Chapter 11

Web-based Information Systems
TP Application Architecture

- **Front-end program**
  - interacts with (possibly wide range of) display devices
  - gathers and validates input, displays output
  - constructs and forward request (e.g., as a RPC or asynchronous message)
  - provides **device-independence** for server

- **Request controller**
  - guides the request execution
  - determines required steps, then executes them by invoking transaction servers
  - usually runs as part of an ACID transaction

- **Transaction server**
  - process that runs application programs doing the actual work of the request
  - almost always runs within the scope of an ACID transaction
  - typically interacts with a DBMS
  - simple applications can be composed into more complex ones (using local proc. call, TRPC, async. messaging, …)
  - makes difference to req. controller fuzzy
Front-end Program Functions

- Gather input and display output (user interaction)
  - form and menu concepts
    - user selects a menu item to identify the type of transaction to be executed
    - front-end program display a (series of) form(s) for gathering input data
    - input data is validated by the front-end program
      - goal: avoid calling the TP-server with incorrect input
      - based on general data type/format, possibly using “cached” data values
  - presentation technologies support form and menu concepts
    - define menus and how to navigate between them
    - define fields and screen layout of forms
    - identify data validation routines to be called for each field

- Construct requests and interpret replies
  - request can take the form of an RPC or (a)synchronous message
  - typically includes
    - user identifier
    - device identifier (e.g., network address)
    - request type
    - input parameters
Web-based IS

- Initial purpose of the WWW: 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/front-end program
    - "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
Forms Technologies and Tools

Design time | Run time
---|---
Thin Client
- HTML/XML
- Forms Editor
- DreamWeaver
- Expression Web
- HTML Forms
- XHTML
- Dynamic HTML
- AJAX

Thick Client
- Visual Studio
- Eclipse-based IDE
- NetBeans
- Visual Basic, C#
- Java
- Eclipse RCP

Web Server
Request Controller

End-user device
Presentation layer may be realized in separate tiers
- client-side presentation using browser, \textit{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 \textit{run within web server environment}

Access to RM layer
- "encapsulated" in appl. logic component
- may also be performed directly \textit{within presentation logic component}
  - special case, if there is no application logic (only data access logic)
Overview

HTML: HyperText Markup Language
CGI: Common Gateway Interface
HTTP: HyperText Transfer Protocol
JSP: Java Server Page
Server Components

- **WWW-Server**
  - core component
  - provides static HTML pages, incl. embedded images, etc. (1, 2)
  - provides Java applets, which may access a DB either directly (5) or via a communication server (6, 7)
  - invokes server-side extensions (3)
  - invokes CGI programs (3)
  - invokes Java servlets, Java Server Pages (JSPs) (8)
  - delivers results of TP interactions (CGI programs 4, Java servlets 9) as dynamically generated HTML to web browser
Server Components (2)

- **DB-Server**
  - manages application data
  - may manage static HTML pages (or fragments)
- **Application-Server**
  - manages and executes request controllers and transaction servers
- **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 (6, 7)
Core Technologies

- **Client-side**
  - Java Applets, Javascript

- **Server-side**
  - Common Gateway Interface (CGI) programs
  - Web Server API
  - Java Servlets
  - Server-side Includes (SSI)
  - 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 (1, 2)
  - 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
    - communication server for load balancing
    - works with unsigned applets
    - communication overhead (longer response times)
  - Type 4 driver
    - direct communication
    - 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

Web Browser (Client) -> RMI

Web Server

HTML images
applets

Applic. Server

Communication Server

DB-Server

persistent (applet) program cache

HTTP

IIOP

Type 3

Type 4

application objects

middleware for heterogeneous and distributed information systems

Middleware for Heterogeneous and Distributed Information Systems

© Prof. Dr.-Ing. Stefan Deßloch
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
    - Connection-Server
    - 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 DB 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
  - Java Servlets
  - Server-Side-Includes
  - 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
- CGI program can access DB-server using DB client APIs
- 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 (3)
- 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 (⑧) 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)

- Server-side scripting
  - HTML page templates contain programming instructions that are
    - to be executed by a server program (web server scripting engine)
    - to build the dynamic content (plain HTML) of the web page
  - popular approaches: PHP, Java Server Pages, ...

- JSPs are 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

- Standard/custom tag libraries, invocation of JavaBeans components
  - separation of programming and page design/development concerns and roles
Session Support

- Stateful interactions may depend on the outcome of previous steps
- 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
- Session support is required
  - session tracking: associate all requests of a single user with a session
  - session state: state information has to be stored and managed by the server/application
Session Tracking

- 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
    - included as tokens or conversation identifiers in client/server interactions

- Common techniques
  - Form variables (hidden form fields)
  - 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 in HTML forms as a hidden variable by server
  
  `<INPUT TYPE=HIDDEN NAME=SID VALUE=4711>`

- Value is transmitted back to web server together with form input, can be used to establish association with session context

- Can be used with all client configurations, supported for every browser and browser configuration

- Forces developer 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 as a path parameter ("/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

- Supported in all configurations and browsers
- Again, 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
    
    \[\text{Set-Cookie: ID=\text{"4711\"}; Version=\text{"1\"}; Path=\text{"/catalog\"}; Max-Age=\text{"1800\"}\] 
    
    is transmitted to browser, together with the HTML document
  - every request to the web server that includes the subdirectory catalog will include \text{Cookie: ID=4711}
  - cookie is valid only for \text{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
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

<table>
<thead>
<tr>
<th></th>
<th>Form variable</th>
<th>URL encoding</th>
<th>HTTP-Cookie</th>
<th>HTTP-Authentication</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros</strong></td>
<td>Independent of browser type and user preferences.</td>
<td>Independent of browser type and user preferences.</td>
<td>Automatic browser support; Independent of HTML document.</td>
<td>Automatic browser and web server support; Independent of HTML document.</td>
</tr>
<tr>
<td><strong>Cons</strong></td>
<td>Has to be included in every HTML page to be displayed by the browser; Dynamic HTML complicates application development.</td>
<td>Dynamic pages; [Complex translation of HTTP requests.]</td>
<td>User configuration needs to permit use of cookies.</td>
<td>Requires user registration and authentication.</td>
</tr>
</tbody>
</table>
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
  - server components
  - server-side scripting
- Client-side, applet-based approaches may be suitable for
  - applications with specific UI requirements
  - data access in intranet environments
- Techniques for managing session state
  - URL encoding, HTTP-Cookies most commonly used