The Design Space of Modern HTML5/JavaScript Web Applications

Marcin Nowak, Cesare Pautasso
University of Lugano, Switzerland
http://saw.inf.usi.ch/
c.pautasso@ieee.org  @pautasso
Marcin.Nowak@sonyx.net
Web application Architecture

Client

Web Browser

Server

Web Server
Web application Architecture

Client

Web Browser

Request

Response

Web Browser

Web Server

Server
Web application Architecture

Client

Web Browser

Application

Server

Web Server

Application

Backend

Database

File System
Very Thin Client

Web Browser

Input/Output

Request → Display

Response

Application

Data

Web Server
Client/Server

Web Browser

Web Server

Input/Output

Display

Application

Data

Request

Response
Rich Client
General Architecture

Web Browser

Display

Application

Cache

Request

Response

Data

Web Server

Server Application

Request

Response

Data
The server is a database

The browser is a terminal
Rich Client Web Application

- Presenter
- Controller
- View-Model

- Local Model
- Local Storage

- View

- UI State
- Templating

- Model-View Data Binding
Rich Client Web Application

Concurrency Management

Remote Model

Synchronization

Presenter Controller View-Model

Local Model

Local Storage

UI State

View

Templating

Model-View Data Binding
Model-View Interaction Pattern

1. MVC - Model View Controller
2. MVP - Model View Presenter
3. MVVM - Model View View-Model
4. MV* - Model View (Whatever)
Model View Controller

- Simple
- Direct mapping from model to view
- No view state
Model View Presenter

- More flexible
- Prone to Presenter overgrowth
- Simple view state
Model View ViewModel

- Most flexible
- Clean separation of view and controller logic
- Simple view state
Model View *

- Roll-your-own
- Good for power applications
Model-View Data Binding

Model (JS):

```javascript
var model = new Person({
    name : "John",
    surname : "Smith",
    age : 123
});
```

View (DOM):

```html
<div class="person">
    Name: <span id="name"></span><br/>
    Surname: <span id="surname"></span><br/>
    Age: <span id="age"></span>
</div>
```
Model-View Data Binding

1. No binding: full refresh required

2. Event-driven: explicit handlers of model changes (mono-directional)

3. Declarative: HTML5 `data-` attributes to bind models to view elements (bi-directional)
View Template Logic

Ensure separation of View and Presenter/Controller/View-Model layers

1. Templates with embedded logic (more expressive, hard to maintain)
2. Logic-less templates (faster)
View State Management

Identify and Persist the current state of the view

1. None (State is lost on refresh)

2. URL routing (# hash)

3. Event-based
Local state persistence

1. Cookies
2. Key-Value (HTML5 LocalStorage, SessionStorage)
3. SQL (WebSQL)
4. IndexedDB (Asynchronous)
Concurrency Management

1. None: Non-deterministic Collisions
2. Operational Transformation (share.js)
3. Locking (Explicit or Implicit)
4. Global Event Serialization (TeaTime)
More Decisions

1. Frameworks
   - MV* Framework
   - HTML Templating Mechanism
   - Eventing Framework

2. Development
   - Programming Language
   - Modularization

3. Portability
   - Target Browser Platform
   - Mobile Support
   - Feature Detection
   - Compatibility with Missing HTML5 Features
# MV* Framework

<table>
<thead>
<tr>
<th>Name</th>
<th>MV*</th>
<th>MVC</th>
<th>MVP</th>
<th>MVVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backbone</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Angular</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spine</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Knockout</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Knockback</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YUI</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Marionette</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>GWT</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Batman</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Template rendering engine

<table>
<thead>
<tr>
<th>Name</th>
<th>Logic-enabled</th>
<th>Logic-less</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dom.js</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Dot.js</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>EJS</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Handlebars</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Mustache.js</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Plates.js</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Pure.js</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Transparency</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Underscore.js</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Event Delivery Method

Work around the limitations of HTTP

1. Polling
2. COMET (streaming and long polling)
3. HTML5 WebSockets, Server-Sent Events
4. SPDY
Programming Language

Java

CoffeeScript

TypeScript

Dart

JavaScript
Modularity

1. None
2. RequireJS
3. Marionette
4. Browserify
Target Browser Platform

1. Google Chrome
2. Microsoft Internet Explorer
3. Mozilla Firefox
4. Apple Safari
5. Opera
6. Amazon Silk
Mobile Support

1. Responsive Design
2. CSS Media queries
3. jQuery mobile
Feature Detection

1. modernizr
2. user agent-string
3. Cutting the Mustard
Compatibility with Missing HTML5 Features

1. CSS
2. Embedded Foreign Browser Frame
3. HTML5 shivs and shims (JavaScript backport)
Conclusion

- There are many emerging frameworks for building rich client applications with HTML5/JavaScript
- Many architectural decisions are required to design rich client applications
- Come and visit the Software Architecture Warehouse http://demo.saw.sonyx.net/ if you would like to make some of those decisions together