Kapitel 7

Web-basierter DB-Zugriff [1]

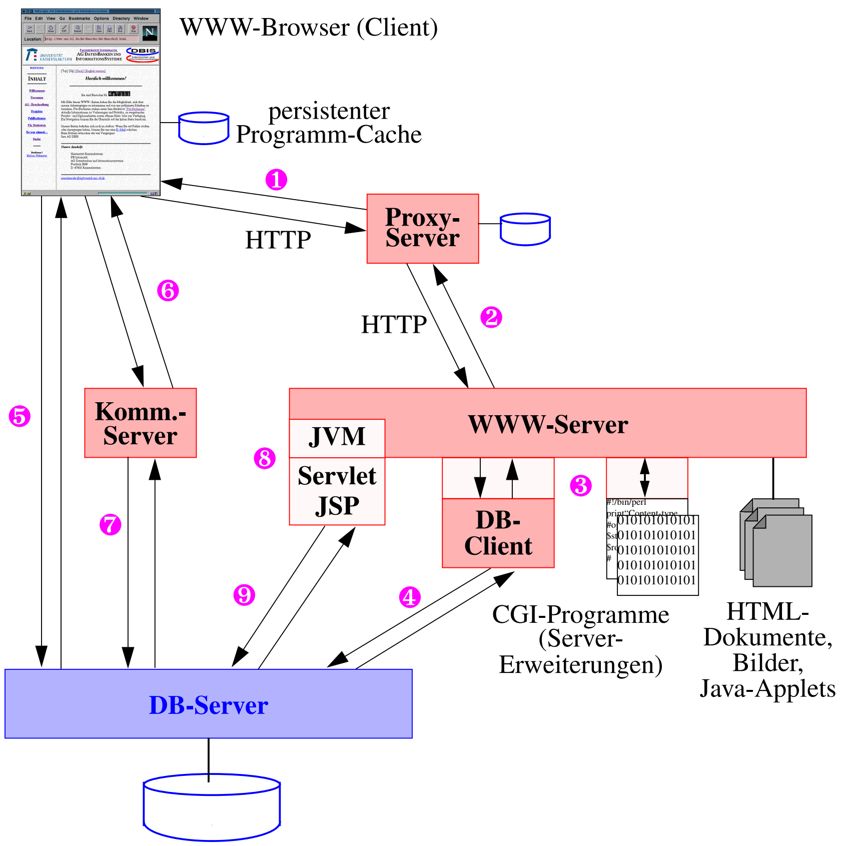
Inhalt

- ❑ Übersicht über die Möglichkeiten des Web-basierten DB-Zugriffs
- ❑ HTTP-basierte Ansätze
- ❑ Applet-basierte Ansätze
- ❑ Spezifische (OR)DBS-Erweiterungen
- ❑ Zusammenfassung

1. Loeser, H.: Techniken für Web-basierte Datenbankanwendungen: Anforderungen, Ansätze, Architekturen, Informatik Forschung und Entwicklung, 1998.

Übersicht (1)

- ❑ Möglichkeiten des Web-basierten DB-Zugriffs



WWW:	World Wide Web
HTML:	HyperText Markup Language
URL:	Uniform Resource Locator
CGI:	Common Gateway Interface
HTTP:	HyperText Transfer Protocol
JVM:	Java Virtual Machine
JSP:	Java Server Page

Übersicht (2)

- Möglichkeiten des Web-basierten DB-Zugriffs (Forts.)
  - WWW-Server
    - zentrale Komponente
    - stellt statische HTML-Seiten, inkl. eingebetteter Bilder, etc., zur Verfügung (❶, ❷)
    - stellt Java-Applets zur Verfügung, die direkt (❺) oder über Kommunikationsserver (❻, ❼) auf DB zugreifen können
    - ruft Server-Erweiterungen auf (❸)
    - ruft CGI-Programme auf (❸)
    - ruft Java-Servlets auf (❽)
    - leitet aus DB-Interaktion (CGI-Programme ❹, Java-Servlets ❾) resultierende HTML-Seiten an Browser weiter
  - DB-Server
    - verwaltet Anwendungsdaten
    - kann auch fertige, statische HTML-Seiten bzw. Teile davon verwalten
  - Proxy-Server
    - puffert Ergebnisse (HTML-Dokumente, Bilder) einer HTTP-Anfrage, um Zugriff auf statische Information zu beschleunigen
    - dynamisch erzeugte (z. B. CGI-Aufruf) und daher speziell markierte Dokumente werden nicht gepuffert
  - Kommunikationsserver
    - kann eingesetzt werden, um Verbindungsweg zum DB-Server für Java-Applets (❻, ❼) herzustellen

Übersicht (3)

- Grundlegende Techniken
  - CGI-Programme
    - dynamische Erzeugung von Dokumenten mittels *CGI* und *HTML-Formularen*
    - WWW-Server startet CGI-Programm in einem eigenen Prozeß
    - CGI-Programm fragt durch WWW-Server gesetzte Umgebungsvariablen ab
    - WWW-Server gibt vom Benutzer in einem HTML-Formular eingegebene Parameter in definierter Weise an CGI-Programm weiter (❸)
    - zum DB-Zugriff kann durch das laufende CGI-Programm Kontakt mit dem DB-Server aufgenommen werden (❹)
    - während seiner Laufzeit erzeugt CGI-Programm ein vollständiges HTML-Dokument und gibt dieses an den (wartetenden) WWW-Server zurück
    - WWW-Server leitet das vom CGI-Programm erzeugte HTML-Dokument an den Web-Browser zurück

Übersicht (4)

- Grundlegende Techniken (Forts.)
  - Server-API
    - Hersteller von WWW-Servern stellen eigene Programmierschnittstelle (*Application Programming Interface, API*) zur Verfügung, damit nicht eigener Prozeß für CGI-Programm erzeugt werden muß
    - Beispiele:
      - NSAPI (Netscape Server API) von Netscape
      - ISAPI (Internet Server API) von Microsoft
    - über Server-API kann WWW-Server um Funktionalität (*Server Application Function, SAF*), die bisher als CGI-Programme bereitgestellt werden mußte, erweitert werden
    - SAFs sind als dynamische Programmbibliotheken zur Verfügung zu stellen, die dem WWW-Server beim Starten hinzugebunden wird
    - über Konfiguration und URL kann der WWW-Server (ähnlich zu CGI) erkennen, ob normaler Dokumentenzugriff oder Aufruf einer SAF (❸)
    - Geschwindigkeitsvorteil gegenüber CGI
      - kein eigener Prozeß
      - DB-Verbindung kann ständig offengehalten werden

Übersicht (5)

- Grundlegende Techniken (Forts.)
  - Server-Side Includes (*SSI*)
    - um spezielle Steuerungsbefehle angereicherte HTML-Dokumente, die vom WWW-Server beim Dokumentenzugriff dynamisch ausgewertet werden
    - SSIs können bspw. genutzt werden, um aktuelles Datum/Uhrzeit bzw. weitere aktuelle Informationen in eine Web-Seite einfließen zu lassen
    - SSIs können auch Betriebssystem-Kommandos oder Anwendungen aufrufen
    - ausgehend von SSIs kann mit Hilfe von CGI bzw. Server-Erweiterungen auf DBs zugegriffen werden

Übersicht (6)

- Grundlegende Techniken (Forts.)
  - Java
    - 1995 von SUN Microsystems veröffentlicht
    - Plattformunabhängigkeit:
      - *Java Bytecode* Konzept ermöglicht es, kompiliertes Programm auf ein anderes System zu übertragen und dort von *Java Virtual Machine (JVM)* ausführen zu lassen
      - JVM interpretiert den Java Bytecode und bildet ihn auf die Maschinenbefehle des jeweiligen Systems ab
    - ***Java Applications***
      - normale, in Java programmierte Anwendungen
      - eignen sich nicht direkt für Web-Anwendungen
      - können aber z. B. für die Implementierung von CGI-Programmen oder Kommunikationsservern eingesetzt werden

Übersicht (7)

- Grundlegende Techniken (Forts.)
  - Java (Forts.)
    - ***Java Applets***
      - werden ähnlich wie HTML-Dokumente auf einem WWW-Server bereitgestellt und vor der Ausführung als Teil einer HTML-Seite (ähnlich wie ein Bild) in einen Java-fähigen Web-Browser geladen (❶, ❷)
      - im Browser notwendigerweise vorhandene JVM übernimmt Ausführung
      - allgemeine Sicherheitsrestriktionen (z. B. nur Netzverbindungen zu dem Rechner, von dem das Applet geladen wurde; kein Zugriff auf lokale Ressourcen)
      - JAR-Dateien (*Java ARchive*) machen es möglich, mehrere bzw. alle Class-Dateien, die ein Applet benötigt, auf einmal übers Netz zu laden
      - *Signed Applets*
        - flexibleres Sicherheitskonzept von JDK (*Java Development Kit*, V. 1.1 und 1.2)
        - in einer JAR-Datei zusammengefaßtes, mit einer digitalen Signatur ausgestattetes Applet
        - stellt sicher, daß das Applet bzw. die Class-Dateien seit Erstellung nicht mehr verändert wurden
        - können persistent auf dem Rechner des Anwenders gespeichert werden
      - für die Realisierung von Applets steht der volle Java-Sprachumfang zur Verfügung

Übersicht (8)

- ❑ Grundlegende Techniken (Forts.)
  - ➪ Java (Forts.)
    - • ***Java-Servlets***
      - ‘SUN-Pendant’ zu Server-Erweiterungen von Netscape und Microsoft für eigenen, komplett in Java implementierten Web-Server
      - Aufnahme in JDK 1.2 und Unterstützung auch durch andere WWW-Server-Hersteller
      - ermöglichen plattform- und herstellerunabhängige Erweiterungen von Web-Servern
      - Voraussetzung: Integration einer JVM in WWW-Server (❽) bzw. Kooperation mit einem entsprechenden Zusatzprozeß
      - Behandlung genau wie bei C-basierten Server-APIs (❷)
      - weiterer Vorteil: dynamisches Binden durch Java-Klassenlader → durchgängiger Server-Betrieb

HTTP-basierte Ansätze (1)

- ❑ ‘HTTP-basierte’ Ansätze umfassen:
  - ➪ ‘klassische’ CGI-Programme
  - ➪ Nutzung von Server-Erweiterungs-APIs
  - ➪ Java-Servlets
  - ➪ SSIs
- ❑ *Zustandslosigkeit*
  - ➪ Problem
    - • Kommunikationsverbindung zwischen Browser und Web-Server besteht nur für die Dauer der Bearbeitung der HTTP-Anfrage
  - ➪ Folge
    - • da mit Hilfe von CGI bzw. Server-API realisierte DB-Clients nur für diese Dauer aktiv sind, können auch Transaktionen nicht länger dauern
    - • negative Auswirkungen auf Laufzeit
      - Neu-Öffnen einer DB-Verbindung für jede HTTP-Anfrage
      - bei CGI: Prozeßneustart für jede HTTP-Anfrage

HTTP-basierte Ansätze (2)

❑ *Sicherheit*

➪ Zugriffsschutz des Web-Servers

- HTTP-Authentisierung
  - Zugriff auf Unterverzeichnisse bzw. den ganzen Server kann auf bestimmte Benutzer eingeschränkt werden
  - beim Zugriff auf geschützten Bereich (*domain*) muß Benutzer sich über Benutzername und Paßwort authentisieren
  - mittels Abbildung des Web-Benutzers auf lokale ID kann DB-Verbindung unter dieser lokalen ID aufgebaut werden
  - für ID-Wechsel sind jedoch spezielle Werkzeuge notwendig; wird normalerweise von WWW-Server nicht unterstützt
  - z. B. suEXEC oder CGIwarp dienen der Ausführung eines CGI-Programms unter einer anderen ID
- Einschränkung des Verbindungsrechts auf bestimmte Internet-Adressen
  - über Konfiguration (des Web-Servers) kann festgelegt werden, von welchen Internet-Adressen aus auf Web-Server oder bestimmte Unterverzeichnisse zugegriffen werden darf und von welchen ein Zugriff verboten ist (*allow/deny*)

HTTP-basierte Ansätze (3)

❑ *Sicherheit* (Forts.)

➪ Übertragungssicherheit

- Verschlüsselungsverfahren
  - SHTTP: *Secure HTTP*
  - SSL: *Secure Socket Layer*
- SSL hat sich wegen seiner Praktikabilität durchgesetzt
  - basiert auf dem RSA-Verfahren
  - benötigt auf Seiten des WWW-Servers ein elektronisches Zertifikat
  - ist der Aussteller dem Browser nicht bekannt, wird Benutzer gefragt, ob er dem Anbieter trauen möchte
  - lehnt Benutzer ab, wird keine verschlüsselte Verbindung aufgebaut
  - Nachteil:
    kurze Schlüssellängen (aufgrund US-amerikanischer Exportrestriktionen) machen SSL für Anwendungen mit besonderen Sicherheitsanforderungen, z. B. Banking, außerhalb der USA unbrauchbar

HTTP-basierte Ansätze (4)

❑ SQL/HTML-Integration

➪ Programmierung

- Beispiel

```
#!/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
# [...]
```


HTTP-basierte Ansätze (5)

❑ SQL/HTML-Integration (Forts.)

➪ Integration von HTML und SQL

- Beispiel

```
<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>
```

HTTP-basierte Ansätze (6)

- SQL/HTML-Integration (Forts.)
  - Macro-Programmierung
    - Beispiel

```
# 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>
%}
```

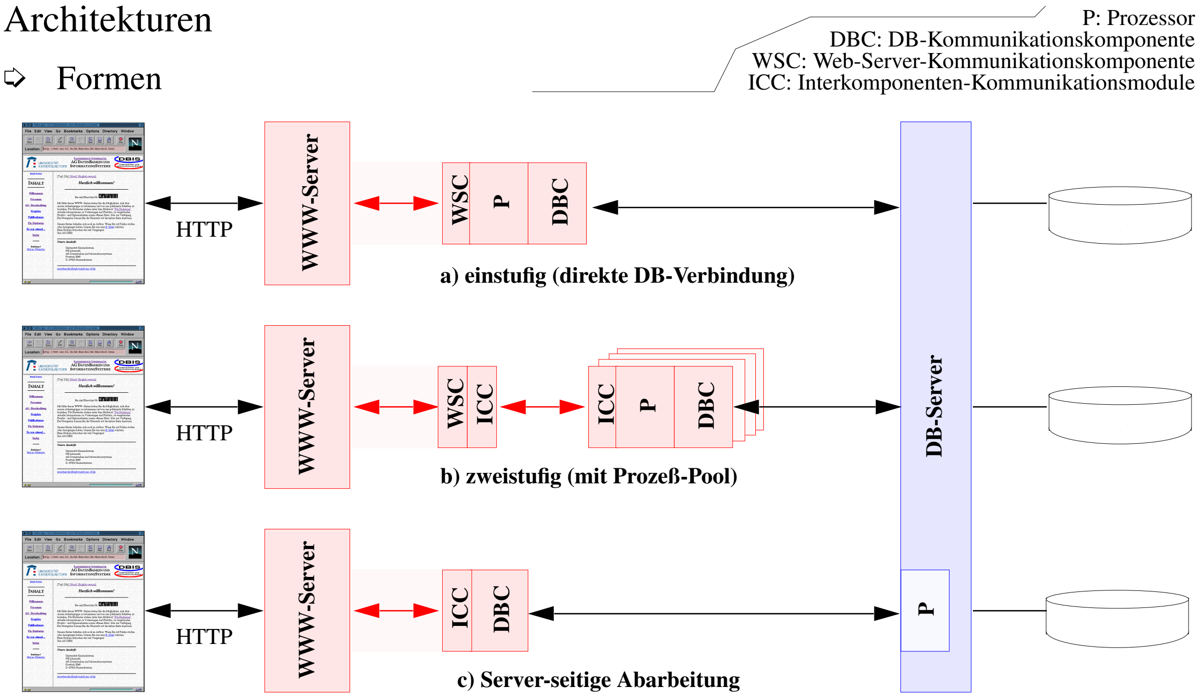

HTTP-basierte Ansätze (7)

- SQL/HTML-Integration (Forts.)
  - Vorteile/Nachteile

	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)

HTTP-basierte Ansätze (8)

- ❑ Architekturen
  - ➪ Formen



HTTP-basierte Ansätze (9)

- ❑ Architekturen (Forts.)
  - ➪ Vergleich der CGI-Architekturen

	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

HTTP-basierte Ansätze (10)

- DB-Server-Integration
  - Speicherung von HTML- und Macrodateien in der DB
    - Vorteile:
      - Makro-Prozessor in DB-Server integriert
        → Zugriff über optimierte Speicherungsstruktur unter Ausnutzung des DB-Puffers ist deutlich effizienter als Zugriff auf das Dateisystem
      - zentrales Repository
        → System leichter administrierbar
      - Web-gestützte Erstellung und Wartung über HTML-Formulare möglich
    - Nachteile:
      - zusätzliche Belastung des DB-Servers (kann zum Engpaß werden)
      - Ablage der Makro- und HTML-Dateien im DB-Server
        → schwierigere Wartbarkeit der Dateien
      - Makro-Prozessor nicht in DB-Server integriert
        → Zugriff auf Dateien im Normalfall effizienter

HTTP-basierte Ansätze (11)

- Realisierung von Zuständen
  - Mehrschrittvorgänge
    → client-seitiger Zustand (Kontext) muß server-seitig in der DB gespeichert werden;
    → es werden *Session-ID* und *User-ID* benötigt
  - Techniken
    - Formularvariable
      - in HTML-Formular wird Session-ID als versteckte Eingabevariable eingetragen
        <INPUT TYPE=HIDDEN NAME=SID VALUE=4711>
      - Wert wird zusammen mit den Benutzereingaben an Web-Server übermittelt, um dort Verbindung zum jeweiligen Vorgang herzustellen
      - Vorteil: für alle Client-Konfigurationen einsetzbar, da durch jeden Browser und jede Browser-Einstellung unterstützt
      - Nachteil: Zwang zu dynamischen HTML-Dokumenten, da Session-ID in alle beim Benutzer angezeigten HTML-Dokumente, die Folgeoperationen auslösen können, eingetragen werden muß
        → Erhöhung der Antwortzeiten
        → Erschwerung der Anwendungsentwicklung
      - nur bedingt einsetzbar für dauerhafte Benutzer-Identifikation; Benutzer müßte sich für jede neue Sitzung initial mit einem URL mit einkodierter Anfrage (“?SID=4711”) an den Web-Server wenden

HTTP-basierte Ansätze (12)

❑ Realisierung von Zuständen (Forts.)

➪ Techniken (Forts.)

- URL-Kodierung
  - Kodierung der Session- oder User-ID in den URL ("/news/4711/ueberblick.html")
    → Web-Server muß Anfrage in Aufruf eines CGI-Programms umsetzen, das die ID ("4711") und das angeforderte Dokument / die gewünschte Aktion extrahiert und das gewünschte Ergebnis herbeiführt
  - Nutzung einer relativen Adressierung für lokale Hyperlinks
    → Web-Server ergänzt dann automatisch die fehlenden Teile des URL (z. B. "/new/4711/")
    → erleichtert server-seitige Verarbeitung, da keine IDs in die Dokumente generiert werden müssen
    → Umsetzung jedoch aufwendig

HTTP-basierte Ansätze (13)

❑ Realisierung von Zuständen (Forts.)

➪ Techniken (Forts.)

- HTTP-Cookie
  - Cookies werden bei jedem Web-Server-Kontakt automatisch durch Browser mitgeschickt und beim Web-Server temporär gespeichert
  - können vielfältig genutzt werden
  - Beispiel:
    Set-Cookie: KNR="4711"; Version="1"; Path="/katalog"; Max-Age="1800"
    → bei jeder Dokumentenanforderung an den Web-Server mit dem Unterverzeichnis *katalog* wird das Cookie *Cookie: KNR=4711* mit übertragen
    → jedoch nur für 1800 Sekunden gültig
  - Nachteil: es gibt Browser, die keine Cookies unterstützen; Cookies können vom Benutzer 'abgeschaltet' werden
- HTTP-Authentisierung
  - über Umgebungsvariable REMOTE_USER kann CGI-Programm Anfrage zuordnen
  - Vorteil: automatische Unterstützung durch Browser und Web-Server
  - Nachteil: vorherige Registrierung der Benutzer und Authentisierung vor jeder Sitzung notwendig

HTTP-basierte Ansätze (14)

- ❑ Realisierung von Zuständen (Forts.)
  - ➯ Problem aller Techniken
    - geeignete Timeout-Werte
      - zur server-seitigen Unterbrechung einer Web-Sitzung bei Untätigkeit des Benutzers
      - wichtig, um nicht benötigte Ressourcen wieder freizugeben
      - insbesondere bei zweistufigen Lösungen wichtig
      - geeignete Timeout-Werte sind applikationsabhängig

HTTP-basierte Ansätze (15)

- ❑ Realisierung von Zuständen (Forts.)
  - ➯ Vergleich

	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

Applet-basierte Ansätze (1)

❑ Neue Möglichkeiten durch Java !

- während HTTP-basierte Verfahren nur server-zentrierte Anwendungen ermöglichen, können nun Anwendungsmodule zur Laufzeit in den Client geladen und dort ausgeführt werden
- Umweg über Web-Server nicht mehr nötig beim Zugriff auf Applikations- bzw. DB-Server

❑ Bewertung

- Grafik
  - HTML unterstützt nur alphanumerische ‘Geschäftsdaten’ in Tabellenform
  - Java kann aufgrund seiner Grafikfähigkeiten auch komplexe Daten, wie Geometrie-/CAD-Daten visualisieren
  - Datenaufbereitung kann durch Applet vorgenommen werden
  - Nachteile:
    - Programm mit Präsentationslogik muß vorher in den Browser geladen werden
    - gesamte Benutzerschnittstelle muß in Java programmiert sein
  - Vorteil:
    - transiente Speicherung von Daten im Programm, lange Transaktionen einfacher realisierbar

Applet-basierte Ansätze (2)

❑ Bewertung (Forts.)

- Netzwerkverbindung, Sicherheit, Zuverlässigkeit
  - Möglichkeit, innerhalb eines Applets Netzwerkverbindungen zu öffnen, z. B. zu
    - DB-Server (❺)
    - einem aus Sicherheitsgründen zwischengeschalteten Verbindungs-Server (❻, ❼)
  - es können prinzipiell beliebig lange Transaktionen und Nutzung von 2PC realisiert werden
  - Nachteil: Java-Sicherheitskonzept
    - ein Applet darf Netzwerkverbindungen nur zu dem Rechner aufmachen, von dem es geladen wurde
      → Web-, DB- und Verbindungsrechner müssen auf demselben Rechner angesiedelt sein, der dann zum Flaschenhals werden könnte
  - durch Nutzung signierter Applets und Gewährung des Verbindungsrechts können jedoch auch DB- und Verbindungs-Server angesprochen werden, die nicht auf dem Rechner des Web-Servers angesiedelt sind

Applet-basierte Ansätze (3)

- Bewertung (Forts.)
  - Ladezeiten
    - im Verhältnis zu HTML höhere initiale Ladezeiten, da Applet (Programmlogik, Benutzerschnittstelle) von Web-Server auf Browser geladen werden muß
    - Abhilfe: persistenter Programm-Cache
      - JAR-Dateien in Kombination mit signierten Applets
        → Applets können client-seitig persistent gehalten werden
    - Alternative: Nutzung von Java-Interfaces, für die erst zur Laufzeit Implementierungen nachgeladen werden

Applet-basierte Ansätze (4)

- JDBC
  - Beispiel: Java-Programm mit JDBC

```
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) {
   ...
```

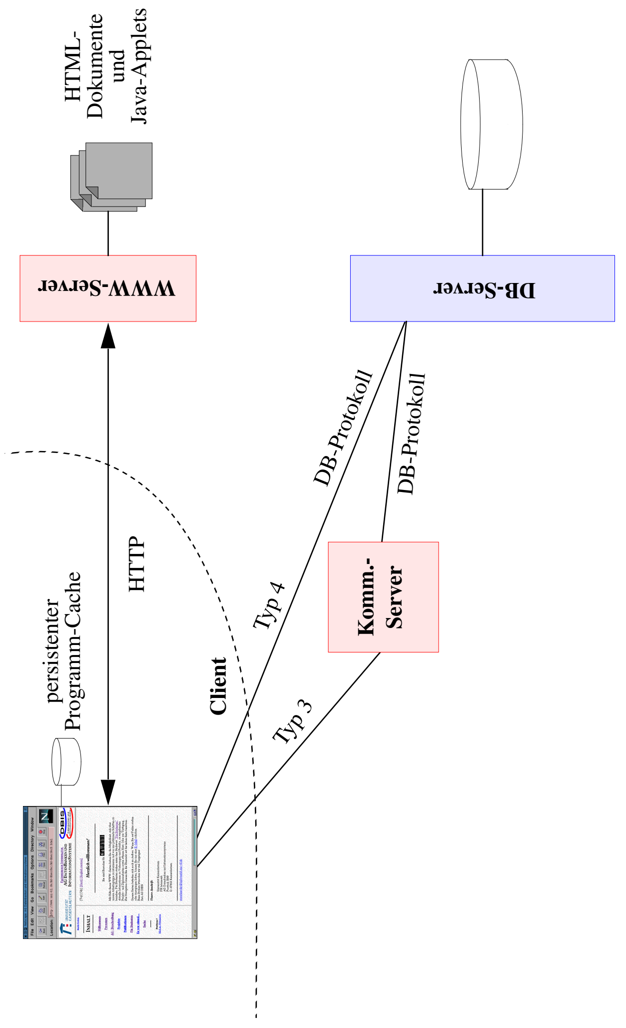
Applet-basierte Ansätze (5)

- SQLJ
  - Beispiel: Java-Programm mit SQLJ

```
// 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) {
   ...
```


Applet-basierte Ansätze (6)

- Architektur und Ablauf (bei Nutzung von JDBC/SQLJ)



Applet-basierte Ansätze (7)

❑ Vergleich: JDBC und SQLJ

	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

Applet-basierte Ansätze (8)

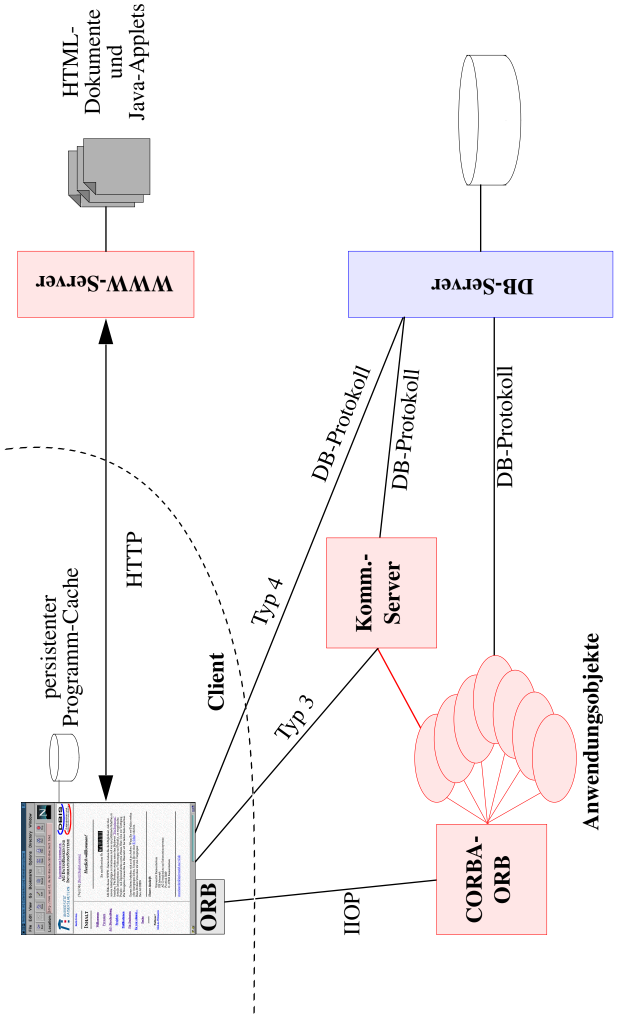
❑ Nutzung von CORBA

➪ Allgemeines

- seit CORBA 2.2 gibt es *Java Language Binding*, das von zahlreichen ORBs (*Object Request Broker*) mit Java-Unterstützung angeboten wird
  (Language Bindings dienen im wesentlichen der Spezifikation von Schnittstellen in IDL)
- seit Version 4 ist in den Netscape-Browser ein ORB integriert
- *Java-IDL*
  - zu CORBA konformer ORB, der über das Kommunikationsprotokoll IIOP (*Internet Inter Orb Protocol*) mit anderen, z. B. server-seitigen, ORBs kommunizieren kann
  - durch Aufnahme der *Java-IDL* in das *JDK 1.2* steht in allen Java-fähigen Browsern ein ORB zur Verfügung
    → Einladen der gesamten CORBA-Laufzeitumgebung in den ORB entfällt
- erhöhte Sicherheit, da Abschirmung des DB-Servers über Zwischenstufe des ORB
- CORBA realisiert Interoperabilität von möglicherweise in verschiedenen Programmiersprachen entwickelten Objekten
  → Anwendungsobjekte können z. B. in C++ implementiert werden
  → Ausführungsgeschwindigkeit für Anwendungslogik steigt
  → CORBA gut für große Anwendungssysteme, insbesondere wenn Zusammenarbeit mit bestehenden Systemen oder Skalierbarkeit wichtig

Applet-basierte Ansätze (9)

❑ Architektur



Applet-basierte Ansätze (10)

❑ Weitere APIs und Lösungen

➪ Java Remote Method Invocation (RMI)

- Möglichkeit des ortstransparenten Methodenaufrufs
- Java-basiertes Pendant zu CORBA
- Nutzung des *Serialization-Interface* von Java
  - Umwandlung eines Java-Objektes in einen über das Netz übertragbaren Byte-Strom, der server-seitig vom RMI-Server in zurückgewandelter Form an das Zielobjekt weitergegeben wird
- vgl. *Java-IDL* (ORB in JDK)

➪ ODMG Java-Binding

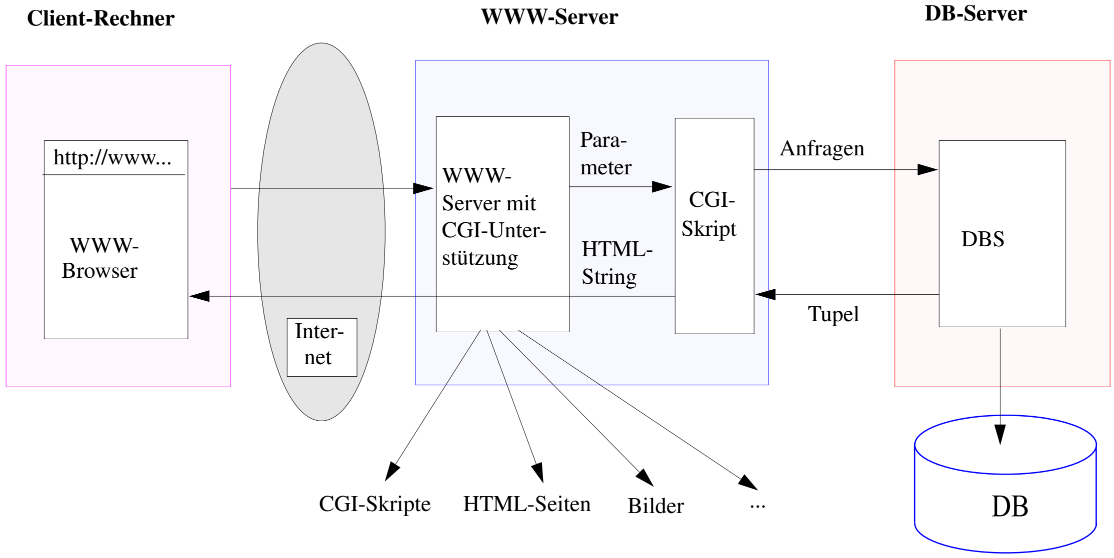
- legt fest, wie in ODL (*Object Definition Language*, Obermenge der IDL) definierte Objekte in Java zur Verfügung gestellt werden können
- API für die Abarbeitung von OQL-Anweisungen (*Object Query Language*) und Handhabung ihrer Ergebnisse
- geeignet für die Anbindung von OODBS

➪ andere Java-DB-APIs

- proprietäre DBVS-Hersteller-spezifische Java-APIs
- standardisierte ODMG-Lösung ist vorzuziehen
- schlecht portierbar

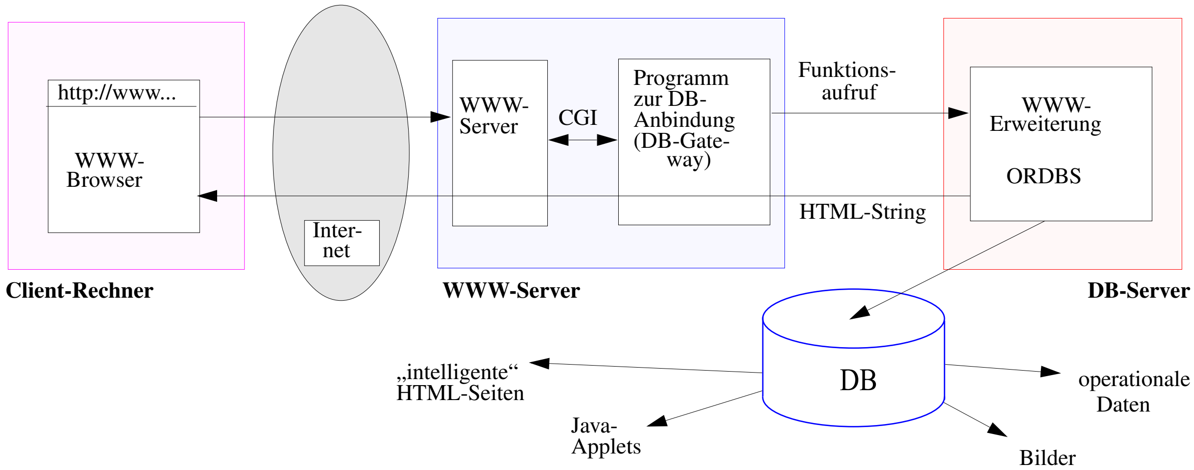
Spezifische (OR)DBS-Erweiterungen (1)

❑ Verwaltung der 'eigentlichen' Web-Dokumente über Web-Server



Spezifische (OR)DBS-Erweiterungen (2)

❑ ORDBVS
- Verwaltung neuer Datentypen
- Zugriff auf externe Daten
- Spezifische Erweiterungen für Web-Zugriff

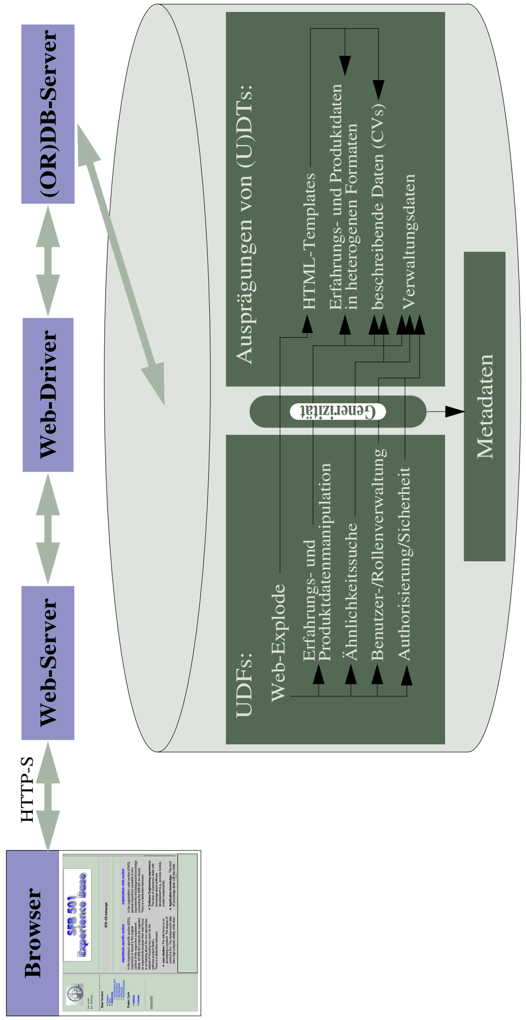


Spezifische (OR)DBS-Erweiterungen (3)

❑ ORDBVS (Forts.)

⇨ Spezifische Erweiterungen für Web-Zugriff (Forts.)

• Beispiel: Erfahrungsdatenverwaltung mit Web-DataBlade von IDS/UDO



Zusammenfassung

❑ Bedeutung der vorgestellten Konzepte wird (wohl) weiter steigen, da Web-basierte Anwendungen immer wichtiger werden

⇨ Web-Browser setzten sich als einfache Benutzerschnittstellen durch

⇨ Netzwerke werden immer leistungsfähiger

❑ HTTP-basierte Anwendungen eignen sich für viele Typen von Anwendungen

❑ Für Anwendungen mit besonderen Anforderungen, z. B. grafische Interaktion, sind Applet-basierte Ansätze (besser) geeignet

❑ Bedeutung von DBS in diesen Web-basierten Anwendungssystemen wird immer höher

⇨ Verwaltung der Anwendungsdaten

⇨ Speicherung von HTML-Dokumenten selbst

⇨ SQLJ: genormte Ablage von Java-Methoden als Stored-Procedures

⇨ besonders ORDBVS wegen Erweiterbarkeit geeignet

- Aufnahme neuer Datentypen
- Integration neuer Verarbeitungsfunktionalität

⇨ *Integrierte Lösungen*