Techniken für Web-basierte Datenbankanwendungen: Anforderungen, Ansätze, Architekturen

Henrik Loeser

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
67653 Kaiserslautern
E-Mail: loeser@informatik.uni-kl.de

Zusammenfassung: Die rasche Verbreitung und einfache Nutzungsmöglichkeit des World Wide Web (WWW) hat heute schon zu einer Vielzahl Web-basierter Datenbankanwendungen geführt. Man geht jedoch davon aus, daß das eigentliche Wachstum des WWW und seiner Anwendungen durch die Fortschritte in der Kommunikations- und Informationstechnik erst noch bevorsteht [10]. In diesem Beitrag stellen wir die bei der Realisierung Web-basierter DB-Anwendungen eingesetzten Techniken vor. Hierzu untersuchen wir zunächst typische DB-Anwendungen, die über das WWW abgewickelt werden, um sie anhand ihrer Kriterien charakterisieren zu können. Daran anschließend diskutieren und analysieren wir nach einem Überblick die unterschiedlichen Realisierungstechniken im Detail.

Schlüsselwörter: Datenbanksysteme, World Wide Web, Datenbank-Gateways, Informationssysteme, HTML, Java

Abstract: The amazing growth in the use of the World Wide Web (WWW or Web) has already led to a large number of Web-based database applications. Due to the continuing progress in the communication and information technologies, however, experts assume “the revolution yet to happen” [10] in the Internet. In this paper, we discuss the concepts and techniques used to implement Web-based DB applications. First, we analyse typical DB applications provided in the Web in order to determine suitable criteria for their evaluation. After the classification of these applications, we survey the spectrum of available implementation techniques, before we discuss each of them in detail.

Keywords: Database Systems, World Wide Web, Database Gateways, Information Systems, HTML, Java

Computing Reviews Classification: H.3.5, H.5.1, H.5.4, C.2.4, I.7.2

1. Einleitung

Im Jahre 1990 begann Tim Berners-Lee am CERN das Projekt *World Wide Web* (WWW oder Web, [5]). Ursprünglich war es zur Verbesserung der internen Informationsbereitstellung gedacht [3]. Dank des einfachen, aber guten Konzeptes von untereinander verknüpften HTML-Dokumenten (*HyperText Markup Language*) und der Integration bisheriger Internet-Dienste über einheitliche Adressen (URL, *Uniform Resource Locator*) unter einer gemeinsamen Oberfläche, dem *Web-Browser*, entwickelte es sich zu dem, was es heute ist: einem immer noch stark wachsendem, nicht überschaubaren Zusammenschluß von Teilnehmern und Informations- und Diensteanbietern aus allen Bevölkerungsgruppen nahezu aller Länder.

Auslöser für dieses starke Wachstum des Web war jedoch nicht nur der einfache Zugriff auf bereitgestellte statische Informationen. Durch die Nutzung des *Common Gateway Interface* (CGI, [8]) ist es möglich, Programme durch den mittels HTTP (*HyperText Transfer Protocol* [11], das Kommunikationsprotokoll im WWW) kontaktierten Web-Server aufrufen und die von diesen Programmen generierten HTML-Seiten als Antwort an die Web-Browser zurückliefern zu lassen. Im Jahr 1995 wurden die Möglichkeiten des Web durch die von SUN Microsystems entwickelte Programmiersprache Java [13] erweitert. In Java implementierte Programme (*Applets*), die in einen plattformunabhängigen *Bytecode* übersetzt und in HTML-Seiten eingebunden werden, können von einem Web-Server geladen und im Browser ausgeführt werden.

Basierend auf diesen Techniken entstanden im Laufe der Zeit viele Web-basierte Anwendungen, kurz Web-Anwendungen. Betrachtet man die sich abzeichnenden Entwicklungen und Trends wie Netzwerk-Computer (NCs) oder elektronische Geräte mit Web-Oberfläche/-Bedienung (Fernseher, Mobiltelefone etc.), so befinden sich das Web und die Anwendungen noch in einer frühen Phase, die eigentliche Revolution steht noch aus [10]. Dies bedeutet, daß in Zukunft der Web-Browser noch stärker als homogene und einfach zu bedienende Benutzerschnittstelle zu Anwendungssystemen und Geräten eingesetzt wird. Wie bei anderen Applikationen auch beruhen viele der Web-Anwendungen auf der Verwendung von Datenbanken (DBs). Dies bedeutet, daß in Zukunft noch stärker Web-basierte DB-Anwendungen entwickelt werden.

In diesem Beitrag wollen wir die bei der Entwicklung Web-basierter DB-Anwendungen eingesetzten Techniken diskutieren. Dazu werden wir im folgenden Kapitel unterschiedliche Web-Anwendungen vorstellen und ihre jeweiligen Anforderungen bzgl. der DB-Anbindung analysieren, wozu wir einen Kriterienkatalog erarbeiten. In Kapitel 3 geben wir einen kurzen Überblick über die verschiedenen Möglichkeiten des Web-basierten DB-Zugriffes, bevor wir in Kapitel 4 detailliert auf HTTP-basierte und in Kapitel 5 auf applet-basierte Verfahren eingehen. Hierbei werden wir bewußt auf die Klassifikation auf dem Markt verfügbarer Werkzeuge verzichten. Dies hat mehrere Gründe: In der noch jungen Geschichte des Webs haben sich die verwendeten Produkte sehr schnell verändert, andere Produkte wurden nur angekündigt, aber nie auf den Markt gebracht, und junge Start-up-Firmen wurden z. T. schon nach kurzer Zeit von großen Unternehmen aufgekauft und deren Produkte umbenannt oder in andere integriert. Wer sich für ein bestimmtes Produkt interessiert und dazu Informationsmaterial besitzt, kann das Produkt mit Hilfe dieses Beitrags jedoch leicht in die entsprechenden Kategorien einordnen und dadurch unsere klassenspezifische Bewertung übernehmen. Den Abschluß dieses Beitrags bildet ein Resümee mit einem Ausblick auf mögliche Entwicklungen des Gespanns Web und Datenbanken.

2. Web-basierte Datenbankanwendungen

Der große Erfolg des WWW mit seinen Millionen von Benutzern resultiert insbesondere aus der einfachen Bedienbarkeit. Dies bedeutet, daß eine Vielzahl von Diensten unter einer gemeinsamen, leicht zu erlernenden Benutzeroberfläche, dem Web-Browser, angeboten wird. Dabei besteht über Hyperlinks die Möglichkeit, einzelne Dokumente miteinander zu verknüpfen, so daß der Anwender durch das Anklicken eines Hyperlinks zum nächsten Dokument gelangt. Über die Nutzung von HTML-Formularen in Verbindung mit dem CGI ist es nun möglich, vom Benutzer eingegebene Daten an den WWW-Server zu schicken und entsprechende (auf den Eingaben basierende, dynamisch erstellte) Antwortseiten an den Web-Browser zur Präsentation zurückzugeben. Eine andere Möglichkeit zur Interaktion besteht in der Verwendung von Java-Applets, mit denen ebenfalls Benutzereingaben entgegengenommen und verarbeitet werden können.

Basierend auf diesen beiden sich in ihren Charakteristika stark unterscheidenden Technologien haben sich in den letzten Jahren viele Web-Anwendungen entwickelt. Wie bei traditionellen Anwendungen auch, beruhen viele von ihnen auf der Verwendung von Datenbanken.

Im folgenden wollen wir zunächst unterschiedliche Web-basierte DB-Anwendungen kurz vorstellen, um dann auf dieser Basis einen allgemeinen Katalog zur Charakterisierung solcher Web-Anwendungen zu erarbeiten. Im Anschluß daran wollen wir noch einmal auf die vorgestellten Anwendungen eingehen und sie auf der Basis unseres erarbeiteten Kriterienkataloges charakterisieren.

2.1 Anwendungen

Im Laufe der Entwicklungsgeschichte des WWW sind unterschiedliche DB-basierte Anwendungen entstanden, deren Spektrum sich durch folgende Beispiele vorstellen läßt:

- ***Adreßdatenbank***: Interessenten können ihre Adresse eingeben und zwecks Informationsanforderung an den Web-Server schicken. Dabei kann zusätzlich z. B. noch die Auswahl aus dem angebotenen Informationsmaterial oder die Möglichkeit zur Eingabe eines Kommentars (*Feedback*-Möglichkeit) angeboten werden.

- ***Elektronisches Gästebuch***: Die Besucher einer Web-Seite können sich mit Namen und Adresse sowie einem Spruch in ein Gästebuch eintragen, in dem geblättert und evtl. auch gesucht werden kann.
- ***Online-Tracking***: Der Benutzer kann durch die Eingabe einer Paket- oder einer Bestellnummer sich über den aktuellen Transport- bzw. Bearbeitungszustand seines Auftrags erkundigen. Hierbei wird suchend auf einen aktuellen und z. T. recht großen Datenbestand zugegriffen.
- ***Online-News***: Basierend auf einer Auflistung der aktuellen Schlagzeilen kann zum ausführlichen Artikel gesprungen werden. Je nach Anbieter kann dabei zwischen öffentlichen und erst nach einer (kostenpflichtigen) Registrierung zugänglichen Teilen unterschieden werden. Oft ist auch eine Suche im vorhandenen Datenbestand möglich.
- ***Nachschlagewerk (Katalog)***: Über eine inhaltliche oder alphabetische Gliederung oder über die gezielte Suche kann die gewünschte Information, z. B. eine Adresse, eine Telefonnummer oder eine Übersetzung, aufgefunden werden. Über Querverweise lassen sich evtl. thematisch verbundene Inhalte, z. B. eine genauere Produktinformation, erreichen.
- ***Bestell-Katalog***: Aufbauend auf der vorherigen Anwendung können nun die gefundenen Artikel oder Dienstleistungen auch bestellt werden. Dazu wird nach und nach ein elektronischer Warenkorb gefüllt, dessen Inhalt zum Abschluß durch Eingabe der Adresse oder der Kundennummer und eventueller Zahlungsmodalitäten bestellt werden kann. Die Daten werden dann in das Bestellsystem übernommen und bearbeitet.
- ***Online-Banking / Online-Broking***: Dem Bankkunden wird durch dieses Web-Angebot die Möglichkeit geboten, viele seiner Bankgeschäfte über das Internet zu führen, d. h. den Kontostand abzufragen, Überweisungen und Wertpapiergeschäfte zu tätigen, Daueraufträge einzurichten, zu löschen oder zu ändern sowie administrative Tätigkeiten wie das Ändern der Zugangskennung (PIN) vorzunehmen. Für die Abwicklung dieser Aktionen werden besondere Sicherheitsvorkehrungen benötigt.
- ***„Web-fähige" Geschäftsanwendungen***: Die Anwender können mit Hilfe des Web-Browsers bisher auf Textmasken oder andere GUIs beschränkte DB-basierte Geschäftsanwendungen bedienen, wie z. B. eine Auftragsbearbeitung. Wichtiges Kennzeichen solcher Anwendungen ist das Abarbeiten von Mehrschritt-Vorgängen, die jeweils Benutzerinteraktion voraussetzen sowie die überwiegende Nutzung innerhalb eines Intranets.

2.2 Kriterienkatalog

Bei den gerade skizzierten Web-Anwendungen lassen sich große Unterschiede in Bezug auf die jeweiligen Systemanforderungen feststellen. Hierzu gehören Differenzen bzgl. der Zugriffsarten, der Datenmengen, der Sicherheits- und anderen Anforderungen, die wir im folgenden systematisieren wollen:

- ***Art des Zugriffs***: Während auf Nachschlagewerke in der Regel nur lesend und auf die vorgestellte Adreß-DB meist nur schreibend zugegriffen wird, erfolgt der Zugriff bei vielen Anwendungen in beiden Modi. Allerdings ist ein überwiegend lesender Zugriff seitens der Web-Anwendung bzw. eine Trennung des Datenbestands bzgl. lesend/schreibend festzustellen, wie z. B. der Bestell-Katalog mit Katalogdaten (lesend), Warenkorb (lesend/schreibend) sowie Bestellsystem (schreibend).
- ***Änderungshäufigkeit / Aktualität der Daten***: In Abhängigkeit von der Änderungshäufigkeit der zugrundeliegenden Datenbasis kann eine Pufferung der Ergebnisdaten einer lesenden Anfrage vorgenommen werden. So können z. B. bei einem Nachschlagewerk die Daten, eine entsprechende Architektur vorausgesetzt, gepuffert werden (siehe Kapitel 4.3), dagegen ist dies z. B. bei Online-Nachrichten in der Regel nicht sinnvoll.
- ***Zahl der gleichzeitigen Zugriffe***: Durch eine hohe Zahl gleichzeitiger Zugriffe kann es je nach eingesetzter Technik und Architektur zu einem Engpaß an Ressourcen kommen. Daher ist in Abhängigkeit vom erwarteten Zugriffsaufkommen ein geeignetes Verfahren zu wählen, um bei gleichzeitiger Anwendungsnutzung durch viele Benutzer trotzdem kurze Antwortzeiten und einen hohen Systemdurchsatz erzielen zu können.
- ***Datenüberlappung der Zugriffe (DB-Sicht)***: Werden von den Benutzern ähnliche Anfragen gestellt oder dieselben Makro-/HTML-Dateien (siehe Kapitel 4.2) verwendet, so läßt sich mit einer geeig-

neten Architektur und einer gezielten Pufferung eine Geschwindigkeitssteigerung erzielen (siehe Kapitel 4.3).

- ***Art der Daten(typen)***: Da HTML nur die Darstellung alphanumerischer Daten sowie von Pixelgrafiken gut unterstützt, müssen für die Darstellung z. B. geometrischer Daten andere Techniken eingesetzt werden.
- ***Datensensivität***: Während bei vielen Anwendungen die übertragenen Daten nicht sehr sensitiv sind, ist dies z. B. beim Online-Banking, bei der Eingabe einer Kreditkarten- oder Kundennummer während einer Bestellung sowie bei der Arbeit mit unternehmensinternen Daten der Fall. Dies bedeutet, daß bei der Datenübertragung Schutzmaßnahmen in Form von Datenverschlüsselung notwendig sind.
- ***Sicherheitsbedarf***: Zwar möchte man immer den Zugriff von Unbefugten auf ein DBS bzw. ein Anwendungssystem vermeiden, jedoch hat diese Anforderung bei sensitiven Daten und Unternehmensanwendungen, seien es Bestell-Kataloge, Geschäftsanwendungen oder Banksysteme, einen besonders hohen Stellenwert. Daher sind gezielt Vorkehrungen zu treffen und das sog. *Backend-System* von der Außenwelt abzuschirmen.
- ***Benutzerauthentisierung***: Einige der Anwendungen werden nur für einen ausgewählten Nutzerkreis angeboten, wie z. B. der Zugriff auf ein Nachrichtenarchiv oder Geschäftsanwendungen in einem Intranet. Hierzu ist beim Anwendungsstart die Authentisierung des Benutzers, meist über eine Kombination aus Name und Paßwort, notwendig.
- ***Benutzeridentifikation***: Eng verknüpft mit der vorhergehenden Eigenschaft ist die Benutzeridentifikation, die für die Personalisierung von Angeboten eingesetzt wird. Jedoch erfordern Anwendungen, wie z. B. ein individuelles Nachrichtenangebot oder der Zugriff auf Tracking-Informationen keine so strengen Sicherheitsanforderungen bei der Identifikation wie z. B. das Online-Banking, so daß der Benutzer an sich anonym bleiben kann.
- ***Anzahl der Arbeitsschritte/Länge einer „Sitzung“***: Während z. B. das Eintragen einer Adresse für die Informationsanforderung oder das Abfragen von Tracking-Information einschrittige Vorgänge, d. h. ohne Anwendungskontext sind, benötigen das Füllen eines Warenkorbs oder Geschäftsanwendungen einen Arbeitskontext. Dies bedeutet, daß ein „Zustand im zustandslosen Web“ seitens des Backend-Systems realisiert werden muß (siehe Kapitel 4.5).
- ***„Verweildauer“***: Je nach Dauer des Aufenthaltes eines Benutzer bei einer Web-Anwendung, d. h. einem Angebot aus mehreren Seiten oder Formularen, sind entsprechende Technologien auszuwählen. So erscheinen z. B. bei einem kurzen Aufenthalt Techniken mit einer relativ langen Lade- bzw. Vorbereitungszeit eher weniger sinnvoll.

2.3 Charakterisierung

Wir wollen nun noch einmal die bereits vorgestellten Anwendungen betrachten und sie basierend auf dem gerade erarbeiteten Kriterienkatalog charakterisieren. Eine umfassende und verdichtete Darstellung gibt die Übersicht in Tabelle 1. Die für eine Anwendung bedeutungsvollen Kriterien sind durch eine Schattierung hervorgehoben.

- ***Adreßdatenbank***: Auf die DB wird (vom WWW aus!) nur schreibend zugegriffen. Da nur einmal die Adresse eingegeben wird, ist die Verweildauer bei dieser Anwendung sehr kurz.
- ***Elektronisches Gästebuch***: Aufgrund des Blätterns im Gästebuch treten lesende Zugriffe häufiger auf als schreibende. Dabei ist die Datenüberlappung wegen des Zugriffs auf die letzten Einträge recht hoch.
- ***Online-Tracking***: Beim Tracking greifen weltweit viele Benutzer meist gleichzeitig auf einen großen und sich ständig ändernden Datenbestand zu. Eine Identifikation erfolgt entweder in Zusammenhang mit einer Authentisierung über Namen und Paßwort oder einfach z. B. über die Eingabe einer gültigen Paketnummer. Um Manipulationen vorzubeugen, muß das Backend-System geschützt werden.
- ***Online-News***: Der Zugriff auf Online-News erfolgt lesend, wobei eine sehr hohe Datenaktualität vorliegt. Oft greifen sehr viele Leute gleichzeitig auf dieselben Übersichtsseiten und aktuellen Artikel zu, die jeweils multimedial, d. h. mit Bildern oder Videosequenzen, aufbereitet sein können. Je nach System (z. B. *Pay-Per-View* für registrierte Kunden) ist z. B. für die Authentisierung (Paßwort-

eingabe) Übertragungssicherheit notwendig. Mit Hilfe einer Benutzeridentifikation können auch personalisierte Nachrichten angeboten werden.

- ***Nachschlagewerk (Katalog)***: Wegen ihres Nutzens sowie der hohen Akzeptanz wird auf die Daten, insbesondere bei Nachschlagewerken, gleichzeitig von vielen Benutzern zugegriffen. Jedoch kommt es dabei abhängig vom jeweiligen Dienst zu mehr oder weniger Datenüberlappung. Auch die Anforderungen bzgl. Systemschutz hängen sehr stark von der konkreten Anwendung und dem Betreiber ab. Bei Produktkatalogen werden die Daten in der Regel multimedial, bei Wörterbüchern und Adreßdaten rein alphanumerisch sein.
- ***Bestell-Katalog***: Anders als bei reinen Nachschlagewerken erfolgt auch ein schreibender Zugriff, zum einen für das Führen des Warenkorbs, zum anderen zum Abschicken der Kundendaten. Da der Warenkorb bei allen Aktionen mitgeführt werden muß, dauert eine Sitzung entsprechend lange an. Zur eigentlichen Bestellabwicklung muß auf das operative Bestellsystem zugegriffen werden, so daß hier entsprechende Sicherheit verlangt wird. Eine Identifikation bei Zugriff auf z. B. den Lieferstatus kann über die Abfrage von Kundennummer und PLZ erfolgen.
- ***Online-Banking / Online-Broking***: Bei diesem Anwendungstypus liegt das Hauptaugenmerk auf der Sicherheit, d. h. auf Abschirmung des Backend-Systems, Benutzerauthentisierung und Übertragungssicherheit. Die Sitzungslänge variiert in Abhängigkeit vom jeweiligen Bankgeschäft. So kann z. B. eine Aktienorder aufgegeben und auf die Kaufbestätigung gewartet oder auch nur der Kontostand abgefragt werden.
- ***Web-fähige Geschäftsanwendung***: Aufgrund der Unterschiede zwischen einzelnen Web-fähigen Anwendungen ist eine genaue Klassifikation schwierig. Gemeinsame Anforderungen sind die Sitzungslänge, d. h., daß Mehrschritt-Vorgänge bearbeitet werden, sowie der Sicherheitsbedarf für das Backend-System, um einer Manipulation durch Unbefugte vorzubeugen.

	Zugriffsart	Aktualität	Gleichzeitiger Zugriff	Datenüberlappung	Datentypen	Datensensivität	Sicherheitsbedarf	Benutzerauthentisierung	Benutzeridentifikation	Sitzungslänge	Verweildauer
Adreßdatenbank	S	0	0-1	0	A	N	1-3	N	N	0	0
Elektr. Gästebuch	sL	?	0-5	8	A	N	1-3	N	N	0	?
Online-Tracking	L	5-9	5-9	0-3	A	N/J	7-9	?	J	?	?
Online-News	L	9	7-9	4-9	AM	N	5-9	N/J	N/J	?	?
Nachschlagewerk	L	1	2-9	4-9	AM	N	3-9	N/J	?	0	?
Bestell-Katalog	Ls	1-5	5-9	4-8	AM	N/J	7-9	N/J	J	5-9	4-9
Online-Banking	LS	3-9	6-9	0-3	A	J	9	J	J	5-9	4-9
Geschäftsanwendung	LS	5-9	5-9	?	AM	N/J	9	J	J	5-9	9
L (lesend), S (schreibend), A (alphanumerisch), M (multimedial) Numerische Werte: 0 (sehr gering) - 9 (sehr hoch), ? unbekannt/nicht möglich											

Tabelle 1: Anwendungsklassifikation

3. Arten der Web-Anbindung

Nachdem wir einen allgemeinen Kriterienkatalog für die Beurteilung von Web-basierten DB-Anwendungen erarbeitet und auf seiner Basis die vorgestellten Anwendungen charakterisiert haben, wollen wir nun im folgenden einen Überblick über die unterschiedlichen Anbindungen von Web-Anwendungen an DBSs geben und daraufhin die zugrundeliegenden Techniken kurz vorstellen. In den nachfol-

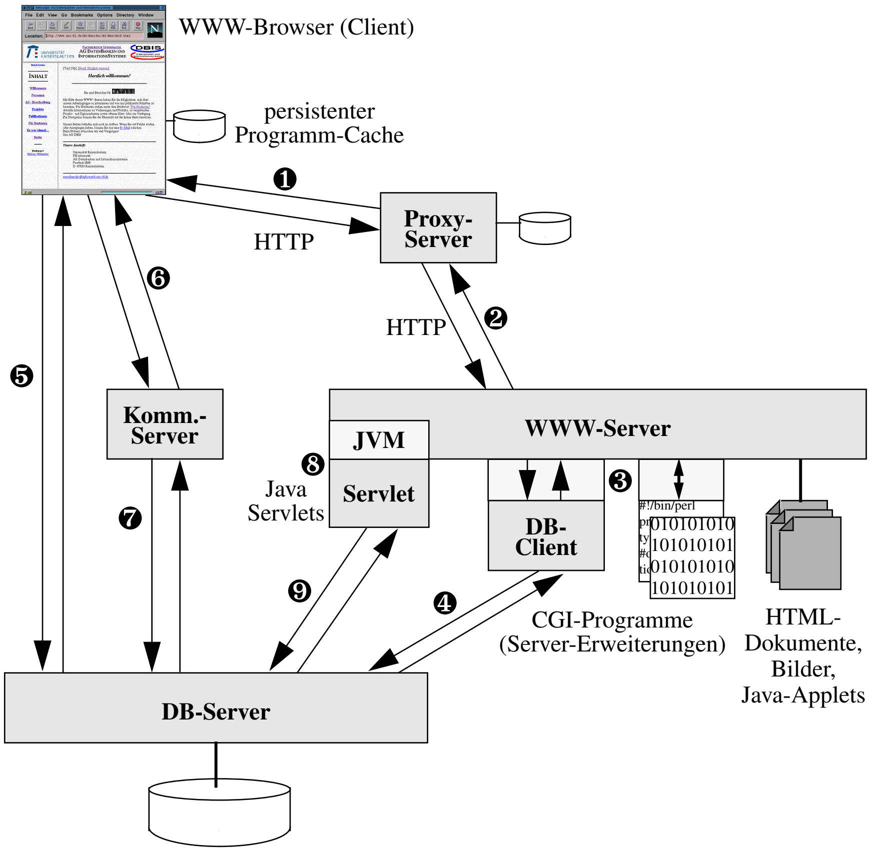

Abb. 1: Web-basierter DB-Zugriff

genden Kapiteln werden wir dann detailliert auf jede von ihnen eingehen, mögliche Realisierungsvarianten vorstellen und die jeweiligen Vor- und Nachteile erörtern.

3.1 Überblick

Zentrale Komponente für die Realisierung einer Web-basierten DB-Anwendung ist der WWW-Server (siehe Abb. 1). Er stellt zum einen die statischen HTML-Seiten inkl. eingebetteter Bilder etc. sowie Java-Applets zur Verfügung, die vom Web-Browser mittels HTTP, evtl. unter Nutzung eines oder mehrerer Proxy-Server[1], angefordert werden können (❶+❷). Der WWW-Server ist aber auch für den aus besonderen HTTP-Anfragen resultierenden Aufruf der CGI-Programme (❸, siehe Kapitel 3.2), von Server-Erweiterungen (❸ mit grauer Hinterlegung, siehe Kapitel 3.3) sowie von Java-*Servlets* (❽, siehe Kapitel 3.5) zuständig. Die dabei u. U. bei einer DB-Interaktion (❹ bzw. ❾) als Ergebnis erzeugten HTML-Seiten werden wiederum vom WWW-Server an den Browser weitergeleitet. Der DB-Server kann neben den eigentlichen Anwendungsdaten (Geschäftsdaten) auch Teile von oder fertige, statische HTML-Seiten verwalten.

Ein andere Möglichkeit zum Web-basierten DB-Zugriff besteht über ein im Browser ausgeführtes Java-Applet (siehe Kapitel 3.5). Es kann bei einer entsprechenden Systemkonfiguration entweder direkt (❺) oder über den Umweg eines Kommunikations-Servers (❻+❼) auf die DB zugreifen [23].

3.2 CGI-Programme

Waren das WWW und HTML in der Anfangszeit nur für den Zugriff und die statische Verknüpfung von Dokumenten ausgelegt, so stellte man schnell fest, daß auch der Zugriff auf und die kontext- bzw.

1.) Ein Proxy-Server puffert Ergebnisse (HTML-Dokumente, Bilder) einer HTTP-Anfrage, um so den Zugriff auf statische Informationen zu beschleunigen und das Netzwerk zu entlasten. Speziell markierte Dokumente sowie Ergebnisse von Anfragen, die auf ein dynamisch erzeugtes Dokument schließen lassen (CGI-Aufruf), werden nicht gepuffert.

parameterabhängige Erzeugung von „dynamischen“ Dokumenten wünschenswert war. Dieses wurde durch die Einführung des CGI [8] und der sog. HTML-Formulare ermöglicht.

Mit Hilfe des CGI kann der WWW-Server die vom Benutzer in einem HTML-Formular eingegebenen Parameter in einer definierten Weise an ein anderes Programm, ein CGI-Programm, weitergeben (❸). Der WWW-Server muß dazu in einem zusätzlichen Prozeß dieses Programm starten, übergibt dann die Parameter und wartet auf die Ergebnisübermittlung durch das CGI-Programm. Dieses fragt nach dem Starten die vom WWW-Server gesetzten und durch [8] spezifizierten Umgebungsvariablen sowie weitere Informationen ab. Daraufhin kann auf die Parameter zugegriffen werden und die eigentliche Programmabarbeitung beginnen, wobei für einen DB-Zugriff der DB-Server kontaktiert wird (❹). Während der Laufzeit des Programms ist nun ein vollständiges HTML-Dokument zu erzeugen, das an den wartenden WWW-Server zurückgegeben wird. Dieser reicht schließlich das Dokument als Ergebnis der HTTP-Anfrage zur Anzeige an den Web-Browser zurück.

3.3 Server-API

Ausgehend von der Tatsache, daß für die Ausführung von CGI-Programmen für jede entsprechende HTTP-Anfrage ein separater Prozeß gestartet werden muß, was zu langen Startzeiten führt, haben einige Hersteller von Web-Servern eigene, untereinander nicht kompatible Programmierschnittstellen (*Application Programming Interface*, API) für die Erweiterung ihres Produktes definiert[2]. Mit Hilfe dieser API ist es nun möglich, den WWW-Server um die Funktionalität der bisherigen CGI-Programme zu erweitern. Die implementierten Funktionen (SAF, *Server Application Function*) müssen als dynamische Programmbibliothek bereitgestellt werden. Diese wird dann beim Starten des jeweiligen Web-Server-Prozesses hinzugebunden. Ähnlich dem CGI kann der WWW-Server über entsprechende Konfiguration anhand des URL unterscheiden, ob ein normaler Dokumentenzugriff oder der Aufruf einer Erweiterungsfunktion zu erfolgen hat. Anders als beim CGI wird eine prozeßinterne Funktion aufgerufen (❸ mit grauem Übergang), so daß hierbei im Vergleich zu einem CGI-Programm ein erheblicher Geschwindigkeitsvorteil resultiert. Durch das Weiterlaufen des Web-Servers nach einer HTTP-Anfrage ist es zudem möglich, DB-Verbindungen ständig geöffnet zu haben, so daß die Startzeiten noch weiter verkürzt werden können.

Da, abgesehen von der Startgeschwindigkeit, keine wesentlichen Unterschiede zu einem CGI-Programm bzgl. des DB-Zugriffs existieren und Produkte meist in beiden Varianten zur Verfügung stehen, werden wir in den folgenden Kapiteln CGI-Programme und Server-API-Aufrufe gemeinsam behandeln.

3.4 Server Side Includes

Basierend auf den ursprünglich im NCSA-Web-Server eingeführten sogenannten *Server Side Includes* (SSI, [34]) haben die meisten Hersteller von Web-Servern ihre Produkte um entsprechende Funktionalität ergänzt und z. T. um zusätzliche Kontrollelemente erweitert [1, 21], teilweise sogar um DB-Kommandos [20, 25]. SSIs sind dabei um spezielle Steuerungsbefehle in Syntax von HTML-Kommentaren angereicherte HTML-Dokumente, die vom Web-Server bei einem Dokumentenzugriff dynamisch ausgewertet werden. Auf diese Weise ist es möglich, z. B. die aktuelle Uhrzeit oder aktuelle Informationen wie Mitteilungen in Teile einer Web-Seite einfließen zu lassen.

Innerhalb dieser Dokumente ist es aber auch möglich, (UNIX-)Kommandos bzw. Anwendungen aufzurufen bzw. CGI-Programme abzuarbeiten, wobei jeweils in Analogie zur normalen CGI-Verarbeitung ein neuer Prozeß erzeugt wird. An diese werden die vorher an das SSI-Dokument übermittelten, benutzerspezifizierten Variablen sowie die Umgebungsvariablen weitergegeben, so daß SSI-Dokumente auch zur regulären Verarbeitung von HTML-Formularen dienen können. Auf diese Art und Weise ist es z. B. möglich, in einem anderen Programm spezifizierte DB-Anweisungen auszuführen und die Ergebnisse in ein, abgesehen von den Steuerungsbefehlen, normales HTML-Dokument einzubetten. Da SSI-Dokumente für den DB-Zugriff in der Regel auf die Mechanismen von CGI und Server-Erweiterungen zurückgreifen müssen, wollen wir sie im folgenden nicht mehr explizit behandeln.

2.) So gibt es u. a. die NSAPI (Netscape Server API, [27]) von Netscape sowie die ISAPI (Internet Server API, [26]) von Microsoft.

Eine besondere Form der SSIs stellen die *Active Server Pages* (ASPs, [25]) von Microsoft dar, die die Anreicherung von Dokumenten um VBScript- und JScript-/JavaScript-Kommandos gestatten. Auch hier wird ein Dokument vor der Übertragung zum Browser vom Web-Server analysiert und die enthaltenen Kommandos abgearbeitet. ASPs stellen im Vergleich zu SSIs, allein schon durch die Mächtigkeit von VBScript und JScript, eine große Funktionalität zur Verfügung, die über den Zugriff auf Formular- und Umgebungsvariablen, die Realisierung von Zuständen bis hin zum DB-Zugriff reicht. Vom Prinzip her sind ASPs mit den SSIs vergleichbar, so daß wir im weiteren Verlauf nicht genauer auf sie eingehen werden.

3.5 Java-basierte Verfahren

Mit der Veröffentlichung der Sprache Java [13] durch SUN Microsystems im Jahr 1995 begann eine neue Ära für die Realisierung von Web-basierten Anwendungen. Durch das Konzept des *Java Bytecode* mit dem Ziel der Plattformunabhängigkeit ist es möglich, ein auf einem System in einer höheren Programmiersprache entwickeltes Programm in seiner kompilierten Fassung auf ein anderes System zu übertragen und dort auf einer *Java Virtual Machine* (JVM) auszuführen. Eine JVM entspricht dabei einem Interpreter, der die Befehle des plattformunabhängigen Java Bytecode auf die Maschinenbefehle des jeweiligen Systems abbildet. Bei den Java-Programmen wird zwischen *Java Applications* und *Java Applets* unterschieden, wobei mittlerweile ein weiterer Begriff geprägt wurde: *Java Servlets*. Java Applications werden wie ganz „normale", in einer anderen Programmiersprache entwickelte Programme installiert und unterscheiden sich mit Ausnahme der zusätzlich benötigten JVM nicht von diesen. Sie eignen sich also im Gegensatz zu Applets und Servlets nicht direkt für Web-Anwendungen, können aber z. B. für die Implementierung von CGI-Programmen oder Kommunikations-Servern eingesetzt werden. Daher wollen wir im folgenden nur Applets und Servlets diskutieren.

Applets

Java Applets werden ähnlich wie HTML-Dokumente, in die sie eingebettet sind, auf einem WWW-Server bereitgestellt und vor der Programmausführung als Teil einer HTML-Seite (wie ein Bild) in einen Java-fähigen Web-Browser eingeladen (Abb. 1, ❶+❷). Nach dem Laden wird das Applet der im Browser notwendigerweise vorhandenen JVM zur Ausführung übergeben. Da mit einem Applet, u. U. sogar ohne Wissen des Anwenders, ein Programm auf einen anderen Rechner geladen wird, unterliegen Applets bei ihrer Ausführung verschiedenen Restriktionen, deren Einhaltung durch die JVM überwacht wird. So ist es z. B. nur möglich, Netzverbindungen zu dem Rechner aufzubauen, von dem das Applet geladen wurde. Zudem ist der Zugriff auf lokale Ressourcen nicht erlaubt, um ein Ausspähen zu verhindern.

Durch die beim Übersetzen erfolgende Überführung jedes Interface und jeder Klasse in je eine Class-Datei muß beim Laden eines Applets über das Netz daher für jede von der Anwendung benötigte Klasse bzw. Interface die entsprechende Class-Datei angefordert werden[3]. Um den dazu jeweils erforderlichen (teuren) Aufbau einer Netzwerkverbindung zu vermeiden, wurden mit der Version 1.1 von Java die sog. JAR-Dateien (*Java ARchive*) eingeführt. Sie können mehrere oder alle Class-Dateien eines Applets enthalten und werden anstelle der einzelnen Dateien geladen, wodurch sich die Applet-Ladezeiten deutlich reduzieren.

Eine weitere Neuerung ab den Versionen 1.1 und 1.2 des *Java Development Kits* (JDK) ist die Ergänzung des Sicherheitskonzeptes. Beschränkten die Sicherheitsauflagen den Einsatz von Applets, z. B. für unternehmensinterne Anwendungen, so können nun vom Benutzer einem signierten Applet (*Signed Applet)* bestimmte Rechte gewährt werden. Ein signiertes Applet ist dabei ein in einer JAR-Datei zusammengefaßtes und mit einer digitalen Signatur versehenes Applet. Durch die digitale Unterschrift soll sichergestellt werden, daß das Applet bzw. die Class-Dateien seit der Erstellung des Archivs nicht verändert wurden und vom Unterschreiber stammen. Eine Verfälschung oder ein Austausch nach der Erzeugung der JAR-Datei, z. B. auf dem WWW-Server durch einen nicht vertrauenswürdigen Mitarbeiter oder auf dem Übertragungsweg durch einen „Angreifer", wird so verhindert. Signierte Applets haben im Vergleich zu normalen Applets und JAR-Dateien den weiteren Vorteil,

3.) Nur das eigentliche Laufzeitsystem ist vorhanden, d. h. die in der jeweiligen Java-Sprachdefinition enthaltenen Klassen.

daß sie persistent auf dem Rechner des Anwenders mittels einer expliziten Installation gespeichert werden können (siehe Abb. 1, persistenter Programm-Cache). Hierdurch ist ein Neuladen über das Netz vor der Ausführung des Applets nicht mehr notwendig, und die Ladezeiten und Startkosten werden, insbesondere bei großen Applets, drastisch reduziert [17].

Da für die Realisierung von Applets prinzipiell der volle Sprachumfang von Java zur Verfügung steht, ergeben sich zahlreiche Möglichkeiten zur Verwirklichung Web- bzw. Java-basierter DB-Anwendungen. Auf sie werden wir in Kapitel 5 genauer eingehen und sie bzgl. ihrer Vor- und Nachteile analysieren.

Servlets

Die *Servlet-API* wurde von SUN Microsystems zunächst als Pendant zu den Server-Erweiterungs-APIs von Netscape und Microsoft für den eigenen, komplett in Java implementierten Web-Server [21] spezifiziert. Durch ihre Aufnahme in das JDK 1.2 sowie durch die Unterstützung von anderen Web-Server-Herstellern gewinnt sie an besonderer Bedeutung [18]. Denn nun ist es auf einfache Weise möglich, plattform- und herstellerunabhängig Erweiterungen von Web-Servern zu entwickeln. Voraussetzung für die Unterstützung von Servlets ist die Integration einer JVM in den Web-Server (❸) bzw. die Bereitstellung und Kooperation mit einem entsprechenden Zusatzprozeß. Genau wie bei den C-basierten Server-APIs werden ankommende HTTP-Anfragen (❷), für welche die vorher festgelegten Bedingungen (Pfadausdrücke) zutreffen, an die jeweiligen Erweiterungsmodule übergeben.

Ein Vorteil von Servlets resultiert aus den Möglichkeiten des dynamischen Bindens, d. h. dem Java-Klassenlader. Hierdurch ist es mittels Konfiguration noch zur Laufzeit möglich, Module hinzuzufügen oder zu entfernen. Ein Neustart des Web-Servers ist nicht notwendig und somit ein durchgängiger Server-Betrieb möglich.

4. HTTP-basierte Ansätze

Nachdem wir im vorangegangenen Kapitel die prinzipiell möglichen Techniken zur Realisierung Web-basierter DB-Applikationen vorgestellt haben, wollen wir nun die diversen Ansätze für (einfache) Anwendungen auf Basis von HTTP vorstellen und ihre jeweiligen Stärken und Schwächen beleuchten. Unter HTTP-basierten Ansätzen fassen wir dabei die „klassischen“ CGI-Programme, Lösungen unter Nutzung der Server-Erweiterungs-APIs inkl. der Java-Servlets sowie die SSIs zusammen. Beginnen wollen wir mit einer Analyse der allgemeinen, diesen Verfahren inhärenten Vor- und Nachteilen bzgl. der DB-Anbindung. Anschließend betrachten wir die unterschiedlichen Ansätze, um aus DB-Anfragen dynamisch HTML-Dokumente zu erzeugen. Im Anschluß daran werden wir auf die möglichen Architekturvarianten eingehen. Ein weiteres Thema wird dann der Speicherungsort der Makro- und HTML-Dateien sein, bevor wir mit einer Diskussion der Lösungsansätze für die Realisierung von Zuständen im Web schließen.

4.1 Allgemeines

Während sich die einzelnen, auf CGI oder Server-API basierenden Werkzeuge für die Web-Anbindung von DBs bzgl. der Spezifikation von SQL-Befehlen und ihrer Architektur unterscheiden, so besitzen sie jedoch aufgrund derselben zugrundeliegenden Technik gemeinsame Vor- und Nachteile, die wir im folgenden erläutern wollen.

Zustandslosigkeit

Das für die Kommunikation im Web verwendete HTTP ist zustandslos, d. h., eine Kommunikationsverbindung zwischen Web-Browser und WWW-Server besteht nur für die Dauer der HTTP-Anfragebearbeitung. Für die mit Hilfe von CGI bzw. der Server-API realisierten DB-Clients bedeutet dies, daß Transaktionen ebenfalls nicht länger dauern können, da sie nur während dieser Zeitspanne aktiv sind. Die kurze Verbindungsdauer bedingt zudem ein Neuöffnen einer DB-Verbindung für jede HTTP-Anfrage, was sich auf die Programmgesamtlaufzeit auswirkt. Bei einem CGI-Programm kommt noch die aus einem Prozeßneustart resultierende Wartezeit hinzu.

Möchte man diese Nachteile umgehen, um z. B. kontextabhängige Mehrschritt-Arbeitsgänge wie das Abholen eines Ergebnisses in Blöcken von 50 Tupeln zu realisieren oder den Verbindungsneuaufbau zu vermeiden, so bedarf es zusätzlicher Techniken, die wir im Rahmen der Architekturmodelle (siehe Kapitel 4.3) sowie der Realisierung von Zuständen besprechen (siehe Kapitel 4.5).

Sicherheit

Bei der Sicherheit von CGI- und Server-API-basierten Lösungen sind die Übertragungssicherheit, d. h. die Abhörsicherheit, und der Zugriffsschutz des Web-Servers zu unterscheiden. Um einen eingeschränkten Zugriff auf bestimmte Inhalte (CGI-Programme eingeschlossen) eines WWW-Servers zu realisieren, können die sich ergänzenden Verfahren der HTTP-Authentisierung sowie das Einschränken des Verbindungsrechtes auf bestimmte Internet-Adressen bzw. Domains angewendet werden[4]. Bei letzteren kann über eine entsprechende Konfiguration des Web-Servers festgelegt werden, von welchen Internet-Adressen aus auf den Web-Server oder bestimmte Unterverzeichnisse zugegriffen werden darf bzw. von welchen Adressen aus dies generell verboten ist (*allow/deny*), um so die Zugriffsmöglichkeit auf eine Abteilung oder ein Unternehmen zu begrenzen und einen Intranet-Server zu realisieren. Damit indirekt verbunden ist auch die Einschränkung eines potentiellen Web-basierten Angriffs auf DB-Server.

Mit dem Verfahren der HTTP-Authentisierung kann der Zugriff auf Unterverzeichnisse oder den ganzen Server auf bestimmte Benutzer eingeschränkt werden [11]. Greift der Benutzer auf geschützte Bereiche (*Domain*) eines Web-Servers zu, so muß er sich über Benutzernamen und Paßwort authentisieren. Mittels einer Abbildung des Web-Benutzers auf eine lokale ID kann der HTTP-Authentisierungsmechanismus genutzt werden, um eine DB-Verbindung unter der Kennung des lokalen bzw. entfernten Benutzers aufzubauen.

Hierfür ist jedoch eine explizite Unterstützung des Web-Werkzeuges oder der Einsatz anderer Hilfsmittel wie z. B. suEXEC [2] oder CGIwrap [29] erforderlich, da „normale“ WWW-Server diesen ID-Wechsel nicht unterstützen. Die letztgenannten Programme dienen dazu, ein CGI-Programm unter einer anderen ID ausführen zu lassen, so daß z. B. auch ein DB-Client unter einer besonderen, mit Restriktionen belegten Benutzer-ID ausgeführt werden kann.

Da die Übertragung von Paßwort-Informationen und anderer vertraulicher Daten unter Nutzung des HTTP nicht sicher ist, bedarf es des Einsatz eines Verschlüsselungsverfahrens. Existierten anfangs SHTTP (*Secure HTTP*, [32]) und SSL (*Secure Socket Layer*, [12]), so hat sich wegen seiner Praktikabilität SSL durchgesetzt und ist heute de facto das alleinige Verfahren. SSL basiert auf dem symmetrischen RSA-Verfahren und benötigt auf Seiten des WWW-Servers ein elektronisches Zertifikat. Ist der Aussteller dem Browser nicht bekannt, z. B. weil sich der Betreiber des WWW-Servers selbst zertifiziert hat, so wird der Benutzer in der Regel gefragt, ob er dem Anbieter trauen möchte. Lehnt er dies ab, wird keine verschlüsselte Verbindung zum WWW-Server aufgebaut. Entscheidender Nachteil von SSL sind die aufgrund der US-amerikanischen Exportrestriktionen kurzen Schlüssellängen, die den Einsatz für Anwendungen mit besonderen Sicherheitsanforderungen, z. B. Banking, außerhalb der USA unbrauchbar machen. Deshalb sind Ergänzungen der im Browser vorhandenen Sicherheitsmaßnahmen nötig, um solche Anwendungen zu realisieren (siehe Kapitel 5.5).

4.2 SQL/HTML-Integration

Bei Web-basierten DB-Anwendungen spielen die Ausführungsgeschwindigkeit und die dabei benötigten Systemressourcen eine wichtige Rolle. Vor der Ausführung ist aber die Anwendung erst zu erstellen, d. h. die DB-Anfragen und die zurückzuliefernden HTML-Dokumente zu spezifizieren. Das „Wie“ dieser Spezifikation wollen wir in diesem Abschnitt betrachten und die möglichen Varianten bez. ihrer Vor- und Nachteile analysieren. Eine Übersicht der erarbeiteten Ergebnisse findet sich in Tabelle 2. Beginnen wollen wir mit dem Vorläufer der heutigen Verfahren, der für besondere Lösungen und kleinere Anwendungen noch zum Zuge kommt: die Programmierung.

4.) Die Verfahren auf Betriebssystemebene, z. B. *Firewalls*, werden nicht betrachtet, siehe z. B. [6].

Programmierung

Da in der Anfangszeit von WWW und CGI keine entsprechenden Werkzeuge zur Anbindung von DBSs von Seiten der DBVS-Hersteller (Datenbankverwaltungssystem) und Drittanbieter zur Verfügung standen, mußte man, wollte man eine Web-DB-Anwendung realisieren, die benötigte Funktionalität selbst programmieren. Auch heute wird diese Lösung noch eingesetzt. Dies ist zum einen der Fall, weil für viele, vor allem kleine, Anwendungen sich die Anschaffung von entsprechenden Werkzeugen nicht lohnt, z. B. für die Anbindung einer Adreß- oder Feedback-DB oder auch die Realisierung eines elektronischen Gästebuchs. Zum anderen kommt eine selbstprogrammierte Lösung dann zum Einsatz, wenn besondere Eigenschaften gefordert sind, die von den verfügbaren Werkzeugen nicht (ausreichend) bereitgestellt werden können, beispielsweise, wenn der gleichzeitige Zugriff auf mehrere DBs gefordert ist.

```
#!/bin/perl
# Msql-Package laden:
use Msql;
# Seitenkopf ausgeben:
print"Content-type: text/html\n\n";
# [...]
# Verbindung mit dem DB-Server herstellen:
$testdb = Msql->connect;
# Datenbank auswaehlen:
$testdb->selectdb(„bookmarks");
# Datenbankanfrage stellen:
$sth = $testdb->query("select name,url from
   bookmarks where name LIKE 'Loe%' order by name");
# Resultat ausgeben:
print"<TABLE BORDER=1>\n";
print"<TR>\n<TH>Name<TH>URL</TR>";
$rows = $sth->numrows;
while ($rows>0)
{
   @sqlrow = $sth->fetchrow;
   print"<tr><td>",@sqlrow[0],"</TD><td><A HREF=\"",
       @sqlrow[1],"\">",@sqlrow[1],"</A></td></TR>\n";
   $rows--;
}
print"</TABLE>\n";
# Seitenende ausgeben
# [...]
```

Abb. 2: CGI-Programmierung (mSQL)

Für die Programmierung einer Web-Anbindung einer DB werden dabei vor allem Sprachen wie Perl (siehe Abb. 2) und C eingesetzt. Während Perl aufgrund der einer Skriptsprache inhärenten Eigenschaften wie der schnellen Erstell- und Änderbarkeit von entsprechenden DB-Anbindungen eingesetzt und geeignete Funktionsbibliotheken von vielen Herstellern bereitgestellt werden, ist C dagegen die „Standardsprache“ für DB-Anwendungsprogrammierung und wird von allen DBVS unterstützt.

Die Vorteile einer selbstentwickelten, ausprogrammierten Lösung liegen in der Möglichkeit, das jeweilige Programm direkt auf die spezifischen Anforderungen zuschneiden zu können. Hierdurch kann im Zweifelsfall gewährleistet werden, daß eine optimale Lösung zum Einsatz kommt. Allerdings sind mit diesem Vorteil zahlreiche Nachteile verbunden. So ist, entwickelt man aus Gründen der Laufzeitgeschwindigkeit in C, für Programmanpassungen ein Neu-Übersetzen notwendig. Für kleinere Projekte lohnt sich meist nicht die Entwicklung einer mehrschichtigen und leistungsfähigeren, dafür aber wesentlich komplexeren Lösung. Zudem sind selbstprogrammierte Lösungen (anfangs) in der Regel fehlerhafter als „Standardwerkzeuge“ für die Web-Anbindung. Eigene Lösungen haben jedoch oft den Vorteil einer wesentlich geringeren Programmgröße, so daß bei Nutzung des CGI deutlich kürzere Programmstartzeiten erzielt werden können und eine kürzere Gesamtausführungszeit resultiert.

```
<HTML><HEAD><TITLE>Bookmark-DB</TITLE></HEAD>
<BODY>
<H1>Bookmark-DB - Anfrageergebnis</H1>
<!-- Eingabeparameter interner Variablen zuweisen -->
<?MIVAR NAME=iname DEFAULT=Loe>$iname<?/MIVAR>
<!-- Tabellenkopf ausgeben -->
<TABLE><TR><TH>Name</TH><TH>URL</TH></TR>
<!-- Anfrage spezifizieren -->
<?MISQL query="select name,url from
   bookmarks where name LIKE '$iname%' order by name;">
<!-- Ergebnis in Tabellenform ausgeben -->
<TR><TD><A HREF="$2">$1</A></TD><TD>$2</TD></TR>
<?/MISQL>
</TABLE></BODY></HTML>
```

Abb. 3: HTML/SQL-Integration (Informix Web DataBlade, [16])

Integration von HTML und SQL

Beginnend mit der Entwicklung von unterstützenden Werkzeugen zur Realisierung Web-basierter DB-Anwendungen wurde HTML von den unterschiedlichen Herstellern um SQL-Befehle ergänzt.

Dies bedeutet, daß in das „HTML-Skelett", welches das spätere Ergebnisdokument festlegt, SQL-Befehle in Verbindung mit entsprechenden Kommandos zur Variablenverarbeitung eingebettet werden (siehe Abb. 3).

Die Integration von SQL-Anweisungen in ein HTML-Dokument hat den Vorteil, daß direkt erkennbar ist, welcher DB-Befehl wann abgearbeitet wird bzw. wie die jeweilige HTML-Ausgabe zustandekommt. Nachteilig kann sich dies jedoch bei großen HTML-Dokumenten und fehlender Strukturierung auswirken, da dann schnell die Übersichtlichkeit verlorengeht. Durch das Ignorieren unbekannter HTML-Tags durch Browser und auch HTML-Editoren können die um SQL-Befehle angereicherten Dokumente auch mit entsprechenden Editoren erstellt werden.

Makro-Programmierung

```
# Datenbank festlegen
%DEFINE{
   DATABASE="bookmarks"
%}
# Anfrage als Funktion spezifizieren
%FUNCTION(DTW_SQL) bquery() {
   select name,url from
   bookmarks where name LIKE 'Loe%' order by name;
   # Ergebnis erfordert eine besondere Ausgabe
   %REPORT{
       <TABLE><TR><TH>Name</TH><TH>URL</TH></TR>
   # Ausgabeformat der Ergebniszeilen festlegen
   %ROW{
       <TR><TD><A HREF="$(V2)">$(V1)</A></TD>
       <TD>$(V2)</TD></TR>   %}
   </TABLE>   %}
%}
# Ausgabesektion beginnt hier
%HTML(REPORT){
<HTML><HEAD><TITLE>Bookmark-DB</TITLE></HEAD>
<BODY>
<H1>Bookmark-DB - Anfrageergebnis</H1>
<!-- Ergebnis ausgeben -->
@bquery()
</BODY></HTML>
%}
```

Abb. 4: Makro-Programmierung (IBM Net.Data, [15])

Ein anderes Verfahren, das ebenfalls häufig zum Einsatz kommt, ist das der sogenannten Makro-Programmierung. Hierbei sind das HTML-Skelett und die DB-Anweisungen voneinander getrennt. Über Platzhalter wird die Position der entsprechenden Anfrageergebnisse festgelegt. Bei der Makro-Programmierung existieren sowohl Ansätze, bei denen Makros und HTML-Skelett in einer Datei in separaten Bereichen gespeichert werden (siehe Abb. 4), als auch Lösungen, bei welchen Makros und die HTML-Dokumentvorlage auf zwei separate Dateien aufgetrennt werden. Letzteres hat den Vorteil, daß das HTML-Skelett mit einem Editor erstellt werden kann. Zudem ist eine Wiederverwendung der spezifizierten SQL-Befehle möglich. So können z. B. mehrere HTML-Dokumente auf einer einzigen Makrodatei basieren, um eine mehrsprachige, länderabhängige Ergebnisausgabe zu realisieren. Durch eine Änderung an den SQL-Befehlen können dann sofort alle Dokumente aktualisiert werden. Nachteilig beim Einsatz zweier Dateien ist, daß nun statt einem zwei Dateizugriffe durchgeführt werden müssen, was die Verarbeitungsgeschwindigkeit u. U. senken kann.

	Programmierung	HTML/SQL-Integration	Makro-Programmierung
Vorteile	• schnelles und evtl. kleines Programm • optimal auf jeweilige Anforderungen angepaßt	• Lesbarkeit durch Plazierung von SQL-Befehlen an den späteren Ort der Daten • Erstellung und Wartung u.U. mit HTML-Editor möglich • Zugriff auf nur eine Datei	• HTML-Datei über HTML-Editor wartbar (2) • übersichtliche Spezifikation aller SQL-Anfragen • Mehrfachverwendung von SQL-Anfragen möglich (2) • schnelleres Parsen möglich, da deutlich kleinere Datei (2)
Nachteile	• unflexibel, evtl. mit Neuübersetzung • für jedes Problem neues Programm	• schnell unübersichtlich wegen der Mischung von HTML und SQL	• schlechte Überschaubarkeit durch Verteilung auf zwei Dateien/Bereiche • zwei bzw. mehrere Dateizugriffe notwendig (2)

Tabelle 2: Vergleich der Spezifikationsverfahren

Allgemein zeichnen sich die Verfahren der Makro-Programmierung dadurch aus, daß durch die Aufteilung auf Makro und eigentliches HTML-Dokument die Lesbarkeit erhöht wird, da auf einen Blick

z. B. alle SQL-Befehle erkennbar sind. Andererseits muß aber, um die Position eines Anfrageergebnisses im HTML-Skelett feststellen zu können, jeweils zwischen zwei Sektionen bzw. Dateien gewechselt werden. Tabelle 2 faßt die Ergebnisse noch einmal kompakt zusammen. Die Vor- und Nachteile mit einer nachgestellten (2) beziehen sich auf Verfahren mit zwei getrennten Dateien.

4.3 Architekturen

Orthogonal zu den verschiedenen Formen der Spezifikation von SQL-Befehlen ist die potentielle Architektur der CGI-basierten Werkzeuge. Unter Architektur verstehen wir den logischen Aufbau (Komponenten) der Werkzeuge und ihre Verteilung auf entsprechende Prozesse. Daher wollen wir im folgenden zunächst mit einem Überblick über die prinzipiell für die Verarbeitung benötigten Komponenten beginnen. Anschließend betrachten wir einstufige, dann zweistufige Verfahren. Zudem ist es möglich, den DB-Server um einen Makroprozessor zu erweitern.

Komponenten

Die einzelnen Werkzeuge für die Realisierung Web-basierter DB-Anwendungen unterscheiden sich zwar in ihrer Funktionalität und ihrer Architektur, doch in ihrem Aufbau beinhalten sie prinzipiell gleiche Komponenten, die wir im folgenden besprechen wollen.

Um die vom Anwendungsentwickler spezifizierten Makros bzw. die um SQL-Befehle erweiterten HTML-Seiten verarbeiten zu können, besitzen alle Werkzeuge als zentrale Komponente einen ***Prozessor*** (**P**, siehe auch Abb. 5). Er ist für die Extraktion der für das gesamte Werkzeug relevanten Informationen aus den Makro- oder HTML-Dateien sowie die daraufhin erfolgende Interaktion mit der ***DB-Kommunikationskomponente*** (**DBC**) zuständig. Sie wird benötigt, um die Verbindung mit dem DB-Server aufzubauen und die spezifizierten DB-Befehle abzuarbeiten. Zur DB-Kommunikation können dabei entweder aus Geschwindigkeitsgründen die herstellereigenen Bibliotheken und API-Funktionen verwendet oder aus Gründen der Portabilität und Herstellerunabhängigkeit ODBC (*Open DataBase Connectivity*) bzw. CLI (*Call Level Interface*) eingesetzt werden. Im Falle eines ausprogrammierten Systems entspricht die eigentliche Anwendungsfunktionalität dem Prozessor.

Ebenso wie mit dem DB-Server wird für den Datenaustausch mit dem Web-Server eine ***Web-Server-Kommunikationskomponente*** (WSC) benötigt. Sie trägt dafür Sorge, daß die vom Anwender spezifizierten und vom Web-Server weitergegebenen Parameter die Prozessor-Komponente erreichen und daß das letztlich vom Werkzeug als Ergebnis gelieferte HTML-Dokument zum Web-Server gelangt. Da die hier besprochenen Verfahren unterschiedliche Aufrufmechanismen verwenden, kann diese Kommunikationskomponente für die Erfüllung ihrer Aufgabe auf bestimmte Parameter innerhalb der Server-API bzw. Umgebungsvariablen im Falle des CGI zurückgreifen.

Kommen zweistufige Verfahren zum Einsatz, so werden für den Datenaustausch zwischen den unterschiedlichen Prozessen noch ***Interkomponenten-Kommunikationsmodule*** (ICC) benötigt. Sie können auf der direkten Programmierung von Sockets bis hin zur CORBA-basierten Kommunikation auf Basis von IIOP beruhen.

Einstufige Lösung

Bedingt durch die im Vergleich zu zweistufigen Lösungen einfachere Entwicklung sind einstufige Architekturansätze das vorherrschende Verfahren für die Realisierung Web-basierter DB-Anwendungen. Allen diesen Verfahren gemein ist die direkte (logische) DB-Verbindung, d. h., gleich, ob gesondertes CGI-Programm oder über Server-API integriertes Funktionsmodul, wird nach dem Aufruf und der Parameterweitergabe durch den Web-Server mittels WSC (siehe Abb. 5a) eine DB-Verbindung (DBC) etabliert und die eigentliche Verarbeitung durch den Prozessor begonnen. Jedoch ist der Verbindungsaufbau zum DB-Server relativ teuer, da auf Seiten des DB-Servers nach der Authentisierung die entsprechenden Rechte des Benutzers, unter dessen Kennzeichen das CGI-Programm bzw. die Server-Erweiterung läuft, überprüft werden muß. Hierdurch und bei CGI-Programmen durch den Start eines eigenen Prozesses bedingt sind einstufige Lösungen langsamer bzgl. der Startzeiten. Dies bedeutet, daß bis zum eigentlichen Verarbeitungsbeginn relativ viele Ressourcen verbraucht werden. Daher sind die einstufigen Verfahren auch nur für wenig frequentierte Anwendungen, wie z. B. eine Adreßdatenbank oder ein Gästebuch, sowie u. U. Einschritt-Vorgänge wie das Online-Tracking geeignet.

Für diese besitzen sie jedoch eine Reihe von Vorteilen. So werden in den Zeiten der Nichtnutzung keine oder nur sehr wenige Ressourcen benötigt, da das CGI-Programm nur bei Bedarf gestartet und Server-Erweiterungen gegebenenfalls vor der ersten Nutzung geladen werden. Auch ist die Installation und Konfiguration einstufiger Lösungen in der Regel deutlich einfacher als die von zweistufigen.

Zweistufige Lösung

Anders als bei den einstufigen Lösungen dient das startende CGI-Programm bzw. die Server-Erweiterung nur zur Kommunikation (mittels ICC) mit dem eigentlichen Verarbeitungsprozeß (P und DBC, siehe Abb. 5b). Nach dem Starten des CGI-Programms, das aufgrund der nicht benötigten Komponenten P und DBC eine deutlich kleinere Programmgröße aufweist, verbindet sich dieses mit einem Prozeß aus dem Pool der ständig verfügbaren Verarbeitungsprozesse und überträgt die vom Web-Server an das WSC-Modul übermittelten Parameter zu diesem Prozeß. Anschließend kann nun dieser die HTML- bzw. Makrodatei auswerten und die darin spezifizierten DB-Anweisungen ausführen. Die erhaltenen DB-Ergebnisse werden nach HTML umgewandelt und das gesamte HTML-Dokument an das CGI-Programm als Ergebnis zurückgegeben. Dieses leitet die HTML-Seite direkt an den Web-Server weiter, welcher wiederum das Dokument zum anfragenden Browser überträgt.

Die Vorteile der zweistufigen Lösungen liegen besonders in den potentiell kürzeren Antwortzeiten und der besseren Lastverteilung. Erstere werden erreicht, da die Verarbeitungsprozesse ständig als Hintergrundprozeß laufen und somit DB-Verbindungen geöffnet halten können. Daher kann der teure Verbindungsaufbau inklusive der Authentisierung entfallen. Zudem können von diesen Prozessen häufig verwendete HTML-/Makrodateien gepuffert werden, was die Verarbeitung noch einmal beschleunigt. Ein weiterer Einfluß auf die gesamte Ausführungsgeschwindigkeit kommt bei CGI-basierten Werkzeugen durch die deutlich kleineren Programme (s. o.), so daß die Startzeiten verkürzt werden können. Eine Lastverteilung kann durch die Verlagerung der Verarbeitungsprozesse auf eine dritte Maschine erreicht werden, so daß jeweils Web-Server, Auswertelogik und DB-Server auf getrennten Rechnern laufen können. Hierdurch eignet sich eine zweistufige Lösung insbesondere für „große" Web-basierte Anwendungssysteme, da zum einen viele gleichzeitig startende, große CGI-Programme vermieden werden und zum anderen eine Lastverteilung auf mehrere Rechner erreicht werden kann. Ein weiterer Vorteil für die Realisierung von Anwendungssystemen resultiert ebenfalls aus den ständig verfügbaren Verarbeitungsprozessen. Durch ihre DB-Verbindungen und evtl. Transaktionen auch

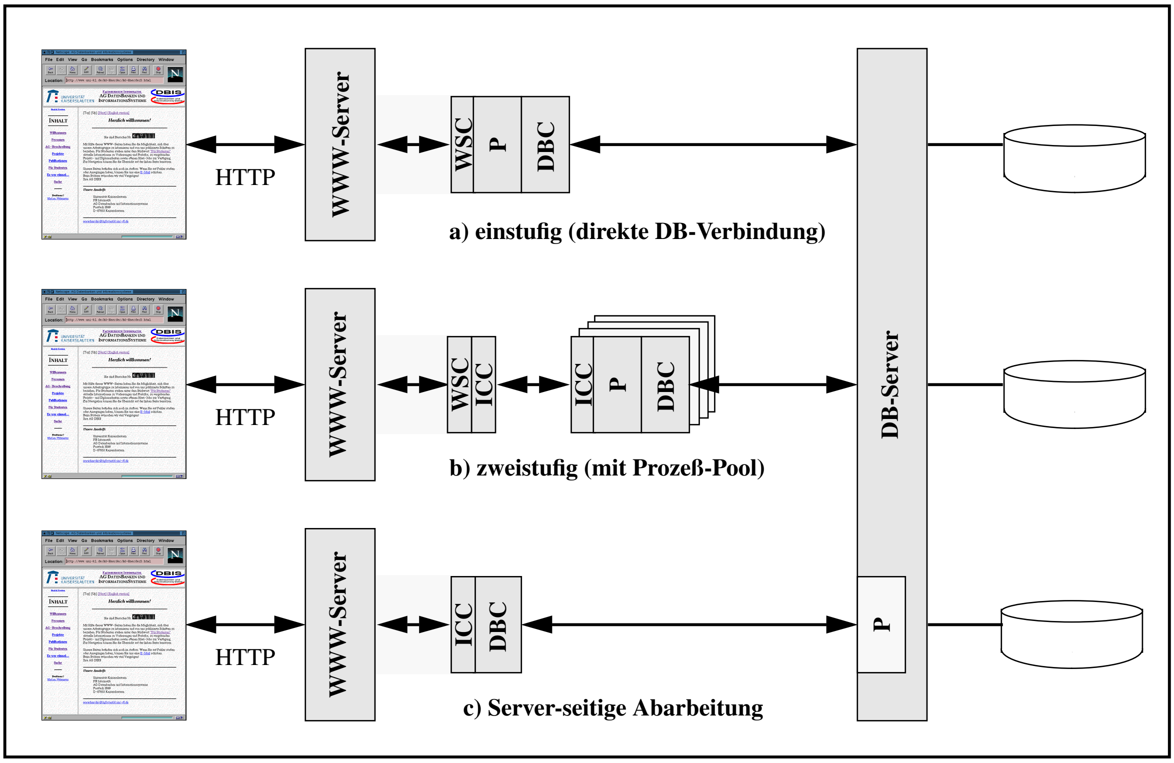

Abb. 5: Architekturen

über das eigentliche Ende der CGI-Verarbeitung hinaus ist es nun möglich, transaktionsorientierte Mehrschritt-Abläufe (“lange” Transaktionen) zu realisieren (siehe auch Kapitel 4.5).

Zweistufige Lösungen zeigen jedoch bei „kleinen“ Anwendungssystemen ihre Nachteile, da bedingt durch den ständig präsenten minimalen Pool an Verarbeitungsprozessen auch in Zeiten der Nichtnutzung Systemressourcen benötigt werden. Ein weiterer Nachteil ist die aus der zusätzlichen Schicht resultierende deutlich höhere Komplexität der Werkzeuge. So sind, sieht man von der schwierigeren Entwicklung für den Hersteller ab, auch die Installation und Konfiguration komplexer, da nun das richtige Zusammenspiel der Komponenten Web-Server/CGI-Programm, CGI-Programm/Prozeß-Pool und Verarbeitungsprozeß/DB-Server einzustellen ist. Außerdem ist auch die Fehlersuche in den auf dieser Basis entwickelten Web-Anwendungen aufwendiger.

Server-seitige Funktionsabarbeitung

Durch die Erweiterungsfähigkeit objekt-relationaler DBVS (ORDBVS), insbesondere durch die benutzerdefinierten Funktionen (*User Defined Functions*, UDFs) ist es möglich geworden, in einer höheren Programmiersprache erstellte Funktionen direkt im DB-Server auszuführen. Unter Nutzung des UDF-Prinzips kann nun auch die Funktionalität des Prozessors in den DB-Server verlagert werden (siehe Abb. 5c). Wesentlicher Vorteil dieser Lösungsvariante ist die verringerte Kommunikation, da lediglich der DB-Server-seitige Aufruf der UDF notwendig ist und als Ergebnis die fertige HTML-Seite in Form eines Bytestrings an das DBC-Modul zurückgegeben wird.

Werkzeuge mit server-seitiger Funktionsabarbeitung können zwar prinzipiell auf einer ein- oder zweistufigen Architektur beruhen, jedoch bringen zweistufige Varianten im Verhältnis zum Aufwand nur geringen zusätzlichen Nutzen. Da der Prozessor und somit die vom Rechenzeitbedarf her teuerste und Code-Größe größte Komponente in den DB-Server verlagert wurde, können durch einen zwischengeschalteten Verbindungs-Server keine wesentlich schnelleren Antwortzeiten erzielt werden. Vorteile kommen nur durch die schon erfolgte Authentisierung.

Eine abschließende Übersicht der Architekturvarianten mit ihren spezifischen Vor- und Nachteilen findet sich in Tabelle 3. Die mit C gekennzeichneten Punkte beziehen sich nur auf CGI-Programme, nicht auf Server-Erweiterungen.

	einstufig	zweistufig	DB-Server-Integration
Vorteile	• einfach zu realisieren • wenig Ressourcen im lastfreien Betrieb • für kleine Lösungen geeignet	• Realisierung “langer” Transaktionen möglich • in der Regel schnellere Antwortzeiten • bessere Lastbalancierung für „große“ Lösungen möglich • Caching von Antworten bzw. HTML-/Makrodateien möglich	• verringerte Kommunikation (Ergebnis ist lediglich ein String)
Nachteile	• im Verhältnis teures Start-up durch relativ großes Programm (C) • benötigt neue DB-Connection, Etablierung ist teuer und langwierig	• bei Nichtnutzung höherer Ressourcenbedarf durch aktiven Prozeß-Pool • aufwendige Installation und Konfiguration • kompliziertere Entwicklung durch eine zusätzliche Schicht	• Zusatzbelastung für den DB-Server • u. U. komplexe Entwicklung

Tabelle 3: Vergleich der CGI-Architekturen

4.4 Speicherungsort von Makrodateien

Während für die meisten Werkzeuge die benötigten Makro- und HTML-Dateien im Dateisystem zur Verfügung gestellt werden müssen, ist eine Speicherung dieser Dateien in der DB bei Ansätzen mit DB-Server-Integration sinnvoll. Wir wollen im folgenden die potentiellen Vor- und Nachteile genauer betrachten.

Die Speicherung der Makro- und HTML-Dateien in der DB zusammen mit den Daten ist bei der DB-Server-Integration für die Ausführungsgeschwindigkeit von großem Vorteil. Da bei diesen Werkzeugen der Makro-Prozessor in den DB-Server integriert wird, ist der Zugriff über die optimierte Speicherverwaltung unter Ausnutzung des DB-Puffers deutlich effizienter als ein Zugriff auf das Dateisystem. Jedoch bringt dies auch mehrere Nachteile mit sich. Zum einen wird der DB-Server zusätzlich belastet und kann u. U. ein Engpaß werden. Zum anderen wird durch die Ablage der Makro- und HTML-Dateien im DB-Server deren Erstellung und Wartung schwieriger, da die Dateien in der Regel vorher ausgelesen, dann editiert und anschließend zurückgeschrieben werden müssen. Andererseits kann aber auch durch die zentrale Verwaltung dieser Dateien durch den DB-Server die Administrierung verbessert werden, da nun die DB als zentrales *Repository* dient, auf das die einzelnen Benutzer mit ihren unterschiedlichen Rechten zugreifen können. Interpretiert man die Makro- und HTML-Dateien als reine Textinformationen, so sind im Gegensatz zu einer Speicherung im Dateisystem auch Web-gestützte Erstellung und Wartung dieser Daten über entsprechende HTML-Formulare durchführbar, d. h., eine Fernwartung der Web-Dokumente wird ermöglicht.

Bei der Ablage im Dateisystem ist eine Fernwartung nicht so einfach realisierbar, dafür können die HTML-Dokumente mit allen verfügbaren HTML-Editoren direkt manipuliert werden. Auch ist ein Zugriff auf das Dateisystem im Vergleich zu einem DB-Zugriff für einen externen Makro-Prozessor im Normalfall effizienter, so daß die Ausführungsgeschwindigkeit für diese Architekturen steigt. Insgesamt ist die Entscheidung, wo die HTML-/Makrodateien abgelegt werden, abhängig von den einzelnen Anforderungen an das Web-Werkzeug und die Arbeitsumgebung.

4.5 Realisierung von Zuständen

Um Mehrschritt-Vorgänge, wie das Führen eines Warenkorbs bei einem Bestell-Katalog, oder besondere Funktionalität, wie das Abholen eines Ergebnisses in mehreren Teilen, zu realisieren, ist es notwendig, den jeweiligen client-seitigen Zustand (Kontext) server-seitig in der DB zu speichern. Dazu muß der Benutzer eindeutig identifiziert werden. Bedingt durch die Zustandslosigkeit des HTTP werden daher für die Realisierung solcher Zustände, d. h. für die Identifizierung eines Benutzers bzw. der jeweiligen „Web-Sitzung“, sogenannte *Session-IDs* eingeführt. Zudem wird für personalisierte Angebote, wie z. B. Nachrichten oder Börsenkurse, eine dauerhafte Benutzeridentifikation benötigt (*User-ID*), um auf eine jeweils neue Authentisierung verzichten zu können. Die für diese Zwecke eingesetzten Techniken stellen wir im folgenden vor.

Formularvariable

Um einen Benutzer während eines Arbeitsvorgangs eindeutig identifizieren zu können, wird in das HTML-Formular die Session-ID als versteckte Eingabevariable eingetragen, z. B.

```
<INPUT TYPE=HIDDEN NAME=SID VALUE=4711>
```

Zusammen mit den Benutzereingaben wird im obigen Beispiel die Variable *SID* mit dem Wert *4711* zum Web-Server übertragen. Dort kann über diesen Wert eine (logische) Verbindung zum jeweiligen Vorgang hergestellt werden. Die Vorteile des Session-ID-Verfahrens liegen in der Unterstützung durch jeden Browser-Typ und jede Browser-Einstellung, d. h., diese Lösung ist für alle Client-Konfigurationen einsetzbar. Ein wesentlicher Nachteil ist jedoch, daß die Session-ID in alle beim Benutzer angezeigten HTML-Dokumente, die mit einem Vorgang in Verbindung stehen und Folgeoperationen auslösen können, einkodiert sein muß. Dies bedeutet bei Bestell-Katalogen in der Regel, daß alle HTML-Dokumente dynamisch erzeugt werden müssen, was den Web-Server mit der dahinterstehenden Infrastruktur erheblich belastet und die Antwortzeiten erhöht. Zudem wird durch den Zwang zu durchweg dynamischen HTML-Dokumenten die Anwendungsentwicklung erschwert.

Das Verfahren eignet sich aufgrund der Kodierung der Session-ID nur bedingt für eine dauerhafte Benutzer-Identifikation. Dieser müßte sich für jede neue Sitzung initial mit einem URL mit einkodierter Anfrage (“?SID=4711”) an den Web-Server wenden, was nicht sehr benutzerfreundlich ist.

URL-Kodierung

Ein Verfahren, das kaum noch eingesetzt wird, ist die Kodierung der Session- oder User-ID in den URL (“/news/4711/ueberblick.html“). Der Web-Server muß die eingehenden Anfragen in Aufrufe eines CGI-Programmes umsetzen, welches wiederum die ID ("4711") und das angeforderte Dokument ("ueberblick.html") oder die gewünschte Aktion extrahiert und ein entsprechendes Ergebnis, evtl. unter Hinzuziehung anderer Programme, zurückliefert.

Verwendet man in den erzeugten Dokumenten für lokale Hyperlinks eine relative Adressierung, so werden vom Web-Browser automatisch die fehlenden Teile des URL, z. B. "/news/4711/", ergänzt. Auf diese Weise müssen keine IDs in die Dokumente generiert werden, was die server-seitige Verarbeitung erleichtert. Andererseits ist die Umsetzung von eingehenden Anfragen in die entsprechenden Aufrufe sehr aufwendig und auch der Einstieg über einen personalisierten URL nicht benutzerfreundlich.

HTTP-Cookie

HTTP-Cookies [22, 28] sind im Gegensatz zu den vorhergehenden Lösungen unabhängig vom eigentlichen HTML-Dokument. Sie werden mit der Meta-Information zu einer HTML-Seite vom Server zum Browser übertragen und dort temporär gespeichert. Wird die Zeile

```
Set-Cookie: KNR="4711"; Version="1"; Path="/katalog"; Max-Age="1800"
```

zusammen mit einem Dokument zum Browser übertragen, so wird, falls der Browser Cookies unterstützt und der Benutzer diese zuläßt, bei jeder Dokumentenanforderung an den Web-Server mit dem Unterverzeichnis *katalog* das Cookie

```
Cookie: KNR=4711
```

zum Web-Server übertragen. Da in unserem Beispiel die Variable *Max-Age* auf 1800 gesetzt wurde, ist der Cookie nur für 30 Minuten (1800 Sekunden) gültig und wird dann vom Browser nicht mehr verwendet und gelöscht. Über einen entsprechend hohen Wert werden Cookies für einen längeren Zeitraum persistent gemacht und dienen somit zur Benutzeridentifikation. Auf diese Weise kann z. B. ein Benutzer in einer Web-basierten DB-Anwendung an dem Punkt fortfahren, an dem er zuvor die Anwendung bzw. den Browser verlassen hat, oder auf personalisierte Informationen zugreifen.

Vorteilhaft beim Einsatz von Cookies ist die automatische Unterstützung durch den Browser, d. h., ist einmal ein Cookie gesetzt, so sendet der Browser bei jedem Web-Server-Kontakt automatisch das Cookie. Hierin liegt aber auch der Nachteil von Cookies. Werden diese vom Browser nicht unterstützt oder hat der Benutzer diese abgeschaltet bzw. verweigert diese, so versagen die Cookie-basierten Web-Anwendungen. Da Cookies genauso wie die in HTML-Formularen verwendeten Variablen behandelt werden und auch mehrere Cookies gleichzeitig eingesetzt werden können, lassen sich viele Informationen bei einer einfachen Verarbeitung ablegen.

HTTP-Authentisierung

Ein weiteres Verfahren zur Realisierung von Zuständen und Sitzungen ist die Nutzung der HTTP-Authentisierung. Über die Abfrage der Umgebungsvariable REMOTE_USER, in der bei einem Zugriff auf einen geschützten Bereich der Name bzw. die User-ID des authentisierten Benutzers steht, kann das CGI-Programm die Anfrage zuordnen. Vorteil dieses Verfahrens ist die automatische Unterstützung durch Browser und Web-Server. Allerdings bedarf es zu einer Nutzung der vorherigen Registrierung der Anwender sowie der Authentisierung zu Beginn jeder Sitzung, so daß dieses Verfahren nicht immer praktikabel ist.

Probleme

Ein generelles Problem, das allen Verfahren zur Realisierung von Zuständen anhaftet, ist die Frage nach geeigneten Timeout-Werten zur server-seitigen Unterbrechung von Web-Sitzungen bei Untätigkeit des Benutzers. Ein zeitgebundener server-seitiger Abbruch ist wichtig, um nicht benötigte Ressourcen (z. B. Kontextdaten in der DB) wieder freizugeben. Dies ist besonders bei zweistufigen Lösungen notwendig, bei denen für jeden Benutzer zur Realisierung "langer" Transaktionen ein eigener Verarbeitungsprozeß etabliert wird. Die Timeout-Werte hängen dabei sehr stark von der jeweiligen Anwendung ab. So wird bei einem Bestell-Katalog ein entsprechender Warenkorb schon nach wenigen Stunden gelöscht, während eine Web-basierte Geschäftsanwendung evtl. mehrere Tage ihren Zustand behält. Bei Suchmaschinen ist das materialisierte Ergebnis, das in mehreren Schritten abgeholt wird, in der Regel schon nach wenigen Minuten wieder gelöscht worden.

Ein abschließender vergleichender Überblick über die vorgestellten Verfahren findet sich in Tabelle 4.

	Formularvariable	URL-Kodierung	HTTP-Cookie	HTTP-Authentisierung
Vorteile	• Unabhängigkeit von Browser-Typ sowie Benutzereinstellungen	• Unabhängigkeit von Browser-Typ sowie Benutzereinstellungen	• automatische Unterstützung durch den Browser • Einsatz unabhängig von der Kodierung in einer HTML-Seite	• automatische Unterstützung durch Browser und Web-Server • Einsatz unabhängig von Kodierung in HTML-Seite
Nachteile	• muß in jeder HTML-Seite, die im Browser angezeigt wird, enthalten sein • Anwendungsentwicklung aufgrund dynamischer HTML-Seiten schwieriger	• aufwendige Umsetzung von HTTP-Anfragen	• nicht von allen Browsern unterstützt • Benutzer kann Cookies abschalten bzw. verweigern	• Benutzer-Registrierung erforderlich

Tabelle 4: Realisierung von Zuständen

5. Applet-basierte Ansätze

Durch die Einführung von Java im Jahr 1995 haben sich die Möglichkeiten zur Realisierung Web-basierter Anwendungen, insbesondere auch von DB-Anwendungen, drastisch vergrößert. Waren durch die HTTP-basierten Lösungen nur server-zentrierte Anwendungen möglich, so ist es mit Hilfe von Java-Applets möglich, Anwendungsmodule zur Laufzeit vom Web-Server in den Client, d. h. den Web-Browser zu laden und dort ausführen zu lassen. Zudem ist für die eigentliche Verarbeitung der Umweg über den Web-Server nicht mehr notwendig, der Applikations- oder der DB-Server können potentiell direkt kontaktiert werden.

Im folgenden wollen wir zunächst wieder allgemeine Vor- und Nachteile, die aus dem Einsatz von Java für Web-basierte DB-Anwendungen resultieren, erarbeiten. Anschließend werden wir dann zuerst Lösungen auf Basis von JDBC, dann unter Nutzung von SQLJ diskutieren. Im Anschluß daran werden wir auf CORBA-basierte Lösungen eingehen. Den Abschluß bilden die Analyse weiterer Lösungsansätze, wie sie u. a. zur Realisierung von Anwendungen mit besonderen Anforderungen notwendig sind, z. B. Online-Banking, sowie ein kurzer Überblick über weitere APIs und Lösungen.

5.1 Allgemeines

Applet-basierte Web-Anwendungen profitieren besonders von der Möglichkeit, Anwendungslogik im Web-Browser ausführen zu lassen und von dort Netzwerkverbindungen zu öffnen. Dies bringt neben den potentiellen Vorteilen für DB-Anwendungen aber auch einige der Sprache Java inhärente Nachteile mit sich, die wir im folgenden gegenüberstellen wollen.

Grafikfähiges Programm

Durch die Möglichkeit, Anwendungen direkt im Browser ausführen zu lassen, stehen nun neben einem Programm mit Zustandsspeicher auch komplexe Datenauswertefunktionen zur Verfügung. Verbunden mit den Grafikfähigkeiten von Java können neben den von HTML nur unterstützten alphanumerischen „Geschäftsdaten“ in Tabellenform auch komplexe Daten, wie z. B. Meßwerte oder Geometrie-/CAD-Daten, visualisiert werden. Daher eignen sich applet-basierte Anwendungen, sieht man von den netzwerkbedingten Nachteilen ab, auch für solche Anwendungstypen.

Während bei HTTP-basierten Verfahren durch die Übertragung des HTML-Dokuments neben den Daten auch die gesamte Markup-Information geliefert wird, kann bei Java-basierten Lösungen eine Beschränkung auf die eigentlichen Daten erfolgen. Mußte bei HTTP-basierten Verfahren z. B. zur Darstellung eines Kursverlaufes einer Aktie server-seitig die Grafik generiert und als eingebettetes Bild übertragen werden, kann die Datenaufbereitung im Browser durch das Applet erfolgen. Jedoch hat dies auch Nachteile, da zum einen das gesamte Programm inklusive der entsprechenden Präsentationslogik vor dem Starten des Applets vom Web-Server in den Browser geladen werden muß. Zum anderen steht für die Darstellung von Formularen und Ausgabedaten nicht mehr HTML mit seiner Einfachheit zur Verfügung, die gesamte Benutzerschnittstelle muß in Java programmiert werden.

Ein Vorteil des im Browser ablaufenden Programms gegenüber reinem HTML ist die allen Programmen inhärente Fähigkeit, einen Zustand transient zu speichern. Hierdurch ist es ohne großen Aufwand möglich, mehrere voneinander abhängige DB-Anweisungen in einer Transaktion auszuführen. Dies bedeutet, daß nun auch „lange" Transaktionen ausgeführt werden können.

Netzwerkverbindungen und Sicherheit, Zuverlässigkeit

Für die Realisierung von DB-Anwendungen besonders interessant ist die Möglichkeit, innerhalb eines Applets Netzwerkverbindungen zu öffnen, z. B. zu einem DB-Server (❺) oder einem aus Sicherheitsgründen dazwischen geschalteten Verbindungs-Server (siehe Abb. 1, ❻+❼). Durch die direkte DB-Kommunikation entfällt der Umweg über den Web-Server, die Zwischenschritte über CGI-Programme sind nicht mehr notwendig. Daher können jetzt prinzipiell beliebig lange Transaktionen unter Nutzung der üblichen zweiphasigen Commit-Protokolle (2PC) realisiert werden. Diesbezüglich nachteilig wirkt sich zunächst das Java-Sicherheitskonzept aus, das Applets nur Verbindungen zu dem Rechner erlaubt, von dem es geladen wurde. Hierdurch bedingt sind sowohl Web-Server als auch DB- bzw. Verbindungs-Server auf einem Rechner anzusiedeln, was zum einen aus Last-, zum anderen aus Sicherheitsgründen nicht sinnvoll ist. Durch die Nutzung signierter Applets und der Gewährung des Verbindungsrechts können aber auch DB- und Kommunikations-Server angesprochen werden, die nicht auf dem Rechner des Web-Servers angesiedelt sind. Dieser Rechner ist dann nicht länger ein potentieller Flaschenhals.

Ladezeiten

Ein großer Nachteil, der jedoch durch den Einsatz neuerer Java-Versionen oder ein abgestimmtes Programmiermodell vermieden werden kann, sind die im Verhältnis zu HTTP-basierten Lösungen initial langen Ladezeiten. Dies rührt daher, daß anfangs zunächst das Applet, welches ja die Programmlogik und die programmierte Benutzerschnittstelle enthält, vom Web-Server in den Browser geladen werden muß.

Durch die Nutzung von JAR-Dateien sowie die Kombination mit signierten Applets können auch Applets bzw. große Teile von ihnen auf der Client-Seite persistent gehalten werden (siehe Abb. 1 und Abb. 8, „persistenter Programm-Cache"), so daß ein Laden vom Web-Server vor jedem Programmstart entfällt. Eine andere Lösung besteht in der Verwendung von Java-Interfaces, für die erst zur Programmlaufzeit die eigentliche Implementierung nachgeladen wird. Auf diese Weise kann mit einem vergleichsweise kleinen Applet und geringen Ladezeiten gestartet werden, die später benötigten Programmodule werden dann bei Bedarf erst zur Laufzeit nachgeladen.

Nachdem wir nun die allgemeinen Vor- und Nachteile von Java-basierten Web-Anwendungen besprochen haben, wollen wir uns jetzt den unterschiedlichen Realisierungsmöglichkeiten solcher Anwendungen zuwenden. Beginnen werden wir dabei mit JDBC.

5.2 JDBC

Schon bald nach der ersten Vorstellung von Java und der Präsentation in HTML-Seiten eingebetteter Applets stellte man fest, daß Java eine DB-Schnittstelle fehlte, um „richtige" Anwendungen zu entwickeln. Daher wurde in Anlehnung an ODBC (*Open DataBase Connectivity*) bzw. das CLI (*Call Level Interface*) und mit einer ähnlichen API die Klassenbibliothek JDBC (*Java DataBase Connectivity*, [14]) definiert. Sie ist seit dem JDK 1.1 in der Sprachdefinition enthalten und wird ständig um weitere Funktionalität ergänzt. So wurde auch die aktuelle Version JDBC 2.0 [37] in eine „Core API" und eine „Standard Extension API" aufgeteilt, um die notwendige Grundfunktionalität von weitergehenden Möglichkeiten zu trennen und somit kommende JDBC-Treiber in ihrer Code-Größe nicht zu groß werden zu lassen. JDBC-Treiber sind dabei Java-Klassen, die die abstrakte Interface-Definition der JDBC-API implementieren und von DBVS-Herstellern und Drittanbietern bereitgestellt werden.

Grundstruktur

Wie aus Abb. 6 ersichtlich, ist für die erfolgreiche Ausführung einer DB-Anfrage das Zusammenspiel unterschiedlicher Klassen notwendig, die alle im Paket *java.sql* enthalten sind. Zu ihnen gehören ein Treiber-Manager (*DriverManager*), eine DB-Verbindungsklasse (*Connection*) sowie Klassen für auszuführende DB-Anweisungen (*Statement*) und die Ergebnismenge (*ResultSet*).

```
import java.net.URL;
import java.sql.*;
class BookmarkQuery{

   public static void main (String args[]) {
       String url   = „jdbc:db2://dbserv:5555/bookmarks";
       String query = „select name,url from bookmarks
       where name LIKE 'Loe%' order by name;";
       try {
// JDBC-Treiber laden
       Class.forName („com.ibm.db2.jdbc.net.DB2Driver");
// Verbindung zum DB-Server aufbauen
       Connection con = DriverManager.getConnection (
           url, „ich", „geheim");
// DB-Anweisung erzeugen
       Statement stmt = con.createStatement ();
// oben spezifizierte Anfrage ausführen
       ResultSet rs = stmt.executeQuery (query);
// Ergebnisse holen und zeilenweise ausgeben
       while (rs.next()) {
           System.out.println(rs.getString(1)+
           " „+rs.getString(2));
       }
// Ergebnismenge, DB-Anweisung und Verbindung schließen
       rs.close();
       stmt.close();
       con.close();
// Fehlerbehandlung und Programmende...
   } catch (SQLException ex) {
   ...
```

Abb. 6: Java-Programm mit JDBC

Da die JDBC-basierte Programmierung abgesehen von der SQL-Syntax und den produktspezifischen Einschränkungen unabhängig von einem DBS oder einem JDBC-Treiber ist, braucht der zu verwendende JDBC-Treiber erst zur Laufzeit spezifiziert und eingeladen werden. Ist der Treiber geladen, kann eine Verbindung zum DB-Server und einer DB aufgebaut werden. JDBC verwendet zur Spezifikation beider Komponenten die URL-Notation, d. h., in unserem Beispiel soll die DB *bookmarks* auf dem Rechner *dbserv* angesprochen werden, wobei ein Kontakt über den Port *5555* hergestellt wird. Ist die DB-Verbindung erfolgreich etabliert worden, so kann für diese Verbindung eine DB-Anweisung erzeugt und die Anfrage ausgeführt werden. Ist auch dies ohne Fehler verlaufen, so können die Ergebniszeilen geholt und weiterverarbeitet werden. Auf ähnliche Weise lassen sich auch andere DB-Anweisungen ausführen.

Die Vorteile in der Verwendung von JDBC liegen in seiner Unabhängigkeit von einer herstellerspezifischen API sowie in seiner Mächtigkeit. Letzteres bedeutet, daß auf der Basis von JDBC sehr komplexe und auch dynamisch konfigurierbare DB-Anwendungen entwickelt werden können, da verwendete Treiber, DB-Server und DBs sowie die Anweisungen an sich erst zur Laufzeit spezifiziert werden müssen. Diese Mächtigkeit ist aber zugleich auch der Nachteil von JDBC, da, wie bei vielen anderen APIs auch, die Programmierung schon für einfache Anwendungen relativ aufwendig ist und entsprechende JDBC-Kenntnisse voraussetzt.

Architektur

Bei der DB-Anbindung mit Hilfe von JDBC werden in Abhängigkeit von ihrer Architektur vier verschiedene Treiber-Typen unterschieden [19]. Für die Realisierung Web-basierter Anwendungen kommen jedoch nur die JDBC-Treiber vom Typ 3 oder 4 in Frage, da Treiber vom Typ 1 (über eine *JDBC-ODBC Bridge*) bzw. Typ 2 (Verwendung von DB-Client-API) auf der Client-Seite auch Nicht-Java-Bibliotheken benötigen. Dies bedeutet, daß nicht die gesamte Applikation einzig mit einem Java-fähigen Browser auskommt, sondern daß noch zusätzliche Software-Installationen auf der Client-Seite notwendig sind. Dies widerspricht jedoch dem Grundgedanken Web-basierter Applikationen. Daher wollen wir im folgenden nur Treiber vom Typ 3 oder 4 betrachten.

JDBC-Treiber vom Typ 3 öffnen keine direkte DB-Verbindung, sondern wickeln die gesamte DB-Kommunikation über einen Verbindungs-Server ab (siehe Abb. 8). Dort erfolgt die Umsetzung der Anweisungen in ein dem DBS bekanntes Format, d. h. die Kommunikation über die Hersteller-API. Vorteile dieses Treiber-Typs sind die Flexibilität bzgl. der Kommunikation sowie die damit zusammenhängende Benutzbarkeit durch alle Arten von Applets, zudem eine erhöhte Sicherheit. Durch den Kommunikationsumweg über den Verbindungs-Server kann der DB-Server auf einem beliebigen Rechner untergebracht werden und muß, möchte man auch unsignierte Applets einsetzen, nicht auf demselben Rechner wie der Web-Server angesiedelt werden. Verwendet man Signed Applets, so kann der Verbindungs-Server auch zur (statischen) Lastbalancierung eingesetzt werden, d. h., Web-, Verbindungs- und DB-Server werden auf jeweils einem eigenen Rechner installiert werden. Durch den Umweg über den Verbindungs-Server kann auch die Systemsicherheit erhöht werden, da bei einer Abschirmung des DB-Servers bzw. Verbindungs-Servers über eine *Firewall* der DB-Server keinen direkten Angriffen mehr ausgesetzt ist. Nachteilig wirkt sich jedoch der Verbindungs-Server auf die Antwortzeiten aus, weil für jeden DB-Server-Kontakt der Umweg über den Verbindungs-Server notwendig ist.

Bezüglich der Antwortzeiten besser sind JDBC-Treiber vom Typ 4, da sie direkten Kontakt zum DB-Server aufnehmen (siehe Abb. 8). Dazu ist eine Implementation des Hersteller-spezifischen Kommunikationsprotokolls notwendig. Bedingt durch die direkte Kommunikation entstehen beim Einsatz unsignierter Applets Einschränkungen bzgl. der Plazierung des DB-Servers. Dieser muß auf demselben Rechner wie der Web-Server laufen, was diesen Rechner je nach System zu einem Flaschenhals macht. Zudem sind spezielle Sicherungsmaßnahmen gegen Angriffe erforderlich, da der Kommunikations-Port des DB-Servers bedingt durch die auf den DB-Server zugreifenden Applets im Internet bekannt ist. Daher sind Treiber vom Typ 4 auch eher für einen Einsatz in einem sicheren Intranet geeignet.

Eine abschließende Zusammenfassung der Bewertung von JDBC findet sich in Tabelle 5. Treiber-spezifische Vor- oder Nachteile sind durch den Treibertyp in Klammern gekennzeichnet.

5.3 SQLJ

Ausgehend von der relativ komplizierten DB-Programmierung auf Basis der JDBC-Schnittstelle wurde nach einer Lösung gesucht, wie sie auch in anderen Programmiersprachen im Einsatz ist. Zunächst unter dem Namen J/SQL, dann unter dem nun verwendeten Namen SQLJ wurde von einem Zusammenschluß mehrerer Hersteller ein Java-Pendant zum klassischen ESQL, also der Spracheinbettung von SQL, vorgeschlagen [33]. Darauf aufbauend wurden bereits auch Erweiterungen, wie z. B. die Integration von Java-Funktionen als *Stored Procedures* in die DB und eine Abbildung von Tabellen auf Java-Klassen, spezifiziert [9].

Grundstruktur

Wie in Abb. 7 zu sehen, werden bei SQLJ ähnlich wie bei ESQL SQL-Anweisungen mit einer vorgestellten Markierung (*#sql*) in den normalen Quelltext eingebettet und müssen vor der eigentlichen Übersetzung durch einen Präprozessor in normale Wirtssprachenkonstrukte umgesetzt werden. Bei SQLJ werden die DB-Anweisungen zur Zeit nur in Aufrufe der JDBC-Schnittstelle umgeformt, eine Abbildung auf eine Hersteller-eigene Java-API ist aber theoretisch auch möglich. Eine Mischung von SQLJ- und JDBC-Kommandos im selben Quelltext ist möglich.

Anders als bei JDBC müssen bei SQLJ die SQL-Anweisungen schon zum Zeitpunkt des Präprozessorlaufs feststehen. Dadurch bedingt steht auch schon die Ergebnisstruktur fest, so daß ein Iterator für die Ergebnismenge definiert werden kann. Hierbei kann man zwei verschiedene Arten von Iteratoren unterscheiden, deren gleichzeitige Verwendung für eine SQLJ-DB-Anfrage sich gegenseitig ausschließt: Bindung über die Position oder Bindung über den Namen. Bei Iteratoren, die über den Namen binden (siehe Abb. 7, Beispiel 1), wird ähnlich einer Record- bzw. Typdefinition für die Ergebnisattribute der Typ und der Name spezifiziert. Nach der Anfrageausführung kann mit dem Iterator die Ergebnismenge durchlaufen werden. Der Zugriff auf die Felder eines Ergebnistupels geschieht über den Spaltennamen, der in der Select-Klausel der Anfrage festgelegt bzw. aufgeführt wurde. Für das Durchlaufen sowie den Attributzugriff ist kein weiteres SQLJ-Konstrukt notwendig. Dies ist auch der Unterschied zu einem aus ESQL her bekannten Cursor. Anders als Cursor sind Iteratoren normale Objekte der Wirtssprache, also Java, und können auch außerhalb von SQLJ-Anweisungen wie Java-Objekte verwendet werden.

Bei Iteratoren, die über die Position binden (siehe Abb. 7, Beispiel 2), werden bei ihrer Definition nur die Typen für die Ergebnisspalten angegeben. Nach dem Ausführen einer Anfrage ist nun für den Ergebniszugriff ein weiteres SQLJ-Konstrukt notwendig. Mit der *FETCH*-Anweisung werden dazu über das Schlüsselwort *INTO* die Felder des aktuellen Ergebnistupels an Java-Variable gebunden. Auf diese kann nun für die weitere Verarbeitung zugegriffen werden. Ebenfalls über *INTO* kann bei *SELECT*-Anweisungen in der Select-Klausel eine Variablenbindung erreicht werden.

SQLJ bietet die Unterstützung für mehrere gleichzeitige DB-Verbindungen. Hierzu muß ein entsprechender Verbindungskontext definiert werden, der den jeweiligen Anweisungen, die sich auf diesen Kontext beziehen sollen, voranzustellen ist. Bei der folgenden Anweisung wird die Anfrage auf der Verbindung *bookmarkdb* ausgeführt, wobei dieser Verbindungskontext vorher definiert sein muß:

```
// Iterator für Ergebnismenge definieren
   #sql public iterator ResRec(
       String name,
       String url );
// Iterator für 2. Beispiel definieren
   #sql public iterator MyPos (String, String);

class BookmarkQueries {
   public static void main (String args[]) {
// Verbindung aufbauen
   ConnectionManager.initContext();
// Beispiel 1
// DB-Anweisung ausführen
   ResRec rr;
   #sql rr={ select name,url from bookmarks
       where name LIKE 'Loe%' order by name };
// Ergebnisse abholen und zeilenweise ausgeben
   while (rr.next())
   {   system.out.println( rr.name()+„ „+rr.url() ); }
// Ergebnismenge schließen
   resRec.close();

// Beispiel 2
   MyPos mp; String rname; String rurl;
// Db-Anweisung ausführen
   #sql mp={ select name,url from bookmarks
       where name LIKE 'Loe%' order by name };
   while (true) {
// über Iterator Ergebnis in eigene Variablen holen
       #sql { FETCH :mp INTO :rname, :rurl };
       if (mp.endFetch()) break;
       System.out.println(rname +„ „ +rurl); }
// Fehlerbehandlung und Programmende...
   } catch (SQLException ex) {
   ...
```

Abb. 7: Java-Programm mit SQLJ

```
#sql [bookmarkdb] rr= { SELECT ... FROM ... WHERE ... };
```

In Analogie zu ESQL bietet SQLJ Anweisungen zur Transaktionssteuerung sowie die Möglichkeit zur Ausführung von DDL- und DML-Befehlen. Bedingt durch seine enge Spracheinbettung werden jedoch keine Konstrukte für die Variablendeklaration (*DECLARE SECTION*) benötigt. Auch fehlt die Definition für dynamische Anweisungen, da diese über das Zusammenspiel mit JDBC erreicht werden können. Insgesamt stehen somit unter Nutzung von SQLJ zahlreiche Möglichkeiten zur DB-Programmierung in einer einfach anwendbaren Art und Weise zur Verfügung.

Aufgrund der Abbildung von SQLJ auf JDBC besitzt SQLJ zur Laufzeit dieselbe Architektur wie JDBC (siehe auch Abb. 8). Ein Überblick über die Vor- und Nachteile von SQLJ und JDBC findet sich in Tabelle 5.

5.4 CORBA-basierte Lösungen

Der Kern des von der OMG (*Object Management Group*) spezifizierten CORBA-Architektur (*Common Object Request Broker Architecture*, [31]), ein ORB (*Object Request Broker*), stellt ein Kommunikations-Framework bereit. Er erlaubt über ortstransparenten Methodenaufruf die einfache Interoperabilität zwischen in unterschiedlichen Programmiersprachen entwickelten Objekten. Von der OMG werden nur die Schnittstellen und Protokolle definiert, Produkte müssen von den jeweiligen Anbietern erworben werden. Zur Spezifikation von Schnittstellen, auch der von eigenen Anwendungsobjekttypen, dient die IDL (*Interface Definition Language*). Für sie existieren sogenannte *Language Bindings*, die die Abbildung der spezifizierten Typen und Methoden auf die jeweiligen Programmiersprachen festlegen. Seit dem Erscheinen von CORBA 2.2 [31] existiert auch ein offizielles *Java Language Binding*, das von den zahlreichen ORBs mit Java-Unterstützung verwendet wird.

	JDBC (Typ 3/4)	SQLJ
Vorteile	• Mächtigkeit • Dynamik • DB-Hersteller-unabhängige API • Portierbarkeit • Sicherheit (3) • Lastbalancierung über Verbindungs-Server (3) • schnelle Kommunikation (4)	• Einfachheit durch Einbettung in Java • DB-Hersteller-unabhängige Lösung • Portierbarkeit • Dynamik/Mächtigkeit über Interaktion mit JDBC
Nachteile	• komplexe Programmierung • längere Antwortzeiten durch Umweg über Verbindungs-Server (3) • ohne Signed Applets Beschränkung bzgl. der Plazierung des DB-Servers (4) • Sicherheit (4)	• nur statisches SQL • erfordert Präprozessor

Tabelle 5: Vergleich von JDBC und SQLj

Wollte man in der Anfangszeit für eine DB-Anwendung ein Applet mit einer CORBA-basierten Kommunikation benutzen, so war es nötig, die gesamte Klassenbibliothek des ORB, d. h. den ORB in den Browser zu laden. Seit der Version 4 ist in den Netscape-Browser ein ORB integriert (siehe Abb. 8). In naher Zukunft wird durch die Aufnahme der sogenannten *Java IDL* in das JDK 1.2 ein ORB in allen Java-fähigen Browsern zur Verfügung stehen, so daß ein Einladen der gesamten Laufzeitumgebung vom Web-Server in den Browser entfällt. Java IDL ist ein zu CORBA konformer ORB, der über das Kommunikationsprotokoll IIOP (*Internet Inter Orb Protocol*) mit anderen, z. B. server-seitigen ORBs kommunizieren kann (siehe Abb. 8).

Architektur und Ablauf

Wie bei den anderen Java-basierten Lösungen auch muß vor dem Applet-Start dieses vom Web-Server in den Browser geladen werden. Durch den Einsatz bestimmter Browser bzw. Java-Versionen kann dabei das Laden der gesamten CORBA-Laufzeitumgebung vermieden werden. Nach dem Starten läßt sich über das Applet eine Verbindung zum server-seitigen ORB mittels IIOP herstellen. Die Adresse des ORB kann dabei als Aufrufparameter in die HTML-Seite, in die das Applet eingebettet ist, einkodiert sein oder über einen *Naming*- oder *Trading-Service* herausgefunden werden. Ein Trading-Service entspricht den Gelben Seiten, d. h., falls der ORB bzw. das eigentlich zu kontaktierende Anwendungsobjekt beim Trading-Service registriert sind, kann das Applet über die Objekteigenschaften die Adresse herausfinden und dieses kontaktieren. Sowohl für die Services als auch den server-seitigen ORB gelten die Einschränkungen bzgl. der Plazierung auf dem Rechner des Web-Servers, falls keine *Signed Applets* eingesetzt werden.

Hat das Applet die Adresse des gewünschten Anwendungsobjektes einmal gefunden, so kann es Methoden dieses Objektes aufrufen. Dies geschieht entweder direkt über den Methodenaufruf eines normalen Objektes oder unter Nutzung des *Dynamic Invocation Interface* (DII). Letzteres wird benötigt, wenn z. B. zum Zeitpunkt der Applet-Entwicklung die Schnittstellen des Anwendungsobjektes noch nicht feststanden und daher keine sogenannten *Client-Stubs* generiert werden konnten. Dies sind die vom Applet aufgerufenen Methoden, die den Funktionsaufruf unter Nutzung der ORBs an das eigentliche Objekt weiterleiten und dazu die übergebenen Argumente in ein über das Internet übertragbares Format und zurück umwandeln (*(un)marshalling*). Das auf diese Weise aktivierte Objekt kann nun die Methode abarbeiten. Da wir in diesem Beitrag ausschließlich DB-Anwendungen betrachten, wird das Anwendungsobjekt über eine bestehende oder noch zu öffnende DB-Verbindung Anweisungen auf einer DB ausführen. Die DB-Anbindung läßt sich dabei direkt über ein Hersteller-spezifisches Protokoll oder über einen Verbindungs-Server, der z. B. ODBC-Aufrufe konvertiert, aufbauen. Nach der Abarbeitung der Methode und eventuell involvierter Subroutinen wird das Ergebnis über die beiden ORBs an das Applet übertragen, in dem, die Nutzung synchroner Methodenaufrufe vorausgesetzt, der Programmlauf fortgesetzt werden kann.

Anders als bei den bisher besprochenen Java-basierten Lösungen kann der Großteil der Anwendungsfunktionalität in den server-seitigen Anwendungsobjekten ausgeführt werden; das Applet braucht im

Prinzip nur noch die Ergebnispräsentation und Benutzerinteraktion zu übernehmen. Die so realisierte dreischichtige Client/Server-Architektur, d. h. das Applet mit Benutzerschnittstelle, die über CORBA ansprechbaren server-seitigen Anwendungsobjekte sowie der zusätzliche DB-Server, weist eine Vielzahl von Vorteilen auf [30], die wir nicht alle im einzelnen besprechen wollen. Sie gelten auch für die in Kapitel 5.5 diskutierten Speziallösungen. Allgemeine Vorteile dieser dreischichtigen Lösungen im Vergleich zu zweischichtigen Varianten wie JDBC-basierten Anwendungen, sind neben der Skalierbarkeit, der höheren Systemverfügbarkeit, der vereinfachten und deutlich preiswerteren Wartung auch die Flexibilität bzgl. der Aufteilung der Anwendungsfunktionalität sowie die Möglichkeit zur einfachen Anbindung von Altsystemen (*Legacy-Applications*).

Für die Realisierung Web-basierter DB-Anwendungen hat eine CORBA-basierte Lösung bei der Verwendung von JDK1.2 bzw. Browsern mit integriertem ORB den Vorteil, daß das Laufzeitsystem nicht vom Web-Server geladen werden muß. Zudem ist durch die Abschirmung des DB-Servers über die Zwischenstufe des ORB mit Anwendungsobjekten, d. h. dem Applikations-Server, eine erhöhte Systemsicherheit gegeben. Durch die Interoperabilität von in unterschiedlichen Programmiersprachen entwickelten Objekten können die Anwendungsobjekte z. B. in C++ implementiert werden, so daß für die Anwendungslogik die Ausführungsgeschwindigkeit steigt. Aufgrund dieser Vorteile sind CORBA-basierte Ansätze eine gute Wahl für die Realisierung von großen Web-basierten Anwendungssystemen, z. B. Web-fähigen Geschäftsanwendungen, insbesondere dann, wenn es auf die Zusammenarbeit mit bestehenden Systemen oder eine entsprechende Skalierbarkeit ankommt.

5.5 Spezial-Lösungen

Für die Realisierung von Web-basierten DB-Anwendungen mit speziellen Anforderungen, z. B. das Online-Banking mit besonderen Sicherheitsanforderungen oder der Interaktion mit Altsystemen (*Legacy-Anwendungen*), werden Java-Applets eingesetzt. Dies ist vor allem durch die Erweiterungsmöglichkeiten von Java begründet, die bei der Verwendung des Gespanns HTML/HTTP gänzlich fehlen. So ist es z. B. in Java möglich, eigene Kommunikationsprotokolle mit besonderen Verschlüsselungsverfahren einzusetzen. Zwar lassen sich Altsysteme, wie wir bereits erwähnt haben, über CORBA-Lösungen einbinden, jedoch ist dies z. B. aus Leistungs- oder Sicherheitsgründen nicht immer sinnvoll.

Deshalb sind auch die Online-Banking-Anwendungen Java-basierte Spezialanwendungen, die eigene Sicherheitsprotokolle einsetzen und für die eigentliche Auftragsabwicklung auf die bestehenden Alt-

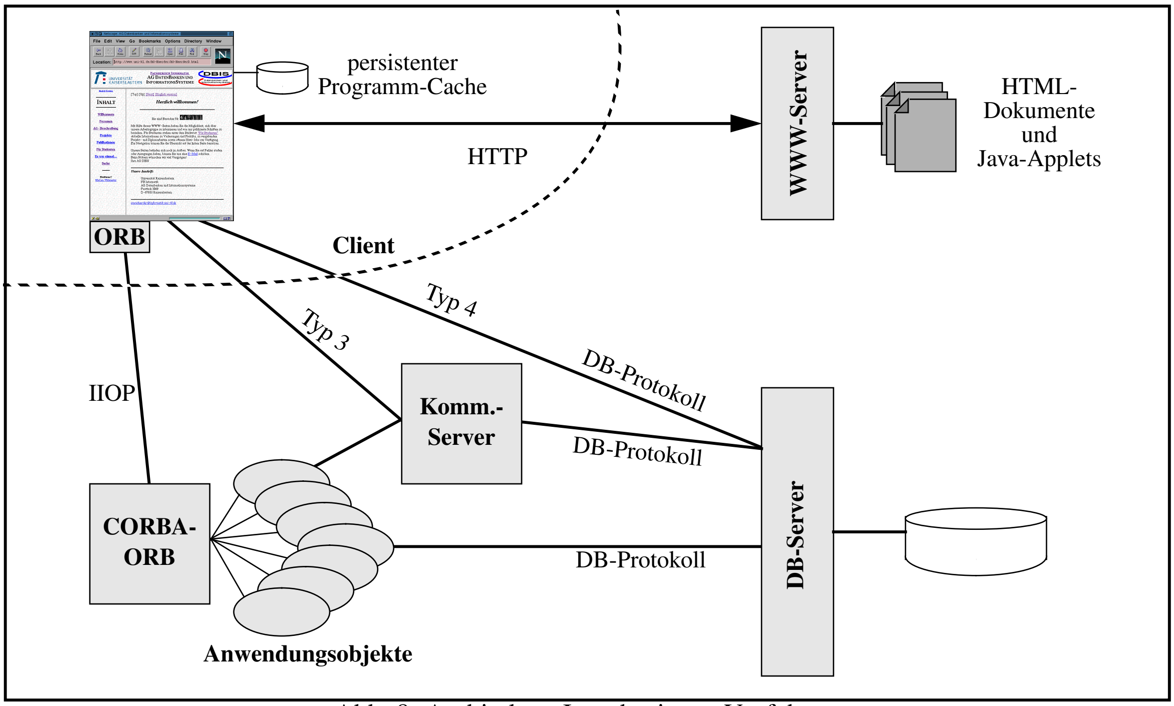

Abb. 8: Architektur Java-basierter Verfahren

systeme zurückgreifen. Zudem ist für bestimmte Service-Angebote, z. B. Ausdruck eines Kontoauszuges, ein Rückgriff auf HTTP/HTML-basierte Lösungen unter speziellen Sicherheitsbedingungen notwendig. Im folgenden wollen wir solch eine Spezialanwendung, ein Applet für Online-Banking, genauer beschreiben und die Lösung unter allgemeinen Gesichtspunkten diskutieren.

Ablauf und Architektur (Beispiel)

Nach dem Einladen des Applets vom Web-Server über HTTPS, also über eine verschlüsselte Verbindung, um ein Verfälschen des Applets während des Ladens zu verhindern, initialisiert sich dieses. Hierbei wird in der Regel unter Benutzerinteraktion, z. B. durch Bewegung der Maus, eine Zufallszahl erzeugt, die für das Aushandeln eines Sitzungsschlüssels der späteren Verbindung zum Kommunikations-Server dient [4]. Die verwendeten Kommunikationsverfahren basieren dabei z. B. auf der in Java vorhandenen Socket-Schnittstelle, ergänzen diese aber um eine entsprechend starke Verschlüsselung (mind. 128 Bit symmetrisch). Nach Etablierung einer Verbindung kann das Applet durch den über *Firewalls* [6] abgeschirmten Kommunikations-Server die gewünschten Dienste ansprechen. Da diese bei Banking-Anwendungen in der Regel auf Mainframe-Lösungen basieren, kommen hierzu spezielle *Gateways* und *Wrapper*, d. h. Programme, die die Altsysteme kapseln und die entsprechenden Originalfunktionen wie z. B. Transaktionsprogramme aufrufen, zum Einsatz. Der Kommunikations-Server dient dabei nur als sichere Verbindung zwischen Applet und dem eigentlichen DB-basierten Anwendungssystem. Sollen etwa besondere Dienste wie das Erzeugen eines (zwecks Ausdruckmöglichkeit) HTML-basierten Kontoauszugs angeboten werden, so kann der Kontoauszug erzeugt und temporär auf dem Web-Server bereitgestellt werden. Das Applet kann mittels HTTP auf das nur kurze Zeit verfügbare Dokument zugreifen und dieses in einem neuen Browser-Fenster zur Anzeige bringen. Auf diese Art und Weise dient das Applet als Schaltzentrale zum Zugriff auf unterschiedlichste Dienste. Der Kommunikations-Server übernimmt die Aufgabe eines sicheren Vermittlers.

Bewertung

Zwar lassen sich auch über die in Java ab JDK 1.2 vorhandenen CORBA-Schnittstellen leistungsfähige Anwendungssysteme realisieren, die u. U. auf Altsysteme zugreifen, jedoch sind für Anwendungen mit speziellen Anforderungen, z. B. Sicherheit, individuelle Lösungsansätze zu entwickeln. Hierbei kann auf bereits für bestimmte Einsatzzwecke entwickelte (Standard-)Produkte oder Lösungen zurückgegriffen werden, z. B. [4]. Insgesamt stellt Java dabei im Vergleich zu HTTP-basierten Lösungen ein viel breiter gefächertes Lösungspotential zur Verfügung. Dies reicht von den einer Programmiersprache inhärenten Realisierungsmöglichkeiten (vergl. auch Kapitel 5.1) über die bereits in der Sprache enthaltenen Funktionsbibliotheken bis hin zu den grafischen Möglichkeiten, die auch die aufbereitete Ausgabe komplexer Daten oder die Schaffung speziell angepaßter Benutzerschnittstellen erlaubt. Daher eignen sich applet-basierte Ansätze besonders für Anwendungen mit besonderen Anforderungen.

5.6 Andere APIs und Lösungen

Neben den gerade ausführlich vorgestellten Lösungen auf Basis von JDBC, SQLJ oder CORBA bzw. spezieller Lösungsansätze existieren diverse weitere Programmierschnittstellen und Lösungen, die wir im folgenden zwecks Einordnung zumindest teilweise noch kurz vorstellen wollen.

Java Remote Method Invocation (RMI)

Die schon in einer frühen Java-Version enthaltene Möglichkeit zum ortstransparenten Methodenaufruf anderer, lokaler oder entfernter Objekte stellt ein rein Java-basiertes Pendant zu CORBA dar. Im Gegensatz zu CORBA wird jedoch aufgrund der Selbstbeschreibungsmöglichkeiten der Java-Objekte keine IDL benötigt, um ein Objekt bzw. seinen Zustand über das Netz zu übertragen. Während bei CORBA Client-Stubs und Server-Skeletons für das *Marshalling* sorgen, kommt bei RMI das ebenfalls in Java verfügbare *Serialization-Interface* zum Einsatz. Es sorgt für die Umwandlung eines Java-Objektes in einen über das Netz übertragbaren Byte-Strom, der server-seitig vom RMI-Server, dem Pendant zum CORBA-ORB, in wieder zurückgewandelter Form an das aufzurufende Objekt weitergeleitet wird.

Da Java-RMI eine proprietäre, reine Java-Fassung von CORBA ist, bezogen auf die Architektur und die Kommunikation ähnliche Vor- und Nachteile wie CORBA besitzt und letzteres ab JDK 1.2 in Java enthalten ist, wird Java-RMI mittelfristig weiter an Bedeutung verlieren.

ODMG Java Binding

Die ODMG (*Object Data Management Group*), die mittlerweile eine Subgruppe der OMG ist, spezifiziert im Standard ODMG 2.0 [7] auch ein *Java Binding* zum Zugriff auf objektorientierte Daten. Es legt fest, wie Typen, die mit der ODL (*Object Definition Language*, eine Obermenge der IDL) spezifiziert wurden, mit ihren Attributen und Zugriffsmethoden in Java zur Verfügung gestellt werden. Weiterhin wird die API für die Abarbeitung von OQL-Anweisungen (*Object Query Language*) und die Handhabung ihrer Ergebnisse spezifiziert.

Die auf dem ODMG-Java-Binding basierenden Schnittstellen können genutzt werden, um Web-basierte Anwendungen mit Zugriff auf Objektdaten zu entwickeln. Während bei HTTP-basierten Verfahren aufgrund der Eignung von HTML für alphanumerische „Geschäftsdaten“, die klassischerweise in RDBVS gespeichert werden, eher selten ODBVS zum Einsatz kommen, können wegen der grafischen Fähigkeiten von Java und der durch das Applet zur Verfügung stehenden Programmlogik zur Datenmanipulation auch auf ODBVS beruhende Web-Anwendungen realisiert werden.

Da die ODMG-Schnittstelle standardisiert ist, ist sie beim Zugriff auf Objektdaten den proprietären Hersteller-APIs vorzuziehen. Ihre Eignung und Bewertung bzgl. der Realisierung von Web-basierten Anwendungen hängt jedoch stark vom eingesetzten ODBVS sowie der Anwendung an sich ab.

Andere Java-DB-APIs

In Ergänzung zu JDBC, SQLJ oder einer ODMG-konformen Schnittstelle bieten die meisten DBVS-Hersteller eigene Java-APIs an, um besondere Merkmale oder Eigenschaften des DBVS besser unterstützen zu können und eine höhere Ausführungsgeschwindigkeit zu erzielen. Bezüglich der Architektur stehen dabei wieder die schon im Rahmen von JDBC besprochenen Varianten zur Verfügung.

Die Vorteile der proprietären Schnittstellen liegen in der engeren Bindung an ein bestimmtes DBVS, seine Datentypen und besonderen Fähigkeiten. Jedoch liegt darin auch der wesentliche Nachteil dieser Ansätze, da sie eine schnelle Anwendungsportierung in der Regel ausschließen.

6. Resümee

In diesem Beitrag haben wir Konzepte und Techniken für die Realisierung Web-basierter DB-Anwendungen vorgestellt und ihre jeweiligen Stärken und Schwächen analysiert. Diese Verfahren bilden die Grundbausteine bei der Entwicklung sowohl kleiner und einfacher als auch großer, komplexer Anwendungssysteme. Während heute nur eine geringe Anzahl reiner DB-basierter Web-Anwendungen existiert und nur vereinzelt eine Web-Oberfläche für bestehende Anwendungen angeboten wird, ist in den nächsten Jahren mit einer starken Zunahme derartiger Web-Anwendungen zu rechnen. Auslöser für diesen Trend ist zum einen die Akzeptanz des Web-Browsers als einfach zu bedienende, homogene Benutzerschnittstelle für Anwendungen aus einem heterogenen Umfeld. Zum anderen sind dies besonders die fortschreitenden Entwicklungen im Telekommunikations- und Netzwerkbereich, die den Web-Anwendern eine immer kostengünstigere und leistungsfähigere Infrastruktur zur Verfügung stellen. Hierdurch bedingt steigen aber nicht nur die potentiell übertragbaren Datenmengen, sondern vor allem auch die Anforderungen und Ansprüche der Benutzer an die Anwendungssysteme in bezug auf die verfügbaren Daten und die Verarbeitungsmöglichkeiten [10].

Bei den vorgestellten Verfahren, die wir zunächst nach HTTP- und applet-basiert unterschieden haben, wurden HTTP-basierte Lösungen auf Basis von CGI-Programmen, Server-Erweiterungen oder SSIs/ASPs als geeignet für viele Anwendungen eingestuft. Über z. B. Cookies zur Benutzeridentifikation und die Wahl einer passenden Prozeßarchitektur lassen sich leistungsfähige und an ihre spezifischen Anforderungen angepaßte Systeme entwickeln, wie z. B. für das Gebiet *Electronic Commerce*. Dieses wurde in diesem Beitrag durch die Anwendung “Bestell-Katalog” charakterisiert. Dieser Anwendungstypus bildet auch die Grundlage für den sich in der Entwicklung befindenden Benchmark für Web-basierte DB-Anwendungen, TPC-W [36].

Für Anwendungen mit besonderen Anforderungen, z. B. Sicherheit oder grafische Interaktion, eignen sich applet-basierte Lösungen. Sie bieten aufgrund des Java-Einsatzes mehr Möglichkeiten für die client-seitige Datenaufbereitung und Benutzerinteraktion sowie die Kommunikation mit DB- und Kommunikations-Servern. Mit JDBC und SQLJ stehen genormte DB-Schnittstellen in Java zur Verfügung.

Interessant scheint bei SQLJ auch die in Zukunft mögliche (genormte) Ablage von Java-Methoden als Stored Procedures in der DB, die noch mehr Möglichkeiten bei der Realisierung bietet. Insgesamt kommen DBs zukünftig eine noch größere Aufgabe zu, da immer häufiger nicht nur die Anwendungsdaten, sondern auch die HTML-Seiten an sich in der DB gespeichert werden. Hierfür scheinen besonders die ORDBVS geeignet zu sein, da sie durch ihre Erweiterungsfähigkeit die Aufnahme neuer Datentypen und die Integration neuer Verarbeitungsfunktionalität erlauben [24, 35]. Durch sie ist es möglich, die im WWW verwendeten Daten auch auf der Typebene des DBS anzubieten. So kann das DBS z. B. auch zur Konsistenzsicherung lokaler Hyperlinks eingesetzt werden. Mit der zunehmenden Unterstützung des WWW und seiner Daten durch DBVS wird es langfristig auch zu „integrierten Lösungen" kommen, die zusätzlich auch Aufgaben eines Web-Servers und einer Suchmaschine übernehmen.

Durch die noch erwartete "Revolution" im Internet, die wachsende Bedeutung von DBs für Web-basierte Anwendungen und die dabei entstehenden neuen Anforderungen bedarf es in Zukunft noch weiterer Entwicklungen und Anstrengungen. Für Forscher, Hersteller und Anwender bedeutet dies eine spannende Zeit. Wie wird sich das Web weiterentwickeln? Wie verändert sich die Bedeutung der DBSs für Web-Anwendungen? Die Zukunft wird es zeigen, warten wir es ab.

7. Literatur

1. *Apache HTTP Server Version 1.3, Module mod_include*, Apache Project, http://www.apache.org/docs/mod/mod_include.html, Juni 1998
2. *Apache HTTP Server Version 1.3, Apache suEXEC Support*, Apache Project, http://www.apache.org/docs/suexec.html, Juni 1998
3. Berners-Lee, T.: *Information Management: A Proposal*, CERN, http://www.w3.org/History/1989/proposal.html, März 1989
4. *X·PRESSO Retail Banking - Architektur*, BROKAT Informationssysteme GmbH, http://www.brokat.de, Juni 1998
5. Cailliau, R.: *A Little History of the World Wide Web*, WWW-Konsortium, http://www.w3.org/History.html, 1995
6. Cheswick, W., Bellovin, S.: *Firewalls and Internet Security - Repelling the Wily Hacker*. Amsterdam: Addison-Wesley Longman 1994
7. Cattell, R., Barry, D., Bartels, D.: *The Object Database Standard: ODMG 2.0*. San Francisco: Morgan Kaufmann Publishers 1997
8. *The CGI Specification*, Version 1.1, University of Illinois, Urbana-Champaign, http://hoohoo.ncsa.uiuc.edu/cgi/interface.html, 1995
9. Clossman, G.: *SQLJ: Java and Relational Databases*, Tutorial, SIGMOD '98, http://www.acm.org/sigmod/sigmod98/eproceedings/, 1998
10. Denning, P., Metcalfe, R.: *Beyond Calculation - The next fifty years of computing*. Boston: Springer-Verlag, 1998
11. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Berners-Lee, T.: *Hypertext Transfer Protocol -- HTTP/1.1*, Network Working Group, Request for Comments: 2068, http://www.w3.org/Protocols/rfc2068/rfc2068, Januar 1997
12. Freier, A. O., Karlton, P., Kocher, P. C. : *The SSL Protocol, Version 3.0 - Internet Draft*, Netscape Communications, http://home.netscape.com/eng/ssl3/ssl-toc.html, Mai 1996
13. Gosling, J., Joy, B., Steele, G.: *The Java Language Specification*, Addison-Wesley, http://java.sun.com:80/doc/language_specification.html, 1996
14. Hamilton, G., Cattell, R.: *JDBC: A Java SQL API, Version 1.20*, ftp://ftp.javasoft.com/pub/jdbc/jdbc-spec-0120.ps, Januar 1997

15. *IBM Net.Data*, IBM Corporation, http://www.software.ibm.com/data/net.data/, 1998
16. *Informix Web DataBlade Module, User's Guide, Version 3.3*, Informix Software Inc., http://www.informix.com/answers/, Mai 1997
17. *JDK 1.2, Security*, Sun Microsystems, Inc., http://www.javasoft.com/products/jdk/1.2/docs/guide/security/index.html, 1998
18. *JDK 1.2, Servlet*, Sun Microsystems, Inc., http://www.javasoft.com/products/jdk/1.2/docs/ext/servlet/index.html, 1998
19. *JDBC Drivers*, Sun Microsystems, Inc., http://java.sun.com/products/jdbc/jdbc.drivers.html, 1998
20. *W3C Jigsaw, SSIResource extension tutorial*, http://www.w3c.org/Jigsaw/User/Tutorials/SSIResource.html, 1998
21. *Java Web Server*, Sun Microsystems Inc., http://jserv.java.sun.com/products/java-server/documentation/webserver1.1/index.html, 1997
22. Kristol, D., Montulli, L.: *HTTP State Management Mechanism*, Network Working Group, Request for Comments: 2109, http://www.w3.org/Protocols/rfc2109/rfc2109, Februar 1997
23. Loeser, H.: *Datenbankanbindung an das WWW - Techniken, Tools und Trends*. In: Tagungsband der GI-Fachtagung 'Datenbanksysteme in Büro, Technik und Wissenschaft' (BTW'97), K. R. Dittrich, A. Geppert (Hrsg.), Informatik aktuell, Ulm, Springer-Verlag, März 1997, S. 83-99
24. Loeser, H.: *Die Nutzung der Erweiterungsmöglichkeiten von ORDBVS für Client/Server-basierte Anwendungssysteme*. In: Tagungsband 10. GI-Workshop Grundlagen von Datenbanken, Konstanz, Juni 1998, Konstanzer Schriften in Mathematik und Informatik, Nr. 63, Mai 1998, S. 77-81
25. *Active Server Pages Technology Feature Overview*, Microsoft Corporation, http://www.microsoft.com/workshop/server/asp/aspfeat.asp, August 1998
26. *Internet Server API Reference*, Microsoft Corporation, http://www.microsoft.com/win32dev/apiext/isapiref.htm, 1997
27. *NSAPI Programmer's Guide*, Netscape Communications Corporation, http://developer.netscape.com/docs/manuals/enterprise/nsapi/index.htm, 1997
28. *Persistent Client State HTTP Cookies*, Netscape Communications Corporation, http://www.netscape.com/newsref/std/cookie_spec.html,1998
29. Neulinger, N.: *CGIWrap - User CGI Access*, http://www.umr.edu/~cgiwrap/, 1998
30. Orfali, R., Harkey, D., Edwards, J.: *The Essential Client/Server Survival Guide. Second Edition*, New York: Wiley Computer Publishing 1996
31. Object Management Group: *The Common Object Request Broker: Architecture and Specification*, ftp://ftp.omg.org/pub/docs/formal/98-02-33.pdf, Februar 1998
32. Rescorla, E., Schiffman, A.: *The Secure HyperText Transfer Protocol - Internet Draft*, Enterprise Integration Technologies, Juli 1995
33. *SQLJ: Embedded SQL for Java, Version 0.9*, Oracle Corporation, http://www.oracle.com/st/products/jdbc/sqlj/sql_specs.html, Juni 1997
34. *Server Side Includes (SSI)*, University of Illinois, Urbana-Champaign, http://hoohoo.ncsa.uiuc.edu/docs/tutorials/includes.html, 1995
35. Stonebraker, M.: *Object-Relational DBMSs - The Next Great Wave*. San Francisco: Morgan Kaufman 1996
36. *TPC Launches New ECommerce Transactional Web Benchmark Effort*, Transaction Processing Performance Council, http://www.tpc.org, 1998
37. White, S., Hapner, M.: JDBC 2.0 API, Version 1.0, Sun Microsystems Inc., http://www.javasoft.com/products/jdbc/jdbc20.ps, Mai 1998