Link zu anderen Typen beschrieben.

Die gestalterischen Bestandteile einer Seite werden vom Konstrukteur durch **`VIEW`**s beschrieben. Der View legt das spätere Aussehen der Seiten fest. Dabei hat jeder Seitentyp seinen View, aber auch einzelne Komponenten können ihre eigenen Views haben. Mit **`EXTEND`** können bestimmte Teile der Views angesprochen werden. Weitere Einzelheiten zu den Views folgen in einem separaten Abschnitt.

Bezeichner für Attribute und Komponenten sind so zu wählen, daß sie durch ihre Namensgebung den Zweck des Attributes oder der Komponente erkennen lassen.

Die PDL-Grammatik

<TypeDefinition>	::=	‘TYPE‘ <ID> <TypeDecl>
<TypeDecl>	::|	<DeclarationPart>
	|	‘:‘ <ID> <TypeMod> <DeclarationPart>
<TypeMod>	::?	‚REF‘
<DeclarationPart>	::?	‚{‚ <Items> ‚}‘
<Items>	::*	<Item>
<Item>	::|	<Attribute>
	|	<Component>
	|	<View>
<Attribute>	::=	‚ATTRIBUTE‘ <ID> <AttrInitPart>
<AttrInitPart>	::?	‚{‚ <AttrInits> ‚}‘
<AttrInits>	::+	<AttrInit>
<AttrInit>	::=	<ID> = <String> ;
<Component>	::=	‚COMPONENT‘ <ID> <CompMod> <TypeDecl>
<CompMod>	::|	
	|	‚?‘
	|	‚+‘
	|	‚*‘
<View>	::=	‚VIEW‘ <ID> <ViewDeclPart>
<ViewDeclPart>	::|	‚{‚ <PVL> ‚}‘
	|	‚EXTEND‘ <ID> ‚{‚ <PVL> ‚}‘
<ID>	::+	[a-z],[A-Z],[0-9]
<String>	::+	<Char>
<PVLl>	::=	*Die PVL wird im Abschnitt Die Page View Language beschrieben.*

Beispiel (ohne Views)

```
TYPE page {
  ATTRIBUTE name
  COMPONENT title: TEXT
}

TYPE air_page: page {
  COMPONENT author {
    ATTRIBUTE name
    ATTRIBUTE mail_address
  }
}

TYPE main_page: air_page {
  COMPONENT overview: TEXT
  COMPONENT index_pages: distrib_page REF
}

TYPE distrib_page: air_page {
  COMPONENT description: TEXT
  COMPONENT documents: document_page REF
}

TYPE document_page: air_page {
  ATTRIBUTE info
  COMPONENT document: TEXT
}

TYPE home_page: distrib_page {
  COMPONENT photo: PIC
}
```

12.1.4 Die Page View Language (PVL)

In AIR enthalten die vom Benutzer erstellten Objekte nur die Inhalte der Attribute und Komponenten, aber keine Angaben darüber, wie sie dargestellt werden. Die Darstellung der Seite bestimmt der Konstrukteur durch Angabe des VIEWs. Da in dem VIEW nur diejenigen Attribute und Komponenten angeordnet werden können, die auch in der Seitenbeschreibung (in PDL) vorkommen, steht der VIEW einer Seite direkt in der Definition dieser Seite (als Methode).

Bei der Bildung von Unterseitenklassen erbt eine Seite die VIEWs aller Oberseitenklassen. Die in dieser Ableitungsfolge vererbten VIEWs können nicht völlig umdefiniert, sondern lediglich erweitert werden. Diese Erweiterung eines VIEWs ist nur dort möglich, wo in der übergeordneten Klasse ein **inner** eingefügt wurde.

Bei der Sprache PVL handelt es sich im Grunde genommen um ein erweitertes TCL, erweitert um den Zugriff auf die Attribute/Komponenten. TCL ist eine stringbasierte, kommando-orientierte Skriptsprache.

Der Vererbungsmechanismus für Views

Bei der Definition eines Views muß angegeben werden, an welchen Stellen er erweitert werden kann. Das geschieht über einen **inner**-Mechanismus, der dem in BETA ähnlich ist. Soll ein Bereich eines Layouts erweiterbar sein, so kann durch Einfügen eines **inner**-Kommandos an dieser Stelle erreicht werden, daß hier die VIEW-Erweiterungen abgeleiteter Klassen eingesetzt werden.

Soll ein View an mehreren Stellen erweiterbar sein, so ist es möglich, das **inner** mit einem Label zu versehen. In den VIEW-Erweiterungen der Unterklassen muß dann durch Angabe von **`EXTEND Label`** festgelegt werden, auf welche Stelle sie sich beziehen. Es ist dabei möglich, auf Labels aller übergeordneten Definitionen zu verweisen. Existieren dabei in einer Ableitungsfolge für eine Stelle mehrere Erweiterungen, werden diese hintereinander eingefügt. Fügt man innerhalb einer Erweiterung einer gewissen Stelle wieder ein **`inner`** mit demselben Label ein, so wird das **`inner Label`** auf die neue Stelle umdefiniert.

Beispiel (mit Views)

```
TYPE page {
  ATTRIBUTE name
  COMPONENT title: TEXT
  VIEW html {
    put „<!DOCTYPE HTML PUBLIC \"//IETF//DTD HTML 3.0//EB\">"
    put „<HTML>"
    put „<HEAD>"
    put „<TITLE>$title</TITLE>"
    inner head
    put „</HEAD>"
    put „<BODY>"
    inner body
    put „</BODY>"
    put „</HTML>"
  }
}



TYPE air_page: page {
  COMPONENT author {
    ATTRIBUTE name
    ATTRIBUTE mail_address
    VIEW html {
      put „<A href=\"mailto:$mail_address\"> $name </A>"
    }
  }
  VIEW html EXTEND body {
    /* erweitert „inner" von page */
    inner textpart
    put „<HR>"
    put [author „html"]
    inner footline
  }
}
```

```
TYPE main_page: air_page {
  COMPONENT overview: TEXT
  COMPONENT index_pages: distrib_page REF
  VIEW html EXTEND textpart {
    put [overview „html‟]
    put „<UL>‟
    forall page in index_pages {
      put „<LI>‟
      put [page „html‟]
      put „<BR>‟
      put [page „description‟ „html‟]
    }
    put „</UL>
  }
}



TYPE distrib_page: air_page {
  COMPONENT description: TEXT
    COMPONENT documents: document_page REF
    VIEW html EXTEND textpart {
      inner mainpart
      put [description „html‟]
    put „<UL>‟
    forall doc in documents {
      put „<LI>‟
      put [doc „html‟]
      put „<BR>‟
      put [doc deref info]
    }
    put „</UL>‟
  }
}



TYPE document_page: air_page {
  ATTRIBUTE info
  COMPONENT document: TEXT
  VIEW html {
    put „Zusammenfassung: $info‟
    put  Originaltext: [document html]‟
  }
}



TYPE home_page: distrib_page {
  COMPONENT photo: PIC
  VIEW html EXTEND mainpart {
    put [photo „html‟]
  }
}
```

Der Zugriff auf Komponenten

Der Zugriff auf Attribute ist mit dem Zugriff auf Inhalte einer Variablen zu vergleichen, der Zugriff auf Komponenten mit einem Funktionsaufruf.

Mit **`put „text“`** wird der Text zwischen den Anführungszeichen ausgegeben, mit **`put „$attributname“`** wird der Inhalt des Attributes **`attributname`** ausgegeben und mit **`put [compname „html“]`** wird der VIEW der Komponente mit dem Namen **`compname`** aufgerufen, um den Inhalt der Komponente in HTML auszugeben.

Optionale Komponenten müssen nicht vorhanden sein, deshalb werden sie in der folgenden Form aufgerufen:

```
ifexists <compname> {Block}
```

`Block` ist der Programmteil in PVL, der aufgerufen wird, wenn die Komponente **`compname`** vorhanden ist. Zum Beispiel:

```
ifexists description { put [description „html“] }
```

Der Aufruf für Komponenten, die mehrfach vorkommen:

```
forall <compname> {Block}
```

Beispiel:

```
forall chapter { put „$chapter“ }
```

12.1.5 Die Page Instance Language (PIL)

Beschreibung

Ein Seitenobjekt in der **Page Instance Language** wird strukturiert durch Start- und Endmarkierungen (Tags), die das Objekt selbst und seine Komponenten begrenzen. Eine Markierung wird immer durch spitze Klammern eingeschlossen. Dabei enthält eine Startmarkierung immer einen Namen, optional gefolgt von einer Reihe von Attributen, in der Form

```
Attributname=“Attributwert“.
```

Welche Attribute dabei vorhanden sein müssen, und welche vorhanden sein können, wird in der Typdefinition festgelegt. Die dazugehörige Endmarkierung enthält denselben Namen mit vorangestelltem ‚/‘-Zeichen, oder als Kurzform nur ein ‚/‘.

Auf oberster Ebene ist der Markierungsname der Name des entsprechenden Seitentyps, den dieses Objekt zu erfüllen hat. Auf Komponentenebene ist es der Komponentenname.

Die **Formulas** dienen der automatischen oder manuellen Erzeugung von Referenzen. Zur automatischen Erzeugung von Referenzen verwendet man **select name**. Der Benutzer muß nach dem Gleichheitszeichen den Selektionsbereich angeben, aus dem die Referenzen generiert werden sollen. Anschließend kann er durch **category** ein Sortierkriterium bestimmen, womit festgelegt wird, daß z.B. nach Namen sortiert werden soll. Mit **info** legt man die Komponenten oder

Attribute aus den referenzierten Seitentypen fest, die als Beschreibung für den Link dienen sollen.

Dagegen kann man mit **<link ref = ... info = ...>** manuell Referenzen erstellen. Hierbei wird bei **link ref** die Zieladresse und bei **info** eine Kurzbeschreibung erwartet.

Die hauptsächliche Arbeit des Endbenutzers besteht darin, die Felder der einzelnen Bestandteile mit sinnvollen Inhalten zu füllen.

Die PIL-Grammatik

```
<ObjectDefinition>    ::|  ‚<‚TYPENAME <neededAttribut> <AttributeList> ‚>‘
                           <InstanceValues>
                           ‚</‘ TYPENAME ‚>‘
                        |  ‚<‚ TYPENAME <neededAttribut> <AttributeList> ‚>‘
                           <InstanceValues> ‚</>‘
<neededAttribut>      ::=  ‚name =‘ STRING
<AttributeList>       ::*  <Attribute>
<Attribute>           ::=  ATTR_NAME ‚=‘ STRING
<InstanceValues>      ::*  <Component>
<Component>           ::|  ‚<‚ COMPONENTNAME <AttributeList> ‚>‘
                           <Values> ‚</‘ COMPONENTNAME ‚>‘
                        |   ‚<‚ COMPONENTNAME <AttributeList> ‚>‘
                           <Values> ‚</>‘
<Values>              ::|  VALUE
                        |  VALUE <Component>
                        |  <Component>
                        |  <Formulas>
<Formulas>            ::*  <Formula>
<Formula>             ::|  ‚select name =‘ STRING
                        |  ‚category =‘ STRING
                        |  ‚info =‘ STRING
                        |  ‚<link ref =‘ STRING ‚info =‘ STRING ‚>‘
```

Hinweise:

- STRING und VALUE sind vom Benutzer auszufüllen.
- Alle anderen großgeschriebenen Wörter sind vom Konstrukteur in der dazugehörigen PDL vorgegeben.
- STRING steht für in Anführungszeichen eingeschlossene Inhalte.

Beispiel

Fortführung des PDL-Beispiels aus “Beispiel (ohne Views)” auf Seite 28:

```
<main_page name="PG269/PG-Startseite">
  <title>Projektgruppe 269 - AIR</>
  <author
    name="Lars Heete"
    mail_address="heete@ls10.informatik.uni-dortmund.de">
  </author>
  <overview>Advanced Information Rearrangement</>
  <index_pages>
    <!-- Die ausgewaehlten Seiten werden durch das select in der
    Klassen- beschreibung auf Seiten vom Typ Distrib_Page
    eingeschraenkt -->
    select name="PG269/*" <!-- interessante_links -->

    <!-- Die Links auf die Seiten werden nach Titel sortiert und
    mit der Inhaltsangabe angezeigt -->
    category="title"
    info="overview"
  </index_pages>
</main_page>


<distrib_page name="PG269/Teilnehmer">
  <title>Veranstalter und Teilnehmer</>
  <author
    name="Lars Heete"
    mail_address="heete@ls10.informatik.uni-dortmund.de">
  </author>
  <description>Teilnehmer der PG 269 A.I.R.</>
  <documents>
    select name="PG269/Homepages/*"
    category="author"
  </documents>
</distrib_page>


<home_page name="PG269/Homepages/Holger's Homepage">
  <title>Holger's Homepage</>
  <author
    name="Holger Isenberg"
    mail_address="isenberg@udo.informatik.uni-dortmund.de">
  </author>
  <photo>holger.gif</>
  <documents>
    select name="Isenberg/Dokumente/*"
    category="titel"
  </>
  <!-- unerlaubte Komponente,da in PDL nicht definiert -->
  <external_links>
    <link ref = „http://www.sinn.los.com" info="Irgendwas">
    <link ref = „http://www.dot.com" info="Noch mehr">
    <link ref = „./Diss.ps" info="Beobachtungen und Analysen zum
    Thema: Autostereogramme als Hintergrundbilder">
  </>
</home_page>
```

```
<distrib_page name="PG269/Interne Infos">
  <title>Internes</>
  <description>
    Restricted area. Authorization given to AIR-men only.</>
  <author
    name="Holger Isenberg"
    mail_address="isenberg@udo.informatik.uni-dortmund.de">
  </author>
  <documents>
    select name="PG269/Protokolle"
    category="datum"
    info="overview"
  </>
  <!-- unerlaubte Unterkomponente, da in PDL nicht definiert -->
  <text>
    <STRONG>Rudi:</STRONG> Treffen am Mittwoch immer von
    10:30 bis 11:45.
  </>
</distrib_page>


<!-- Seitentyp existiert nicht -->
<protocol_page name="PG269/Protokolle/Protokoll_1">
  <title>Protokoll vom 8.11.</>
  <author
    name="Holger Isenberg"
    mail_address="isenberg@udo.informatik.uni-dortmund.de">
  </author>
  <date>11:45 08.11.1995</>
  <protocol> Protokoll-Text</>
</protocol_page>
```

12.1.6 Zusammenhang zwischen PIL- und PDL-Grammatik

Beschreibung

Aus den Beispielen ist ersichtlich, daß zwischen den beiden Grammatiken für die PDL und PIL ein sehr enger Zusammenhang besteht.

Während die PDL die Bestandteile eines Seitentypes festlegt, kann anschließend in der PIL der Benutzer diese Bestandteile mit Inhalten füllen. In der PIL gibt es nur Attribute und Komponenten, die bereits in der PDL definiert worden sind. Auch können in der PIL keine neuen, d.h. noch nicht definierten, Seitentypen verwendet werden. Benötigt man also einen neuen Seitentyp, so muß der Konstrukteur diesen neu anlegen.

Der Aspekt der Views aus der PDL-Grammatik kann vernachlässigt werden, da für die Erstellung von Seitenobjekten keine Layoutinformationen erforderlich sind. Hier werden sie einfach weggelassen.

Man kann aus jeder PDL-Beschreibung einen Rahmen für ein PIL-Objekt generieren. Dazu übernimmt man einfach die IDs der Attribute und Komponenten der entsprechenden PDL und deren Oberklassen. Ebenso verfährt man bei geschachtelten Komponenten. Dabei läuft man durch alle übergeordneten Seitentypen. Die Attribute und Komponenten der übergeordneten Seitentypen werden *vor* den eigenen Bestandteilen eingefügt.

Eine Ausnahme bilden Attribute, diese stehen in der PIL in der Kopfzeile. So benötigt jedes Seitenobjekt grundsätzlich einen eindeutigen Namen, da diese Information schon beim Anlegen eines Seitenobjekts festgelegt werden muß.

Schwieriger wird es, wenn der **CompMod** der Komponenten in der PDL eine andere Auftrittshäufigkeit als *genau einmal* hat. Jetzt können Bestandteile nie oder beliebig oft auftreten. Die genaue Auftrittshäufigkeit kann man bei der Erzeugung eines solchen Typs im voraus nicht abschätzen, also muß der Benutzer beim Bearbeiten des Seitenobjekts entweder die überzählige Komponente entfernen, oder zusätzliche hinzufügen.

Beispiele

Das einfachste Beispiel. Ein leeres Objekt:

PDL:
TYPE page{
ATTRIBUTE name
}
	PIL:
	<page name = „Bsp1“>
	</page>
	
	oder
	<page name = „Bsp1“>
	</>

Übertragung von Komponenten und Attributen aus der PDL in PIL:

Aus **ATTRIBUTE tag** wird in der Kopfzeile tag = “ “.

Aus **COMPONENT test** wird **<test> </>** oder **<test> </test>**.

Ein weiteres Beispiel. Ein abgeleitetes Objekt

PDL:

```
TYPE page{
  ATTRIBUTE name
  COMPONENT titel:TEXT
}

TYPE divpage : page{
  ATTRIBUTE inhalt
  COMPONENT text : TEXT
}

TYPE div2page : divpage{
  COMPONENT datum : date
}
```

PIL:

```
<divpage name ="Bsp2" inhalt = „ein Text">
  <titel>Ein abgeleitetes Objekt</>
  <text> Das abgeleitete Objekt enthaelt auch alle Komponenten
         und Attribute des vererbenden Seitentyps.
  </>
</divpage>

<div2page name ="Bsp3" inhalt = „ein Text und ein Datum">
  <titel> Ein zweifach abgeleitetes Objekt </>
  <text> Das abgeleitete Objekt enthaelt auch alle Komponenten
         und Attribute der vererbenden Seitentypen. Dieses
         geschieht ueber beliebig viele Stufen.
  </>
  <datum> 1.1.96 </datum>
</div2page>
```

Verschiedene CompMod. Wiederholung der verschiedenen Modi der Komponenten:

Modus	Auftreten
keine Angabe	genau einmal
?	null oder einmal
*	null bis unendlich
+	eins bis unendlich

PDL:

```
TYPE page{
  ATTRIBUTE name
  COMPONENT einmal : TEXT
  COMPONENT null-eins ? : TEXT
  COMPONENT null-unendlich * : TEXT
  COMPONENT eins-unendlich + : TEXT
}
```

Erlaubte Seitenobjekte:

```
<page name = „Bsp4"><!-- Alle einmal -->
  <einmal>         </>
  <null-eins>      </>
  <null-unendlich> </>
  <eins-unendlich> </>
</page>

<page name = „Bsp5"><!-- Minimalanzahl -->
  <einmal>         </>
  <eins-unendlich> </>
</page>
```

```
<page name = „Bsp6"><!-- alle im oberen Bereich -->
  <einmal>          </>
  <null-eins>       </>
  <null-unendlich>  </>
  <null-unendlich>  </>
  <eins-unendlich>  </>
  <eins-unendlich>  </>
</page>
```

```
<page name = „Bsp7"><!-- gemischt -->
  <einmal>          </>
  <null-unendlich>  </>
  <null-unendlich>  </>
  <eins-unendlich>  </>
</page>
```

Beispielrahmen für ein solches Typobjekt:

```
<page name = „BspRahmen"><!-- genau einmal -->
  <einmal>          </>
  <!-- entfernen oder behalten -->
  <null-eins>       </>
  <!-- entfernen oder behalten oder vervielfaeltigen -->
  <null-unendlich>  </>
  <!-- behalten oder vervielfaeltigen -->
  <eins-unendlich>  </>
</page>
```

Geschachtelte Komponenten. a) Eine Diskussion:

PDL:

```
COMPONENT diskussion{
  COMPONENT these: TEXT
  COMPONENT meinungen{
    COMPONENT pro: TEXT
    COMPONENT kontra: TEXT
  }
}
```

Eine dazu passende Seiteninstanz:

PIL:

```
<diskussion>
  <these> a + b = c </these>
  <meinungen>
    <pro> stimmt </pro>
    <kontra> stimmt nicht </kontra>
  </meinungen>
</diskussion>
```

b) Eine Tabelle:

PDL:

```
COMPONENT tab{
  ATTRIBUTE titel
  COMPONENT zeile * {
    COMPONENT eintrag * : TEXT
  }
}
```

Dazu beispielhaft eine Tabelle mit zwei Zeilen je drei Einträgen:

PIL:

```
<tab titel = „Tabelle 1“>
  <zeile>
    <eintrag> A </>
    <eintrag> B </>
    <eintrag> C </>
  </zeile>
  <zeile>
    <eintrag> ja </>
    <eintrag> nein </>
    <eintrag> vielleicht </>
    </zeile>
</tab>
```

12.2 Der Parser

Der Parser übersetzt die Sprachen PDL und PIL, in denen Seitentypen und Seiteninstanzen definiert sind, in eine vom AIR-System einfacher benutzbare Form, die in der C++ Implementierung von AIR mehrfach genutzt werden kann. Während der Bearbeitung der PDL- und PIL-Eingaben des Benutzers wird der Parser von verschiedenen Stellen aus aufgerufen: z.B. zum Eintragen in AIRbase und zur Darstellung in HTML.

Um die Implementierung einfacher zu halten, ist das Design des Parsers objektorientiert. Anstelle eines einfachen Syntaxbaumes, der normalerweise von einem Parser ausgegeben wird, erstellt dieser Parser einen Baum von Objekten, die zusätzlich zu den eingelesen Werten Methoden auf diese enthalten. Diese Methoden werden entsprechend dem jeweiligen Anwendungsgebiet definiert. Die Parser-Klasse von AIRcontrol enthält nur Standardmethoden der Objekte, die entsprechend den Anforderungen erweitert werden können.

12.2.1 Ablauf des Parsing

Als Basis wird ein Typ-Speicher (**`AIR_TypeStore`**) verwendet. Der **`AIR_TypeStore`** verwaltet die Informationen, die durch die Seitenbeschreibungen gegeben sind. Zu Beginn muß zunächst ein Objekt der Klasse **`AIR_TypeStore`** angelegt werden (**`new AIR_TypeStore`**) und um die benötigten Basistypen ergänzt werden (**`addType()`**).

Eingelesen werden die Seitentypen mit der Methode **`addType()`** der Klasse **`AIR_TypeStore`**.

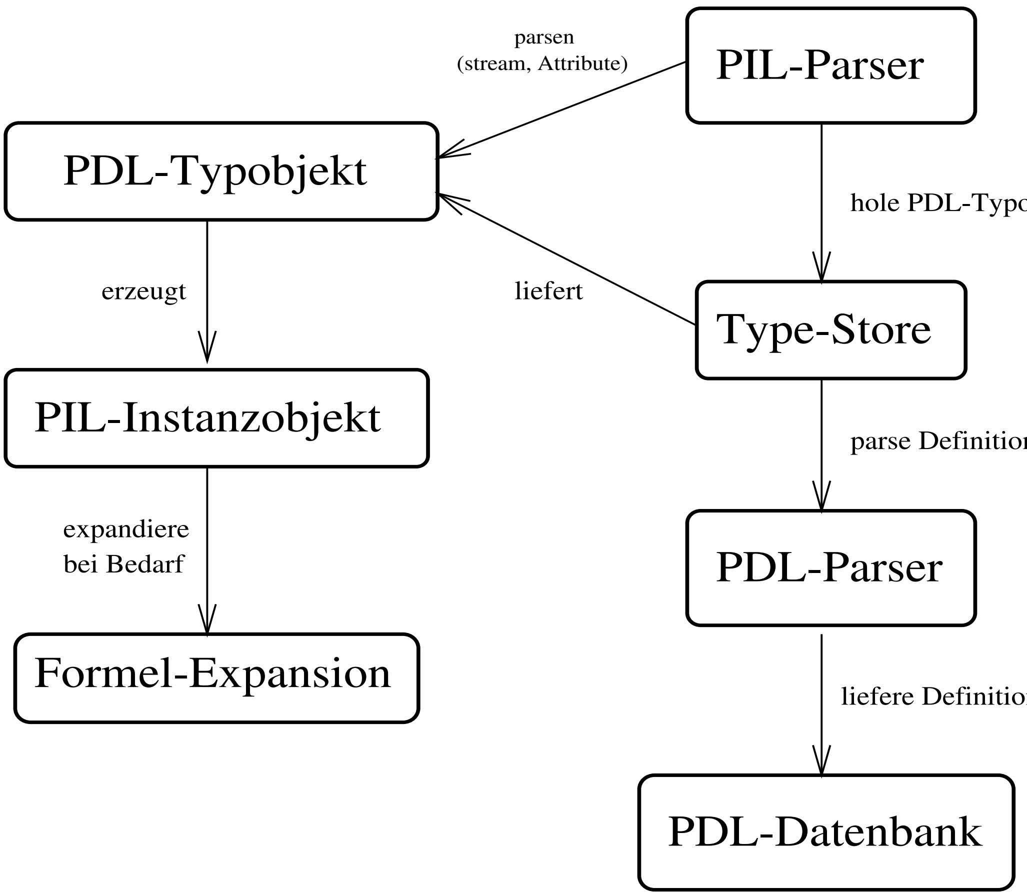
Mit der Methode **`lookupType()`** erhält man den unter dem angegebenen Namen eingetragen Datentyp.

Ein Objekt der Klasse **`AIR_TypeStore`** enthält eine Methode, die einen Parser für die PIL bereitstellt. Dieser ist ein Objekt der Klasse AIR_InstanceParser. Es enthält die Methoden **`par-`**

seObjekt zum Einlesen einer Seiteninstanz und die Methode **parseTag** zum Einlesen eines einzelnen Tags.

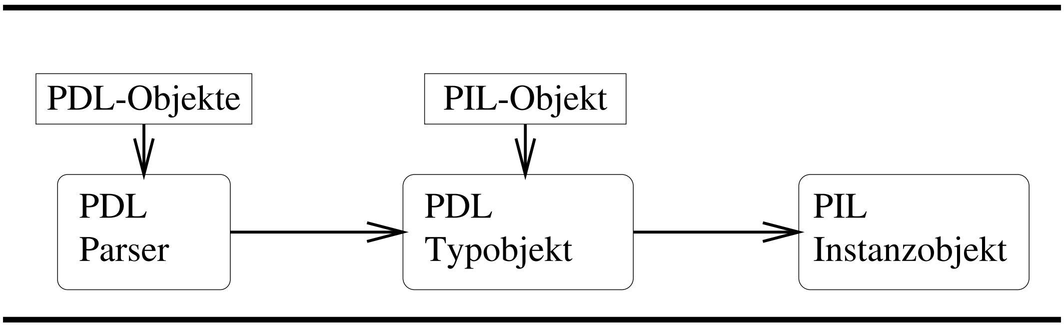
Das Ergebnis beim Einlesen einer Seiteninstanz ist ein Objekt der Klasse **AIR_Instance**.

Bei allen Parsevorgängen werden die Informationen zu den verwendeten Datentypen aus dem AIR_TypeStore geholt. Dabei wird bei Bedarf auf die Datenbank zugegriffen.



Ablauf des Parsvorgangs

12.2.2 Ergebnis des Parsing



Ebenen des Parsers

Ein Objekt der Klasse **`AIR_Instance`** bietet Methoden an, um den Inhalt dieses Objektes auszulesen. In einem anderen Teil des AIR-Systems kann dadurch die Klasse **`AIR_Instance`** z.B. um eine Methode zur Darstellung in HTML erweitert werden.

Alle Seitentypen werden in einer Vererbungshierarchie, ausgehend von dem Typ *Page*, in PDL definiert. Aus diesen Definitionen erzeugt der PDL-Parser ein AIR_Type-Objekt. Jeder Seitentyp wird dabei auf ein AIR_Type-Objekt abgebildet, das dann Methoden anbietet, um Informationen über Attribute und Komponenten und Views zu erhalten. Jede Komponente kann dabei wieder ein AIR_Type-Objekt erhalten.

Ein AIR_Type-Objekt enthält nach dem Parse-Vorgang also die gesamte Information der zugehörigen PDL-Beschreibung. Diese Information wird auch für das Parsing eines PIL-Objektes benötigt. Hierzu bietet jedes AIR_Type-Objekt eine Methode, die mit einer PIL-Beschreibung als Eingabestrom aufgerufen wird.

Ist diese PIL-Beschreibung korrekt, erhält man dann das Instanzobjekt für diese Seite. Dieses AIR_Instance-Objekt kann dann z.B. über zusätzlich definierte Viewmethode in HTML ausgegeben werden.

Als Schnittstelle zum gesamten AIR-System stehen folgende Operationen und Klassen bereit, die in den nächsten Abschnitten beschrieben werden.

- class AIR_InstanceParser
- class AIR_TypeStore
- class AIR_Type
- class AIR_Instance
- class AIR_FormulaInstance

12.2.3 Parserklassen

class AIR_InstanceParser

Ein AIR_InstanceParser wird erzeugt für einen AIR_TypeStore, der ihm die Typobjekte bereitstellt. Er ist in der Lage, aus einem Eingabestrom die PIL-Beschreibung eines Seitenobjektes zu lesen. Als Ergebnis erhält man ein Objekt der Klasse AIR_Instance.

- Konstruktoren
  - AIR_InstanceParser(AIR_TypeStore types)
    Erzeugt Instanzparser, der auf Typspeicher **`types`** zurückgreift.

- Methoden
  - AIR_Instance* parseObject(istream& in)
    Liest eine Seitenbeschreibung aus dem Eingabestrom und liefert dessen Wert.
  - static void parseTag(istream& in, String& name, list<Attr>& attrs)
    Liefert Name und Attributliste eines Tags aus dem Eingabestrom.

class AIR_TypeStore

Ein AIR_TypeStore liefert dem Parser für Typnamen die entsprechenden Typobjekte. Daneben enthält er Konstruktoren für besondere Wertobjekte, wie Formeln.

- Methoden
  - void addType(String name, AIR_Type* type)
    void addType(istream& in)
    Registriere neuen Typ unter dem angegebenen Namen.
  - AIR_Type* lookupType(String name)
    Liefere Typobjekt für den angegebenen Namen.

class AIR_Type

AIR_Type stellt alle Informationen über einen Seiten- bzw. Komponententyp bereit. Daneben macht es den Konstruktor zugänglich, der eine Instanz dieses Typs aus einem Eingabestrom liest.

- Konstruktor
  - AIR_Type(AIR_Type *parent, createProc create)
    Erzeugt neues Typobjekt mit der Vaterklasse parent und der Instanzerzeugungsprozedur create.
- Methoden
  - virtual void deriveType()
    Leite neuen Typ von bestehendem ab.
  - void defineConstant(String name, String value)
    Definiere Attribut als Konstanten Wert.
  - void defineAttribute(String name, AttrMod mod)
    Füge Attribut hinzu.
  - void defineComponent(String name, AIR_Type* type)
    Komponentendefinition.
  - AIR_Type* getComponent(String name)
    Liefert den Typ der angegebenen Komponente.
  - void defineView(String name, String view)
    void defineView(String name, String part, String view)
    Viewdefinition.
  - String getView(String name)
    String getView(String name, String part)
    Liefert View.

- AIR_Instance* readInstance(istream& in, list<Attr>& attrs)
  liest eine Instanz dieses Typs von dem Eingabestrom

class AIR_Instance

AIR_Instance repräsentiert einen Knoten in dem Wertebaum, den der Parser liefert. Es ist es möglich, für alle Unterkomponenten dieses Wertes weitere AIR_Instance Objekte zu erhalten.

- Konstruktor
  - AIR_Instance(AIR_Type *type, istream& in, list<Attr>& attrs)
    Erzeugt neue Instanz mit dem type type aus dem Eingabestrom input.
- Methoden
  - AIR_Instance* getComponent(String name)
    liefert den Wert einer Komponente.
  - String getAttribute(String name)
    liefert den Wert eines Attributes.

class AIR_FormulaInstance

Die Klasse AIR_FormulaInstance ist abgeleitet von AIR_Instance. Objekte dieser Klasse werden vom InstanceParser an den Stellen eingesetzt, an denen in dem Seitenobjekt Formeln stehen. Das Formelobjekt liefert dann erst bei Bedarf die Werte, indem es die Formel auswertet. Dabei muß sichergestellt sein, daß der Wert mit dem angeforderten Typ übereinstimmt.

- Konstruktor
  - AIR_FormulaInstance(AIR_Type *type, list<Attr>& attrs, String fml)
    Erzeugt ein Formelobjekt.

12.2.4 Verwendung des Parsers

Das Ergebnis eines korrekten Parsevorgangs ist ein Baum von Instanzobjekten. Um nun eigene Operationen auf diesem Baum durchführen zu können, muß man eigene Instanzklassen ableiten können, die diese Operationen unterstützen. Die AIR_InstanceParser Klasse ist daher so angelegt, daß man beim Erzeugen eines Parsers festlegen kann, welche Instanzklassen verwendet werden. Dazu dient das AIR_TypeStore Objekt, mit dem der Parser erzeugt wurde. In diesem TypeStore ist für jede Instanzklasse ein Typobjekt (Klasse AIR_Type) einzufügen, daß u.a. einen Pointer auf eine create Funktion der Instanzklasse enthält. Diese Funktion erzeugt dann ein Objekt dieser selbstdefinierten Instanzklasse.

Beispiel

```
class compilerInst: public virtual AIR_Instance {
  public:
    virtual void executeView(ostream& output, String name);
};



class compilerStringInst: public compilerInst,
  public AIR_StringInstance {
    static AIR_Instance *create(AIR_Type *type, istream& in,
                                list<Attr>& attrs);
};
```

```
class compilerCompoundInst: public compilerInst,
  public AIR_CompoundInstance {
    static AIR_Instance *create(AIR_Type *type, istream& input,
                                list<Attr>& attrs);
};
```

Hier wird mit der Klasse CompilerInst zunächst die Klasse AIR_Instance um die Operation executeView erweitert. Alle anderen selbstdefinierten Instanzklassen erben zum einen von dieser CompilerInst Klasse die Operation execute View, zum anderen benötigen sie aber auch die Operationen der vordefinierten Instanzklasse. Da AIR_Instance virtuelle Basisklasse für compilerInst und AIR_StringInstance ist, werden bei mehrfacher Vererbung die Variablen der AIR_Instance Klasse nur einmal angelegt.

Zum Erzeugen eines Parsers wird dann zunächst ein AIR_TypeStore Objekt angelegt, in dem die selbstdefinierten Instanzklassen registriert werden. Dazu erzeugt man für jede Instanzklasse ein AIR_Type Objekt, das mit einer Konstruktorfunktion parametrisiert wird, und fügt es mit addType unter dem PDL-Typnamen in das AIR_TypeStore Objekt ein.

```
AIR_TypeStore *types = new AIR_TypeStore();

types->addType
        („string‟, new AIR_Type(compilerStringInst::create));

types->registerCompoundType
        (new AIR_Type(compilerCompoundInst::create));

AIR_InstanceParser* parser = new AIR_InstanceParser(types);

// PIL-Objekt von der Standardeingabe lesen
compilerInst *inst
    = dynamic_cast<compilerInst*>AIR_parser->parseObject(cin);

// HTML ausgeben
inst->executeView(cout, html);
```

12.2.5 Beispiel zur Funktionsweise des AIR_TypeStore

Ein AIR_TypeStore wird initialisiert mit den vom AIR-System vorgegebenen Basistypen. Wird nun ein unbekannter Seitentyp angefordert, so holt der TypeStore sich aus der Datenbank die entsprechende PDL-Definition und erzeugt daraus die Typbeschreibung. Dieser Vorgang soll nun anhand eines Beispiels erläutert werden.

Der Instanzparser benötigt das Typobjekt für den Seitentyp air_page. Mit

```
types->lookupType(„air_page‟);
```

fordert er vom TypeStore dieses an. Der TypeStore kennt diesen Typ noch nicht, und läßt sich von der Datenbank die Definition liefern.

```
TYPE air_page:page {
  COMPONENT author {
    ATTRIBUTE name:STRING
    ATTRIBUTE mail_address:STRING
    VIEW html {
      put „<A href=\"mailto:$mail_address\">$name</A>"
    }
  }
  VIEW html {
    <!-- erweitert „inner" von page -->
    inner body
    put „<HR>"
    put [author html] <!=
    inner footline
  }
}
```

Ein PDL-Parser liest diese Definition ein und fordert wiederum von dem TypeStore das Typobjekt für die Vaterklasse *page* an. Er erzeugt ein Typobjekt der Klasse AIR_CompoundType mit dem vom Typestore erhaltenen, abzuleitenden Vaterklasse. Die Attribute, Komponenten und Viewbeschreibungen werden dann mit den Methoden **defineAttribute**, **defineComponent** und **defineView** in diesem AIR_CompoundType Objekt registriert.

Kapitel 13

AIRbase

13.1 Einleitung

AIRbase stellt die Datenbank des AIR-Projektes dar. Die Aufgabe dieser Datenbank ist es, die in diesem Projekt verwendeten Seitenobjekte zu erzeugen, zu speichern und wieder zur Verfügung zu stellen.

Innerhalb des AIR-Systems wird ein konkretes Seitenobjekt definiert mittels der PIL. Die Elemente dieses Seitenobjektes wurden von der PDL abstrakt vorgegeben. Das konkrete Aussehen dieses Seitenobjektes wird wiederum von der PVL fesgelegt. Aus dieser Konstellation ergeben sich eine Reihe von Anforderungen für die Datenbank.

Das Erscheinungsbild eines Seitenobjektes wird von der PVL festgelegt, indem dort verschiedene Objekte (Texte, Grafiken, etc.) angeordnet werden. Es kann zum Beispiel diese Anordnung variiert werden, es können einzelne Objekte wiederholt vorkommen usw. Um an dieser Stelle zu gewährleisten, daß dem Gestaltungswillen möglichst wenig Grenzen gesetzt sind, sollten diese Objekte kein integraler Bestandteil der Beschreibung sein, sondern separat verfügbar. Erst dann können sie beispielsweise immer wieder verwendet werden.

Es ist also auch Aufgabe der Datenbank, diese Objekte zu speichern, zur Verfügung zu stellen und zu verwalten.

Da diese Objekte auch über verschiedene Attribute verfügen können, wie z.B. einem Haltbarkeitsdatum, ist es eine weitere Aufgabe von AIRbase, diese Attribute zusätzlich zu den Objekten zu verwalten. Im Rahmen dieser Attributverwaltung werden auch verschiedene Spezialfunktionen, die sich auf die Attribute beziehen, zur Verfügung gestellt.

Des weiteren wird von AIRbase auch die Page-Definition-Language (PDL) verwaltet. Diese Funktionen sind denen der Objekt-Verwaltung sehrähnlich, wobei sie sich im Moment noch auf das Speichern und Zurückgeben der PDL beschränkt.

13.2 Verwendung der AIRbase

Die AIRbase ist als eine Klasse in C++ implementiert. Um diese Klasse verwenden zu können, ist es notwendig, die Datei AIRBase.h einzubinden. Die Datenbank wird initialisiert, wenn in dem aufrufenden Programm ein Objekt vom Typ AIR_Base erzeugt wird.

Um die Attributverwaltung den Erfordernissen anzupassen, muß eventuell in der Datei AttrDefs.h der Name der Attributdatei <AttrFileName> angepaßt werden.

13.3 Die Schnittstellen

In der Klasse AIRbase sind die folgenden Funktionen deklariert.

13.3.1 Allgemein

AIR_Base();

Konstruktor der AIR_Base.

~Air_Base();

Destruktor der AIR_Base.

13.3.2 Objektfunktionen

String CreateObject (String name, String TypID)

Anforderung: CreateObject erhält als Eingabe den Namen, in der Form (Domain/Identifier), sowie den Typidentifikator des Objektes. Es wird ein Template generiert, das dem Typen entspricht und zum Bearbeiten in das Dateisystem geschrieben wird. Die Rückgabe enthält den Dateinamen.

bool RemoveObject (String name)

Anforderung: Die Eingabe ist der Objektname in der Form (Domain/Identifier). Die Funktion liefert den Wert true zurück, falls ein Objekt mit diesem Namen in der angegebenen Domain aus der Datenbank entfernt werden konnte.

String CheckOutObject (String name)

Anforderung: Die Eingabe ist der Objektname in der Form (Domain/Identifier). Die Ausgabe ist der Name der Datei in die das Objekt geschrieben wurde.

Wird der Funktion ein Wildcard-Ausdruck übergeben, so enthält die Datei alle Objekte, auf die der Wildcard-Ausdruck zutrifft. Diese Objekte werden alle in eine Datei geschrieben.

bool CheckInObject (String name)

Anforderung: Die Eingabe ist der Name einer Datei. In dieser Datei ist ein Objekt enthalten, das dann in die Datenbank eingecheckt wird.

Es können nur Objekte eingecheckt werden, die vorher auch ausgecheckt wurden. Die Funktion liefert den Wert true zurück, wenn ein Objekt erfolgreich eingecheckt wurde.

String ReadOnlyObject (String name)

Anforderung: Die Eingabe ist der Objektname in der Form (Domain/Identifier). Die Ausgabe ist der Name der Datei in die das Objekt geschrieben wurde.

Wird der Funktion ein Wildcard-Ausdruck übergeben, so enthält die Datei alle Objekte, auf die der Wildcard-Ausdruck zutrifft. Diese Objekte werden alle in eine Datei geschrieben.

Im Unterschied zu CheckOutObject kann ein so gelesenes Objekt nicht wieder eingecheckt werden.

bool RemoveLog (String name)

Anforderung: Die Eingabe ist der Objektname in der Form (Domain/Identifier). Die Ausgabe ist TRUE, wenn die Operation erfolgreich war.

Die Funktion löscht ein Attribut, das beim Auschecken eines Objektes mit der Funktion CheckOutObject gesetzt wurde. Nach dem Aufruf von RemoveLog kann ein Objekt wieder ausgecheckt werden ohne daß es zuvor mit CheckInObject eingecheckt wurde.

13.3.3 PDL-Funktionen

String CheckOutPDL (String name)

Anforderung: Die Eingabe ist der Name der PDL in der Form (Domain/Identifier). Die Ausgabe ist der Name der Datei in die die PDL geschrieben wurde.

bool CheckInPDL (String name)

Anforderung: Die Eingabe ist der Name einer Datei. In dieser Datei ist eine PDL enthalten, die dann in die Datenbank eingecheckt wird.

Die Funktion liefert den Wert true zurück, wenn eine PDL erfolgreich eingecheckt wurde.

String ReadOnlyPDL (String name)

Anforderung: Die Eingabe ist der Name der PDL in der Form (Domain/Identifier). Die Ausgabe ist der Name der Datei in die die PDL geschrieben wurde.

Im Unterschied zu CheckOutPDL kann eine so gelesene PDL nicht wieder eingecheckt werden.

String CreatePDL (String name, String TypID)

Anforderung: CreatePDL erhält als Eingabe den Namen, in der Form (Domain/Identifier), sowie den Typidentifikator der PDL. Es wird ein Template generiert, das zum Bearbeiten in das Dateisystem geschrieben wird. Die Rückgabe enthält den Dateinamen.

13.3.4 Attribut-Verwaltung

Variablen

Die folgenden Funktionen enthalten einige wiederkehrende Übergabewerte, die an dieser Stelle erläutert werden sollen.

Es handelt sich dabei um:

- String objname
- Der eindeutige Name eines Objektes.
- String attrname
- Die Bezeichnung eines Attributes.
- String attrval
- Der Wert, respektive der Inhalt eines Attributes.

Funktionen

bool SetAttribute (String objname, String attrname, String attrval)

Mit dieser Funktion wird für ein Objekt ein bestimmtes Attribut gesetzt. Sollte dieses Attribut für das angegebene Objekt bereits gesetzt sein, wird es von der Funktion überschrieben.

Die Funktion liefert immer true (1) zurück, da bis jetzt keine Beschränkungen für einen der drei Übergabewerte vorliegen, und somit die Operation immer erfolgreich ist.

String GetAttribute (String objname, String attrname)

Diese Funktion liefert den Wert eines gegebenen Attributes für ein gegebenes Objekt zurück.

Falls für dieses Objekt kein Attribut des angegebenen Names vorliegt, oder eventuell das Objekt selbst nicht existiert, wird ein leerer String zurückgegeben.

bool SetCompileDateTime (String objname)

Diese Funktion speichert die aktuelle Compilierzeit für das gegebene Objekt ab, um sie für die Haltbarkeitsdatum-Abfrage (ListUpdateObjects) bereitzustellen.

Der Rückgabewert ist true, wenn das Attribut CompileTime für das gegebene Objekt gesetzt werden konnte.

bool SetBestBefore (String ObjName, int day, int Month, int Year)

Diese Funktion setzt für das angegebene Objekt ein Haltbarkeitsdatum. Die Übergabe des gewünschten Datums erfolgt in der Form:

Day	der Tag des Monats	(1 - 31)
Month	der Monat des jahres	(1 - 12)
Year	das Jahr - vierstellig	(xxxx)

Man erhaelt dieses Haltbarkeitsdatum zurück, indem man

```
GetAttribute("ObjName", BestAttr)
```

aufruft. der Wert, den man hier zurückbekommt, ist dann allerdings der String eines Longintegers vom Typ time_t aus <time.h>.

bool FindFirstUpdate(String& objname)

Diese Funktion beginnt eine Suche nach Objekten, deren Haltbarkeitsdatum abgelaufen ist. Die Daten, die hier zugrunde gelegt werden, sind einerseits die aktuelle Systemzeit und andererseits das Attribut "BestBefore", also das Haltbarkeitsdatum eines Objektes. Der Name eines gefundenen Objektes wird in objname zurückgegeben.

Wird kein entsprechendes Objekt gefunden, wird eine 0 als Funktionswert zurückgegeben.

bool FindNextUpdate(String& objname)

Diese Funktion sucht das nächste Objekt, dessen Haltbarkeitsdatum abgelaufen ist. Der Name eines gefundenen Objektes wird in objname zurückgegeben.

Kann kein weiteres Objekt gefunden werden, wird eine 0 als Funktionswert zurückgegeben.

Attribute

Es gibt einige bereits vordefinierte Attribute, die von verschiedenen Spezialfunktionen benutzt werden. Diese Attribute können natürlich auch ganz regulär mit den Grundfunktionen SetAttribute und GetAttribute benutzt werden. Und zwar handelt es sich im einzelnen um:

BestBefore. Dieses Attribut steht für das Haltbarkeitsdatum eines Objektes. Objekte, die dieses Attribut nicht haben, haben demnach auch keine Gültigkeitsbeschränkung.

Dieses Attribut ist in der Datei AttrDefs.h vordefiniert als BestAttr.

CompileTime. In diesem Attribut ist der Zeitpunkt festgehalten, zu dem das Objekt zuletzt kompiliert wurde. Dieser Zeitpunkt umfaßt die Uhrzeit und das Datum. Dieses Attribut wird von der Attributverwaltung nicht automatisch erstellt, sondern muß mit SetCompileDateTime explizit gesetzt werden.

Dieses Attribut ist in der Datei AttrDefs.h vordefiniert als CompAttr.

13.3.5 Verzeichnisfunktionen

stream DirPDL (String str =“*")

Anforderung: Die Eingabe dieser Funktion ist ein Suchbegriff. Es werden die Namen aller PDLs ausgegeben, die dem Suchbegriff entsprechen. Ein Suchbegriff kann die folgende Form haben:

```
cout << DirPDL ("DOMAIN=pg269/Homepages AUTHOR=AIR-Craft");
```

Die Default-Eingabe ist das Wildcard *. Hierbei würden alle vorhanden PDLs ausgegeben.

stream DirPIL (string str =“*")

Siehe DirPDL.

13.4 Technische Details

13.4.1 Allgemeines

Die AIRBase besteht momentan aus den folgenden Dateien:

AIRbase allgemein:

AIRBase.h	Header-Datei mit den Deklarationen der Klasse AIR_Base
AIRBase.C	C-Datei mit den Definitionen der KLasse AIR_Base

Attributverwaltung

AttrDefs.h	Header-Datei mit Konstantendefinitionen
DBObject.h	Header-Datei mit den Deklarationen der Klasse DBObject (der Attributverwaltung)
DBObject.C	C-Datei mit den Definitionen der Klasse DBObject
ClassAttrMap.h	Header-Datei mit den Deklarationen der Klasse Class_AttrMap
ClassAttrMap.C	C-Datei mit den Definitionen der Klasse Class_AttrMap

Die Klasse AIR_Base verwendet zur Verwaltung von Zeichenketten nicht die üblichen char-Pointer sondern die String-Klasse von Lars Heete aus der Datei <String.h>.

13.4.2 Objektverwaltung

Es wurde entschieden, daß für die Funktionen der Objektverwaltung das Werkzeug RCS zu Hilfe genommen wird. RCS bietet für dieses Projekt mehrere Vorteile.

Zum ersten wäre da der Umstand, das RCS ein Werkzeug ist, welches sich in der Anwendung bereits bewährt hat. Der Funktionsumfang von RCS und seine Möglichkeiten sind bekannt und ihre Funktionalität gewährleistet. Des weiteren bietet RCS Mechanismen welche die Sicherheit der bei diesem Projekt entstehenden Objekte garantiert.

So können Objekte vor unbefugtem Zugriff geschützt werden. Unbefugt soll in diesem Zusammenhang bedeuten, daß jeder Zugriff auf die Objekte, der nicht über das AIR-System erfolgt, unterbunden wird. Es kann so auch verhindert werden, daß mehrere Personen gleichzeitig dasselbe Objekt bearbeiten.

Außerdem beinhaltet RCS die Möglichkeit einer Versionsverwaltung für die Objekte. Es können also problemlosältere Versionen der Objekte wiederhergestellt werden. Dies ist allerdings eine Funktion, die bisher nicht explizit gefordert wurde.

13.4.3 Attributverwaltung

Beim Aufruf der Attributverwaltung, also beim Aufruf der Klasse AIR_Base, wird eine Datei mit bereits erstellten Attributen vollständig eingelesen. Der Name dieser Datei wird festgelegt über die Konstante AttrFileName aus der Datei AttrDefs.h. Die Datei selbst ist eine einfache Textdatei, bei der jede Zeile eine Wertegruppe darstellt. Eine solche Wertegruppe teilt sich auf in Objektname, Attributname und Attributwert, die jeweils durch ein Semikolon getrennt sind.

Die eingelesenen Daten werden intern gespeichert und verwaltet mit Hilfe einer Map, einer Datenstruktur, in der Werte mittels eines Schlüsselindizes gespeichert werden.

Die hier verwendete Map entstammt dem Library-Paket STL.

13.5 Ausblick

Da bis zum Stichtag, dem 15.6.1996, die Datenbank nicht eingebunden worden ist, läßt sich leider nicht feststellen, ob sie in dem Kontext des AIR-Systems überhaupt funktionsfähig ist.

Objektverwaltung (???).

Die Attributverwaltung präsentiert sich im Moment als nicht besonders elegant, aber immerhin funktionsfähig. Für diese Funktionsfähigkeit kann man sich natürlich nur unter Testbedingungen, und nicht innerhalb des AIR-Systems verbürgen. Wenn sich die Anforderungen für die Attributverwaltung nicht ändern sollten ist sie in der vorliegenden Form durchaus praktikabel und benutzbar.

Einige Dinge bleiben bei der Attributverwaltung noch zu überlegen. Ist es zum Beispiel tatsächlich praktisch, alle Attribute im Speicher zu halten, oder ist bereits abzusehen, daß man mit dieser Methode irgendwann auf Engpässe stößt? Und spätestens, wenn man soweit ist, daß man einige Daten auf der Platte halten muß, sollte man auch darüber nachdenken, ob man die Daten nicht vielleicht in einem anderem Format als dem reinen Textformat speichert.

13.6 Fileverwaltung

13.7 Motivation

Die Fileverwaltung hat die Aufgabe, Objekte aufzunehmen, zu speichern und zur weiteren Bearbeitung verfügbar zu halten. Auf die Objekte selbst darf nur über die Fileverwaltung zugegriffen werden, um Fehler aufgrund gleichzeitiger konkurrierender Bearbeitung zu vermeiden. Darüberhinaus sollen sogenannte Domains eine Strukturierung der Daten ermöglichen. Domains sind Bezeichner, unter denen der AIR-Benutzer sinnzusammenhängende Daten sammeln kann, beispielsweise die Daten aller PG-Teilnehmer in der Domain "'PG-Mitglieder"'. Eine derartige Einteilung versetzt uns in die Lage, zukünftig Benutzern gezielt Zugriffsrechte auf einzelne Domains zu erteilen oder zu verweigern, was die universelle Einsetzbarkeit von AIR deutlich erweitert.

13.7.1 Implementation

Das Aufgabenprofil der Fileverwaltung entspricht im wesentlichen der eines Software-Version skontrollsystems. Es bot sich daher an, das weit verbreitete Tool RCS als Implementationsgrundlage zu wählen, und die für AIR notwendigen Funktionalitaeten auf RCS abzubilden. Dies hat auch den zusätzlichen angenehmen Nebeneffekt, daß für AIR alle früheren Versionen eines Objekts zur Verfügung stehen und bei Bedarf wiederbeschafft werden können.

Der Aufbau der Fileverwaltung ist wie folgt: Alle Objekte werden an der mit AIRHOME bezeich
neten Stelle im Verzeichnisbaum doppelt abgelegt: Einmal die neueste Version eines Objekts im Klartext, und einmal als RCS-File. Während sich der Pfad des Klartext-Objekts unmittelbar aus der Stringkonkatenation **AIRHOME+Domain+Objektname** ergibt, wird das RCS-File unter **AIRHOME+Domain+"'RCS/"'+Objektname** ermittelt. Diese doppelte Speicherung ist erforderlich, um einerseits AIR einen flüssigen Zugriff auf benötigte Objekte zu ermöglichen (es wäre viel zu ineffizient, ein Objekt bei Bedarf immer wieder auszuchecken), und andererseits die gewünschten Zugriffskontrollen zu gewährleisten.

Die Implementationsschnittstelle der Fileverwaltung liegt in der AIRBase-Klasse. Sie bietet fol gende Zugriffsfunktionen an:

int CreateDomain (const char *domain);

Lege die gewünschte Domain samt zugehörigem RCS-Directory an. Nicht existierende Zwischendirectories werden dabei automatisch angelegt.

int DeleteDomain (const char *domain);

Lösche die angegebene Domain inklusive aller in ihr gespeicherten Objekte komplett. Es sollte offensichtlich sein, daß diese Funktion nur mit allergrößter Vorsicht benutzt werden (und an der Bedienungsoberfläche durch entsprechende Sicherungsabfragen geschützt) werden sollte.

int DomainExists (const char *domain);

Zeigt an, ob eine Domain unter dem angegebenen Namen existiert.

int SaveFile (const char *file, const char *domain);

Speichert das angegebene File in der angegebenen Domain und checkt es automatisch in RCS ein. Sollte dieses File nicht bereits existiert haben, wird es automatisch angelegt.

int EditFile (const char *file, const char *domain);

Checkt das angegebebe File zur Bearbeitung in das aktuelle Arbeitsverzeichnis aus und markiert es als gelockt, um eine konkurrierende Bearbeitung zu verhindern.

int GetFile (const char *file, const char *domain);

Checkt das angegebene File zur Ansicht in das aktuelle Arbeitsverzeichnis aus. Es kann jedoch nicht veraendert oder wieder eingechecked werden.

int DeleteFile (const char *file, const char *domain);

Loescht das angegebene File aus dem RCS-Repository und der Fileverwaltung. Aus offensichtlichen Gruenden sollte diese Funktion mit angemessenen Absicherungen benutzt werden.

int FileExists (const char *file, const char *domain);

Stellt fest, ob ein File des angegebenen Namens existiert.

FILE *GetObject (const char *file, const char *domain);

Liefert den Filepointer der angegebenen Datei. Es sollte noch einmal ausdruecklich darauf hingewiesen werden, dass ausschliesslich ueber diese Funktion auf den Inhalt des gespeicherten Objektes zugegriffen werden soll, um eine saubere Zugriffskontrolle zu etablieren.

Kapitel 14

Implementierung

14.1 Übersicht

Zur Implementierung von AIR haben wir das objektorientierte Modell gewählt. Dieses Modell bietet sich hier besonders an, da in AIR Seitenobjekte und Seitenbeschreibungen ebenfalls objektorientiert modelliert werden. Die Objekte der PIL und PDL werden so direkt auf Objekte in C++ Ebene abgebildet.

Zur Verwaltung der Seiten- und Typobjekte wird je ein Instanz- und Typ-Speicher verwendet. Über den Typ-Speicher werden neue Seitentypen in das System eingefügt und bestehende verändert. Die Seiten- und Typobjekte werden intern in PDL und PIL abgelegt. Während der Laufzeit des Systems werden diese Beschreibungen wieder in die entsprechenden C++ Objekte umgesetzt. Das System und der Benutzer erhält so immer die aktuellen Definitionen.

Eingelesen werden die Typobjekte von einem Parser für die Seitenbeschreibungssprache PDL. Die Scanner- und Parser-Generatoren **lex** und **yacc** vereinfachen die Umsetzung der PDL-Grammatik in die Parserimplementierung.

Zur Implementierung des PIL-Parsers kann kein Generator verwendet werden, da die Symbole (<tags>) erst zur Laufzeit des Systems bekannt sind.

Die Instanz- und Typ-Speicher verwenden intern ein Datenbank-Interface oder andere Möglichkeiten zur permanenten Speicherung. Bei der Auswahl eines Datenbanksystems besteht hier keine Einschränkung, da die Seiten- und Typobjekte in Form von PIL und PDL Texten gesichert werden.

Bei der Auswahl des Datenbanksystems werden so die jeweiligen Anforderungen an das AIR-System berücksichtigt. Für einen relativ kleinen Datenbestand, wie er in dem Anwendungsbeispiel verwendet wird, reicht eine einfache Sicherung in einem normalen Dateisystem aus. In größeren Systemen können beliebige relationale oder objektorientierte Datenbank-System eingesetzt werden.

Ein weiteres Ziel unserer Implementierungsform ist die Möglichkeit zur einfachen Erweiterung der Datentypen. Neue Typklassen werden als Basistypen definiert, um andere Datentypen, als die bestehenden, in den Seitenobjekten zu verwenden. Z.B. Film- und Tondokumente.

Die Implementierung von Selektionsformeln, setzt auf dem Basistyp **List** auf. Dieses Typobjekt bietet Methoden zur Verwaltung von Seitenobjekten in Listen. Es steht zusätzlich eine Methode bereit, die aus einer Selektionsformel, aus einem angegeben Bereich, z.B. den Seitennamen, eine Liste von Instanzobjekten generiert.

Aus dem Ergebnis können so Index und Inhaltsverzeichnis über den View generiert werden.

14.1.1 Details der Seiten- und Typobjekte

Ein Typobjekt, als Abbildung einer Seitenbeschreibung aus PDL, enthält die Informationen des Seitentyps, die über Methoden abgefragt werden. Jedes Typobjekt enthält einen Konstruktor für ein Instanzobjekt dieses Typs. Zur Generierung des Instanzobjekts wird die Objektbeschreibung in PIL-Form eingelesen.

Ein Instanzobjekt, als Abbildung eines Seitenobjekts aus PIL, stellt Methoden bereit, um den Inhalt der Attribute und Komponenten zu lesen und zu ändern. Die Komponenten werden ebenfalls als Instanzobjekte behandelt. Zur Darstellung des Objekts (z.B. in HTML) wird die Klasse des Instanzobjekts erweitert. Aus einer allgemeinen View-Klasse und der entsprechenden Instanzklasse wird diese abgeleitet.

Über eine Methode (View) des Instanzobjektes kann eine Darstellung generiert werden, die von dem angegebenen Parameter abhängt (HTML für einen WWW-Server). Die View-Methode eines Instanzobjektes für eine gesamte Seite erzeugt die Darstellung über Aufrufe der View-Methoden der einzelnen Komponenten.

14.2 Übersicht der C++ Klassen

% Bild von der Ableitungshierarchie

14.2.1 Typ-Verwaltung

AIR_Type

AIR_Type stellt alle Informationen über einen Seiten- bzw. Komponententyp bereit. Daneben macht es den Konstruktor zugänglich, der eine Instanz dieses Typs aus einem Eingabestrom liest.

`object/AIRType.h`

AIR_TypeStore

Typobjekte werden in PDL eingelesen und können über den Namen werden augelesen werden.

`object/AIRTypeStore.h`

AIR_TypeBase

Diese Klasse existiert in nur einer Instanz im AIR System. Es werden hier Funktionen des TypeStores bereitgestellt, die die Objekte über die Datenbank-Schnittstelle (class DBObject) verwalten.

`dbase/AIRType.h`

AIR_TypeCoder

Ein Objekt vom Typ AIR_Type wird in lesbarer PDL ausgegeben.

`object/AIRTypeCoder.h`

14.2.2 Instanz-Verwaltung

AIR_Instance

AIR_Instance stellt alle Informationen über ein Instanzobjekt bereit. Attribute und Komponenten werden über verändert und gelesen. Der Typname der Instanz ist im Attribut **typename** erreichbar.

`object/AIRInstance.h`

AIR_InstanceStore

AIR_InstanceStore ermöglicht das Einlesen von PIL-Objekten in das AIR-System. Über den Objektnamen liefert der InstanceStore ein AIR_Instance-Objekt zurück.

`object/AIRInstanceStore.h`

AIR_InstBase

Diese Klasse existiert in nur einer Instanz im AIR System. Es werden hier Funktionen des InstanceStores bereitgestellt, die die Objekte über die Datenbank-Schnittstelle (class DBObject) verwalten.

`dbase/InstBase.h`

AIR_InstParser

`object/AIRInstParser.h`

AIR_InstCoder

Ein Objekt vom Typ AIR_Instance wird in lesbarer PIL ausgegeben.

`object/AIRInstCoder.h`

AIR_TextInst

Der Basistyp **text** wird in dieser Klasse definiert. Als Klasse zur Darstellung ist **AIR_TextView** definiert.

`object/AIRInstance.h`

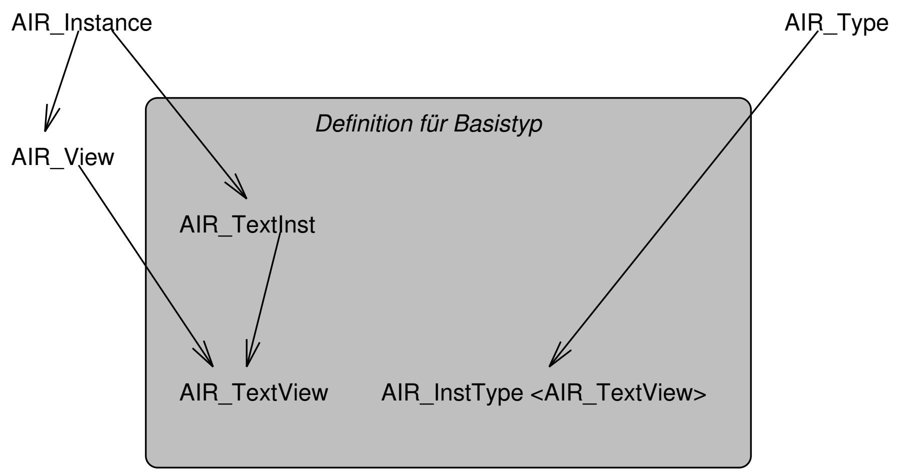
14.3 Erweiterung um neue Basistypen

Das Typ-System ermöglicht eine einfache Erweiterung um neue Basistypen. Dazu wird zunächst eine von AIR_Instance oder einem anderen Basistyp abgeleitete Klasse als Ursprung benutzt, die als Basis für Objekte des neuen Typs gelten soll. Zusätzlich ist eine Klasse zur Darstellung notwendig. Diese wird abgeleitet von der neuen Objektklasse und einer, der Ursprungsklasse, entsprechenden AIR_View Klasse (s. Abb.).

Im Typ-Speicher wird der neue Basistyp über ein Template

```
addType(„neuer_Typname‟, new AIR_InstType<neuer_ViewTyp>)
```

eingeführt.



Definition von Basistypen

Kapitel 15

AIR-Show - Eine Anwendungsbeispiel für AIR

15.1 Installation

Zum Betrieb von AIR-Show wird ein http-Server verwendet. Hierzu kann jeder Server, der CGI-Scripts unterstützt, benutzt werden. „apache" der für viele Systeme frei erhältlich ist, bietet sich z.B. an.

Die wichtigsten Komponenten von AIR-Show sind folgende ausführbare Files:

AIRimport

AIRimport.sh

AIRgen

AIRgen.sh

air.sh

Diese Files müssen in das Verzeichnis „cgi-bin" des http-Servers kopiert werden. Es wird vorausgesetzt, das die Verzeichnisse „htdocs" und „icons" existieren. Bei Bedarf kann dies in *air.sh* geändert werden.

Zusätzlich werden die Files **`logo.gif`** und **`logo_klein.gif`** von AIR-Show in dem Verzeichnis **`icons`** erwartet.

Die Seitentypen und Objekte stehen in den Verzeichnissen **`${AIR_DB_HOME}/objects`** und **`${AIR_DB_HOME}/types`**. Die Variable AIR_DB_HOME muß in **`AIRgen.sh`**, **`AIRimport.sh`** und **`air.sh`** angepaßt werden.

Vor der ersten Anwendung muß einmal **`air.sh`** im Verzeichnis cgi-bin aufgerufen werden. Danach erst ist die Startseite index.html erreichbar.

15.2 Beschreibung der Komponenten

AIRImport.sh und AIRgen.sh werden als CGI-Scripts vom http-Server aufgerufen. Diese starten dann **`AIRimport`** bzw. **`AIRgen`** und **`air.sh`**. AIRimport erzeugt ein Formular zum Einlesen eines Seitenobjekts und schreibt die Daten in PIL-Form zurück. AIRgen generiert die Ansicht eines Seitenobjekts. air.sh schreibt das Inhaltsverzeichnis in das File **`index.html`**, das den Einstiegspunkt für AIR-Show darstellt.

Beide Programme erwarten als Parameter den View-Namen und den Objekt-Namen in der CGI-Parameter-Form. Ein Aufruf wäre z.B.:

```
AIRimport /html/gui
```

Damit wird das Seitenobjekt „gui“ über den View-Namen „html“ generiert.

Die Seitenobjekte werden als Dateien in dem Verzeichnis **`${AIR_DB_HOME}/objects`** gespeichert. Der Filename ist der Objektname mit der Endung .pil. Die Seitentypen stehen in **`${AIR_DB_HOME}/types`** mit Typnamen und Endung .pdl.

Kapitel 16

Erstellung eines Beispielsystems

16.1 Motivation

Um die Einsatzfähigkeit des AIRsystems zu testen, haben wir beschlossen, teilweise das kommentierte Vorlesungsverzeichnis der Informatik in unsere definierten Sprachen umzusetzen.

In dem momentanen Zustand des Systems lassen sich noch keine Verweise automatisch über die **PIL** generieren, dieses erfolgt für die Titelseite über ein CGI-Script. Diese Titelseite nennt sich **`index.html`**, sie enthält zentriert (über mehrere Zeilen) den offiziellen Titel des Schriftstücks einschließlich der Angabe des entsprechenden Semesters. Darunter stehen dann als Verweise die einzelnen Lehrveranstaltungen.

Im ersten Teil dieses Kapitels wird aus der Sicht des *Konstrukteurs* geschildert, wie die passenden Seitentypen entworfen und erste Seitenobjekte angelegt werden. Ferner wird gezeigt, wie Seitentypen an neue Anforderungen angepaßt werden können.

In dem zweiten Teil dieses Kapitels wird aus der Sicht des *Benutzers* erklärt, wie diese Seiten über HTML-Formulare erzeugt und weiter verarbeitet werden können.

16.2 Das Beispielsystem aus Sicht des Konstrukteurs

Als erstes muß der *Konstrukteur* von seinem Kunden ermitteln, welche Bestandteile oder Inhalte die zu erzeugenden Seiten haben sollen, und wie diese abgebildet werden sollen. Der Kunde, wenn er nicht gleichzeitig auch der *Konstrukteur* ist, kommt mit den Sprachen (**PDL** und **PVL**) des Systems nicht direkt in Berührung. Sobald der *Konstrukteur* diese Anforderungen ermittelt hat, kann er mit dem Entwurf der Seitentypen beginnen.

Es werden folgende Anforderungen an das System festgelegt:

- Das kommentierte Vorlesungsverzeichnis der Informatik enthält verschiedene Seitenstrukturen. So soll eine Überblickseite vorhanden sein, auf der man zu den einzelnen Lehrveranstaltungen verzweigen kann. Ferner sollen Seiten zu den einzelnen Lehrveranstaltungen existieren.
- Die Lehrveranstaltungen sollen in einer einheitlichen Form dargestellt werden. Mögliche inhaltliche Bestandteile sind:

  - Titel der Lehrveranstaltung
  - Umfang der Veranstaltungen
  - Ort der Veranstaltungen
  - Typ der Veranstaltung (Vorlesung, Übung, Seminar, etc.)
  - Beginn der Lehrveranstaltung
  - Veranstalter oder Übungsleiter
  - Inhalt der Lehrveranstaltung
  - Hörer der Lehrveranstaltung
  - Voraussetzungen zum Besuch
  - Hinweise zum Besuch der Lehrveranstaltung
  - Literaturempfehlungen
  - Anforderungen an die Studierenden
  - Mögliche Folgeveranstaltungen
  - Prüfungsmöglichkeiten
- Es müssen nicht alle Bestandteile auf jeder Seite auftauchen. Die Reihenfolge der abgebildeten Komponenten ist aber identisch.
- Jeder Bestandteil hat ein vom Konstrukteur vorgegebenes Aussehen (nach Mitsprache des Kunden).

16.2.1 Anmerkungen zu den Grammatiken

Die genaue Beschreibung und Syntax der Sprachen steht im Endbericht der Projektgruppe im Kapitel der Sprachen.

Hier nur eine kurze Zusammenfassung:

PDL. Attribute werden mit **`ATTRIBUT`** und Komponenten mit **`COMPONENT`** definiert. Komponenten enthalten die View-Bestandteile ihrer Unterkomponenten. Nach dem Doppelpunkt wird der Typ des Bestandteils (ein Grundtyp wie **`TEXT`** oder ein vererbender Typ) angegeben.

PVL. Mit **`putcomp`** und **`putattr`** werden die Inhalte der Komponenten und Attribute in HTML ausgegeben. Mit **`putstr`** wird ein normaler String in HTML geschrieben. Dabei ist ein String eine zusammenhängende Zeichenfolge oder bei enthaltenen Leerzeichen eine durch doppelte Anführungzeichen markierte Zeichenfolge. Um doppelte Anführungzeichen schreiben zu können muß diesen ein Backslash vorangestellt werden. Durch Angabe **`putstr -n`** wird ein String in eine neue Zeile der HTML-Datei geschrieben. Steht innerhalb eines Views der Text **`view`** wird der View des entsprechenden Labels eingefügt. An den Stellen, wo ein **`inner`** (eventuell mit Label) angegeben ist, kann der Seitentyp in abgeleiteten Typen erweitert werden.

PIL. Die Namen der Komponenten und Attribute werden in spitzen Klammern angegeben. Die eingetragenen Inhalte erhält der Konstrukteur vom Kunden. Absätze in Textbereichen werden durch eine Leerzeile gekennzeichnet. Die geerbten Attribute und Komponenten erscheinen ebenso wie die des eigentlichen Types.

16.2.2 Entwurf der PDL-Typen

Als erstes benötigt man einen grundlegenden Seitentypen. Diesen findet man in allen möglichen Systemen. Dieser Typ heißt *SEITE*. Aus diesem Typ werden alle weiteren Seitentypen abgeleitet.

Für dieses System braucht man folgende weiteren Typen (dabei werden Typen in normaler Schrift nicht genauer vorgestellt, da für Seitentypen die Verweise enthalten, im System noch keine Behandlung vorhanden ist):

- **TITELSEITE**
- **GLIEDERUNGSSEITE**
- **VERANSTALTUNG**
- **VORLESUNG**
- **UEBUNG**
- **PRAKTIKUM**
- **SEMINAR**
- **LEHRVERANSTALTUNG**

Zur Vereinfachung des Systems gibt es hier nur wenige verschiedene Seitentypen. Sie werden alle Veranstaltungen über den Seitentyp **LEHRVERANSTALTUNG** definiert. Die veranstaltungsartspezifischen Typen liefern nur Zusatzinformationen zum Typ **VERANSTALTUNG**, und werden von diesem abgeleitet.

Diese Vereinfachung erschwert allerdings die spätere leichte Erweiterbarkeit.

Das Aussehen von Grundtypen wie **TEXT** wird nicht näher beschrieben.

Der Seitentyp SEITE

```
TYPE SEITE {
  ATTRIBUTE name
  COMPONENT titel : TEXT
  VIEW html_head TCL{
    putstr -n <html>
    putstr "<!-- HTML generiert von AIR email:“
    putstr “pg269@udo.informatik.uni-dortmund.de -->"
    putstr -n <head>
    putstr -n <title>
    putcomp titel
    putstr -n </title>
    putstr -n </head>
    putstr -n "<body bgcolor=\"#d0d0d0\" text=\"#000000\""
    putstr "link=\"#0000ee\" vlink=\"#551a8b\"“
    putstr “alink=\"#ff0000\">"
    putstr "<a href=\"/index.html\">"
    putstr -n "<IMG SRC=\"/icons/logo_klein.gif\"></a><p>"
  }
  VIEW html_tail TCL{
    putstr "</body>"
    putstr "</html>"
  }
```

```
    VIEW html TCL{
      view "html_head"
      inner kopfzeile
      inner inhalt
      inner fusszeile
      putstr "<HR>"
      putstr "<FORM method=get"
      putstr "action=\"/cgi-bin/AIRgen/html_form/\""
      putstr -n "[attribute name]\">"
      putstr "<input type=\"submit\" value=\"&Auml;ndern\">"
      putstr "</FORM>"
      view "html_tail"
    }
  }
```

Obwohl dieser Grundtyp kompliziert erscheint, ist er einfach zu verstehen. Bestimmte Teile (z.B. die sechstletzte bis vorletzte Zeile in **`VIEW html TCL`** dienen zur Weiterverarbeitung der Inhalte durch HTML-Formulare. In **`html_head`** wird zusätzlich zu den Grundlagen (Bestandteile und Erweiterbarkeitsstellen), ein Kommentar zu AIR eingefügt, eine Hintergrundfarbe für alle abgeleiteten Seiten sowie die Farben für Links (sowohl noch nicht besuchter wie besuchter) definiert, ein fester Verweis zur Indexseite eingebaut.

Der Grundtyp **SEITE** besteht aus dem Attribut **name** und der Komponente **`titel`**, die vom Typ **TEXT** ist. Über das eindeutige Attribut **name** kann man später das Objekt ansprechen. Durch die Komponente **titel** wird ein Titel definiert, der als Titel der HTML-Seite verwendet wird.

Dieser Seitentyp kann an drei Stellen erweitert werden. An der Stelle **inner kopfzeile** wird die Seite um eine Kopfzeile ergänzt. Wo **inner inhalt** steht, findet die hauptsächliche Ergänzung statt. Dann kann noch eine Fußzeile an der Stelle **inner fusszeile** eingefügt werden.

Der Typ VERANSTALTUNG

```
TYPE VERANSTALTUNG {
  ATTRIBUTE veranst_nr ?
  ATTRIBUTE anzahl_veranst ?
  COMPONENT veranst + {
    ATTRIBUTE wochentag
    ATTRIBUTE zeit
    ATTRIBUTE ort

    VIEW html TCL{
      putstr -n "<tr> <td>"
      putattr -n wochentag
      putstr -n <td>
      putattr -n zeit
      putstr -n <td>
      putattr -n ort
    }
  }

  VIEW html TCL{
    inner personen
    putstr -n "<p> <ul> <table> <tr> <td>"
    putattr -n veranst_nr
    putstr <td>
    putattr -n anzahl_veranst
    inner typ
    putstr </table>
    putstr -n "<table>"
    withall veranst { view html }
    putstr "</table> </ul> <p>"
  }
}
```

Mit dem Typen **`VERANSTALTUNG`** wird eine Veranstaltung definiert. Die Veranstaltung enthält die optionalen Attribute **veranst_nr** und **`anzahl_veranst`**, dieses wird durch das Fragezeichen hinter der Deklaration angegeben. Diese Attribute müssen in der PIL nicht ausgefüllt werden. Dagegen ist die Komponente **`veranst`** mindestens einmal erforderlich, dieses ist durch das Pluszeichen angegeben. Diese enthält die Unterkomponenten **`wochentag`**, **`zeit`** und **`ort`**.

Die Komponente **`veranst`** wird als Tabelle in HTML dargestellt. Bei der Darstellung des gesamten Typs wird eine Erweiterungsstelle **`inner personen`** eingefügt, dann als Tabelle die optionalen Attribute ausgegeben, ebenfalls mit einer Erweiterung (**`inner typ`**). Dann wird eine unsortierte Liste mit den Elementen der Komponente **`veranst`** hinzugefügt.

Der Typ **`VERANSTALTUNG`** wird in reiner Form in keinem PIL-Objekt auftauchen, sondern immer als einer der nachfolgenden erweiterten Typen.

Der Typ VORLESUNG

```
TYPE VORLESUNG: VERANSTALTUNG {
  COMPONENT dozent + : NAME
  VIEW html EXTEND typ TCL{
    putstr -n V
  }
  VIEW html EXTEND personen TCL{
    putstr -n "Dozent:"
    withall dozent {
      view html putstr " "
    }
  }
}
```

Verwendet Hilfstyp NAME:

```
TYPE NAME: text {
  ATTRIBUTE lines = "1"
}
```

Dieser Typ ist eine Erweiterung bzw. Ableitung des Typs **`VERANSTALTUNG`**, er erbt alle Bestandteile dieses Typen, ohne sie in seiner Beschreibung detailliert zu erwähnen. Er ergänzt den View (**`inner personen`**), um die Angabe der Dozenten, die in je eine Zeile geschrieben werden, und fügt ein **`V`** nach der Anzahl der Veranstaltungen (**`inner typ`**) ein.

Der Typ UEBUNG

```
TYPE UEBUNG: VERANSTALTUNG{
  COMPONENT tutor + : NAME
  VIEW html EXTEND typ TCL{
    putstr "UE"
  }
  VIEW html EXTEND personen TCL{
    putstr "&Uuml;bungsleiter: "
    withall tutor {
      view html putstr " "
    }
  }
}
```

`UEBUNG` ist analog **`VORLESUNG`** aufgebaut. Nur wird anstelle von Dozenten Übungsleiter geschrieben, und als Typ **`UE`** angegeben.

Der Typ PRAKTIKUM

```
TYPE PRAKTIKUM: VERANSTALTUNG {
  COMPONENT leiter + : NAME
  VIEW html EXTEND typ TCL{
    putstr "P"
  }
  VIEW html EXTEND personen TCL{
    putstr -n "Leiter: "
    withall leiter {
      view html putstr " "
    }
  }
}
```

Praktikum ist analog zu **`VORLESUNG`** aufgebaut.

Der Typ SEMINAR

```
TYPE SEMINAR: VERANSTALTUNG {
  COMPONENT leiter + : NAME
  VIEW html EXTEND typ TCL{
    putstr "S"
  }
  VIEW html EXTEND personen TCL{
    putstr -n "Leiter:"
    withall leiter {
      view html putstr " "
    }
  }
}
```

Seminar ist analog zu **`VORLESUNG`** aufgebaut.

Der Seitentyp LEHRVERANSTALTUNG

```
TYPE LEHRVERANSTALTUNG : SEITE {
  ATTRIBUTE einstufung ?
  ATTRIBUTE beginn ?
  COMPONENT inhalt: TEXT
  COMPONENT hoerer ? : TEXT
  COMPONENT voraussetzungen ? : TEXT
  COMPONENT hinweise ? : TEXT
  COMPONENT literatur {
    COMPONENT text ? : TEXT
    COMPONENT eintrag * : TEXT

    VIEW html TCL{
      putcomp text
      putstr <UL>
      withall eintrag {
        putstr -n "<li>"
        view html
      }
      putstr </UL>
    }
  }
  COMPONENT anmeldung ? : TEXT
  COMPONENT anforderungen ? : TEXT
  COMPONENT folgeveranst ? : TEXT
  COMPONENT pruefungen ? : TEXT
  COMPONENT vorlesung ? : VORLESUNG
  COMPONENT uebung ? : UEBUNG
  COMPONENT praktikum ? : PRAKTIKUM
  COMPONENT seminar ? : SEMINAR

  VIEW html EXTEND kopfzeile TCL{
    if [exists -a einstufung] {
      putattr einstufung
    }
    putstr "<HR>"
  }

  VIEW html EXTEND inhalt TCL{
    putstr -n "<H1>"
    putcomp titel
    putstr "</H1>"
    withall vorlesung { view html }
    withall uebung { view html }
    withall seminar { view html }
    withall praktikum { view html }

    if [exists -a beginn] {
      putstr -n "Beginn: "
      putattr beginn
    }
    putstr "<H2>Inhalt:</H2>"
    putcomp inhalt

    withall hoerer {
      putstr "<H2>H&ouml;rer:</H2>"
      view html
    }
```

```
        withall voraussetzungen {
          putstr "<H2>Voraussetzungen:</H2>"
          view html
        }

        withall Hinweise {
          putstr "<H2>Hinweise:</H2>"
          view html
        }

        putstr "<H2>Literatur:</H2>"
        putcomp literatur
        if [exists anmeldung] {
          putstr "<H2>Anmeldung:</H2>"
          putcomp anmeldung
        }

        if [exists anforderungen] {
          putstr "<H2>Anforderungen:</H2>"
          putcomp anforderungen
        }

        if [exists folgeveranst] {
          putstr "<H2>Weiterf&uuml;hrende Veranstaltungen:</H2>"
          putcomp folgeveranst
        }

        if [exists pruefungen] {
          putstr "<H2>Pruefungen:</H2>"
          putcomp Pruefungen
        }
      }
    }
```

Dieser Seitentyp bildet alle Lehrveranstaltungen ab. Er ist abgeleitet vom Grundseitentyp **SEITE**, wobei er zwei Erweiterungsstellen ergänzt. Er übernimmt das Attribut **name** und die Komponente **titel** (dieses geschieht über den Ableitungsmechanismus, die Namen erscheinen im Typen nicht).

LEHRVERANSTALTUNG enthält zwei optionale Attribute **einstufung** und **beginn**. Das erste Attribut gibt die Einstufung im Studium (Grundvorlesung, Praktikum oder Programmierkurs, Seminar oder Hauptstudium) an. Das zweite benennt das Datum des Lehrveranstaltungsbeginns.

Ferner gibt es dreizehn Komponenten, von denen alle außer zwei optional sind. Diese zwei Komponenten **inhalt** und **literatur** beschreiben den Inhalt der Lehrveranstaltung und die Literaturempfehlungen. Vier Komponenten sind abgeleitet von den abgeleiteten Veranstaltungstypen **VORLESUNG**, **UEBUNG**, **PRAKTIKUM** und **SEMINAR**, sie heißen wie der Typ in kleiner Schrift. Die sonstigen optionalen Komponenten sind **hoerer**, **voraussetzungen**, **hinweise**, **anmeldung**, **anforderungen**, **folgeveranstaltungen** und **pruefungen**}, ihr Inhalt ergibt sich aus der Namensgebung.

Außer den abgeleiteten Komponenten ist nur die Komponente **literatur** komplex. Die übrigen enthalten nur Text. Die Komponente **literatur** umfaßt zwei Unterkomponenten **text** und **eintrag**, wobei die erste optional ist, wogegen die zweite null bis unendlich oft auftauchen darf. Bei dem View wird zuerst der Inhalt der Unterkomponente **text** ausgegeben und anschließend falls vorhanden, eine Liste mit einzelnen Literatureinträgen.

Der View der Grundseite **SEITE** wird an der Stelle **inner kopfzeile** ergänzt. Wenn das Attribut **einstufung** ausgefüllt ist, wird in die HTML-Datei nach der **BODY**-Deklaration der Inhalt der Einstufung und eine horizontale Linie eingefügt.

Außerdem wird an der Stelle **inner inhalt** erweitert, dieses geschieht in der folgenden Reihenfolge der Komponenten und Attribute: **titel** als große Überschrift, dann die abgeleiteten Veranstaltungstypen in der Reihenfolge **vorlesung**, **uebung**, **seminar** und **praktikum**, danach der **beginn**, der **inhalt**, die **hoerer**, die **voraussetzungen**, die **hinweise**, die **literatur**, die **anforderungen**, die **folgeveranstaltungen** und die **pruefungen**.

16.2.3 Erzeugung von Seitenobjekten

In diesem Abschnitt wird anhand von ein paar **LEHRVERANSTALTUNG**-Seitenobjekten die Verwendung der **PIL** vorgestellt. Dabei stehen die Bestandteile der Grammatik in Schreibmaschinenschrift und die Inhalte in normaler Schrift.

Zum leichteren Verständnis ist in den folgenden Beispielen die Reihenfolge der Attribute und Komponenten identisch mit der des Seitentypen **LEHRVERANSTALTUNG**. Dieses Vorgehen erleichtert dem **Benutzer** das vollständige Ausfüllen der leeren PIL- Objekte. Grundsätzlich dürfen die Komponenten und Attribute in beliebiger Reihenfolge auftreten, bei langen Seitenobjekten kann dann schnell der Überblick verloren gehen.

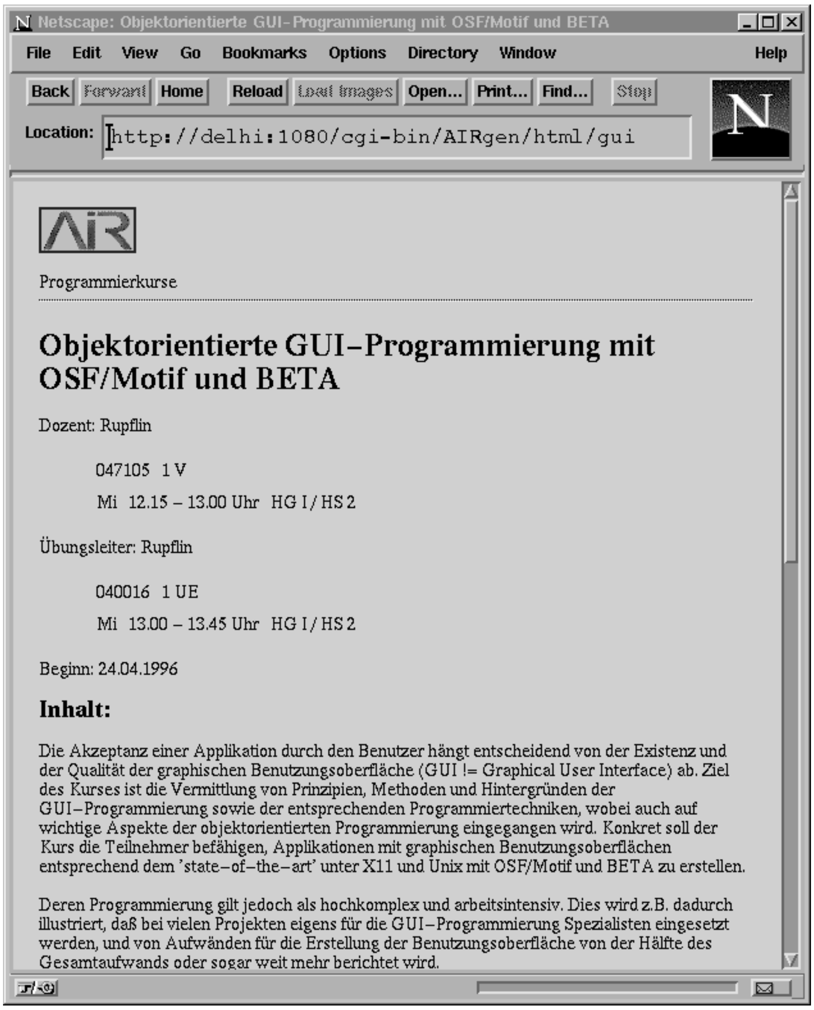


Abbildung einer von AIR generierten Beispiel-Seite

Das Seitenobjekt RS

Die Vorlesung Rechnerstrukturen wird in der **`PIL`** beschrieben. Dabei werden nicht alle optionalen Attribute oder Komponenten verwendet, so fehlt z.B. das Attribut **`beginn`** und die Komponente **`pruefungen`**. Bei der Komponente **`uebung`** tritt die Unterkomponente **`tutor`** gleich fünfmal auf. Gleich in der ersten Zeile sieht man das geerbte Attribut **`name`** und ein paar Zeilen später die Komponente **`titel`**. In der Komponente **`vorlesung`** sind außer der Komponente **`dozent`** des Typs **`VORLESUNG`** auch die Bestandteile des Typs **`VERANSTALTUNG`** enthalten. Analog gilt dieses für die Komponente **`uebung`**.

```
<LEHRVERANSTALTUNG
    name="RS"
    einstufung="Vorlesungen im Grundstudium">
  <titel>Rechnerstrukturen</titel>
```

```
<inhalt>
  Während die Vorlesung Programmierung die Nutzung eines
  Programmiersystems in den Vordergrund stellte, kommt die
  Vorlesung Rechnerstrukturen ‚von unten‘, indem sie in die
  wesentlichen Grundlagen zur Realisierung von Rechnersystemen
  einführen soll. Den Kern dieser Vorlesung bilden die
  technische Darstellung von Information im Rechner,
  Hardwarebausteine in heute gängiger Verwendung, sowie die
  Architektur des Von-Neumann-Rechners mit ihrer
  Maschinensprache- (bzw. Assembler-) Programmierung. Die
  restlichen Kapitel der Vorlesung haben eher
  Überblickscharakter. Zwei umreissen, wie die Hardware durch
  Systemsoftware ergänzt wird (Betriebssysteme,
  Programmiersprachen und ihre Übersetzer). Zwei weitere
  Kapitel bieten unter den Stichwörtern ‚Parallelität‘ und
  ‚Verteilung‘ einen Ausblick auf bereits gängige Varianten der
  Von-Neumann Architektur. Der Stoff dieser Überblickskapitel
  wird dann im Hauptstudium durch jeweils eigene Vorlesungen
  vertieft werden.
</inhalt>

<hoerer>Kerninformatiker vor dem Vordiplom
</hoerer>
<voraussetzungen>
  Vorlesung ‚Programmierung‘
</voraussetzungen>
<hinweise>
  Es finden Übungen zur Vorlesung statt. Die Teilnahme an den
  Übungen wird sehr empfohlen.
</hinweise>
<literatur>
  <text>
    Die Vorlesung orientiert sich an dem gleichnamigen
    Skriptum (Autoren: K. Echtle, H. Krumm, W. Banzhaf),
    das zu Beginn der Vorlesungszeit in der
    Skriptenverkaufsstelle verfügbar sein sollte.
  </text>
</literatur>

<vorlesung
    veranst_nr="+040001"
    anzahl_veranst="4">
  <veranst
      wochentag="Di"
      zeit="12.15 - 13.45 Uhr"
      ort="HG III / HS 1">
  </veranst>
  <veranst
      wochentag="Do"
      zeit="14.15 - 16.00 Uhr"
      ort="HG III / HS 1" >
  </veranst>
  <dozent>Banzhaf</dozent>
</vorlesung>
```

```
    <uebung
        veranst_nr="040002"
        anzahl_veranst="2">
      <veranst
          wochentag="n.V."
          zeit=""
          ort="">
      </veranst>
      <tutor>Beckmann</tutor>
      <tutor>Eiss</tutor>
      <tutor>Knaup</tutor>
      <tutor>Lind</tutor>
      <tutor>Kursawe</tutor>
    </uebung>
  </LEHRVERANSTALTUNG>
```

Das Seitenobjekt EfP

Die Vorlesung Einführung in das funktionale Programmieren wird als weiteres **LEHRVERANSTALTUNG**-Objekt erzeugt. Diesmal ist das optionale Attribut **beginn** vorhanden, die optionalen Komponenten sind wie üblich nicht alle vorhanden. Optionale Komponenten oder Attribute, die nicht verwendet werden, läßt man einfach weg. Die Komponente **literatur** ist in diesem Objekt in vollem Umfang enthalten, sowohl die Unterkomponente **text** als auch mehrere vom Typ **eintrag** erscheinen.

```
<LEHRVERANSTALTUNG
    name="EfP"
    einstufung="Programmierkurse"
    beginn="15. April 1996">
  <titel>Einführung in das funktionale Programmieren</titel>
  <inhalt>
    Im 1. Semester wurde ins Programmieren im allgemeinen und ins
    objektorientierte Programmieren im besonderen eingeführt. In
    Programmierung II wird der funktionale Programmierstil
    vorgestellt anhand der Programmiersprache ML. Darüberhinaus
    werden funktionale mit objektorientierten sowie
    logikorientierten Sprachkonzepten und Programmiermethoden
    verglichen.
  </inhalt>
  <hoerer>
    Student(inn)en der Kern- und Angewandten Informatik ab 2.
    Semester
  </hoerer>
  <literatur>
    <text>
      Grundlage der Vorlesung ist ein Skript, das zu Beginn der
      Lehrveranstaltung erworben werden sollte. Als ergänzende
      Literatur wird mindestens eines der folgenden Bücher
      empfohlen
    </text>

    <eintrag>
      L. C. Paulson, ML for the Working Programmer, Cambridge
      University Press 1991 (Exemplare in der Lehrbuchsammlung)
    </eintrag>
```

```
    <eintrag>
      C. Reade, Elements of Functional Programming,
      Addison-Wesley 1989 (Exemplare in der Lehrbuchsammlung)
    </eintrag>
    <eintrag>
      R. Bird, Ph. Wadler, Introduction to Functional
      Programming, Prentice-Hall 1988 (deutsch: Einführung in
      die funktionale Programmierung, Hanser 1992)
    </eintrag>
    <eintrag>
      R. Bosworth, A Practical Course in Functional Programming
      using Standard ML, McGraw-Hill 1995
    </eintrag>
    <eintrag>
      J. D. Ullman, Elements of ML Programming, Prentice-Hall
      1994
    </eintrag>
    <eintrag>
      C. Myers, C. Clack, E. Poon, Programming with Standard ML,
      Prentice-Hall 1993
    </eintrag>
    <eintrag>
      R. Stansifer, ML Primer, Prentice-Hall 1992
    </eintrag>
    <eintrag>
      A. Wikström, Functional Programming using Standard ML,
      Prentice-Hall 1987
    </eintrag>
    <eintrag>
      R. Sethi, Programming Languages: Concepts and Constructs,
      Addison-Wesley 1989 (Gegenüberstellung verschiedener
      Sprachstile; Exemplare in der Lehrbuchsammlung)
    </eintrag>
  </literatur>

  <vorlesung
      veranst_nr="040015"
      anzahl_veranst="2">
    <veranst
        wochentag="Mo"
        zeit="12.15 - 14.00 Uhr"
        ort="EF 50 / HS 1">
    </veranst>
    <dozent>Padawitz</dozent>
  </vorlesung>

  <uebung
      veranst_nr="040016"
      anzahl_veranst="1">
    <veranst
        wochentag="n.V."
        zeit=""
        ort="">
    </veranst>
    <tutor>Hallmann</tutor>
    <tutor>Huwig</tutor>
  </uebung>
</LEHRVERANSTALTUNG>
```

Das Seitenobjekt PKProlog

Dieses PIL-Objekt beschreibt den Programmierkurs Prolog. Es enthält außer den zwingend vorgeschriebenen Bestandteilen nur noch Informationen über die Vorlesung und Übung. Durch die Leerzeilen im Fließtext wird ein Absatz angegeben.

```
<LEHRVERANSTALTUNG
    name="PKProlog"
    einstufung="Programmierkurse">
  <titel>Programmierkurs Prolog</titel>
  <inhalt>
    Dieser Programmierkurs soll Prinzipien einer
    nichtprozeduralen Programmiersprache anhand der logischen
    Programmierung vermitteln. Dabei soll die Möglichkeit im
    Vordergrund stehen, PROLOG durch umfangreiche praktische
    Übungen als effizientes Werkzeug kennenzulernen.
    Daneben werden Voraussetzugen sowohl für die Stammvorlesung
    Künstliche Intelligenz als auch für Seminare und
    Projektgruppen in diesem Bereich erworben. Schließlich wird
    ein kurzer Einblick in den Bereich Constraint Logic
    Programming gegeben.
    Der Kurs befaßt sich im ersten Teil mit den Grundbegriffen
    der logischen Programmierung, wie Hornklausellogik,
    Resolution und Unifikation und liefert damit Logikkenntnisse
    und Grundlagen für einen Vergleich mit anderen
    Programmiersprachen.
    Danach wird das Ausführungsmodell von PROLOG mit
    Backtracking, Rekursion, expliziter Ablauf-, sowie Ein- und
    Ausgabekontrolle und Metaprädikate besprochen. Zudem wird
    die Programmiertechnik anhand von Übungen erläutert.
    Praktische Programmierübungen am Rechner, vor allem aus dem
    Bereich Künstliche Intelligenz, bieten die Möglichkeit, zum
    einen den Umgang mit der Programmiersprache PROLOG gut
    einzuüben und zum anderen, deskriptive Konzepte
    kennenzulernen, wie sie bei modernen Programmiersprachen
    verwendet werden.
  </inhalt>

  <literatur>
    <eintrag>
      Bratko, I. 1990. Prolog Programming for Artificial
      Intelligence. 2nd Edition. Addison Wesley.
    </eintrag>
    <eintrag>
      Sterling L. & Shapiro, E. 1986. The Art of Prolog. Advanced
      Programming Techniques. MIT Press.
    </eintrag>
  </literatur>
```

```
    <vorlesung
         veranst_nr="047097"
         anzahl_veranst="3">
      <veranst
           wochentag="zweiwöchiger Kompaktkurs vom 18.3. - 29.3."
           zeit="9.00 - 12.00 Uhr"
           ort="HG I / HS 2">
      </veranst>
      <dozent>Markhof</dozent>
      <dozent>Rieger</dozent>
    </vorlesung>

    <uebung
         veranst_nr="047098"
         anzahl_veranst="4">
      <veranst
           wochentag="zweiwöchiger Kompaktkurs vom 18.3. - 29.3."
           zeit="13.00 - 17.00 Uhr"
           ort="HG I / HS 2">
      </veranst>
      <tutor>Markhof</tutor>
      <tutor>Rieger</tutor>
    </uebung>
</LEHRVERANSTALTUNG>
```

Vorgabe für Benutzer - Ein neues leeres Seitenobjekt

Nach drei konkreten Seitenobjekten stellen wir jetzt ein unausgefülltes Seitenobjekt vor. Hinweise zum Ausfüllen stehen rechts davon in normaler Schrift. Als Anzahl (Anz.) wird die Auftrittshäufigkeit der Komponenten oder Attribute angegeben (1, 0-1, 0-N, 1-N), dabei steht N für eine beliebig große Zahl. Wenn Auftrittshäufigkeit 0 gewählt wird, dann wird das Attribut oder die Komponente weggelassen, bei einer höheren Auftrittshäufigkeit als 1 wird der Bestandteil vervielfältigt.

PIL-Grammatik	Anz.	Hinweise
<LEHRVERANSTALTUNG name=" "	1	Eindeutiger Name, geerbtes Attribut von **SEITE**
einstufung=" "	0-1	Wenn bekannt, z.B. Praktikum
begin=" " >	0-1	Wenn bekannt, Tag der ersten Veranstaltung
<titel> </titel>	1	nötig, Titel der Veranstaltung; geerbte Komponente
<inhalt> </inhalt>	1	Inhalt der Veranstaltung
<hoerer> </hoerer>	0-1	Wenn erwünscht, mögliche Interessenten
<voraussetzungen> </voraussetzungen>	0-1	Wenn nötig, Voraussetzungen an den Besuch
<hinweise> </hinweise>	0-1	Wenn erwünscht, Hinweise zur Veranstaltung

PIL-Grammatik	Anz.	Hinweise
<literatur>	1	nötig ⇒ mindestens eine Unterkomponente vorhanden
<text> </text>	0-1	wenn erwünscht, Literaturempfehlung als Text
<eintrag> </eintrag>	0-N	beliebige Anzahl an Literaturstellen
</literatur>		
<anmeldung> </anmeldung>	0-1	wenn nötig, Anmeldebedingungen
<anforderungen> </anforderungen>	0-1	wenn erwünscht, Anforderungen an die Studierenden
<folgeveranst> </folgeveranst>	0-1	wenn bekannt, anschließende Veranstaltungen
<pruefungen> </pruefungen>	0-1	wenn erwünscht, Informationen zu Prüfungen
<vorlesung	0-1	nötig, wenn Teil der Veranstaltung
veranst_nr=""	0-1	wenn bekannt, Veranstaltungsnummer eintragen
anzahl_veranst="" >	0-1	wenn bekannt, Anzahl angeben
<veranst	1-N	beliebige Anzahl an Veranstaltungen > 1
wochentag=" "	1	wenn bekannt, angeben, sonst N.V.
zeit=" "	1	wenn bekannt, sonst leer lassen
ort=" ">	1	wenn bekannt, sonst leer lassen
</veranst>		geerbt von **VERANSTALTUNG**
<dozent> </dozent>	1-N	nötig, wenigstens eine Person muß angegeben werden
</vorlesung>		geerbt von **VORLESUNG**
<uebung	0-1	nötig, wenn Teil der Veranstaltung
veranst_nr=" "	0-1	wenn bekannt, Veranstaltungsnummer eintragen
anzahl_veranst=" ">	1	wenn bekannt, angeben, sonst N.V.
<veranst	1-N	beliebige Anzahl an Veranstaltungen > 1
wochentag=" "	1	wenn bekannt, angeben, sonst N.V.
zeit=" "	1	wenn bekannt, sonst leer lassen
ort=" " >	1	wenn bekannt, sonst leer lassen
</veranst>		geerbt von **VERANSTALTUNG**

PIL-Grammatik	Anz.	Hinweise
<tutor> </tutor>	1-N	nötig, wenigstens eine Person muß angeben werden
</uebung>		geerbt von **UEBUNG**
<praktikum	0-1	nötig, wenn Teil der Veranstaltung
veranst_nr=" "	0-1	wenn bekannt, Veranstaltungsnummer eintragen
anzahl_veranst=" ">	1	wenn bekannt, angeben, sonst N.V.
<veranst	1-N	beliebige Anzahl an Veranstaltungen > 1
wochentag=" "	1	wenn bekannt, angeben, sonst N.V.
zeit=" "	1	wenn bekannt, sonst leer lassen
ort="">	1	wenn bekannt, sonst leer lassen
</veranst>		geerbt von **VERANSTALTUNG**
<leiter> </leiter>	1-N	nötig, wenigstens eine Person muß angeben werden
</praktikum>		geerbt von **PRAKTIKUM**
<seminar	0-1	nötig, wenn Teil der Veranstaltung
veranst_nr=" "	0-1	wenn bekannt, Veranstaltungsnummer eintragen
anzahl_veranst=" " >	1	wenn bekannt, angeben, sonst N.V.
<veranst	1-N	beliebige Anzahl an Veranstaltungen > 1
wochentag=" "	1	wenn bekannt, angeben, sonst N.V.
zeit=" "	1	wenn bekannt, sonst leer lassen
ort=" " >	1	wenn bekannt, sonst leer lassen
</veranst>		geerbt von **VERANSTALTUNG**
<leiter> </leiter>	1-N	nötig, wenigstens eine Person muß angeben werden
</seminar>		geerbt von **SEMINAR**
</LEHRVERANSTALTUNG>		

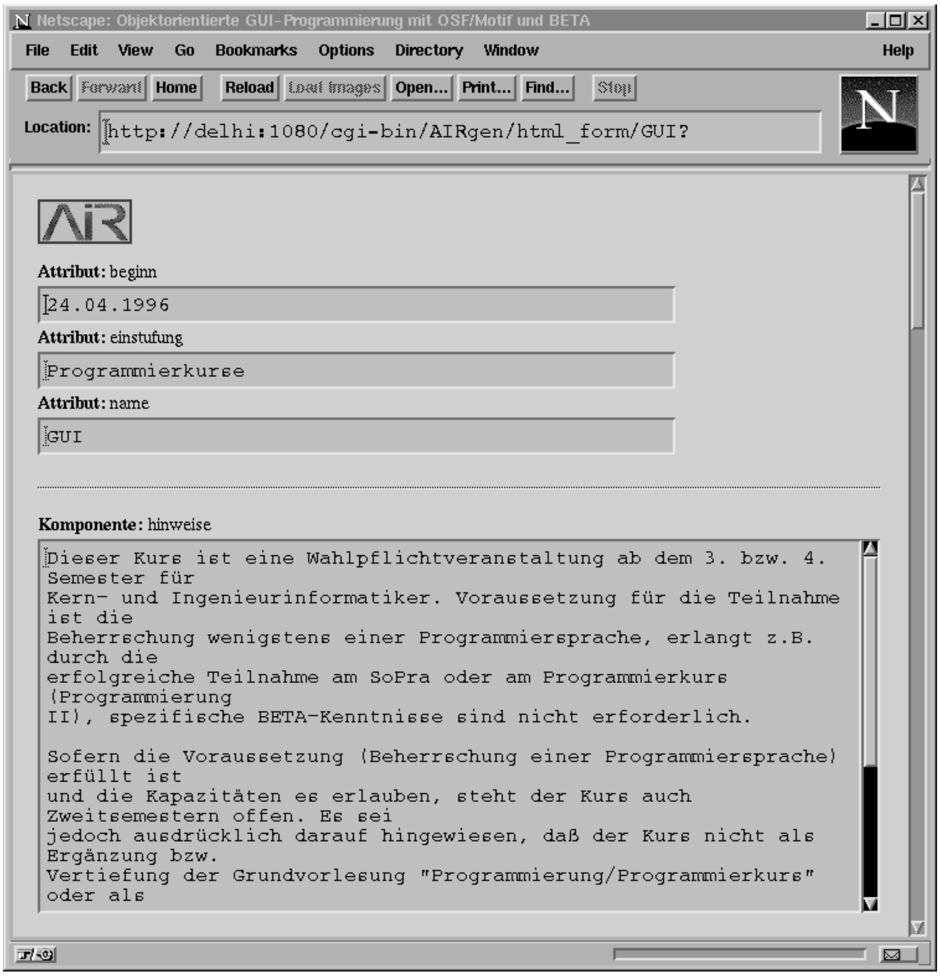


Abbildung eines gefüllten Beispiel-Formulares

Änderungen von Seitenobjekten

Nachdem der **`Benutzer`** mit Hilfe des **`Konstrukteurs`** seine Seitenobjekte erstellt hat, erstellt das AIRsystem daraus die HTML- Seiten, wobei das Layout durch die Views der Typen vorgegeben ist.

Diese Seiten kann der **`Benutzer`** zur Laufzeit durch zwei Methoden ändern. Die erste erfolgt durch direkte Manipulation der PIL-Objekte (nur in Zusammenarbeit mit dem **`Konstrukteur`**). Die zweite Möglichkeit ist die Änderung der Objekte anhand von HTML- Formularen. Dieser Eingriff in das System steht auch dem **`Benutzer`** uneingeschränkt offen, da nur die Inhalte der Seiten geändert werden können. Diese Möglichkeit wird im zweiten Abschnitt dieses Kapitels erläutert. Kommen wir zurück zur ersten Methode. Stellt der **`Benutzer`** Fehler in seinen Inhalten fest, oder hat der Inhalt sich überholt, kann er das entsprechende Seitenobjekt öffnen um den Inhalt korrigieren. Nach Abspeicherung erscheint die korrigierte Seite im AIRsystem. Analog kann man seine Seiten um noch nicht genutzte optionale oder wiederholte Bestandteile ergänzen, indem man diese Teile in das PIL-Objekt mit ihren Inhalten einträgt. Auch diese erscheinen nach dem Abspeichern zur Laufzeit auf den HTML-Seiten.

16.2.4 Anpassungen der Typen an neue Anforderungen

In diesem Abschnitt zeigen wir, wie Seitentypen oder andere Typen an geänderte Anforderungen angepaßt oder neue Wünsche in das bestehende System integriert werden können.

Auch PDL-Typen können zur Laufzeit geändert werden. Das Ändern von Typen kann notwendig werden, wenn das Layout sich als nicht geeignet herausstellt, oder der **Kunde** andere Vorstellungen hatte.

Eine einfache Änderungen in unserem Beispielsystem ist z.B. der Wunsch einer neuen Hintergrundfarbe. Nach der Durchführung der Änderung erscheinen alle Seiten mit der neuen Hintergrundfarbe, da die Modifikation im Grundseitentyp **SEITE** vorgenommen wird. In dem Typ wird die Zeile

```
putstr -n "<body bgcolor=\"#d0d0d0\" text=\"#000000\" "
```

durch

```
putstr -n "<body bgcolor=\"#d1d1d1\" text=\"#000000\""
```

ersetzt.

Alle Anpassungen, die keine Strukturveränderung bewirken (z.B. Kommentare, Farbeinstellungen, Schriftgrößen usw.) können genauso leicht durchgeführt werden. Man muß dabei beachten, daß sich die durchgeführten Anpassungen auf alle Objekte auswirken, die diesen Typ oder einen davon abgeleiteten verwenden.

Umfangreiche Änderungen erfordern einen erheblich höheren Aufwand. In feingranularen Typsystemen kann man leicht Anpassungsstellen erkennen. Dort besteht aber die Gefahr, durch viele Ableitungsebenen den Überblick über die beteiligten Typen, Attribute und Komponenten zu verlieren. Bei sehr umfangreichen Typen kann eine kleine Änderung einen fast vollständigen Neuentwurf (View) erfordern. Am Besten wählt man eine mittlere Typgröße. Man muß berücksichtigen, daß eine Änderung eines Types auf alle abgeleiteten Typen Auswirkungen hat.

Die Erweiterung um neue Typen wird durch eine hohe Anzahl an vorhandenen Erweiterbarkeitsstellen (**inner**) unterstützt, da leicht neue Attribute und Komponenten eingefügt werden können. Man leitet einen neuen Typ von einem alten ab, dann müssen nur die neuen Bestandteile definiert und der View zur passenden **inner**- Stelle des alten Typen erzeugt werden. Schon steht ein neuer Typ zu Verfügung.

Wenn man bestehende Seitentypen durch neue optionale Bestandteile ergänzt, hat es keine Auswirkungen auf bestehende Objekte, die diesen Typen direkt oder abgeleitet verwenden, außer es entstehen durch die neuen Bestandteile Namenskonflikte. Dabei wird vorausgesetzt, das der View höchstens um eine optionale Ausgabe der neuen Bestandteile ergänzt wird. Unproblematisch sind Viewanpassungen, wenn sie allein aus Einfügung von Erweiterungsstellen (inner) bestehen.

In unserem Beispielsystem könnte eine solche Anpassung so aussehen:

- Im Typen **LEHRVERANSTALTUNG** wird eine zusätzliche optionale Komponente **maxteilnehmer** eingeführt.
- Im View **view** ... extend ... inhalt wird nach Ausgabe der Komponente **hoerer** durch folgende vier Zeilen ergänzt:

```
if [exists maxteilnehmer]{
  putstr "<H2>maximale Anzahl an Teilnehmern</H2>"
  putcomp maxteilnehmer
}
```

Diese Typanpassungen hat keine Auswirkungen auf die existierenden Seitenobjekte **`RS`** und **`EfP`**, die unverändert abgebildet werden. Alle PIL-Objekte vom Typ **`LEHRVERANSTALTUNG`**, auch die bereits existierenden, können nun die neue Komponente verwenden.

Problematisch sind Auftrittshäufigkeitsänderungen von **`0-1`** oder **`0-N`** nach **`1`** oder **`1-N`** oder das Streichen eines Bestandteils. In diesem Fall können bestehende Seitenobjekte, gegen die neuen Kriterien des Typen verstoßen. Entweder müssen bei der nächsten Aktualisierung dieser Objekte die jetzt zwingend vorgeschriebenen Elemente (wenn nicht bereits vorhanden) hinzugefügt und die entfallenden gelöscht werden, oder es gibt Probleme. Dieses ist der Fall, wenn die neuen zwingenden Bestandteile nicht in allen existierenden Objekten gebraucht werden, oder ein Gestrichener nicht entfallen darf. Als einfache Lösung kann auf die Änderung der Auftrittshäufigkeit oder das Streichen verzichtet werden. Gibt es allerdings nur einen sehr niedrigen prozentualen Anteil an Objekten dieses Typs, der den neuen Kriterien nicht genügt, sollte die Anpassung durchgeführt, und für die abweichenden Seitenobjekte ein eigener Typ angelegt werden.

Gibt es komplexe Bestandteile, die in verschiedenen anderen Typen verwendet werden können, definiert man diese als eigenen Typen (in unserem Beispielsystem der Typ **`VERANSTALTUNG`**).

Nehmen wir an unser Kunde will das die Komponente **`inhalt`** nicht nur einen Text, sondern auch eine Auflistung von einzelnen Teilen enthält. Dann benötigen wir Unterkomponenten in dieser Komponente. Aber auch die Komponente **`literatur`** verwendet einen solchen Bestandteil, der zu einem eigenen Typen definiert werden kann. Dieser neue Typ kann dann ebenfalls für die Komponente **`inhalt`** gebraucht werden. Die Anpassungen sehen so aus:

Als erstes entwerfen wir den neuen Typ.

```
TYPE TEXT_LISTE {
  COMPONENT text ? : TEXT
  COMPONENT eintrag * : TEXT
  VIEW html TCL{
    putcomp text
    putstr <UL>
    withall eintrag {
      putstr -n "<li>"
      view html
    }
    putstr </UL>
  }
}
```

Danach ändern wir den Seitentyp **LEHRVERANSTALTUNG**.

```
TYPE LEHRVERANSTALTUNG : SEITE {
   ...
   COMPONENT inhalt: TEXT {
      COMPONENT textliste ? : TEXT_LISTE
      VIEW html TCL{
         if [exists textliste]{ view html }
      }
   }
   ...
   COMPONENT literatur : TEXT_LISTE
   ...
   VIEW html EXTEND inhalt TCL{
      ...
      putstr "<H2>Inhalt:</H2>"
      putcomp inhalt
      ...
      putstr "<H2>Literatur:</H2>"
      putcomp literatur
      ...
   }
}
```

Diese Anpassungen beeinflussen die bereits existierenden Seitenobjekte **RS** und **EfP** nicht, da die Komponente **inhalt** immer noch vom Typ **TEXT** abgeleitet ist. Die zusätzliche Unterkomponente ist optional, d.h. sie muß nicht vorhanden sein.

16.3 Das Beispielsystem aus der Sicht des Benutzers

16.3.1 Beispiel Vorlesungsverzeichnis

In diesem Abschnitt wird beschrieben, wie ein **Benutzer** die Seiten eines Vorlesungsverzeichnisses wartet, pflegt und neue Seiten hinzufügt. Gerade bei diesen Tätigkeiten zeigen sich die Vorteile von AIR, da z.B. auf den Verteilseiten die Verweise inklusive ihrer Sprungadressen automatisch generiert werden oder beim Erstellen neuer Seiten Layout-Gestaltungen wie z.B. Kopfzeilen, Fußzeilen oder Logos nicht umständlich hineinkopiert oder gar neu geschrieben werden müssen, sondern einzig die Angabe des Seitentyps ausreicht, um das entsprechende Layout zu erhalten.

Im Folgenden beziehen sich alle Ausführungen auf das praktische Beispiel des Vorlesungsverzeichnisses. Es kann weltweit eingesehen werden, um jedoch Änderungen vorzunehmen, muß man sich an einer Paßwortabfrage als berechtigter Benutzer zu erkennen geben.

Das Warten der vorhandenen Seiten auf einem Server bedeutet in der Regel, daß man folgende vier Schritte nacheinander durchführt:

1. Blättern in den vorhandenen Seiten des Vorlesungsverzeichnis
2. Ändern einer vorhandenen Seite
3. Hinzufügen einer Seite
4. Überprüfen der geänderten und der neuen Seiten (und des Inhaltsverzeichnisses)

16.3.2 Blättern in den vorhandenen Seiten des Vorlesungsverzeichnis

Der Einstieg in das Vorlesungsverzeichnis erfolgt über eine Titelseite, die sich von den anderen Seiten stark unterscheidet. Während alle Seite, die in dem elektronischen Vorlesungsverzeichnis vorkommen, grundsätzlich eine gewisse Ähnlichkeit haben sollen, besitzt die Titelseite anderseits ihr eigenes Layout.

Unter dem Titel folgt das Inhaltsverzeichnis, in dem die Namen aller vorhandenen Seiten angegeben sind. Diese Namen sind Links zu den angegebenen Seiten, d.h. durch Anklicken gelangt man auf diese Seite. Unten am Fuße der Seite sind zwei Knöpfe angebracht. Auf dem einen steht **`<Ändern>`** und auf dem anderen steht **`<Hinzufügen einer neuen Seite>`**. Der Knopf **`<Ändern>`** ist auf allen Seiten an dieser Position zu finden. Durch das Betätigen erhält man die Möglichkeit, die Inhalte der Seite zu verändern.

Der Knopf **`<Hinzufügen>`** ist nur auf der Titelseite des Vorlesungsverzeichnis vorhanden. Durch das Drücken dieses Knopfes kann man neue Seiten in das Vorlesungsverzeichnis einfügen.

Wählt man eine Seite an, z.B. **`Hardwarepraktikum`**, so gelangt man auf die entsprechende Seite des Vorlesungsverzeichnis. In der Kopfzeile der Seite steht *Praktikum*, da es sich um eine Veranstaltung aus diesem Bereich handelt. In der ersten Zeile steht in großen Buchstaben der Titel der Veranstaltung, darunter in einer oder mehr Zeilen Zeit, Ort, Dozent, Umfang und Nummer der Veranstaltung. Manchmal (optional) steht der Beginn der Veranstaltung darunter. Darauf folgt der Inhalt der Vorlesung und am Ende der Seite wieder optional "Hörer", "Voraussetzungen" und "Literaturhinweise".

Am unteren Ende dieser und jeder anderen Seite befindet sich ein Knopf "Ändern". Auch hier erhält man durch Drücken dieses Knopfes die Möglichkeit, die Inhalte der Seite zu verändern, wie weiter unten beschrieben wird.

16.3.3 Ändern einer vorhandenen Seite

Um eine Seite verändern zu können, muß man sich als berechtigter Benutzer autorisieren. Drückt man auf den Knopf **`<Ändern>`**, so erscheint eine Paßwortabfrage[1]. Nach erfolgreicher Eingabe des Paßwortes hat man sich als berechtigter Benutzer autorisiert und man erhält ein Formular.

In diesem Formular existiert für jede Komponente, die in der Seite vorkommt, ein eigener Bereich. Diese Bereiche sind mit den Texten der Seite ausgefüllt. Der Benutzer kann nun nach Belieben weiteren Text hinzufügen und andern. Am Fuße der Seite gibt es einen Knopf **`<Ändern>`**. Durch die Betätigung werden die gewünschten Änderungen vorgenommen und es erscheint die neue, aktualisierte Seite.

16.3.4 Hinzufügen einer neuen Seite

Das Hinzufügen einer neuen Seite ist dem Ändern einer Seite sehr ähnlich.

Der einzige größere Unterschied ist der, daß nach Anwählen dieses Menüpunkts sich ein Fenster öffnet, in dem sich alle vorhandenen Seitenlayouts befinden. Durch Anwählen eines Layouts öffnet sich auch hier ein neues Fenster, in dem jede Komponente einen eigenen, anfangs unbeschriebenen, editierbaren Bereich hat. Auch in diesem Fenster befinden sich am Fuße der Seiten der bekannte Knopf **`<Ändern>`**.

1. Das Paßwort können Sie bei der Projektgruppe erfragen.

16.3.5 Überprüfen der geänderten und der neuen Seiten (und des Inhaltsverzeichnisses)

Bei jeder Veränderung der Seiten wird die HTML-Darstellung sofort generiert. Man kann also unmittelbar mit der Überprüfung der geänderten Seiten beginnen. Insbesondere das Inhaltsverzeichnis wird auf diese Weise immer auf dem neuesten Stand gehalten und neu erstellte Seiten werden direkt mit aufgelistet.