# Java, XML und Servlets zur Integration datenbankbasierter Applikationen im Web

Albrecht Schmidt und Günther Specht

TU München, Orleansstr. 34, 81667 München
Albrecht.Schmidt@in.tum.de
specht@in.tum.de

**Zusammenfassung** Am Institut für Informatik der Technischen Universität München werden zwei bestehende Technologien, digitale Bibliothekssysteme und multimediale Datenbanken, in einem Web-basierten Kontext integriert, d. h. ein Benutzer soll letztendlich über einen der gängigen WWW-Browser auf die schon vorhandenen digitalen Bibliotheken als auch auf multimediale Informationen zugreifen können. Eine besondere Rolle spielt dabei die Verwendung von Java als Implementationssprachen und XML als (zunächst internem) Datenformat zum Zusammentragen der Information aus den verschiedenen Datenbanken.

## 1 Einführung und Überblick

An der Technischen Universität München läuft im DFG-Schwerpunktprogramm "Verteilte Verarbeitung und Vermittlung digitaler Dokumente" [DFG98] das Projekt OMNIS/2. Darin wird die Integration von Bibliothekssystemen und multimedialen Datenbanken untersucht. Da schon Vorarbeiten in umfangreichen anderen Projekten erbracht worden sind, stand am Anfang die Frage, wie man möglichst große Teile davon übernehmen könne. Dabei ging es nicht nur um das Wiederverwenden von Programmcode, sondern vor allem um die Nutzbarmachung und Integration großer Mengen an Datenbeständen, insbesondere Bibliotheksdatenbanken.
Eine zweite Anforderung bestand in der Verwendung und Evaluierung von modernen Technologien. Nach einer Planungsphase und der Diskussion verschiedener Designvorschläge kam der Entschluß, die Implementation in Java durchzuführen. Dabei spielte einerseits die Plattformunabhängigkeit der Sprache eine große Rolle, andererseits auch die gute Einbettung in die vorhandene WWW-Architekur [Wor98b], insbesondere die Möglichkeit HTTP-gestützter Kommunikation durch URL-Klassen und Servlets [Sun98].
Eine weitere Anforderung bestand in der Integration bestehender Projekte. Da diese bereits über einen WWW-Anschluß verfügten, lag die Idee nahe, diesen auch zu nutzen. Die übliche HTML-Ausgabe dient aber lediglich der Darstellung berechneter Resultate. Insbesondere gehen durch Konvertierung der internen Datenstrukturen nach HTML viele semantische Informationen verloren, die für eine Weiterverarbeitung sehr hilfreich wären. Basierend auf der Idee, statt nach HTML in eine

Sprache zu konvertieren, die auch Semantik in die Dokumentenstruktur miteinbezieht, wurde zusätzlich zur Standard-HTML-Ausgabe eine XML-Ausgabe [Wor98a] (mit modifizierter URL) implementiert, in der die Daten die nötige semantische Tiefe besitzen.
Die genannten Technologien entwickeln sich zudem immer mehr zu einem Standard, so daß eine Vielzahl an Werkzeugen zur Verfügung steht, die deren Einsatz enorm erleichtert.
Ziel dieses Papiers ist es zu zeigen, wie man unter Einbeziehung von XML, Java und Servlets als Kerntechnologien, mehrere Anwendungen in einer Plattform integrieren und an das WWW anschließen kann.

## 2 Bibliothekssysteme

Bei den in der Einführung erwähnten Bibliothekssystemen handelt es sich im wesentlichen um die verschiedenen Ausprägungen des an der TU München entwickelten OMNIS-Systems [KVB97], das in einer Vielzahl an Projekten im Einsatz ist und für das daher mehrere sehr umfangreiche Datenbanken existieren. Es verwaltet die typischen, im Bibliothekskontext auftretenden Dokumentenarten. Bei der Literaturrecherche hat man nicht nur Zugriff auf die herkömmliche Attribute wie Autor, Titel, Verlag, sondern kann auch im Volltext der ersten Seiten eines Dokuments, also im Abstract und/oder Inhaltsverzeichnis suchen.
Ist die Suche erfolgreich gewesen, so besteht die Möglichkeit, sich noch einen optischen Eindruck von einer gescannten Version der gefundenen Dokumente zu machen und danach das Dokument herunterzuladen und auszudrucken, sofern es vollständig vorliegt. Ansonsten kann es – je nach Version und Kontext – evtl. bestellt werden.
Intern arbeitet OMNIS mit drei verschiedenen Attributarten. Da gibt es die von traditionellen Bibliothekskatalogen her bekannten Strukturfelder wie Autor, Titel, Jahr, Verlag. Zusätzlich steht ein Teil des Dokuments im suchbaren Volltext zur Verfügung, der das Ergebnis eines mittels OCR analysierten Scannens ist. Bilddaten stellen das Dokument in seiner ursprünglichen Form dar. Sie können sowohl als Bitmap als auch als Postscript vorliegen. In der aktuellen Version ist noch keine Suche und Verlinkung der in der Datenbank liegenden Quellen möglich.

## 3 Multimediale Datenbanken

Multimediale Datenbanken unterscheiden sich von herkömmlichen Bibliothekssystemem in zweierlei Hinsicht. Zum einen können sie mit neueren kontinuierlichen Medientypen umgehen, im wesentlichen Video und Audio. Dies allein ist schon eine wünschenswerte Eigenschaft, die vielen herkömmlichen Bibliothekssystemen fehlt.
Zum anderen unterstützen sie auch Verlinkungen zwischen Objekten, die über die aus HTML bekannten Verfahren hinausgehen. So enthalten gängige Modelle zu multimedialen Datenbanksystemen auch bidirektionale, getypte und $n : m$-Links, wie es im Dexter-Modell [HS94], das unserer Planung zugrunde liegt, der Fall ist. Typische Anwendungsbereiche in

unserem Kontext sind Literaturverzeichnisse, wo das als Referenz aufgeführte Schriftstück auf Maus-Klick verfügbar gemacht werden kann, oder Zitate, die gleich auf die bezuggenommene Textstelle verweisen können. Im allgemeinen werden sich die genannten Anforderungen nicht mit reinem HTML [Wor98c] erfüllen, auch wenn die Mächtigkeit dieser Sprache durch immer neue Standards und Entwicklungen weiter zunimmt. Soll es z. B. möglich sein, in einem Bild interaktiv einen Link durch Umrahmen der Ankerfläche mit einem Polygon zu setzen, so erfordert dies den Einsatz von Java-Applets.
Im Projekt MultiMAP, ebenfalls an der TU München, wird ein solches multimediales Datenbanksystem entwickelt, das sowohl multimediale Objekte als auch die erwähnten komplexen Linkstrukturen basierend auf dem Dexter-Modell unterstützt [HS94]. Auch hier gibt es schon eine Reihe an Anwendungs-Datenbanken.

## 4 Designziele

Betrachtet man die Eigenschaften der beiden vorgestellten Systeme, so würden sie sich in natürlicher Weise zu etwas Mächtigerem ergänzen; dazu muß man jedoch Möglichkeiten des einen in das andere übernehmen. Das würde zum Beispiel die angesprochenen vernetzten Literaturverzeichnisse ermöglichen, aber auch Annotationen, die ein Benutzer zu Schriftstücken einfügen könnte, oder thematisch begründete Verbindungen.
Somit ergibt sich als Designziel ein System, das die Eigenschaften von Bibliothekssystemen und multimedialen Datenbanken vereint. Eine einfache Code-Integration der Systeme OMNIS und MultiMAP war jedoch aufgrund jeweils eigener Datenformate und -strukturen nicht möglich. Die gewählte Zielspezifikation bestand aus einer Integrationsschicht über beiden Einzelkomponenten, die auf die darunterliegenden Systeme zurückgreift und leicht um weitere Module erweiterbar ist. Diese Integrationsschicht benötigt aber mehr semantische Informationen, als die HTML-Ausgaben der beteiligten Systeme liefern. Da es aber verhältnismäßig einfach ist, einen WWW-Anschluß von einer HTML-Ausgabe auf eine XML-Ausgabe umzustellen – in unserem Fall war nur reine textuelle Substitution im erzeugenden Code nötig – wurde das OMNIS-System, das schon an das WWW angebunden war [CVW95], lediglich um eine XML-Ausgabe erweitert. Es hat sich herausgestellt, daß es so möglich ist, alle in den internen Datenbanken vorhandenen relevanten Informationen unserer integrativen Anwendung zugänglich zu machen.
Für XML als zukünftigem Standard gibt es mittlerweile eine ganze Reihe an Bibliotheken, Programmen, Parsern und Prozessoren, die sich an die W3C-Empfehlung halten und somit einen reibungslosen Datenaustausch ermöglichen.
Ebenso sollte unsere integrative Anwendung selbst keinerlei proprietäre Protokolle verwenden. Es wird oft noch viel Zeit und Mühe darauf verwendet, eigene Kommunikationsprotokolle zu entwerfen. Das ist bei Web-basierten Anwendungen in vielen Fällen gar nicht nötig, da mit dem HTTP-Protokoll [Wor98d] bereits ein Standard zur Verfügung

steht, der die gängigsten Fälle abdeckt. Weiterentwicklungen wie HTTP-NG [Wor98e] sollten bei längerfristigen Projekten auch in die Planung miteinbezogen werden, da sich so in Zukunft Sackgassen vermeiden lassen. Gerade die Java-eigenen URL-Klassen reichen oft vollkommen zur Kommunikation aus.
Weiterhin sollte eine integrative Anwendung auch skalierbar sein. Der Einsatz einer objektorientierte Sprache wie Java bietet die Möglichkeit, weitere Anwendungen in Modulen unterzubringen, die – soweit sie XML als Ausgabe liefern – sich leicht in die Gesamtarchitektur einbinden lassen. Die Initialisierung dieser Module könnte in der `init()`-Methode eines Servlets geschehen. Für weniger häufig gebrauchte Module ist sogar ein dynamisches Laden denkbar.
Zusammmenfassend lassen sich unsere Designziele folgendermaßen beschreiben:

1. Verwendung von standardisierten, plattformunabhängigen Technologien
2. Modularität und Skalierbarkeit
3. größtmögliche Wiederverwendbarkeit bestehender Datenbanken und Applikationen

## 5 Architektur

Diese Designziele übertragen sich auf natürliche Weise in die Architekur, die in diesem Abschnitt vorgestellt wird.

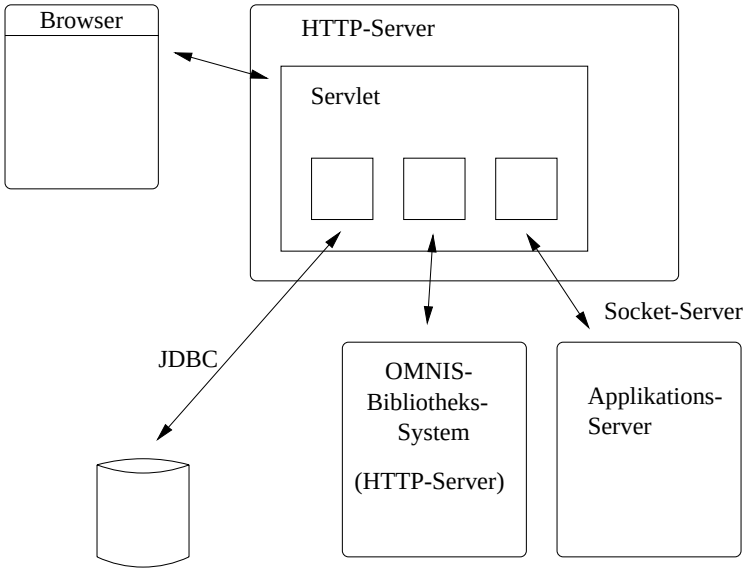


**Abbildung1.** Unser Applikations-Schema

Da wir uns als Ziel gesetzt haben, möglichst nur über Standard-Protokolle zu kommunizieren und einen WWW-Anschluß zur Verfügung zu stellen, liegt es nahe, den von uns benutzten und notwendigen HTTP-Server mit Hilfe von Servlets zu einem vollständigen Applikationsserver auszubauen. Da die zu implementierenden Methoden eines Servlets genau den verschiedenen HTTP-Anfragen (`GET, PUT, POST` etc.) entsprechen, können wir unsere Kommunikation ohne großen Mehraufwand über das HTTP-Protokoll abwickeln. Diese Überlegungen resultierten im linken oberen Teil unsere Applikationsschems, wie es in Abbildung 5 zu sehen ist: Browser und HTTP-Server kommunizieren nur über einen Standard-HTTP-Port.
Vom Browser eines Benutzers kommen Anfragen, die in den Servlets verarbeitet werden. Das können Anforderungen des Browsers selbst sein, wie sie etwa durch Aktivieren eines Bookmarks oder Anforderung einer neuen URL ausgelöst werden, möglicherweise sind sie auch von Java-Applets z. B. in interaktiven Bildern oder in HTML-Seiten eingebettete JavaScript-Teile gestellt.
Diese Anfragen werden entsprechend ihrer Semantik aufgespalten und an die für sie zuständigen Module weitergereicht. Wenn es darum geht, einen Link zu verfolgen, wird in der Link-Datenbank das Linkziel bestimmt, eine neue Seite zusammengestellt und an den Web-Browser des Benutzers geschickt. Bei einer Stichwortsuche in einer Bibliotheksdatenbank werden die Suchbegriffe in das entsprechende URL-Format übersetzt, abgeschickt, und die XML-Antwort interpretiert, mittels eines XSL-Prozessors [Wor97] nach HTML übersetzt und an den Browser zurückgegeben.
Diese Grundideen übersetzen sich auf natürlich Weise in eine Java-Umgebung. Durch Servlets erweitert man den Web-Server um eigene Prozeduren. In der `init()`-Routine werden allgemeine Initialisierungen vorgenommen wie das Anlegen von Datenbankobjekten, die wiederum Verbindungen zu Datenbanken aufbauen, oder die Instanziierung von Hüllklassen für Datenquellen wie der OMNIS-Datenbank. Weniger häufig benötigte Module, wie etwa Autorenkomponenten, können während des laufenden Betriebs nachgeladen werden.
Die Verbindung zu der erwähnten Linkdatenbank kann innerhalb eines eigenen Moduls über einen JDBC-Anschluß und damit ebenfalls über eine standardisierte Schnittstelle bewerkstelligt werden. Wenn traditionelle Internet-Socket-Server integriert werden sollen, so kann die Kommunikations-Verbindung über Java-Sockets, einem weiteren Element der Java-API, hergestellt werden. In diesem Fall ist allerdings eine Emulation des von der integrierten Anwendung geforderten proprietären Protokolls notwendig, was je nach Protokoll recht aufwendig werden kann. Handelt es sich bei dem Socket-Server um einen HTTP-Server, kann mit ihm über die WWW-konformen Routinen in `java.net.HttpURLConnection` kommuniziert werden.
Viele Datenbankhersteller [Inf98,Ora98] verfolgen übrigens ein ähnliches Konzept, nur daß sie von das Pferd quasi von der anderen Seite aufzäumen. Sie integrieren in Datenbanken immer mehr Funktionen, die vorher vom Betriebssystem oder von Spezialapplikationen wie HTTP-Servern übernommen wurden, und vereinfachen so die konzeptuelle Projekt-

Planung. JDBC ist unter diesem Gesichtspunkt ein Versuch der Vereinheitlichung auf Programmiersprachenseite.

## 6 Servlets

Zwar sind Servlets in gewisser Weise ein Sprachelement von Java, jedoch ist es im Anwendungsszenario oft sinnvoll, sie als einfache Möglichkeit zu sehen, einen speziell auf die eigenen Bedürfnisse zugeschnittenen HTTP-Server zu erstellen. Dies gilt in besonderer Weise, da Servlets mit Infrastruktur außerhalb der eigenen Programmumgebung sehr eng zusammenarbeiten.
Insofern hat die Verwendung von Servlets größere Auswirkungen auf das Design des Gesamtsystems, als es bei der bloßen Verwendung eines weiteren Teils der Java-API der Fall wäre. Einerseits machen sie als Erweiterung des HTTP-Servers es überflüssig, einen eigenen Applikations-Server zu schreiben. Andererseits impliziert die oftmals als Vorteil angeführte Tatsache, daß Servlets als Threads und nicht als Prozesse laufen, daß Teile des Systems "Thread-sicher" sein müssen, d. h. sie sich gegen die Probleme absichern, die üblicherweise entstehen, wenn mehrere Prozesse auf dieselben Ressourcen zugreifen. Insbesondere bei Datenbankanbindungen können Probleme entstehen, da JDBC-Treiber von ihrer Architektur her nicht Thread-sicher sind und somit Thread-sichere Hüll-Klassen oder Container geschrieben werden müssen.
Eine mögliche Lösung ist, in einem `java.util.Vector` eine Liste von offenen Datenbankverbindungen zu halten. Wenn nun ein Thread eine Datenbankanfrage absetzen will, greift er über eine `sychronized`-Methode auf die Liste zu, holt sich eine Verbindung und löscht das entsprechende Element in der Liste. Sobald er seine Antworten erhalten hat, fügt er die Verbindung wieder in die Liste ein. Abbildung 6 zeigt einige Code-Fragmente, die dies verdeutlichen.
Die einfache Handhabe von Threads ohne das explizite Setzen und Freigeben von Sperren macht es einfach, solche Konzepte schnell in Java umzusetzen. Im Gegensatz zu einer entsprechenden C- oder C++-Implementation ist der zusätzliche Programmieraufwand eher gering. Wohl aber ist der Planungsaufwand höher.

## 7 Java

Der Vorteil, den der Einsatz von Java gebracht hat, ist neben der Plattformunabhängigkeit die problemlose Einbettung in das WWW. Das in der Standard-API enthaltene Packet `java.net` stellt alle nötigen Kommunikationsroutinen zur Verfügung, ohne daß ein Einbinden von Nicht-Standard-Bibliotheken erforderlich wäre.
Durch diese auf transparente Kommunikation ausgelegten Sprachelemente bietet sich Java für die Entwicklung von Anwendungen wie unserer an. Die Entwicklungzeiten waren kurz, schon bald stand ein Prototyp zur Verfügung.
Ein weiterer Punkt ist die relativ "billige" Parallelität durch mehrere Threads, die bei Verwendung von Servlets fast automatisch entsteht.

```
class DataBase {
  ...
  public synchronized Connexion getConnexion()
    throws SQLException, ClassNotFoundException {
    if (Connexions.isEmpty ()) {
       // Vektor ist leer
       // evtl. temporäre Verbindung aufbauen
       ...
    }
    else {
      c = (Connexion) Connexions.elementAt(0);
      Connexions.removeElementAt(0);
    }
  }

  public synchronized void releaseConnexion(Connexion c) {
    if (Connexions.size() >= MaxNumberOfConnexions) {
      // evtl. temporäre Verbindungen auflösen
      ...
    }
    else { Connexions.addElement(c); }
  }
}
```

**Abbildung2.** Datenbankzugriff in Threads

Der Trade-Off dafür sind die höheren Entwicklungskosten für die benötigten Klassen, die Thread-sicher sein müssen. Diese Kosten würden aber bei jeder Parallelarchitektur, die Threads benutzt, anfallen. Gerade die einfache Handhabe von Threads mittels `synchronized`-Methoden stellt eine wesentliche Vereinfachung gegenüber den von C bzw. C++ bekannten POSIX-Threads dar.
Auch für Zugriffe auf relationale Datenbanken stellt Java ein Standard-Sprachelement zur Verfügung. Während bisher Embedded-SQL-Code oder spezielle Bibliotheken verwendet werden mußten, kann jetzt die Datenbank über eine nicht mehr herstellerspezifische JDBC-Schnittstelle angesprochen werden, die Zeitersparnis im Vergleich zu den anderen Techniken ist signifikant.

## 8 XML und HTML

Wie anfangs beschrieben, war die Umstellung einer HTML-Ausgabe auf XML ein wichtiger Punkt im Integrationsprozeß. Anhand eines Beispiels soll die Rolle von XML in unserem Kontext verdeutlicht werden. Abbildung 3 zeigt eine etwas vereinfachte HTML-Ausgabe eines WWW-Bibliothekssystems. Da ist einerseits die URL des Linkziels, wo das Dokument

abrufbar ist, andererseits sichtbar am Bildschirm der Autor und der Titel. Zusätzlich ein Anker für den Link, der zu dem Dokument führt.

```
<A HREF= URL>
M. Usterautor, Wie man in Java programmiert
</A>
```

**Abbildung3.** Ergebnis einer Suche in HTML

```
<BUCH>
  <URL>URL</URL>
  <AUTOR>M. Usterautor</AUTOR>
  <TITEL>Wie man in Java programmiert</TITEL>
</BUCH>
```

**Abbildung4.** Ergebnis einer Suche in XML

Für eine Applikation, die diesen HTML-Text als Eingabe erhält, ist es sehr schwierig, auf die semantische Struktur zu schließen, zumal diese noch von Anfrage-Ergebnis zu Anfrage-Ergebnis verschieden sein kann. Im Gegensatz dazu enthält die XML-Version in Abbildung 4 zu jeder Ausgabe die Meta-Information, um welche Art von Daten es sich handelt. Der Name des Autors ist von einem `AUTOR`-Tag-Paar umgeben. Ebenso verhält es sich mit URL und Titel. Zusätzlich ist der Block von einem `BUCH`-Tag umgeben, das als zusätzliche Information enthält, daß es sich bei dem umschlossenen Text um die Beschreibung eines Buches handelt. Wie oben beschrieben, ist ein Umstellen der Ausgabe einer bestehenden Applikation von HTML nach XML eine relativ "billige" Lösung, da man oft lediglich die Ausgabe-Tags verändern muß. Besonders wenn Techniken wie Server-Parsed-HTML-Templates, wie es bei OMNIS der Fall war, zum Einsatz kommen, fällt eine Umstellung leicht. Wenn man weiter das HTTP-Protokoll benutzt, um seine Anfragen zu stellen, besteht die Haupt-Integrationsaufgabe darin, die entprechenden Anfragen in einer URL nachzumodellieren und diese dann abzuschicken.
Die Rolle von XML in unserer Anwendung kann noch weiter gefaßt werden. Bei verschiedenen heterogenen Modulen, die in größerem Maße voneinander abhängen, kann XML als intermodulares Austauschformat verwendet werden.
XML selbst enthält keine Informationen darüber, wie es von einer Anwendung visuell oder akustisch dargestellt werden soll. Im Gegensatz zu HTML gibt es keine Konvention, daß bestimmte Tag-Namen eine Bedeutung wie "Drucke das folgende Wort kursiv" haben. Zu diesem Problem

gibt es mehrere mögliche Lösungsansätze. Einerseits können proprietäre Methoden Einsatz finden, d. h. ein XML-Quelltext wird eingelesen, ein Parse-Baum wird aufgebaut, und aus diesem eine Ausgabe generiert. Andererseits kann man auch Standard-Techniken verwenden wie Style-Sheets oder XSL-Prozessoren [Wor97]. Style-Sheets bieten sich an, wenn die Struktur des XML-Textes im wesentlichen der Struktur der gewünschten visuellen Darstellung entspricht, da sie für jedes XML-Tag die visuellen oder akustischen Attribute angeben. XSL-Prozessoren geben größere Freiräume, da sie ein (fast) beliebiges Traversieren des von der Dokumentenstruktur implizierten Syntaxbaums erlauben.
Gegenwärtig läßt sich die Tendenz ausmachen, bei Client/Server-Systemen die Darstellungslogik zu möglichst großen Teilen in den Client zu verlegen. Wo die Mächtigkeit von Style-Sheets bzw. XSL-Prozessoren nicht ausreicht oder die geforderte Interaktivität nicht bietet, werden Java-Applets eingesetzt.

## 9 Zusammenfassung

Wir haben gesehen, wie man Web-basierte Anwendungen integrieren kann, ohne große Veränderungen an den Strukturen oder Quelltexten der beteiligten Module vornehmen zu müssen. Dies geschah aus der Grundidee, einen WWW-Port, der auf Anfragen eine für graphische Darstellung der Ergebnisse aufbereitete Antwort liefert, von einer HTML- auf eine XML-Ausgabe umzustellen und als generischen Zugang zu einer Applikation zu betrachten.
Java leistet in diesem Zusammenhang besonders wertvolle Dienste, da es viele WWW-konforme Sprachelemente enthält und die Entwicklungszeiten kurz sind.

## Literatur


[CVW95] Alexander Clausnitzer, Pavel Vogel, and Stephan Wiesener. WWW-Interface to the OMNIS/Myriad Literature Retrieval Engine. In *The Third International World-Wide Web Conference: Technology, Tools and Applications, Darmstadt*, 1995.

[DFG98] DFG. DFG-Schwerpunktprogramm: Verteilte Verarbeitung und Vermittlung digitaler Dokumente. http://www.graphics.uni-bonn.de/dfgspp.VVVDD, 1998.

[HS94] F. Halasz and M. Schwartz. The Dexter Hypertext Reference Model. *Communications of the ACM*, 37(2):30–39, 1994.

[Inf98] Informix. Developing datablade modules for informix dynamic server with universal data option. http://www.informix.com/informix/whitepapers/, 1998.

[KVB97] Wolfgang Kowarschick, Pavel Vogel, and Rudolf Bayer. ELEKTRA: An Electronic Article Delivery Service. In *8th International Conference on Database and Expert Systems, Tolouse*, July 1997.

[Ora98] Oracle, Inc. Cartridges. http://www.oracle.com/cartridges/, 1998.

[Sun98] Sun. Java Documentation. http://www.java.sun.com/docs/index.html, 1998.

[Wor97] World Wide Web Consortium. A Proposal for XSL. http://www.w3.org/TR/NOTE-XSL.html, August 1997.

[Wor98a] World Wide Web Consortium. Extensible Markup Language. http://www.w3.org/XML/, 1998.

[Wor98b] World Wide Web Consortium. Homepage. http://www.w3.org/, 1998.

[Wor98c] World Wide Web Consortium. HTML 4.0 Specification. http://www.w3.org/TR/REC-html40/, 1998.

[Wor98d] World Wide Web Consortium. HTTP – Hypertext Transfer Protocol. http://www.w3.org/Protocols/, 1998.

[Wor98e] World Wide Web Consortium. Hypertext Transfer Protocol - Next Generation. http://www.w3.org/Protocols/HTTP-NG/, 1998.