Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Technische Universität Kaiserslautern

Chapter 6
Web-based Information Systems

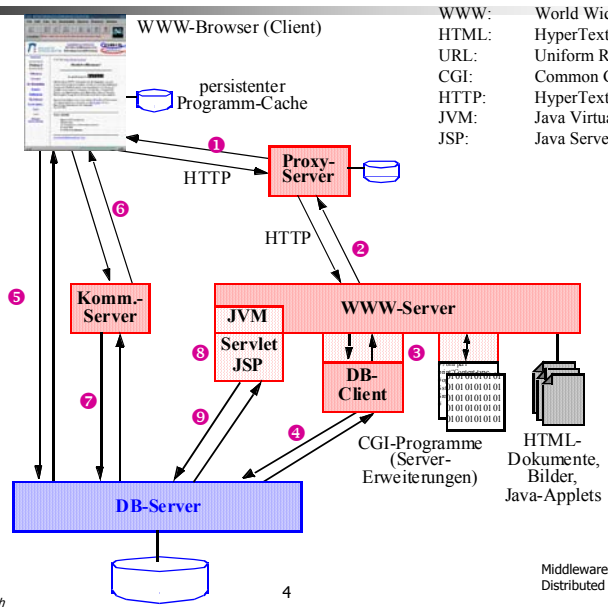
Role of the WWW for IS

- Initial purpose: sharing information on the internet
  - technologies
    - HTML documents
    - HTTP protocol
  - web browser as client for internet information access
- For Information Systems: connecting remote clients with applications across the internet/intranet
  - "web-enabled" applications
    - extend application reach to the consumer
    - leverage advantages of we technologies
  - web browser as a universal application client
    - "thin client"
    - no application-specific client code has to be installed
  - requirements
    - content is coming from dynamic sources (IS, DBS)
    - request to access a resource has to result in application invocation
    - session state: tracking repeated interactions of the same client with a web server

Detour: What is HTTP?

- HTTP is a simple request/response protocol
- Client: Program that submits a request
  - GET /MyPath/MyHomePage.html HTTP/1.1
    Host: www.myserver.com
    Accept: text/html
    User-Agent: IE 5.5
  - POST /myFunctions/reverse HTTP/1.1
    Host: www.myserver.com
    Content-Type: text/plain
    Content-Length: 12
    Hello, World
- Server: Program that processes a request and returns a response (if applicable)
  - HTTP/1.1 200 OK
    Content-Type: text/html
    <HEAD>
    <TITLE>My Homepage</TITLE>
    </HEAD>
- No state is maintained between request/response pairs
- Based on TCP, connection oriented
- Can easily reach across firewalls

Übersicht



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

Server-Komponenten

- 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

Server-Komponenten (2)

- 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) oder speziell markierte Dokumente werden nicht gepuffert
- Kommunikationsserver
  - kann eingesetzt werden, um Verbindungsweg zum DB-Server für Java-Applets (❻, ❼) herzustellen

Grundlegende Techniken

- HTTP-basiert
  - CGI-Programme
  - Server-API
  - Server-Side-Includes
  - Java Servlets
- Applet-basiert
  - Java Applets

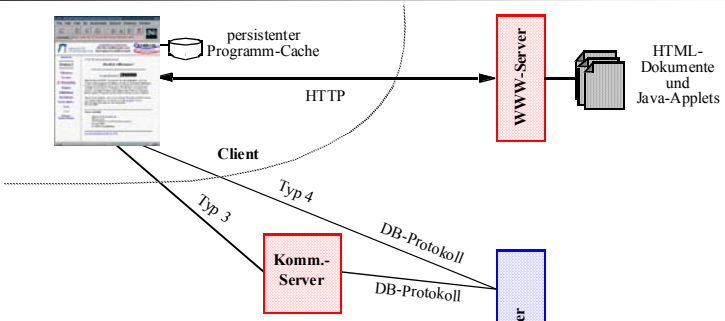
Java Applets

- 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, ab 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

Applet-basierte Ansätze

- Ziel: anwendungsspezfische, dynamische clients in den web browser integrieren
- Möglichkeiten
  - Anwendungsmodule können zur Laufzeit in den Client geladen und dort ausgeführt werden
  - Zugriff auf Applikations- bzw. DB-Server durch das Applet

DB-Zugriff - Architektur und Ablauf

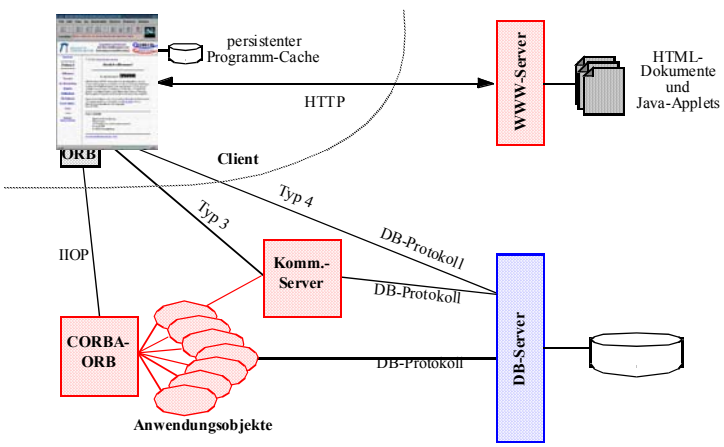


- Vergleich bei Nutzung von JDBC/SQLJ
  - Typ 3: erhöhte Sicherheit, Lastbalancierung über Verbindungs-Server, aber: längere Antwortzeiten durch Umweg über Verbindungs-Server
  - Typ 4: schnelle Kommunikation, aber: ohne Signed Applets Beschränkung hinsichtlich der Platzierung des DB-Servers, Sicherheit

Interaktion mit Anwendungskomponenten

- CORBA
  - 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 IDL-PL-Abbildung)
  - Java-IDL
    - zu CORBA konformer ORB, der über *IIOP (Internet Inter Orb Protocol)* mit anderen, z. B. server-seitigen, ORBs kommunizieren kann
  - durch Aufnahme der Java-IDL in das JDK (ab V1.2) steht in allen Java-fähigen Browsern ein ORB zur Verfügung
    - Einladen der gesamten CORBA-Laufzeitumgebung in den ORB entfällt
- Java Remote Method Invocation (RMI)
  - vgl. Java-IDL (ORB in JDK)
  - Interoperabilität mit CORBA
    - RMI over IIOP

Architektur – CORBA-Interaktion



Applet-basierte Ansätze

- Weitere APIs und Lösungen (Forts.)
  - 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

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, die einzelne Interaktionen überdauern

Bewertung (Forts.)

- Netzwerkverbindung, Sicherheit, Zuverlässigkeit
  - Möglichkeit, innerhalb eines Applets Netzwerkverbindungen zu öffnen, z. B. zu
    - DB-Server (❺)
    - Verbindungs-Server (❻, ❼)
    - Anwendungsserver
  - es können prinzipiell beliebig lange Transaktionen und Nutzung von 2PC realisiert werden
- Problembereiche
  - Java-Sicherheitskonzept
    - ein (nicht signiertes) Applet darf Netzwerkverbindungen nur zu dem Rechner aufmachen, von dem es geladen wurde
    - Web-, DB- und Verbindungsrechner müssten dann 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 → dieser Rechner ist dann nicht länger Flaschenhals
  - Freigabe der Verbindungsrechte zu den Serversystemen (Sicherheit)

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

HTTP-basierte Ansätze

- Idee
  - Web Server kann aufgrund einer Client-Anforderung eine Programmkomponente ausführen, welche die benötigte Resource (z.B. HTML-Dokument) dynamisch erzeugt
- Ansätze
  - CGI-Programme
  - Server-API
  - Server-Side-Includes
  - Java Servlets, Java Server Pages (JSPs)

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 (wartenden) WWW-Server zurück
- WWW-Server leitet das vom CGI-Programm erzeugte HTML-Dokument an den Web-Browser zurück

Server-API (für Server-Erweiterungen)

- 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

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

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 Hersteller-unabhängige 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
- Primärer Ansatz zur Realisierung von Webanwendungen in J2EE
  - Web-Anwendungsserver integriert Unterstützung für, Interaktion von Webkomponenten (e.g., Servlets) und Anwendungskomponenten (EJBs)

SQL/HTML-Integration

- Beispiel: Programmierung (hier mit Perl-Skript)

```
#!/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”);

. . .
```

SQL/HTML-Integration (Forts.)

- Beispiel: Programmierung (hier mit Perl-Skript, Forts.)

```
. . .
# 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
# [...]
```


SQL/HTML-Integration (Forts.)

- Beispiel: direkte Integration

```
<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>
```

SQL/HTML-Integration (Forts.)

- Beispiel: 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> %}
%}
```

. . .

SQL/HTML-Integration (Forts.)

- Beispiel: Makro-Programmierung (Forts.)

. . .

```
# Ausgabesektion beginnt hier
%HTML(REPORT){
<HTML><HEAD><TITLE>Bookmark-DB</TITLE></HEAD>
<BODY>
<H1>Bookmark-DB - Anfrageergebnis</H1>
<!-- Ergebnis ausgeben -->
@bquery()
</BODY></HTML>
%}
```

SQL/HTML-Integration (Forts.)

- Vorteile/Nachteile

	Programmierung	**Integration**	**Makro**
Vorteile	Schnelles und evtl. kleines Programm; Optimal auf jeweilige Anforderung angepasst.	Lesbarkeit durch Platzierung der SQL-Befehle an den späteren Ort der Daten; Erstellung und Wartung mit HTML-Editor möglich; Zugriff auf nur eine Datei.	HTML-Datei über HTML-Editor wartbar *; Übersichtliche Spezifikation aller SQL-Anweisungen; Mehrfachverwendung von SQL-Anfragen möglich *; Schnelles Parsen, da deutlich kleinere Datei *.
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 *.

** Im Falle getrennter Makro- und HTML-Dateien*

HTTP-basierte Ansätze - 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 Password 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

HTTP-basierte Ansätze - Sicherheit (Forts.)

- Zugriffsschutz des Web-Servers (Forts.)
  - 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*)
- Übertragungssicherheit
  - Verschlüsselungsverfahren
    - SHTTP: *Secure HTTP*
    - SSL: *Secure Socket Layer,* 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

HTTP-basierte Ansätze - Zustandslosigkeit

- 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

Realisierung von Zuständen

- Mehrschrittvorgänge (z.B. Führen eines Warenkorbs)
  - client-seitiger Zustand (Kontext) muss server-seitig in der DB gespeichert werden;
  - es werden *Session-ID* und *User-ID* (erlaubt Verzicht auf jeweils neue Authentifizierung) benötigt
- Techniken
  - Formularvariablen
  - URL-Kodierung
  - HTTP-Cookies
  - HTTP-Authentisierung
- Nutzung der Techniken
  - explizit durch den Programmierer
  - impizit über höherwertige Schnittstellen
    - Bsp.: HttpSession-Interface für Servlets

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

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 in erzeugten Dokumenten
  - Web-Browser ergänzt dann automatisch die fehlenden Teile des URL (z. B. "/news/4711/")
  - erleichtert server-seitige Verarbeitung, da keine Ids in die Dokumente generiert werden müssen
- Umsetzung jedoch aufwendig, personalisierte URL nicht benutzerfreundlich

HTTP-Cookies

- Cookies sind unabhängig vom eigentlichen Web-Dokument
- werden mit der Meta-Information zu einer HTML-Seite vom Server zum Browser übertragen und dort temporär gespeichert
- werden (solange gültig) bei jedem Web-Server-Kontakt automatisch durch Browser mitgeschickt
- Beispiel:
  - Zeile *Set-Cookie: KNR="4711"; Version="1"; Path="/katalog"; Max-Age="1800"* wird mit einem Dokument zum Browser übertragen
  - 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

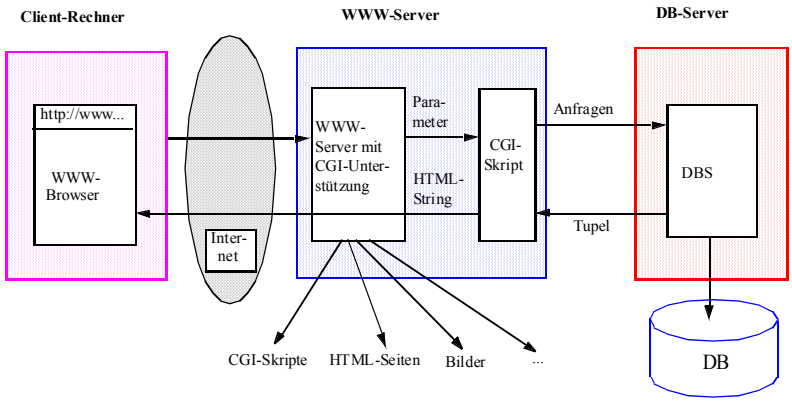
Vergleich

	Formular-variable	**URL-Kodierung**	**HTTP-Cookie**	**HTTP-Authentisierung**
Vorteile	Unabhängigkeit von Browser-Typ und Benutzereinstellungen.	Unabhängigkeit von Browser-Typ und Benutzereinstellungen.	Automatische Unterstützung durch den Browser; Einsatz unabhängig von der Kodierung in HTML-Seite.	Automatische Unterstützung durch Browser und Web-Server; Einsatz unabhängig von der 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.	Benutzerregistrierung erforderlich.

- 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

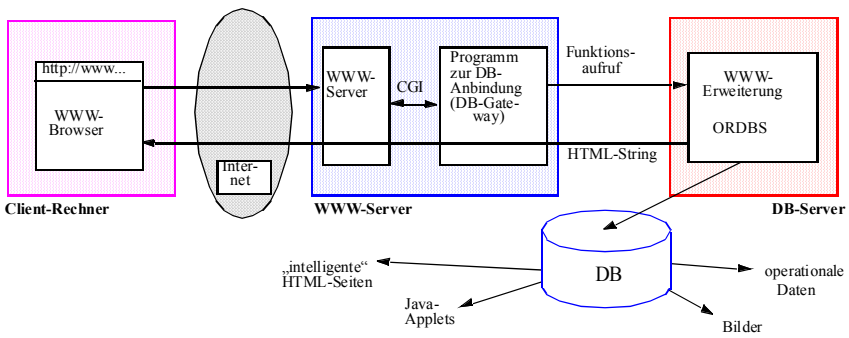
Spezifische ORDBS-Erweiterungen

- Verwaltung der Web-Dokumente durch Web-Server



Spezifische ORDBS-Erweiterungen (Forts.)

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



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 Web-basierten Anwendungssystemen
  - Verwaltung der Anwendungsdaten
  - Speicherung von HTML-Dokumenten selbst
  - besonders ORDBVS wegen Erweiterbarkeit geeignet
    - Aufnahme neuer Datentypen
    - Integration neuer Verarbeitungsfunktionalität
    - Spezifische (kommerziell verfügbare) Erweiterungen für Web-Zugriff
  - Integrierte Lösungen