

Itinerary Planner: A Mashup Case Study

Shyam Govardhan and George Feuerlicht

Faculty of Information Technology,
University of Technology, Sydney, Australia
ssgovard@it.uts.edu.au, jiri@it.uts.edu.au

Abstract. The wide adoption of Web Services and the availability of web APIs are transforming the web into a programmatic environment for developing innovative web applications that combine information from various sources to provide a rich user experience. These mashup applications are characterized by rapid development using existing data sources and the use of new technologies such as AJAX, JSON, etc. Developers often focus on delivering rich functionality via the browser environment and pay little attention to the design and maintainability of the applications. In this paper we describe our experience in developing an Itinerary Planner travel application, and discuss the challenges associated with developing mashups. In the conclusion, we briefly discuss the lessons learned in addressing these challenges and how these lessons can be applied to future mashup projects.

Keywords: WEB 2.0 mashups, user interface design, data integration design

1 Introduction

Over the past decade the web has undergone a number of dramatic transformations. Initially emerging as a platform for information retrieval in the mid 1990s and evolving into a comprehensive e-business (electronic business) platform at the beginning of this century. The recent proliferation of publicly available web APIs (Application Programming Interfaces) and AJAX enabled websites is transforming the web into an environment for collaborative development and information sharing with extensive user participation. This new web platform, known as WEB 2.0 is characterized by specialized, publicly available data sources (e.g. mapping data, weather information, etc.) and it is accessible via *light-weight* APIs that can be used to create value-added services in the form of mashups [1].

Well-established and open programming interfaces such as the Google Maps API allow developers to integrate static data with dynamic mapping interfaces and produce highly interactive and visual applications. Many of the mashups use geographic data, and studies indicate that more than 80% of all information can be represented in geographic terms [2].

Free access to online data aggregation services such as Yahoo Pipes¹ and information sources such as Google Maps², Google Search³, Picasa⁴, Flickr⁵, Amazon⁶ and Ebay⁷ has created an environment where users can participate in the development of applications that combine information from multiple sources and produce novel applications. Sophisticated use of AJAX in popular applications such as Gmail⁸ has resulted in users expecting the native look and feel of desktop applications in dynamic web environments. Scripting languages such as Javascript have evolved from being a tool used for simple tasks such as menu navigation and HTML validation to being a powerful tool for Web 2.0 development [3].

This transformation of the web brings many new opportunities, but at the same time poses new challenges to the developers of web applications. Software developers need to deal with disparate types of data such as spatial data, images, music clips and video clips sourced from different web sites, and under varied levels of quality of service (i.e. latency, reliability, security, etc). While developers adapt to this dynamic environment by learning new languages, APIs and tools, at present there are no widely accepted design and development methods to ensure that the rich functionality of mashups is achieved without sacrificing performance, maintainability and reuse. The *light-weight* programming models used in the development of mashup applications avoid the complexities of more mature programming environments (e.g. Web Services) but lack the reliability and robustness that is essential for the development of mission-critical applications. It is becoming clear that traditional software design principles need to be adapted to this new application environment.

In this paper we illustrate the issues that developers of mashup applications face using a travel application - Itinerary Planner Mashup (IPM). IPM is a Single-page application [4], [5] that allows users to create an itinerary of the destinations (cities) that they plan to visit and display these destinations on a map. Users can display additional information about each destination including weather data (derived from Yahoo Weather RSS feeds) and other local information (derived using the Google Ajax Search API). A working prototype of the application is available at <http://www.redlifejacket.com>. (The IPM application is password protected; access can be granted upon request).

The next section (section 2) focuses on design issues, discussing user interface and data integration design. The following section (section 3) describes the implementation of the IPM application, and section 4 is a discussion of the technical considerations including security and performance. In conclusion (section 5) we summarize the lessons learnt from the IPM case study and discuss how we applied these experiences to new projects. While the IPM application is relatively small, the

¹ <http://pipes.yahoo.com/pipes/>

² <http://www.google.com/apis/maps/>

³ <http://code.google.com/apis/ajaxsearch/>

⁴ <http://code.google.com/apis/picasaweb/overview.html>

⁵ <http://www.flickr.com/services/api/>

⁶ <http://aws.amazon.com>

⁷ <http://developer.ebay.com/common/api/>

⁸ <http://www.gmail.com>

issues encountered are generic and the lessons learned could be applied to other mashup applications.

2 IPM Design Considerations

In this section we consider two main design stages that characterize most mashup applications: User interface design (section 2.1) and Data integration design (section 2.2).

2.1 User Interface Design

An important design consideration for the IPM application was to make the most effective use of the entire web browser window to display the map and present users with additional information as and when needed.

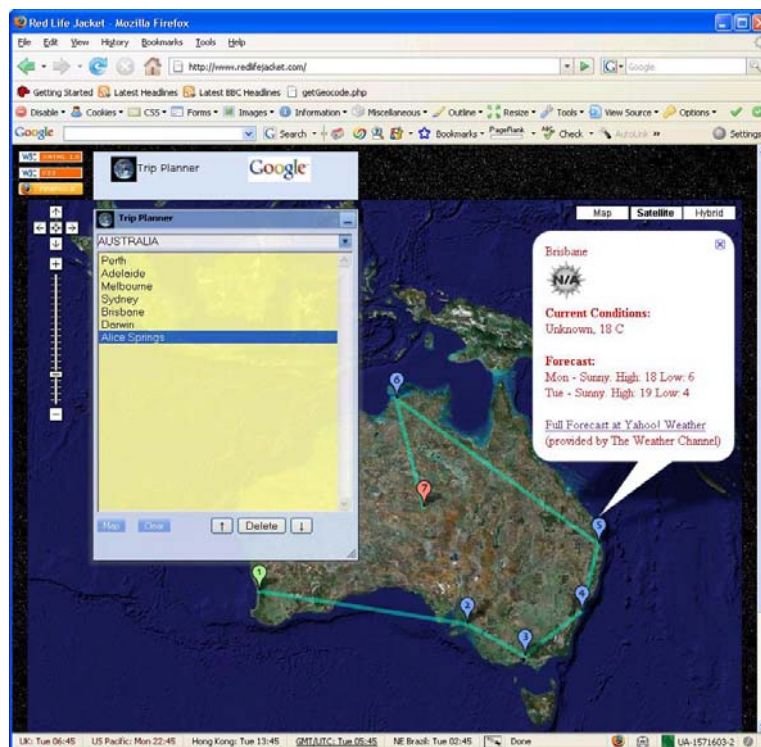


Fig. 1. Planning a travel route using Google Maps API and Yahoo Weather RSS feeds

User interface features such as moveable and resizable windows were implemented using the Dojo Floating pane widget. The text auto-completion feature is implemented using the Dojo combo-box widget.

The Trip Planner floating pane allows users to type the country name into a Dojo combo box widget; the Trip Planner then populates the list-box with the list of major cities and charts the route on the map as illustrated in Figure 1. When users select (click on) a place marker on the map, the application displays the current weather information using the Yahoo weather RSS feeds.

Selecting a destination from the list-box and clicking the “Google” icon causes the application to launch a new floating pane containing the search results that relate to the selected destination. The search floating pane includes Video, Blog, News, Book, Local and Web results as shown in Figure 2. The place markers on the map are numbered according to the order specified in the Trip Planner list and the colors indicate the start (green) and end (red) of the trip.

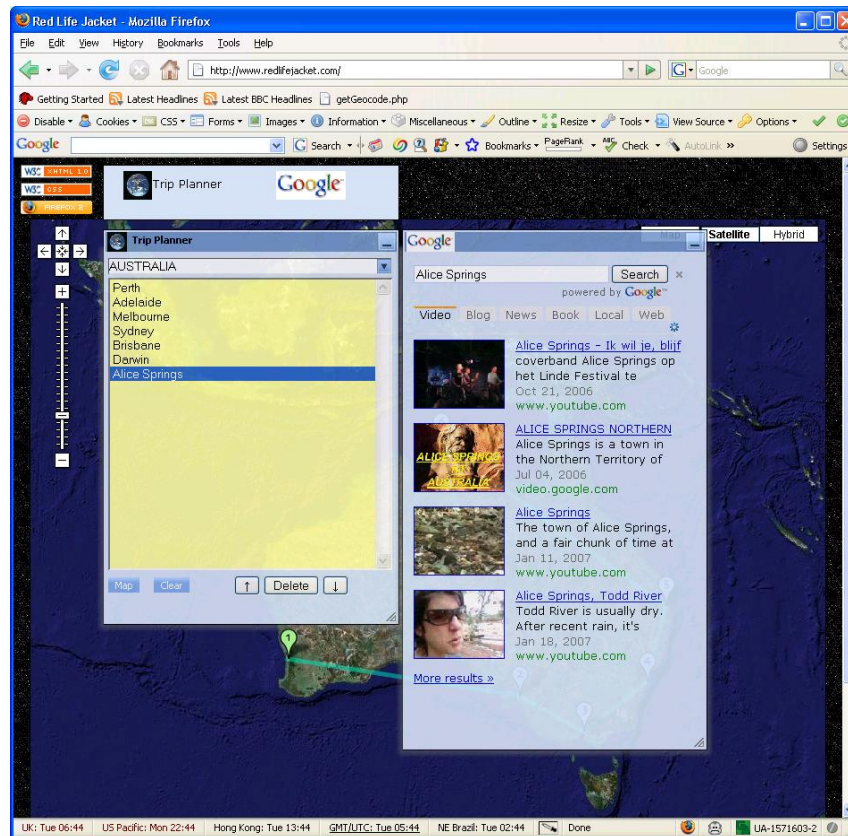


Fig. 2. Using the Google Ajax Search API to display local information for a selected city

2.2 Data Integration Design

As illustrated in Figure 3, the IPM application combines content from a number of different sources and presents an integrated view of the information to the user. The

Google Maps API retrieves Google Maps, the Google Ajax Search API allows customized display of Google Search results and the Yahoo RSS Weather feeds provide the current weather and forecast information. While the Google Maps API and the Google Ajax Search API are accessed directly by the client-side Javascripts, the Yahoo RSS feeds are retrieved by a server-side PHP script. The PHP XML_RSS⁹ Parser was used to process the RSS feeds and return the latest weather forecast information as a JSON data structure as explained in section 3.

The following factors need to be considered when addressing the data requirements of mashup applications:

- i) what information is required and where to source it from
- ii) Data synchronization requirements, i.e. how frequently should the information be refreshed
- iii) decisions about the locality of data, i.e. should the information be replicated locally

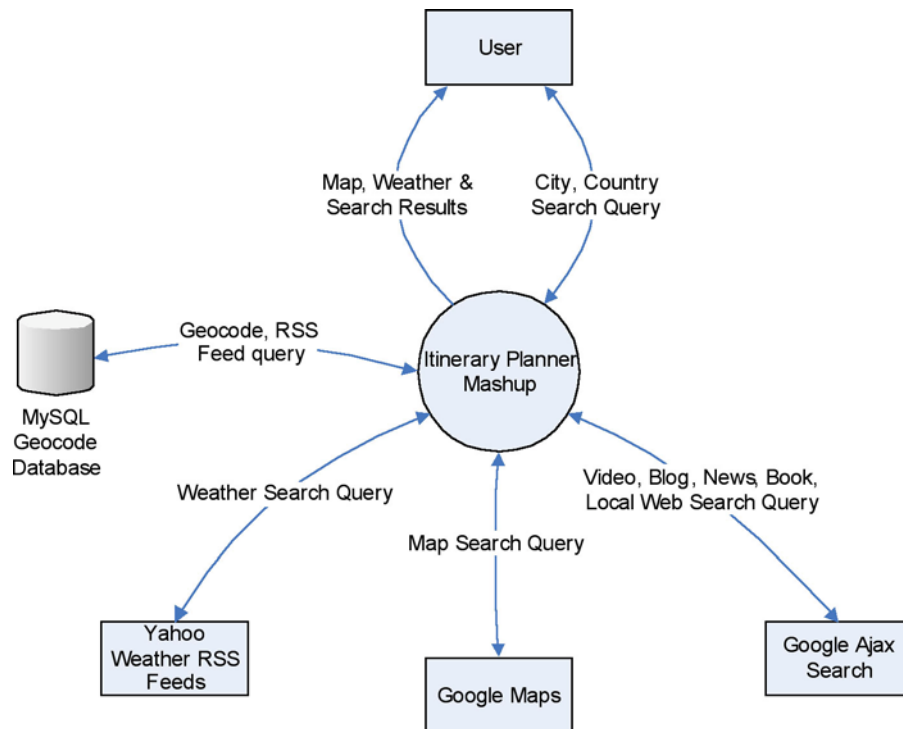


Fig. 3. Context diagram of the IPM application showing local and remote data sources

The above design decisions impact on the correctness of the information, application performance and availability of data. Excessive asynchronous API requests between the client and the server affect application performance, resulting in

⁹ http://pear.php.net/package/XML_RSS

“chatty” applications. While the chattiness of mashups is heavily influenced by the API features of third party information sources such as Google Maps and Google Ajax Search, application performance can be significantly improved by storing static data in a local database. Storing static data (e.g. city/country listings, Yahoo weather RSS Feed location ID settings, latitude and longitude values for the various locations) avoids the need for continuously refreshing this information. The caching of latitude and longitude values in a local (MySQL) database was an important factor in optimizing the performance of the IPM application. Real-time geocode resolution would incur a heavy cost in terms of latency and the number of geocode requests sent over the network during map rendering.

3. IPM Implementation

The IPM application relies heavily on the AJAX features available in the Dojo Javascript Toolkit. The following sections describe the implementation of the various features used in the IPM application. The text auto-completion feature (implemented with the Dojo combo-box widget) and the geocode lookup feature both use the JSON format to interact with the server.

3.1 Country lookup Auto-completion

Each character typed in the combo-box results in a GET request being sent to the `getCountries.php` script on the server. The “type” GET parameter indicates that the client expects the response in JSON format and the “matching” GET parameter indicates that the client expects a list of all countries beginning with “a”.

Request

<http://www.redlifejacket.com/getCountries.php?type=json&matching=a>

Response (JSON)

```
{ "AFGHANISTAN": "AFGHANISTAN", "ALGERIA":  
"ALGERIA", "ARGENTINA": "ARGENTINA", "AUSTRALIA":  
"AUSTRALIA", "AUSTRIA": "AUSTRIA" }
```

3.2 City list lookup

When the “onValueChanged” event is triggered in the combo-box, a GET request is sent to the `getCities.php` script. The “country” GET parameter is set to AUSTRALIA, indicating that the client expects a list of major cities within Australia. The response from the server is a comma separated list of cities. The callback Javascript function populates the listbox with the list of cities fetched from the server.

Request

<http://www.redlifejacket.com/getCities.php?country=AUSTRALIA>

Response (CSV)

Adelaide, Alice
Springs, Brisbane, Darwin, Melbourne, Perth, Sydney

3.3 Geocode lookup

When users click on the “Map” button, a GET request is sent to the getGeocode.php script on the server with the “cityList” GET parameter set to a comma-separated list of cities. The response from the server is in JSON format. The data structure used for the JSON response is an array of cities. Each city element has the Name, Latitude, Longitude, Description and YahooRssFeed attributes.

Request

<http://www.redlifejacket.com/getGeocode.php?citylist=Sydney>

Response (JSON)

Params Headers Response

```
[{"name": "Sydney", "lat": -33.8666666666667, "lng": 151.2, "yahooRssFeed": "http://xml.weather.yahoo.com/forecast/rss?p=ASXX0112&u=c", "description": "<div class='\"placemark\"'>Sydney<div>\n<img src='\"http://l.yimg.com/us.yimg.com/i/us/we/52/30.gif\"' /\><br /\>\n <b>Current Conditions:</b><br /\>\n Partly Cloudy, 14 C<br /\><br /\>\n <b>Forecast:</b><br /\>\n Mon - Mostly Sunny. High: 13 Low: 5<br /\>\n Tue - Sunny. High: 13 Low: 6<br /\>\n <br /\>\n<a href='\"http://us.rd.yahoo.com/dailynews/rss/weather/Sydney_AS/*http://weather.yahoo.com/forecast/ASXX0112_c.html\"'>Full Forecast at Yahoo! Weather</a><br /\>\n (provided by The Weather Channel)<br /\>\n </div></div>"}]
```

The getGeocode.php script performs a number of sequential tasks. Firstly, the city information is retrieved from the local database using the following SQL query:

```
SELECT city.name, latitude, longitude, yahooLocationId
FROM city, country
WHERE city.Country_idCountry = country.idCountry
AND country.name like '%AUSTRALIA%'
```

Name	latitude	longitude	yahooLocationId
Brisbane	27° 28' S	153° 2' E	ASXX0016
Melbourne	37° 49' S	144° 58' E	ASXX0075
Sydney	33° 52' S	151° 12' E	ASXX0112

Then the latitude and longitude values are converted into decimal format and a Yahoo RSS feed URL constructed using the YahooLocationId values. The “u=c” GET parameter indicates that the client expects the result in Celsius. Next, the Yahoo RSS feed is parsed and the “description” element stored. Finally, a new instance of the JSON service is created and the JSON encoded city list array is returned back to the client (Table 1).

1.	<code>\$sql = "select name, latitude, longitude, yahooLocationId from city where lcase(name) in \$str";</code>
2.	<code>\$city['lat'] = getLatLonAsDecimal(\$row['latitude']);</code>
3.	<code>\$city['lng'] = getLatLonAsDecimal(\$row['longitude']);</code>
4.	<code>\$city['yahooRssFeed'] = "http://xml.weather.yahoo.com/forecastrss?p=" . \$row['yahooLocationId'] . "&u=c";</code>
5.	<code>\$city['description'] = "<div class=\"placemark\">" . \$city['name'] . parseRss(\$city['yahooRssFeed']) . "</div>";</code>
6.	<code>\$json = new Services_JSON();</code>
7.	<code>echo \$json->encode(\$cities);</code>

Table 1. Code snippet from the getGeocode.php script

4 Technical Considerations

In this section we discuss the technical challenges encountered when developing mashup applications, including security, performance, and debugging.

4.1 Security

The widespread adoption of AJAX has resulted in security vulnerabilities such as exposure to Cross-site scripting (XSS) [6], Cross-Site Request Forgery (CSRF) [7] and phishing [8], [9]. The increasing trend towards the use of JSON formatting instead of XML makes an attack against the data transport mechanism (e.g. Javascript Hijacking) possible as the security model of web browsers does not support the use of Javascript for the transport of confidential information and allows attackers to circumvent the “Same Origin” policy [10]. The typical way of using JSON responses

from the server (i.e. by invoking the `eval()` function) can expose the application to attacks. One way to address this issue is to enclose the JSON response with Javascript comment characters (`/* */`) on the server-side and remove these comment characters before evaluating the response on the client-side.

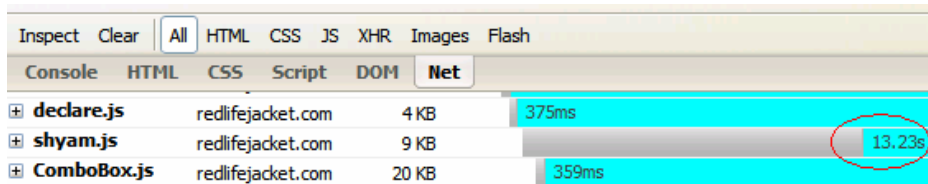
In our IPM application we initially considered using XML as the format for data interchange with server-side PHP scripts, but it soon became evident that JSON was a more viable option due to its seamless integration with the client-side Javascript application. XML was found to be too verbose, and considerably less programming effort was required to exchange data between the client and server using JSON as this avoids the parsing of XML messages. However, as noted above, there are security considerations that need to be addressed when using JSON as the data interchange format.

The server-side PHP scripts, used in the IPM application, are accessed using REST-based interactions [11]. These scripts extract the GET parameters from the URL and pass them as arguments to SQL queries to fetch data from the database. It would be possible for a malicious user to perform a SQL injection attack by passing invalid arguments to the server-side scripts. For instance, in the absence of any safeguards against such an attack, a hacker could delete all data from the underlying MySQL database. The application can be protected against these attacks by using the “`mysql_real_escape`” function to escape special characters used to hijack the SQL queries and by using Prepared Statements to bind variables into the SQL query at runtime.

While there are well established standards, such as WS-Security and WS-Secureconversation for securing XML based messages using the SOAP protocol [12], so far there has been little progress on this front for JSON based message exchanges. While some techniques are available that mitigate against this risk [13] the security issues in the context of Javascript remain largely unresolved.

4.3 Performance

Although the Dojo Javascript toolkit provides a rich collection of widgets for sophisticated AJAX application development, there are significant performance tuning and optimization issues that need to be addressed during the development and deployment of mashup applications. For example, the initial loading of the IPM application was significantly impacted by the time taken to load the Dojo widgets. As shown in Figure 4, the Firebug profiling tool was used to identify bottlenecks in the application (e.g. `shyam.js` file took 13.23s to load). This delay could be attributed to dependency resolution in the Dojo framework. Delaying the loading of these widgets to later in the application execution, in response to user interaction could reduce the time taken for system initialization.



File	Source	Size	Time
declare.js	redlifejacket.com	4 KB	375ms
shyam.js	redlifejacket.com	9 KB	13.23s
ComboBox.js	redlifejacket.com	20 KB	359ms

Fig. 4. Identifying performance bottlenecks using the Firebug profiling tool.

The Dojo community aims to address the performance tuning issues [14] by focusing on four areas, namely, download, parsing, instantiation and deferred download. Dependency resolution through synchronous widget requests using the `dojo.require()` statement results in latency and network overhead. A significant reduction in the network I/O and CPU load times can be achieved by configuring the web server to cache static content such as Javascript files without requesting status information [15].

4.4 Debugging and Validation Tools

Since most of the processing happens within the context of the web browser, traditional server-side debugging techniques such as setting different log levels and log file inspection are not adequate in AJAX development. Specialized debugging tools such as Firebug¹⁰ help troubleshoot AJAX applications. Firebug provides several useful features such as Javascript debugging, profiling, network monitoring, CSS editing, CSS visualization and HTML inspection. Firebug was used extensively during development to troubleshoot event handling and Dojo IO binding issues. Other tools such as the Mozilla DOM Inspector¹¹, Mozilla Venkman Debugger¹² and Firefox Inspect Element¹³ are also useful in AJAX development.

4.4.1 XHTML Validation

Ensuring that the application conformed to the W3C XHTML validation standards was a key consideration during development. XHTML conformance would allow better integration of the application to other mashups, feeds, web services and third-party APIs. This requirement was imposed upon the application as an after-thought (after the selection of Dojo). This section briefly discusses the issues encountered in meeting this requirement using the Dojo framework. Use of the “dojotype” attribute resulted in XHTML validation errors. There are two ways of using widgets in Dojo: 1) creating the widget using specialized Dojo attributes in HTML, 2) creating the

¹⁰ <http://www.getfirebug.com/>

¹¹ <http://www.mozilla.org/projects/inspector/>

¹² <http://www.mozilla.org/projects/venkman/>

¹³ <https://addons.mozilla.org/en-US/firefox/addon/434>

widget programmatically in Javascript. While the first method is easier to implement, it does not produce valid XHTML. Creating widgets programmatically in Javascript, results in simple and clean HTML. The following example illustrates how the Dojo Taskbar widget was converted from HTML to Javascript.

Method 1: Create the widget using specialized Dojo attributes in HTML

```
<div dojoType="TaskBar" id="mytaskbar"
resizable="false"; class="headContainer">
</div>
```

Method 2: Create the widget programmatically in Javascript

```
<div id="taskbar" class="taskbar"></div>
<script type="text/javascript">
    function initTaskBar() {
        var properties = {
            id:"mytaskbar",
            resizable:false
        }
        var taskbarNode = dojo.byId('taskbar');
        taskbar =
        dojo.widget.createWidget("TaskBar",properties,taskbarNode);
    }
</script>
```

4.5 Cross-platform Issues

Given the prevalence of web browser incompatibilities and the varying levels of conformance to standards amongst web browser vendors [3], there is an increasing need for proven, tested and verified frameworks to assist the developers of mashup applications. Several competing client-side frameworks such as the Google Web Toolkit (GWT)¹⁴, Dojo¹⁵, Prototype¹⁶, Scriptaculous¹⁷, Mochikit¹⁸, Yahoo UI¹⁹, etc aim to alleviate the cross-browser compatibility issues by hiding the browser specific functionality, and allowing developers to focus on high level user interface design using widgets. Although these frameworks facilitate fast development of small to medium sized web applications, their suitability for mission-critical and scalable, enterprise applications is yet to be determined.

¹⁴ <http://code.google.com/webtoolkit/>

¹⁵ <http://dojotoolkit.org/>

¹⁶ <http://www.prototypejs.org/>

¹⁷ <http://script.aculo.us/>

¹⁸ <http://www.mochikit.com/>

¹⁹ <http://developer.yahoo.com/yui/>

5 Conclusions

Currently there is extensive developer interest in Web 2.0 and more specifically in the development of second generation web applications using AJAX. With the emergence of free online data aggregation and manipulation services such as Yahoo Pipes²⁰, the level of interest is likely to increase even more dramatically.

At present, mashups are used primarily for developing innovative applications with emphasis on providing rich user experience and without much consideration for the long-term viability of such applications. Many of these applications will not survive the test of time and end up as “disposable applications” with short lifetimes. The suitability of mashups for the development of mission-critical, high availability applications is yet to be determined and requires careful design consideration.

It is inevitable that new standards and best practices will emerge to address the current security and performance challenges. However, given the fast-changing nature of the underlying information sources and the continuous emergence of new specialized APIs, there is an urgent need for methods to support the seamless integration of various types of data. Different types of mashup applications share many common features and design patterns are emerging that are applicable to mashup applications.

The experience gained during the implementation of the IPM application and described in this paper was invaluable for our next project, the Olympic Torch Relay Mashup (OTRM) application. Similar to the IPM application, we have adopted a solution that uses a local database to store static information to improve the performance of the application. The city, region and the scheduled date of arrival of the torch are stored in a local MySQL database along with the latitude and longitude values which are derived using the Google Maps API. (Beijing Olympic torch relay route is represented in a chronological form at <http://torchrelay.beijing2008.cn/en/journey/>.)

Another important lesson learnt from the implementation of the IPM application is that the representation of information as required by the user-interface needs to be taken into account when choosing a suitable data structure for storing the data. While the IPM application uses the text auto-completion feature provided by the Dojo combo-box widget, the OTRM application uses the Dojo Tree widget to store the region and city information. Both applications required transformation of relational data stored in a local database into a hierarchical structure for easy traversal using Javascript when the data is being displayed. Further research is needed to identify data transformation patterns that are frequently used in geospatial mashup applications and to provide generic solutions for such requirements.

References

1. www.oreillynet.com. *O'Reilly -- What Is Web 2.0*. 2008 [cited 30 January 2008]; Available from:

²⁰ <http://pipes.yahoo.com/pipes/>

- <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html?page=1>.
2. Zhao, L., *Integrating rank correlation techniques with GIS for marketing analysis*, in *GeoComputation 2000*. 2000: University of Greenwich, United Kingdom.
 3. Doernhoefer, M., (2006) *JavaScript*. SIGSOFT Softw. Eng. Notes, Vol. 31 (4): p. 16-24, ISSN 0163-5948
 4. Wikipedia *Single page application*. [cited 2007; Available from: http://en.wikipedia.org/wiki/Single_page_application.
 5. Mesbah, A., *Ajaxifying Classic Web Applications*, in *Companion to the proceedings of the 29th International Conference on Software Engineering*. 2007, IEEE Computer Society.
 6. Neville-Neil, G., (2005) *Vicious XSS*. Queue, Vol 3 (10): p. 12-15, ISSN 1542-7730
 7. Johnson, D., A. White, and A. Charland, *Enterprise Ajax*. 1 ed. 2008: Prentice Hall. ISBN 978-0-13-224206-0
 8. Fette, I., N. Sadeh, and A. Tomasic, *Learning to detect phishing emails*, in *Proceedings of the 16th international conference on World Wide Web*. 2007, ACM Press: Banff, Alberta, Canada.
 9. Yu, D., et al., *JavaScript instrumentation for browser security*, in *Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 2007, ACM Press: Nice, France.
 10. Chess, B., Y. Tsipenyuk O'Neil, and J. West, *JavaScript Hijacking*. 2007.
 11. www.ics.uci.edu. *Architectural Styles and the Design of Network-based Software Architectures*. 2002 [cited 31 January 2008]; Available from: <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.
 12. Bhargavan, K., et al., (2007) *Secure sessions for Web services*. ACM Trans. Inf. Syst. Secur., Vol. 10 (2): p. 8, ISSN 1094-9224
 13. *A Note on "JavaScript Hijacking"*. 2007 [cited; Available from: <http://dojotoolkit.org/node/619>.
 14. *Performance Optimization*. 2007 [cited; Available from: <http://dojotoolkit.org/book/dojo-book-0-4/part-6-customizing-dojo-builds-better-performance/performance-optimization>.
 15. *Improving performance of Dojo-based web applications*. 2007 [cited; Available from: <http://lazutkin.com/blog/2007/feb/1/improving-performance/>.