Web Application Engineering
Introduction

Matthew Dailey

Information and Communication Technologies
Asian Institute of Technology

Readings

Readings for these lecture notes:

- SEIA, Chapters 1–2
- Henderson, *Building Scalable Web Sites*, Chapter 1
- Ruby et al., *Agile Web Development with Rails*, Chapter 1

These notes contain material © Andersson, Greenspun, and Grumet, 2005; Henderson, 2006; Ruby et al., 2016.

Outline

Course introduction
What is web application engineering?

WAE is the construction and maintenance of large, complex Web applications. The main design goals:

- Easy management of the content
- Easy software maintenance
- High performance
- Scalability
- High availability
- Connections between people

Course introduction
What is special about Web applications?

WAE is similar to traditional software engineering in that it needs

- Requirements analysis
- System design
- Testing and quality assurance
- Performance evaluation

Some concerns take the forefront, however:

- Security: you must design as if your system is under constant attack.
- Continuous development process: Web sites evolve over time, and so do the applications running them.
- Untrained user base: your users will expect to be able to figure out how to use the application without training.

Course introduction
What Web application engineering is not about

WAE is not about:

- Visual design (this is actually important for construction of an engaging Web site that will keep your users coming back, but comes in a long second to functionality).
- Graphics and animation
- Java applets

We mainly focus on the back end: the software architecture and technology used on the server side.

Unfortunately, the dividing line is not clean — Web applications increasingly depend on client-side software that affects the application architecture.

Course introduction

The project in this course

In this class you will build a Web application including user authentication, session management, content management, and so on from the ground up.

- Your application should be social. That is, it should put people together to either share knowledge or work collaboratively.
- One important type of social Web application is the online community. See SEIA Chapter 1 for more information about communities and their importance. In a nutshell, according to Greenspun, a community is a "group of people with varying degrees of expertise in which the experts attempt to help the novices improve their skills."
- A second type of social Web application automates complex collaborative workflow in an enterprise or between individuals.

Course introduction
The project in this course

Note that AIT is a community!

The main benefit of online communities: more people get opportunities to learn, and equally important, more people get opportunities to teach.

The techniques and technologies we study will be equally useful for e-commerce or other types of Web sites.

Your Web application will be "small" enough to build with standard tools that come bundled with Linux and other operating systems.

Outline

Web technology background

The basic Web technologies

Fundamentally, the Web is a marriage of three standard technologies:

- The URI or Uniform Resource Identifier
- HTTP, the Hypertext Transfer Protocol
- HTML, the Hypertext Markup Language

Web resources are named by URIs, e.g. `http://www.cs.ait.ac.th/`. HTML allows embedding of links to URIs in documents, and HTTP is the protocol for requesting and transferring Web resources over TCP/IP.

Web technology background

Relationship with TCP/IP

HTTP runs on TCP/IP, a suite of layered packet network protocols:

- Link layer: provides direct communication between hosts connected by the same physical medium. TCP/IP can actually run on top of many different link layers, e.g., 802.3 (Ethernet).
- Network layer: the "Internet Protocol" (IP) provides *un*reliable transmission of "datagrams" from source to destination, possibly over multiple networks.
- Transport layer: end-to-end transmission of data with error handling, retransmit, flow control, and so on. The two main choices are TCP for a reliable "stream" or UDP for connectionless transmission of datagrams to a specific application (port) at the destination.
- Application layer: application-specific protocols for communication of data over the network. Examples are SMTP (email), SSH (secure remote login), and HTTP.

Web technology background
HTTP

HTTP/1.1, the “Hypertext Transfer Protocol”:

- An application-layer protocol running on top of TCP/IP
- Defined by Fielding et al., RFC 2616.
- A client-server, request-response protocol:
  - The client (browser) makes a TCP connection to the server’s port 80 (by default).
  - The client sends a request, e.g. `GET / HTTP/1.0`.
  - The server does some crunching then returns a document, e.g.,

```
HTTP/1.1 200 OK
Date: Tue, 22 Aug 2006 00:21:45 GMT
Server: Apache/1.3.34 (Unix) mod_fastcgi/2.4.2 PHP/4.4.2 mod_ssl/2.8.25 OpenSSL/0.9.7e-p1
Connection: close
Content-Type: text/html

<HTML>
<HEAD>
   <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
   <META NAME="Author" CONTENT="Olivier Nicole">
   <TITLE>AIT CSIM Program</TITLE>
</HEAD>
...
```

Web technology background
Origins of Web applications

The Web was originally conceived as a system for document sharing, by Tim Berners-Lee.

The first Web server and first Web browser debuted in 1990.

Soon people figured out it would be cool to request the output of a program, rather than (or in addition to) static documents.

The Common Gateway Interface (CGI) debuted in 1993. The Web server would create a new process to run the selected program, set environment variables corresponding to the HTTP request, and start the program running.

Performance was improved by building programming language interpreters directly into the Web server so that new processes did not have to be created. E.g. `mod_perl` for Apache arrived in 1996.

Web technology background
Origins of Web applications

Web applications began with scripts in Perl or other languages that performed calculations then output HTML directly.

In the mid-1990s, server page technology began to gain popularity.

With server pages, the developer writes HTML pages and inserts code wherever necessary for dynamic content. PHP, ASP, JSP, ERB all take this approach.

Web technology background

Limitations of the HTTP medium

Fast invocation of programs over HTTP means we could now build application serving millions of users per day.

But even with fast invocation of programs, HTTP is fundamentally request-response and stateless.

It is up to you (the Web application developer) to use some tricks to give the illusion of state even though the communication medium is stateless.

There are three ways to maintain state over a series of response/request interactions:

- Hide state information in documents delivered to the user.
- Store information about user sessions on the server.
- Respond to user events client-side, without starting a new HTTP request/response cycle.

In practice a single Web application will use all three strategies.

Web technology background

Client-side state information

There are a few main ways to “store” state information client-side:

- Add information to HTML forms (`hidden` variables)
- Request the user’s browser to accept a cookie
- Add information to the URIs in a document
- Place information in invisible DOM (Document Object Model) elements

All of these techniques are frequently useful but have security issues and limited capacity. Almost all significant applications also require server-side state storage.

Web technology background

Server-side persistence

Static web sites can use the server's file system as the source of persistence.

Dynamic web sites require more sophisticated means of persistence. The relational model is good for frequent updates to structured data. RDBMSs pass the ACID test:

- Atomicity (transactions are all or none)
- Consistency (the database moves from one consistent state to another)
- Isolation (partially completed transactions are not visible to any other user)
- Durability (completed transactions should survive power outages and so on)

Web technology background

Server-side persistence

For big data applications where reading is much more common than writing, NoSQL databases are becoming more common:

- Key-value databases
- Column family databases
- Graph databases
- Schemaless databases

Web technology background
Server-side persistence

NoSQL is becoming more prevalent. But the majority of Web applications still use relational database management systems (RDBMSs) as the source of persistence. Why?

- Easy-to-learn standardized SQL interface.
- ACID-compliance.
- Scaling to thousands of concurrent users gracefully.
- Powerful object-relational mapping frameworks are readily available.

Web technology background
Building RDBMS-backed Web applications

Once you decide to use a RDBMS, building a traditional Web application boils down to these steps:

- Develop the data model.
- Determine the set of legal transactions on the data model.
- Design the page flow.
- Implement each page.

Simple, no?

Yes, until your application grows. We'll discuss ways to manage complexity as an application grows.

Web application frameworks help by forcing structure on the software running on the application server, outside the RDBMS.

Web technology background

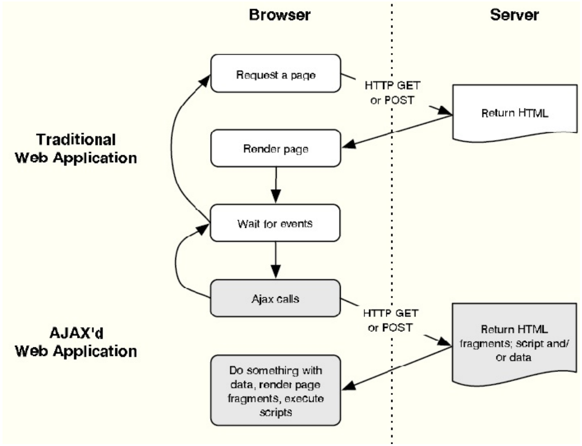
Client-side scripting

Client-side scripting is increasingly important as the libraries' capabilities and server-side framework integration improve:

- Event handlers can respond to user actions directly and modify the DOM without making HTTP requests. This means we have multiple user interactions per HTTP request, improving performance.
- Ajax support in the browser allows us to execute non-blocking HTTP requests and asynchronously update the DOM when the response is received, without reloading the page.
- HTML5 Web socket support in the browser gives us a two-way persistent connection to a server, enabling real-time server push.

Web technology background

Client-side scripting



Browser
Server
Traditional Web Application
AJAX'd Web Application
Request a page
HTTP GET or POST
Return HTML
Render page
Wait for events
Ajax calls
HTTP GET or POST
Return HTML fragments; script and/ or data
Do something with data, render page fragments, execute scripts

Thomas et al. (2014), Fig. 21.1

Web technology background

Client-side scripting

Back to the future: mainframe $\rightarrow$ client-server $\rightarrow$ traditional Web app $\rightarrow$ RIAs and mobile apps

Client-side scripting makes it more difficult to develop clean, maintainable Web applications, but does not affect the basic principles.

We'll first look at how to structure applications on the server side, then later in the semester we'll look at patterns and antipatterns for client-side scripts.