The World Wide Web

- Huge distributed system consisting of millions of clients and servers for accessing linked documents
- Started at CERN, European Particle Physics Laboratory in Geneva early 1990's
- Since 1994 controlled by the WWW Consortium (CERT and MIT)
- www.w3.org
Structure of this chapter

- Architecture overview
- Communication
- Processes
- Naming
- Synchronization
- Caching and Replication
- Fault Tolerance
- Security
Overview of WWW

- Servers maintain a collection of documents
- Documents are stored as files or generated on request
- URL (Universal Resource Locator) (cmp. IOR in CORBA)
- Client application: browser
- Browser renders document
The World Wide Web

Overall organization of the Web.

1. Get document request
2. Server fetches document from local file
3. Response

Client machine
- Browser
- OS

Server machine
- Web server
- Database
Document model

- Static and dynamic documents
- Documents can contain references to other documents: hyperlinks
- HTML - Hypertext Markup Language
A simple Web page embedding a script written in JavaScript.

```
<HTML>
<BODY>
<H1>Hello World/H1>
<P>
<SCRIPT type = "text/javascript">
    document.writeln ('<H1>Hello World</H1>);
    // Write a line of text
</SCRIPT>
</P>
</BODY>
</HTML>
```
Document Object Model

- Rooted tree - standardized
- Aka dynamic HTML
- Interface specified in CORBA IDL and translated to various different prog. languages
- HTML vs. XML
- Extensible Markup Language
An XML definition for referring to a journal article.
<?xml version "1.0">
<!DOCTYPE article SYSTEM "article.dtd">
<article>
  <title>Prudent Engineering Practice for Cryptographic Protocols</title>
  <author><name>M. Abadi</name></author>
  <author><name>R. Needham</name></author>
  <journal>
    <jname>IEEE Transactions on Software Engineering</jname>
    <volume>22</volume>
    <number>12</number>
    <month>January</month>
    <pages>6 – 15</pages>
    <year>1996</year>
  </journal>
</article>
### Document Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Subtype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Plain</td>
<td>Unformatted text</td>
</tr>
<tr>
<td></td>
<td>HTML</td>
<td>Text including HTML markup commands</td>
</tr>
<tr>
<td></td>
<td>XML</td>
<td>Text including XML markup commands</td>
</tr>
<tr>
<td>Image</td>
<td>GIF</td>
<td>Still image in GIF format</td>
</tr>
<tr>
<td></td>
<td>JPEG</td>
<td>Still image in JPEG format</td>
</tr>
<tr>
<td>Audio</td>
<td>Basic</td>
<td>Audio, 8-bit PCM sampled at 8000 Hz</td>
</tr>
<tr>
<td></td>
<td>Tone</td>
<td>A specific audible tone</td>
</tr>
<tr>
<td>Video</td>
<td>MPEG</td>
<td>Movie in MPEG format</td>
</tr>
<tr>
<td></td>
<td>Pointer</td>
<td>Representation of a pointer device for presentations</td>
</tr>
<tr>
<td>Application</td>
<td>Octet-stream</td>
<td>An uninterrupted byte sequence</td>
</tr>
<tr>
<td></td>
<td>Postscript</td>
<td>A printable document in Postscript</td>
</tr>
<tr>
<td></td>
<td>PDF</td>
<td>A printable document in PDF</td>
</tr>
<tr>
<td>Multipart</td>
<td>Mixed</td>
<td>Independent parts in the specified order</td>
</tr>
<tr>
<td></td>
<td>Parallel</td>
<td>Parts must be viewed simultaneously</td>
</tr>
</tbody>
</table>

Six top-level MIME types and some common subtypes.
Architectural Overview

• Early enhancement:
  – CGI - Common Gateway Interface
  – User data can come from e.g., HTML form
  – CGI script sophistication: the sky's the limit

• Server-side scripts

• Applets: pass precompiled code to browser

• Servlet: precompiled code executed in server
The principle of using server-side CGI programs.
Architectural Overview (2)

An HTML document containing a JavaScript to be executed by the server
Architectural Overview (3)

Architectural details of a client and server in the Web.
Communication

- HTTP - HyperText Transfer Protocol
- Based on top of TCP
- HTTP 1.0 - nonpersistent
- HTTP 1.1 - persistent connections
- Client can issued pipelined requests
a) Using nonpersistent connections.
b) Using persistent connections
## HTTP Methods

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head</td>
<td>Request to return the header of a document</td>
</tr>
<tr>
<td>Get</td>
<td>Request to return a document to the client</td>
</tr>
<tr>
<td>Put</td>
<td>Request to store a document</td>
</tr>
<tr>
<td>Post</td>
<td>Provide data that is to be added to a document (collection)</td>
</tr>
<tr>
<td>Delete</td>
<td>Request to delete a document</td>
</tr>
</tbody>
</table>

Operations supported by HTTP.
# HTTP Messages (1)

The diagram illustrates an HTTP request message, which consists of the following components:

- **Request line**: This is the first line of the message and contains three parts: operation, reference, and version.

  - **Operation**: The type of operation (e.g., GET, POST).
  - **Reference**: The resource being requested.
  - **Version**: The version of HTTP being used.

- **Request message headers**: These are name-value pairs that provide additional information about the request. They are separated by a delimiter and grouped under a header.

- **Message body**: This section contains the data associated with the request.

### Table: Request Message Headers

<table>
<thead>
<tr>
<th>Message header name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message header name</td>
<td>Value</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Message header name</td>
<td>Value</td>
</tr>
</tbody>
</table>

(a) HTTP request message
HTTP response message.
### HTTP Messages (3)

Some HTTP message headers:

<table>
<thead>
<tr>
<th>Header</th>
<th>Source</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Client</td>
<td>The type of documents the client can handle</td>
</tr>
<tr>
<td>Accept-Charset</td>
<td>Client</td>
<td>The character sets are acceptable for the client</td>
</tr>
<tr>
<td>Accept-Encoding</td>
<td>Client</td>
<td>The document encodings the client can handle</td>
</tr>
<tr>
<td>Accept-Language</td>
<td>Client</td>
<td>The natural language the client can handle</td>
</tr>
<tr>
<td>Authorization</td>
<td>Client</td>
<td>A list of the client's credentials</td>
</tr>
<tr>
<td>WWW-Authenticate</td>
<td>Server</td>
<td>Security challenge the client should respond to</td>
</tr>
<tr>
<td>Date</td>
<td>Both</td>
<td>Date and time the message was sent</td>
</tr>
<tr>
<td>ETag</td>
<td>Server</td>
<td>The tags associated with the returned document</td>
</tr>
<tr>
<td>Expires</td>
<td>Server</td>
<td>The time how long the response remains valid</td>
</tr>
<tr>
<td>From</td>
<td>Client</td>
<td>The client's e-mail address</td>
</tr>
<tr>
<td>Host</td>
<td>Client</td>
<td>The TCP address of the document's server</td>
</tr>
<tr>
<td>If-Match</td>
<td>Client</td>
<td>The tags the document should have</td>
</tr>
<tr>
<td>If-None-Match</td>
<td>Client</td>
<td>The tags the document should not have</td>
</tr>
<tr>
<td>If-Modified-Since</td>
<td>Client</td>
<td>Tells the server to return a document only if it has been modified since the specified time</td>
</tr>
<tr>
<td>If-Unmodified-Since</td>
<td>Client</td>
<td>Tells the server to return a document only if it has not been modified since the specified time</td>
</tr>
<tr>
<td>Last-Modified</td>
<td>Server</td>
<td>The time the returned document was last modified</td>
</tr>
<tr>
<td>Location</td>
<td>Server</td>
<td>A document reference to which the client should redirect its request</td>
</tr>
<tr>
<td>Referer</td>
<td>Client</td>
<td>Refers to client's most recently requested document</td>
</tr>
<tr>
<td>Upgrade</td>
<td>Both</td>
<td>The application protocol the sender wants to switch to</td>
</tr>
<tr>
<td>Warning</td>
<td>Both</td>
<td>Information about the status of the data in the message</td>
</tr>
</tbody>
</table>
Processes - Clients

- Web browser
- Plug-ins
- Helper applications
- Web proxy
Clients (1)

Using a plug-in in a Web browser.
Clients (2)

Using a Web proxy when the browser does not speak FTP.
Processes - Servers

- Example: Apache server - dominant on Unix
- Core module: demultiplexer/scheduler
- Modules:
  - modify fields in request records
  - contain handlers that are invoked by core module
  - use URIs for references
General organization of the Apache Web server.
Server Clusters (1)

The principle of using a cluster of workstations to implement a Web service.
- horizontal distribution -
Server Clusters (2)

(a) The principle of TCP handoff.
A scalable content-aware cluster of Web servers.
Naming

- URI - Uniform Resource Identifier
  - URL - Uniform Resource Locator
  - URN - Uniform Resource Name
    - true identifier
    - globally unique
    - location-independent
    - persistent

- URI syntax is determined by *scheme*
Uniform Resource Locators (1)

Often-used structures for URLs.

b) Using only a DNS name.

c) Combining a DNS name with a port number.

d) combining an IP address with a port number.
## Uniform Resource Locators (2)

<table>
<thead>
<tr>
<th>Name</th>
<th>Used for</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>http</td>
<td>HTTP</td>
<td><a href="http://www.cs.vu.nl:80/globe">http://www.cs.vu.nl:80/globe</a></td>
</tr>
<tr>
<td>file</td>
<td>Local file</td>
<td>file:/edu/book/work/chp/11/11</td>
</tr>
<tr>
<td>data</td>
<td>Inline data</td>
<td>data:text/plain; charset=iso-8859-7,%e1%e2%e3</td>
</tr>
<tr>
<td>telnet</td>
<td>Remote login</td>
<td>telnet://flits.cs.vu.nl</td>
</tr>
<tr>
<td>tel</td>
<td>Telephone</td>
<td>tel:+31201234567</td>
</tr>
<tr>
<td>modem</td>
<td>Modem</td>
<td>modem:+31201234567;type=v32</td>
</tr>
</tbody>
</table>

**Examples of URLs.**
Uniform Resource Names

<table>
<thead>
<tr>
<th>&quot;urn&quot;</th>
<th>Name space</th>
<th>Name of resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>urn</td>
<td>ietf</td>
<td>rfc:2648</td>
</tr>
</tbody>
</table>

The general structure of a URN
Synchronization

• Non-issue:
  – In the WWW: servers never exchange information with other servers! (or clients with clients for that matter)
  – WWW was initially considered a read-mostly system.

• But... what about web authoring?
Caching and Replication

- Web Proxy Caching
- Cooperative caching
- Active caching
  - applet is sent by server to proxy cache to generate response there
Web Proxy Caching

The principle of cooperative caching

1. Look in local cache

2. Ask neighboring proxy caches

3. Forward request to Web server

HTTP Get request

Web proxy

Cache

Client

Web server

Web proxy

Cache

Client

Client

Client

Client

The principle of cooperative caching
Server replication

- Mirroring
- Content Delivery Networks (CDN) (aka Content Distribution Network)
- Example: Akamai using virtual ghosts
- Look up ghosting.com
Server Replication

The principle working of the Akami CDN.

1. Get base document
2. Document with refs to embedded documents
3. Get embedded documents
4a. Get embedded documents from local cache or server (if not already cached)
4b. Embedded documents
5. Embedded documents
Fault Tolerance

- Client side caching
- Server replication
- No special facilities in HTTP to assist fault tolerance
Security

- Secure Socket Layer (SSL)
  - TLS - RFC 2246
- Sandwich approach
- Mutual authentication
The position of TLS in the Internet protocol stack.
TLS with mutual authentication.
Distributed Coordination-Based Systems

Chapter 12
Introduction to Coordination Models

A taxonomy of coordination models (adapted from [cabri.g2000])

<table>
<thead>
<tr>
<th>Coupled</th>
<th>Uncoupled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temporal Coupled</td>
<td>Temporal Uncoupled</td>
</tr>
<tr>
<td>Direct</td>
<td>Mailbox</td>
</tr>
<tr>
<td>Meeting oriented</td>
<td>Generative communication</td>
</tr>
</tbody>
</table>
Taxonomy of Coordination Models

• Referential vs. Temporal
• Direct communication
• Mailbox
• Meeting oriented
  - cmp. events, publish/subscribe systems
• Generative communication
Generative Communication

- Introduced in the Linda programming system
- Collection of independent processes makes use of a shared, persistent data store.
- Processes can put any type of record into the store.
- Processes specify associative query on tuples
Jini

• Temporal and referential uncoupling of processes
• Coordination system called JavaSpaces
• Shared data space that stores tuples
• Tuples are typed sets of references to Java objects
• Multiple JavaSpaces may exist in one Jini system
Overview of Jini

The general organization of a JavaSpace in Jini.
JavaSpaces

- Implemented in Java
- `write()`, `read()`, `take()`, ...
- Read: blocking, max. timeout, non-blocking
- Tuple matching through 'templates'
- Challenge: efficient implementation
The layered architecture of a Jini System.
Communication

• Based on Java RMI
• Java Spaces
• Event and notification subsystem
  - client can register itself with object using a listener object. Callback via RMI
  - all registrations are subject to a lease
Communication Events

Using events in combination with a JavaSpace

1. Request notification for T
2. Insert a copy of C
3. Notify when C is inserted
4. Look for tuple that matches T
5. Return C (and optionally remove it)
Processes

- Tricky bit: implementation of JavaSpaces Server
  - How to simulate associative addressing without massive searching
  - How to distribute tuple instances among machines and locate them later
A JavaSpace can be replicated on all machines. The dotted lines show the partitioning of the JavaSpace into subspaces.

b) Tuples are broadcast on WRITE

c) READs are local, but the removing of an instance when calling TAKE must be broadcast
Processes (2)

Unreplicated JavaSpace.

b) A WRITE is done locally.
c) A READ or TAKE requires the template tuple to be broadcast in order to find a tuple instance
Processes (3)

Partial broadcasting of tuples and template tuples.
Naming

- No *traditional* naming service provided
- JavaSpace can implement naming service
- Jini provides specialized lookup service
  - ServiceID
  - Service
  - Attribute Sets
The Jini Lookup Service (1)

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceID</td>
<td>The identifier of the service associated with this item.</td>
</tr>
<tr>
<td>Service</td>
<td>A (possibly remote) reference to the object implementing the service.</td>
</tr>
<tr>
<td>AttributeSets</td>
<td>A set of tuples describing the service.</td>
</tr>
</tbody>
</table>

The organization of a service item.
The Jini Lookup Service (2)

Examples of predefined tuples for service items.

<table>
<thead>
<tr>
<th>Tuple Type</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServiceInfo</td>
<td>Name, manufacturer, vendor, version, model, serial number</td>
</tr>
<tr>
<td>Location</td>
<td>Floor, room, building</td>
</tr>
<tr>
<td>Address</td>
<td>Street, organization, organizational unit, locality, state or province, postal code, country</td>
</tr>
</tbody>
</table>
Leasing

- Referenced object can keep track of who references it
- But... what happens in the case of failures?
- Lease expiration, renewal
- Best effort!!!
Synchronization

- Blocking read/write

- Transactions
  - 2 phase-commit
  - Nested transactions
  - JavaSpace can participate in a transaction
  - Transactions can span multiple JavaSpaces
Synchronization of Transactions

The general organization of a transaction in Jini. Thick lines show communication as required by Jini’s transaction protocol.
Caching and Replication

- Fault tolerance for lookup services
- Fault tolerance for communications via RMI via TCP
Security

- Java RMI security model
  - stack introspection
  - privilege checks via Java security manager
- JAAS - Java Authentication and Authorization Service
- PAM - Pluggable Authentication Module
The position of PAM with respect to security services.