Libraries, API's, Frameworks

- browsers are not perfectly standardized
- DOM and CSS coding is messy and complicated
- web services are ever more complex

- how do we make it easy to create applications?

- libraries of common Javascript operations

- API's, often Javascript, to access services

- frameworks: development environments for integrated client & server programming
YAHOO.util.Connect = {
    _msxml_progid: [
        'MSXML2.XMLHTTP.5.0',
        'MSXML2.XMLHTTP.4.0',
        'MSXML2.XMLHTTP.3.0',
        'MSXML2.XMLHTTP',
        'Microsoft.XMLHTTP'
    ],
    createXhrObject:function(transactionId) {
        var obj, http;
        try {
            http = new XMLHttpRequest();
            obj = { conn:http, tId:transactionId };
        }
        catch(e) {
            for (var i=0; i<this._msxml_progid.length; ++i){
                try {
                    http = new ActiveXObject(this._msxml_progid[i]);
                    obj = { conn:http, tId:transactionId };
                    break;
                }
                catch(e){}
            }
        }
        finally {
            return obj;
        }
    },...
Javascript libraries

- library of Javascript functions that typically provides
  - easier access to DOM
  - convenience functions for arrays, iterators, etc.
  - uniform interface to Ajax
  - visual effects like fading, flying, folding, ...
  - drag and drop
  - in-place editing
  - extensive set of widgets: calendar, slider, progress bar, tabs, ...

- there are lots of such libraries
  - jQuery, jQueryUI, Dojo, Yahoo User Interface (YUI), mooTools,
    Prototype / Scriptaculous, ...

- see http://code.google.com/apis/libraries/
  - single library for uniform access to ~10 Javascript libraries
  - experiment at http://code.google.com/apis/ajax/playground
Basic structure of Ajax code in browser

```javascript
var req;
function geturl(s) {
    if (s.length > 1) {
        url = 'http://www.cs.princeton.edu/~bwk/phone3.cgi?' + s;
        loadXMLDoc(url); // loads asynchronously
    }
}
function loadXMLDoc(url) {
    req = new XMLHttpRequest();
    if (req) {
        req.onreadystatechange = processReqChange;
        req.open("GET", url);
        req.send(null);
    }
}
function processReqChange() {
    if (req.readyState == 4) { // completed request
        if (req.status == 200) // successful
            show(req.responseText); // could be responseXML
    }
}
function show(s) { // show whatever came back
    document.getElementById("place").innerHTML = s
}
```
jQuery example

```html
<script>
    function geturl(s) {
        if (s.length > 1) {
            var url = 'http://www.cs.princeton.edu/~bwk/phone3.cgi?' + s;
            $.get(url, function(res) {
                $('pre').empty().append(res);
            });
        }
    }
</script>
<form name=phone>
    Type here:
    <input type="text" id="pat" onkeyup='geturl(pat.value);'>
</form>
<pre id="place"></pre>
```
Debugging Javascript

• it's hard
• use var declarations, check balanced quotes, braces, brackets, ...
• in Chrome
  - "wrench" / Tools / Javascript console
• in Firefox
  - Tools / Web developer / Web Console

• use console.log to write debugging output
  - like printf
  - much better than alert( ... ) for most things
Google maps API (version 3)

```html
<style type="text/css">
  html { height: 100% }
  body { height: 100%; margin: 0px; padding: 0px }
  #map { height: 100% }
</style>

<script type="text/javascript">
  src="http://maps.google.com/maps/api/js?sensor=true">
</script>

<script type="text/javascript">
  function initialize() {
    var latlong = new google.maps.LatLng(40.34705, -74.65495);
    var opts = {
      zoom: 18, center: latlong,
      mapTypeId: google.maps.MapTypeId.HYBRID 
    );
    var map = new google.maps.Map(document.getElementById("map"), opts);
    var marker = new google.maps.Marker({
      position: latlong, map: map, title:"You are here, more or less" });
  }
</script>

<body onload="initialize()">
  <div id="map" style="width:100%; height:100%"></div>
</body>
</html>
```
Web [Application] Frameworks

- conventional approach to building a web service
  - write ad hoc client code in HTML, CSS, Javascript, ... by hand
  - write ad hoc server code in [whatever] by hand
  - write ad hoc access to [whatever] database system
- so well understood that it's almost mechanical
- web frameworks mechanize (parts of) this process
- lots of tradeoffs and choices
  - what client and server language(s)
  - how web pages are generated
  - how web events are linked to server actions
  - how database access is organized (if at all)
- can be a big win, but not always
  - somewhat heavyweight
  - easy to lose track of what's going on in multiple layers of generated software
  - work well if your application fits their model, less well if it doesn't
- examples:
  - Ruby on Rails
  - Django
  - Google Web Toolkit
  - Zend (PHP), ASP.NET (C#, VB.NET), and many others
Django

• by Adrian Holovaty and Jacob Kaplan-Moss (released July 2005)

• a collection of Python scripts to

• create a new project / site
  - generates Python scripts for settings, etc.
  - configuration info stored as Python lists

• create a new application within a project
  - generates scaffolding/framework for models, views

• run a development web server for local testing

• generate a database or build interface to an existing database
• provide a command-line interface to application
• create an administrative interface for the database
• ...

Django Reinhart, 1910-1953
Django web framework

• write client code in HTML, CSS, Javascript, …
  - Django template language helps separate form from content
• write server code in Python
  - some of this is generated for you
• write database access with Python library calls
  - they are translated to SQL database commands

• URLs on web page map mechanically to Python function calls
  - regular expressions specify classes of URLs
  - URL received by server is matched against regular expressions
  - if a match is found, that identifies function to be called and arguments to be provided to the function
Conventional approach to building a web site

- user interface, logic, database access are all mixed together

```python
import MySQLdb
print "Content-Type: text/html"
print
print "<html><head><title>Books</title></head>"
print "<body>"
print "<h1>Books</h1>"
print "<ul>
connection = MySQLdb.connect(user='me', passwd='x', db='my_db')
cursor = connection.cursor()
cursor.execute("SELECT name FROM books ORDER BY pub_date DESC")
for row in cursor.fetchall():
    print "<li>%s</li>" % row[0]
print "</ul>"
print "</body></html>"
connection.close()
```
Model-View-Controller (MVC) pattern

- an example of a design pattern
- model: the structure of the data
  - how data is defined and accessed
- view: the user interface
  - what it looks like on the screen
  - can have multiple views for one model
- controller: how information is moved around
  - processing events, gathering and processing data, generating HTML, ...
- separate model from view from processing so that when one changes, the others need not
- used with varying fidelity in
  - Django, App Engine, Ruby on Rails, XCode Interface Builder, ...
- not always clear where to draw the lines
  - but trying to separate concerns is good
Django approach

- generate framework/skeleton of code by program

# models.py (the database tables)

from django.db import models

class Book(models.Model):
    name = models.CharField(max_length=50)
    pub_date = models.DateField()

# views.py (the business logic)

from django.shortcuts import render_to_response
from models import Book

def latest_books(request):
    book_list = Book.objects.order_by('-pub_date')[:10]
    return render_to_response('latest_books.html',
                               {'book_list': book_list})

# urls.py (the URL configuration)

from django.conf.urls.defaults import *
import views

urlpatterns = patterns('',
                       (r'latest/$', views.latest_books),
)
Database linkage

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': '/Users/bwk/dj1/mysite/sql3.db', ...
    }
}

from django.db import models
class Books(models.Model):
    isbn = models.CharField(max_length=15)
    title = models.CharField(max_length=35)
    author = models.CharField(max_length=35)
    price = models.FloatField()

BEGIN;
CREATE TABLE "db1_books" (
    "id" integer NOT NULL PRIMARY KEY,
    "isbn" varchar(15) NOT NULL,
    "title" varchar(35) NOT NULL,
    "author" varchar(35) NOT NULL,
    "price" real NOT NULL
);
```
URL patterns

- regular expressions used to recognize parameters and pass them to Python functions
- provides linkage between web page and what functions are called for semantic actions

```python
urlpatterns = patterns('',
    (r'^time/$', current_datetime),
    (r'^time/plus/(\d{1,2})/$', hours_ahead),
)
```

- a reference to web page `time/` calls the function `current_datetime()`
- tagged regular expressions for parameters: `url time/plus/12` calls the function `hours_ahead(12)`
Administrative interface

- most systems need a way to modify the database even if initially created from bulk data
  - add / remove users, set passwords, ...
  - add / remove records
  - fix contents of records
  - ...

- often requires special code

- **Django generates an administrative interface automatically**
  - loosely equivalent to MyPhpAdmin

```python
urlpatterns = patterns('',
    ...
    # Uncomment this for admin:
    # (r'^admin/', include('django.contrib.admin.urls')),
```
Google Web Toolkit (GWT)  (first available May 2006)

- write client (browser) code in Java  
  - widgets, events, layout loosely similar to Swing
- test client code on server side  
  - test browser, or plugin for testing with real browser on local system
- compile Java to Javascript and HTML/CSS  
  - [once it works]
- use generated code as part of a web page  
  - generated code is browser independent (diff versions for diff browsers)
- can use development environments like Eclipse  
  - can use JUnit for testing
- strong type checking on source  
  - detect typos, etc., at compile time (unlike Javascript)
- doesn't handle all Java runtime libraries  
  - ?
- no explicit support for database access on server  
  - use whatever package is available
"Same Origin Policy"

- "The same origin policy prevents a document or script loaded from one origin from getting or setting properties of a document from another origin. This policy dates all the way back to Netscape Navigator 2.0." (Mozilla)

- "The SOP states that JavaScript code running on a web page may not interact with any resource not originating from the same web site." (Google)

- basically Javascript can only reference information from the site that provided the original code

- BUT: if a page loads Javascript from more than one site (e.g., as with cookies from third-party sites), then that JS code can interact with that third-party site
GWT assessment

- problem: Javascript is irregular, unsafe, not portable, easily abused

- solution: use Java, which is type-safe, standard, portable

- translate Java to Javascript to either be browser independent or tailored to specific browser as appropriate
- can take advantage of browser quirks, make compact code, discourage reverse engineering
- can provide standardized mechanisms for widgets, events, DOM access, server access, AJAX, RE's and other libraries, ...

- in effect, treat each browser as a somewhat irregular machine and compile optimized code for it specifically
GWT vs Django

- focusing on different parts of the overall problem

- **GWT provides**
  - reliable, efficient, browser-independent Javascript (from Java)
  - extensive widget set
  - no help with database access, generating HTML, ...

- **Django provides**
  - no Javascript help
  - no widgets
  - easy database access; template language for generating HTML, ...
  - easy linkage from URLs on web page to Python functions

- is GWT + App Engine a good combination?
Assessment of Web Frameworks

• **advantages**
  - takes care of repetitive parts
    - more efficient in programmer time
  - automatically generated code is likely to be more reliable, have more uniformity of structure
  - "DRY" (don't repeat yourself) is encouraged
  - "single point of truth"
    - information is in only one place so it's easier to change things
  - ...

• **potential negatives**
  - automatically generated code
    - can be hard to figure out what's going on
    - can be hard to change if you don't want to do it their way
  - systems are large and can be slow
  - ...

• **read Joel Spolsky's "Why I hate frameworks"**
  
Assessment of Ajax-based systems

• potential advantages
  - can be much more responsive (cf Google maps)
  - can off-load work from server to client
  - code on server is not exposed
  - continuous update of services

• potential negatives
  - browsers are not standardized
  - Javascript code is exposed to client
  - Javascript code can be bulky and slow
  - asynchronous code can be tricky
  - DOM is very awkward
  - browser history not maintained without effort

• what next? (changing fast)
  - more and better libraries
  - better tools and languages for programming
  - better standardization?
  - will the browser ever replace the OS?