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

Technische Universität Kaiserslautern

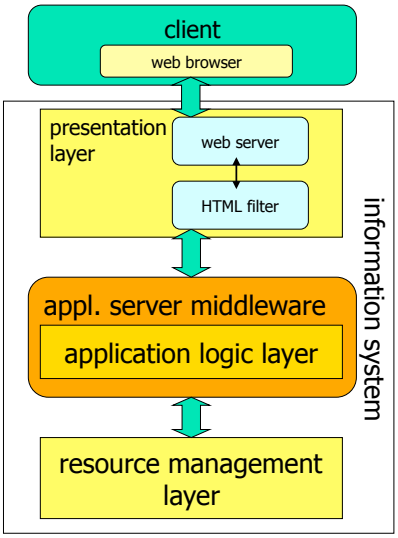
Chapter 10
Web-based Information Systems

Role of the WWW for IS

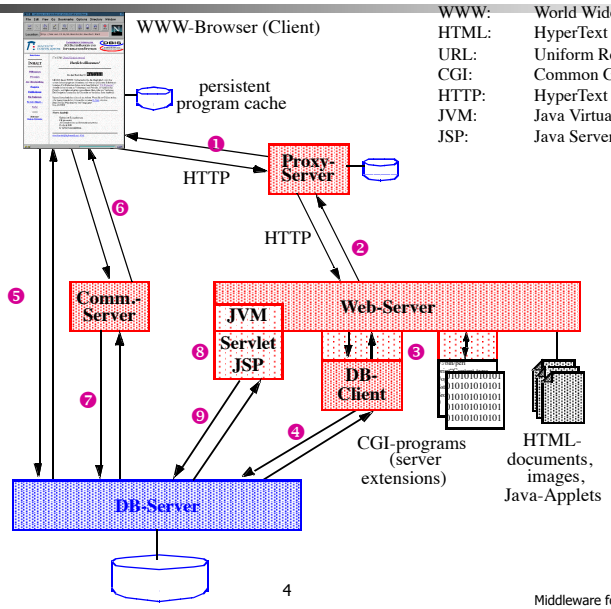
- Initial purpose: sharing information on the internet
  - technologies
    - HTML documents
    - HTTP protocol
  - web browser as client for internet information access
- For Information Systems: connecting remote clients with applications across the internet/intranet
  - "web-enabled" applications
    - extend application reach to the consumer
    - leverage advantages of web 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

Architecture for Web-IS

- Presentation layer may be realized in separate tiers
  - client-side presentation using browser, *client components (optional)*
  - server-side presentation done by web server, dynamic HTML generation (HTML filter)
- Presentation components interact with application logic components
  - managed by appl. server, or *run within web server environment*
- Access to RM layer
  - "encapsulated" in appl. logic component
  - may also be performed directly *within presentation logic component*
    - special case



Overview



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 Components

- WWW-Server
  - core component
  - provides static HTML pages, incl. embedded images, etc. (❶, ❷)
  - provides Java applets, which may access a DB either directly (❺) or via a communication server (❻, ❼)
  - invokes server-side extensions (❸)
  - invokes CGI programs (❸)
  - invokes Java servlets, Java Server Pages (JSPs) (❽)
  - delivers results of DB interactions (CGI programs ❹, Java servlets ❾) as dynamically generated HTML to web browser

Server Components (2)

- DB-Server
  - manages application data
  - may manage static HTML pages (or fragments)
- Proxy-Server
  - caches results (HTML documents, images) of an HTTP request to improve response time for static information requests
  - dynamically generated or specially marked documents are not cached
- Communication-Server
  - can be used to support DB-Server connectivity for Java applets (❻, ❼)
- Application-Server (not included in our chart!)

Core Technologies

- Client-side
  - Java Applets
- Server-side
  - Common Gateway Interface (CGI) programs
  - Web Server API
  - Server-side Includes (SSI)
  - Java Servlets
  - Java Server Pages (JSP)

Client-Side Approaches

- Goal: application-specific, dynamic clients integrated into the web browser
- Capabilities
  - application modules can be downloaded (at run-time) to the client and executed
    - don't need to be pre-installed prior to invocation
  - module can access application server or DB server
    - may result in performance benefits
  - module can manage state across multiple interactions

Java Applets

- Application component embedded in HTML page (similar to images), stored on the web server
- Dynamically transferred to the client to be executed in a Java-enabled web browser (❶, ❷)
  - requires JVM integrated into web browser or loaded by Java plug-in
- Applets can use full Java language support
- JAR files (Java ARchive) can be used to package all class files needed by an applet for download over the network
- General security restrictions (untrusted applets)
  - no access to local resources
  - network communication restricted to server of origin
- Signed Applets
  - Security concept for applets (since JDK 1.1)
  - JAR file contains digitally signed applet files and digital certificate
    - guarantee that applet has not been modified after signed by providing party
    - client may trust applets, grant permissions based on certificates
  - Can be stored persistently on the client side

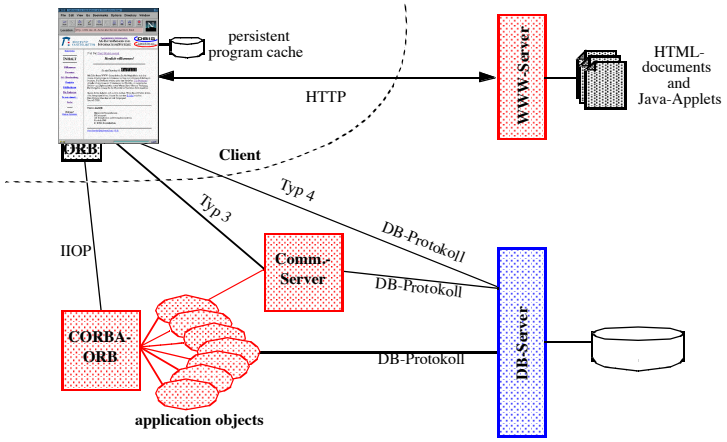
Applet-based DB Access

- Using JDBC/SQLJ
  - Type 3 driver
    - improved security
    - connection server for load balancing
    - works with unsigned applets
    - communication overhead (longer response times)
  - Type 4 driver
    - direct communication (i.e., connection server overhead)
    - requires either signed applets or identical DB-server and web server location
- Additional alternatives and APIs
  - ODMG Java-Binding
    - for access OODBMS
    - defines language binding for ODL (Object Definition Language) to Java
    - API for executing OQL(Object Query Language) statements and processing of results
  - other Java-DB APIs
    - proprietary, DBMS-vendor-specific Java-APIs
    - applet implementations not portable

Interaction with Application Components

- CORBA
  - since CORBA 2.2: *Java Language Binding*, supported by numerous ORBs with Java support
  - Java-IDL
    - CORBA-compliant ORB, can communicate with server objects and server-side ORBs using *IIOP (Internet Inter Orb Protocol)*
    - available in all Java-enabled browsers as part of JDK 1.2
      - avoids downloading CORBA runtime
- Java Remote Method Invocation (RMI)
  - interoperability with CORBA/EJB
    - RMI over IIOP

Applet-based Architecture for Web-IS



Evaluation: Advantages

- Enhanced UI-support
  - HTML only supports presentation of alpha-numeric data, potentially in tabular form
  - applets can leverage Java to process data and visualize complex data structures (e.g., geometry/CAD data)
    - complete UI has to be implemented in Java
    - transient storage of state within applet, across multiple user interactions
- Connectivity, transactions
  - applet may connect directly to
    - DB-Server (❺)
    - Connection-Server (❻, ❼)
    - Application-Server
  - applet state can preserve DB-connections across interactions
    - long, multi-step transactions
    - distributed transactions

Evaluation: Disadvantages

- Loading time
  - higher initial loading time due to downloading applet (application logic, UI) from web server
  - solution: persistent program cache
    - in combination with signed applets
    - applets can be held persistently at client
  - alternative: use of Java interfaces, delaying download of implementation
- Java security
  - unsigned applet can only connect back to server of origin
    - web, DB/connection server have to reside on the same machine (-> bottleneck)
  - this restriction can be avoided by using signed applets and appropriate client security policies
- No adequate support for combined client/server-side transactions
  - passing transaction context to server-side web components not supported by HTTP
- Requires enabling/allowing connection to server systems from web clients
  - may be suitable for the intranet
  - questionable for internet scenarios (security, firewalls, …)

Client-Side Processing

- JavaScript
  - object-oriented scripting language (syntax similar to Java)
  - can be embedded in or referenced from within a web page
  - script can interact with the Document Object Model (DOM) of the web page
    - can manipulate the page/HTML
  - can react to user events
- Web browser
  - needs to support the JavaScript language
  - interprets the script elements
- Main use
  - implement user interface logic of a web page
  - validate web forms
  - make web pages more interactive, dynamic and responsive

Ajax

- Originally a shorthand for „Asynchronous JavaScript and XML"
- Indicates a set of technologies used for web applications
  - presentation based on XHTML, CSS
  - dynamic display and interaction using Document Object Model
  - data interchange and manipulation using XML, XSLT
  - asynchronous data retrieval using XMLHttpRequest
  - JavaScript (binding everything together)
- Ajax „variations"
  - use of another scripting language (e.g., VBScript)
  - data interchange based on JavaScript Object Notation (JSON) or others
- Main benefits
  - avoids reloading complete pages, only interacts with server for subset of the content or avoids interactions altogether
  - user perceives web application as faster, due to asynchronous loading of data
  - state preserved inside JavaScript variables across multiple interactions
    - main container page is not reloaded

Server-side Approaches

- Idea
  - web server can execute program component based on client request
    - may perform data access operations
    - can interact with other business logic components (e.g., EJBs, JavaBeans, …)
  - program dynamically generates required resource (e.g., HTML document)
- Approaches
  - CGI programs
  - Server API
  - Server-Side-Includes
  - Java Servlets, Java Server Pages (JSPs)

Common Gateway Interface (CGI) Programs

- Dynamic generation of HTML documents based on CGI and HTML forms
- Web server starts CGI program in a separate process
- CGI program inspects environment variables set by web server
- Web server communicates parameters provided in HTML forms to CGI program in a well-defined manner (❸)
- CGI program can access DB-server using DB client APIs (❹)
- CGI program generates HTML document and returns it to the web server as the result of the program execution
- Web server passes the resulting HTML back to the client (web browser)

Server API (for Server Extensions)

- Web server vendors provide proprietary APIs to avoid creation of separate process for CGI program
- Examples:
  - NSAPI (Netscape Server API), Netscape
  - ISAPI (Internet Server API), Microsoft
- Means to extend web server capabilities with additional functions (Server Application Function, SAF) that previously had to be realized using CGI
- SAFs are provided as dynamic program libraries, linked to web server at startup time
- Web server can distinguish regular HTML document access from SAF invocation based on URL and configuration data (❸)
- Performance advantage over CGI
  - avoids creation of separate process
  - DB-connection can be kept open

Java Servlets

- SUNs response to server extensions by Netscape, Microsoft for Java-based web server
- Included in JDK 1.2, supported by many web server implementations
- Supports platform-independent and vendor-independent extensibility of web servers
- Primary approach for realizing web applications in J2EE
  - web application server integrates support for and interaction of web components (e.g., servlets) and application components (EJBs)
- Requires integration of JVM in web server (❽) or cooperation of web server with associated JVM process
- Follows the same model as C-based server APIs
- Additional advantage: dynamic binding of Java class loader -> uninterrupted web server execution

Server-Side Includes (SSI)

- Directives included in HTML document as HTML extensions
- Dynamically evaluated by web server when document is requested by client
- Can be used to
  - include current date, time or other status information into the web page
  - invoke applications and OS commands
  - access DB-server
- Web-server-specific extensions

Java Server Pages (JSPs)

- Based on SSI, servlets
  - JSPs are translated (once) into servlets for execution
- Mixes static HTML with embedded JSP constructs for presenting dynamic content
  - scripting elements
    - Java code to be included in servlet
  - directives
    - controls overall structure of servlet
  - actions
    - allow for use of existing components

SQL/HTML-Integration

- Example: Programming (here using Perl-Script)

```
#!/bin/perl
# load Msql-Package:
use Msql;
# generate header:
print"Content-type: text/html\n\n";
# [...]
# connect to DB-Server:
$testdb = Msql->connect;
# select DB:
$testdb->selectdb(„bookmarks“);
# execute query:
$sth = $testdb->query(“select name,url from
   bookmarks where name LIKE ‘Loe%’ order by name”);
. . .
# generate result presentation:
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”;
# generate footer
# [...]
```


SQL/HTML-Integration (cont.)

- Example: direct integration

```
<HTML><HEAD><TITLE>Bookmark-DB</TITLE></HEAD>
<BODY>
<H1>Bookmark-DB – query results</H1>
<!– assign input parameters to internal variables -->
<?MIVAR NAME=iname DEFAULT=Loe>$iname<?/MIVAR>
<!– produce table header -->
<TABLE><TR><TH>Name</TH><TH>URL</TH></TR>
<!– specify query -->
<?MISQL query=”select name,url from
   bookmarks where name LIKE ‘$iname%’ order by name;”>
<!– produce result HTML table -->
<TR><TD><A HREF=”$2”>$1</A></TD><TD>$2</TD></TR>
<?/MISQL>
</TABLE></BODY></HTML>
```

SQL/HTML-Integration (cont.)

- Example: Macro-Programming

```
# specify database
%DEFINE{
  DATABASE="bookmarks"
%}
# define query as a function
%FUNCTION(DTW_SQL) bquery() {
  select name,url from
  bookmarks where name LIKE 'Loe%' order by name;
  # result requires special form of output
  %REPORT{
    <TABLE><TR><TH>Name</TH><TH>URL</TH></TR>
  # determine format of result rows
  %ROW{
    <TR><TD><A HREF="$(V2)">$(V1)</A></TD>
    <TD>$(V2)</TD></TR>%}
  </TABLE>%}
%}
```

. . .

SQL/HTML-Integration (cont.)

- Example: Macro-Programming (cont.)

```
# output section starts here
%HTML(REPORT){
<HTML><HEAD><TITLE>Bookmark-DB</TITLE></HEAD>
<BODY>
<H1>Bookmark-DB – query results</H1>
<!– output results -->
@bquery()
</BODY></HTML>
%}
```

SQL/HTML-Integration (cont.)

- Comparison of approaches

	Programming	**Integration**	**Macro**
Advantage	fast, possibly small program; optimally adjusted to specific requirements.	Improved readability through placement of SQL commands at intended location of result data; HTML editor may support creation, modification of document; Access to only one file.	HTML document can be manipulated using HTML editor *; Organized specification of all SQL statements; Reuse of SQL statements possible *; Faster parsing due to smaller file size *.
Disadvantage	inflexible, changes may require recompilation; separate program for each task.	Mix of SQL and HTML may become confusing for more complex, non-trivial applications.	Readability suffers due to separation of HTML and SQL into different areas/files; Access to multiple files required *.

** If macro and html are kept in separate files*

Session State

- Server-side approaches are based on HTTP
- HTTP is a stateless protocol
  - does not provide direct support for storing information that persists across HTTP interactions
  - problems
    - DB-clients realized with CGI, Server-APIs are only "active" for the duration of a single interaction
      - no transactions across multiple requests
      - new connection has to be obtained for every HTTP request resulting in DB-access
    - negative impact on response times
- State information has to be stored and managed by the application

Managing Session State

- Session may consist of multiple steps (e.g., managing a shopping cart)
  - client state (context) needs to be stored/managed and made available to server components
  - *Session-ID* and *User-ID* (to avoid repeated authentication) required
- Techniques
  - Form variables
  - URL encoding
  - HTTP-Cookies
  - HTTP-Authentication
- Use of the techniques
  - explicitly by the programmer
  - implicitly through higher-level programming interfaces
    - Example: HttpSession-Interface for Servlets

Form Variables

- Session-ID is included into HTML forms as a hidden variable
  <INPUT TYPE=HIDDEN NAME=SID VALUE=4711>
- Value is transmitted to web server together with form input, can be used to establish association with session context
- **Advantage**: can be used with all client configurations, supported for every browser and browser configuration
- **Disadvantage**: forced to use dynamic HTML documents with form submissions for all interactions, because Session-ID needs to be inserted into all HTML documents potentially causing subsequent stateful interactions
  - response times suffer
  - complicates application development

URL Encoding

- Encoding session/user-ID in the URL ("/news/overview;id=4711")
  - pages are dynamic, each URL on the page is "personalized" by the server
  - client request now "re-transmits" the id
  - web server/CGI program needs to extract the ID from the URL and perform the appropriate action
- All pages need to be dynamic
- Overall, realization is complex, personalized URL is not user-friendly

HTTP-Cookies

- Independent of web documents
- Cookies are pieces of text that can be transmitted by the server, together with the meta data of the HTML, stored temporarily at the client
- Automatically included in web server interaction by the browser (until cookie is invalidated)
- Example:
  - the string
    Set-Cookie: ID="4711"; Version="1"; Path="/catalog"; Max-Age="1800"
    is transmitted to browser, together with the HTML document
  - every request to the web server that includes the subdirectory *catalog* will include *Cookie: ID=4711*
  - cookie is valid only for *1800 seconds*
- Disadvantage: cookies may be disabled by the browser/client

HTTP-Authentication

- REMOTE_USER environment variable can be used by CGI program to correlate requests
- Advantage: automatically supported by browser and web server
- Disadvantage: user **registration** and authentication before every session

Comparison

	Form variable	**URL encoding**	**HTTP-Cookie**	**HTTP-Authentication**
Pros	Independent of browser type and user preferences.	Independent of browser type and user preferences.	Automatic browser support; Independent of HTML document.	Automatic browser and web server support; Independent of HTML document.
Cons	Has to be included in every HTML page to be displayed by the browser; Dynamic HTML complicates application development.	Dynamic pages; Complex translation of HTTP requests.	User configuration needs to permit use of cookies.	Requires user registration and authentication.

- APIs can help reduce complexity (e.g., servlet APIs)
- Problem with all techniques: choosing timeout values
  - for server-side termination of session due to inactivity of user
  - important for releasing server resources
  - suitable values for timeout are application-dependent

REST

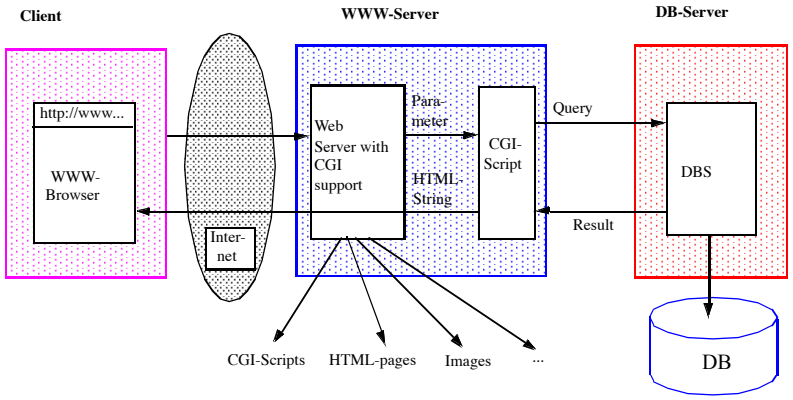
- REST (Representational State Transfer)
  - architectural style for building large-scale distributed hypermedia systems
    - see dissertation by Roy Thomas Fielding (2000)
- Four main principles
  - resource identification through URI
  - uniform interface
    - HTTP GET, PUT, POST, DELETE
  - self-descriptive messages
    - resources are decoupled from their representation
    - content can be accessed in a variety of formats (content negotiation)
  - stateful interactions through hyperlinks
    - no session state maintained on the server
    - explicit state transfer using self-contained messages
- Basis for scalability of the web
  - caching, clustering, load balancing

RESTful Web Services

- Web services provided purely based on the above principles
  - alternative to WSDL/SOAP-based „big" web services
- Rationale
  - perceived to be simple
    - leverages existing, well-known standards (HTTP, XML, URI, MIME)
  - light-weight infrastructure that requires only minimal tooling, is inexpensive
    - similar to building dynamic web site
  - REST and Ajax complement each other nicely
- Drawbacks and limitations
  - (still) needs careful design and enumeration of resources to be exposed, mapped to generic interface, and of data representations used
    - more flexible, but more format variations to account for
  - only RPC-style interactions, HTTP-only
  - requires manual implementation of reliability, transactions

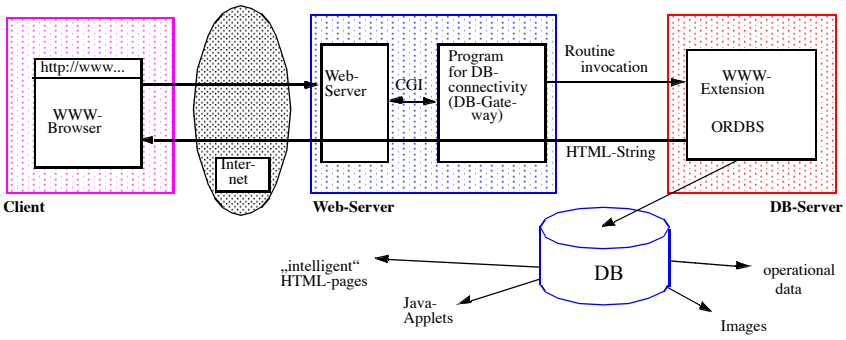
Specific ORDBMS-Extensions

- Web documents managed by web server



Specific ORDBMS-Extensions (cont.)

- ORDBMS
  - Management of new data types
  - Access to external data
  - Specific extensions for web access



Summary

- Web-based applications become ubiquitous, concepts for web-based IS are therefore increasingly important
  - web browser as a simple, uniform user interface
  - networks become more powerful
- Server-side, HTTP-based approaches suitable for wide range of applications
- Client-side, applet-based approaches may be suitable for
  - applications with specific UI requirements
  - in intranet environments
- Role of DBS in web-based application systems
  - Management of application data
  - Management of HTML documents
  - ORDBMS especially suitable (extensibility)
  - Integrated solutions