Software Architecture in Practice

Chapter 7: The World Wide Web - A Case Study in Interoperability

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890

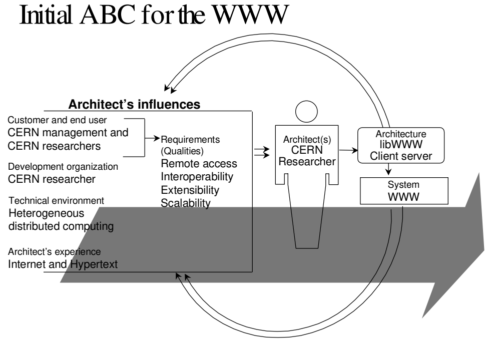
Sponsored by the U.S. Department of Defense

Lecture Objectives

This lecture will enable students to

- **understand the relationship between the architecture of the World Wide Web (WWW) and the architectural environments that spawned it**
- **understand techniques used by the WWW's software developers to support scalability**
- **understand how shifting requirements can cause an architecture to change over time**

Initial ABC for the WWW



Original Requirements/Qualities

Promote access across networks

Support a heterogeneous computing environment

Avoid centralized control

Support access to existing data

Give users the ability to add data

Permit data analysis

Support live links

Original Non-Requirements

Graphics and multimedia

Different link types

Visual history

Copyright enforcement

Security

Privacy

Mark-up format

Growth of WWW

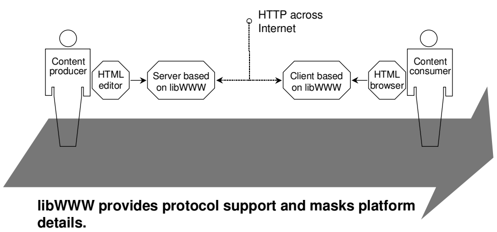
Month	No. of Web sites	Percent of .com sites	Hosts per Web server
6/93	130	1.5	13,000
12/93	623	4.6	3,475
6/94	2,738	13.5	1,095
12/94	10,022	18.3	451
6/95	23,500	31.3	270
1/96	100,000	50.0	94
6/96	230,000 (est.)	68.0	41
1/97	650,000 (est.)	62.6	NA

Requirements Come and Go

With the success of the Web came new requirements. For example:
- **success: cyberporn**
- **requirement: content labeling (PICS)**

What would have been the fate of this requirement in the original proposal?

WWW Architectural Approach: Client-Server



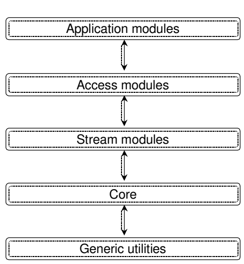
libWWW provides protocol support and masks platform details.

Remote access is achieved by building WWW on top of Internet.

Meeting the Requirements: libWWW

Dynamic view of the architecture is client-server.

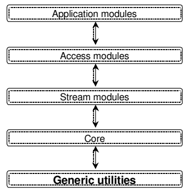
Developer's view is a *layered* architecture.



Generic Utilities

Basic building blocks for the system

- **network management**
- **data types (such as container class)**
- **string manipulation utilities**



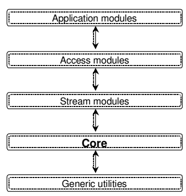
Core

Skeletal function of a WWW application

- **network access**
- **data management**
- **parsing**
- **logging**

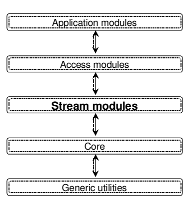
Core does nothing. Actual work done by

- **plug-ins**
- **call-outs**



Stream Modules

***Abstraction* of a stream of data**

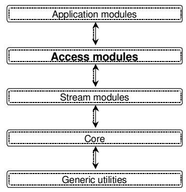


Access Modules

Skeleton for network protocol-aware modules

- HTTP
- WAIS
- FTP
- Telnet
- rlogin
- Gopher
- local file system

Easy to add new protocols

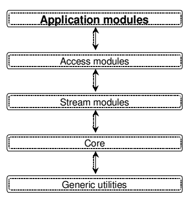


Application Modules

Not an actual application; again, a skeleton

Set of functionality useful for writing applications

- caching
- logging
- registering proxy server and gateways
- history maintenance



libWWW Lessons Learned

libWWW has had many releases.

Lessons learned from the many iterations include

- Formalized application program interfaces (APIs) are required.
- APIs must be layered.
- APIs must support a dynamic, open-ended set of features.
- APIs must be thread safe.

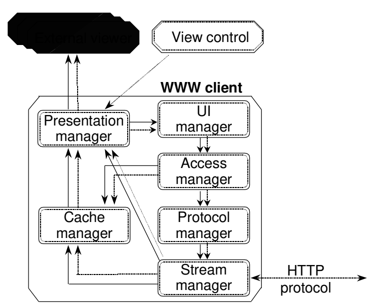
Not all of these goals have been met!

Dynamic View of Typical WWW Architecture

We will look at a typical client server.

Layers of libWWW disappear in the runtime view.

Typical WWW Client



Client Components

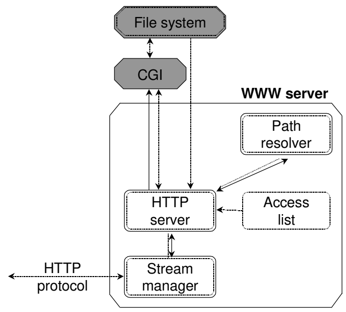
User interface (UI) manager is responsible for the look and feel of the UI (e.g., Web browser).

Presentation manager delegates document types to viewers.
- **external: QuickTime movies**
- **internal: HTML to UI manager**

UI manager captures user's request for URL and passes to access manager.

Access manager determines if URL has been cached and if it has not, it initiates retrieval through protocol and stream managers.

WWW Server



Server Components - 1

Hypertext transfer protocol (HTTP) server receives request for URL, and passes that to the path resolver, which determines the file location for the document (assuming local).

HTTP server checks the access list to see if access is permitted and then gets the document from the file system.

Server Components - 2

Common gateway interface (CGI) is a special document type that allows customized access to other data or programs.

HTML is served by the HTTP server back to client.

Extending the Server

CGI allows
- dynamic documents
- addition of data to existing databases
- customized queries
- clickable images

CGI scripts
- can be written in a variety of languages
- run as separate processes

Problems With CGI

Extensibility and "put" principally supported by CGI scripts in libWWW applications

However
- CGI has security holes
- CGI is not portable

One solution: Jigsaw resource objects

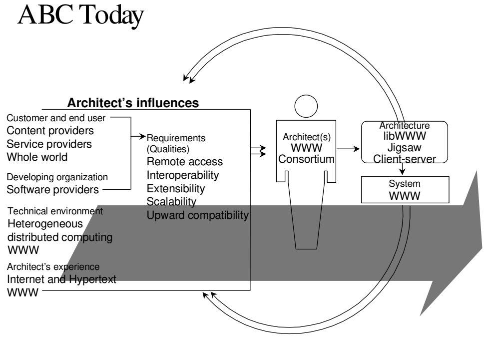
New Server Base

Jigsaw is an object-oriented server.
- written in Java
- intended to replace libWWW for server side

Jigsaw addresses
- portability
- security
- ease of creation
- extensibility (through use of resource objects)
- efficiency (through minimization of file accesses)

The architectural solution is Jigsaw.

ABC Today



How WWW' s Qualities Are Achieved - 1

Allow a heterogeneous computing environment: Define a new protocol realizable on any platform and isolate platform dependencies into libWWW layer.

Be non-centralized: Each data item can have links to other data items; system is client-server with servers communicating via universal naming scheme.

How WWW' s Qualities Are Achieved -2

Allow access to existing data: User specified viewers.

Give users the ability to add data: Links are defined at runtime from existing data.

Support scalability by open-ended skeletons that allowed protocol-aware modules to be plugged in.

Lecture Summary

Exponential growth of the WWW was allowed by

- use of a flexible client-server architecture
- a well-defined protocol
- no centralized control
- libWWW as a *skeleton* into which protocols were added

This growth changed the technical environment, causing iterations of the ABC.

Discussion Questions -1

1. We have identified a number of qualities that made the WWW successful: interoperability, portability, remote access, and scalability. Which of these do you think contributed most substantially to the Web's success? If any of these qualities had been sacrificed, would the Web still have been successful? What trade-offs did these quality goals entail in the architecture of applications based upon libWWW?

Discussion Questions -2

2. The WWW did not have performance as one of its early quality goals. This is unusual for a successful system. What, if anything, does this say about the future of computing?