Web 2.0: Building Enterprise Portals

George Feuerlicht,
Shyam Govardhan

Faculty of Information Technology
University of Technology, Sydney
Level 2, Building 10, 235 Jones Street,
Broadway 2007
e_mail: jiri@it.uts.edu.au, ssgovard@it.uts.ed.au

***Abstract**: Various types of Web resources are being made available programmatically using APIs and through data feeds, facilitating the development of highly interactive applications with rich media content - the so called Web 2.0 applications. Most current Web 2.0 applications are simple mashups that combine information from multiple sources and provide a Windows-like user interface. More sophisticated Web programming techniques based on portable and reusable application components are becoming available and enable the construction of highly innovative applications that deliver real business value to organizations. In this paper we discuss the challenges of developing Web 2.0 applications and describe the design of an advanced mashup application, the Lenovo Olympic Podium, that illustrates the potential of Web 2.0.*

Keywords: Web 2.0, AJAX, Mashup, Google Gadgets

1. Introduction

Mashups have emerged recently as a popular alternative to Web Services for accessing Web resources, and are rapidly becoming an integral part of the service-oriented computing paradigm. Mashups combine diverse sources of data available via public APIs (Application Programming Interfaces) such as the Google Maps API (http://picasa.google.com/), YouTube API (http://www.youtube.com/), Picasa Web Albums API (http://picasa.google.com/) and through data feeds such as Atom and RSS feeds.

Web Services have been used extensively for the implementation of transactional e-business (electronic business) applications, and a large number of Web Services standards have been specified to provide security and to ensure reliability of e-business applications. Some experts argue that as a result of adding layers of functionality, the Web Services stack has become excessively complex and that this complexity has impacted on its widespread adoption and deployment [1]. While the complexity associated with the use of Web Services can be justified in mission-critical (transactional) enterprise applications, there are many other types of Web applications that are primarily concerned with accessing and aggregating various data sources and do not require such a comprehensive infrastructure. This has resulted in the emergence of a new Web 2.0 environment that relies on light-weight programming solutions, and provides an alternative to accessing Web resources using Web Services.

Web 2.0 [2] applications are characterized by programmatic access to Web resources via light-weight APIs that are used to create value-added services in the form of mashups using the AJAX programming environment. The rapid growth of various Web APIs (Application Programming Interfaces) and their fast adoption by the developers of Web applications are transforming the Web into a programmatic platform [3].

However, the sophistication (and complexity) of the Web 2.0 programming environment is growing as mashup programmers attempt to address more demanding user requirements. 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 [4]. More recently, advanced programming techniques based on light-weight portable Web applications such as Google Gadgets that can be deployed on the third-party websites using Gadget Syndication, have been added to the Web 2.0 programming environment. While simple mashup applications can be implemented relatively easily, more sophisticated use of advanced AJAX programming techniques requires the re-evaluation of programming models and design techniques to ensure that the resulting Web applications are maintainable and perform according to specifications.

We have reported on the challenges of developing mashup applications in an earlier publication [5]. In this paper we describe the implementation of an advanced mashup application, the Lenovo Olympic Podium (LOP) that was developed by a cross-functional global team from Google and Lenovo (http://2008.lenovo.com/), and illustrates the potential of the Web 2.0 platform. The main purpose of the LOP application is to display information about the various Olympic venues and torch relay destinations, using multiple sources of data drawn from various customs feeds (Olympic event schedules, etc), news feeds, Picasa images, Google maps, YouTube video clips, RSS feeds, and Google Calendar. The approach used for developing the LOP portal involved first implementing a working prototype mashup application (Olympic Mashup). This was then followed by using more advanced techniques to implement the final LOP application.

In this paper we first describe the techniques and strategies employed during the development of the initial *proof-of-concept* Olympic Mashup prototype (section 2). The Olympic Mashup prototype was used to demonstrate the potential of the Web 2.0 programming platform and to identify user requirements more precisely. We then discuss how the lessons learned from implementing the prototype were applied during the design and implementation of the Lenovo Olympic Podium (section 3). In conclusions (section 4) we discuss the potential for such sophisticated mashup applications to deliver real business value to organizations.

2. Initial Proof-of-Concept Prototype

Prior to commencing work on the Lenovo Olympic Podium, an initial proof-of-concept working prototype (Olympic Mashup) was developed to explore the capabilities of the available APIs and their suitability for the Olympics project. In the following sections we discuss the design considerations involved in implementing the prototype. We consider four separate areas of mashup application design: User Interface Design (section 2.1), Data Integration Design (section 2.2), Performance (section 2.3), and Security (section 2.4).

2.1 User Interface Design

An important design consideration for mashup applications is to make the most effective use of the available web browser window to display the maps and present users with additional information as and when needed. User interface features such as moveable and resizable windows are implemented using the Dojo Floating pane widget [6].

Figure 1 shows the Olympic Mashup user interface. The torch relay destinations are displayed in a tree format in the floating pane on the left. When users click on a location, the application displays search results from Google, a slideshow of images for the selected location and charts the torch relay route on the underlying map.

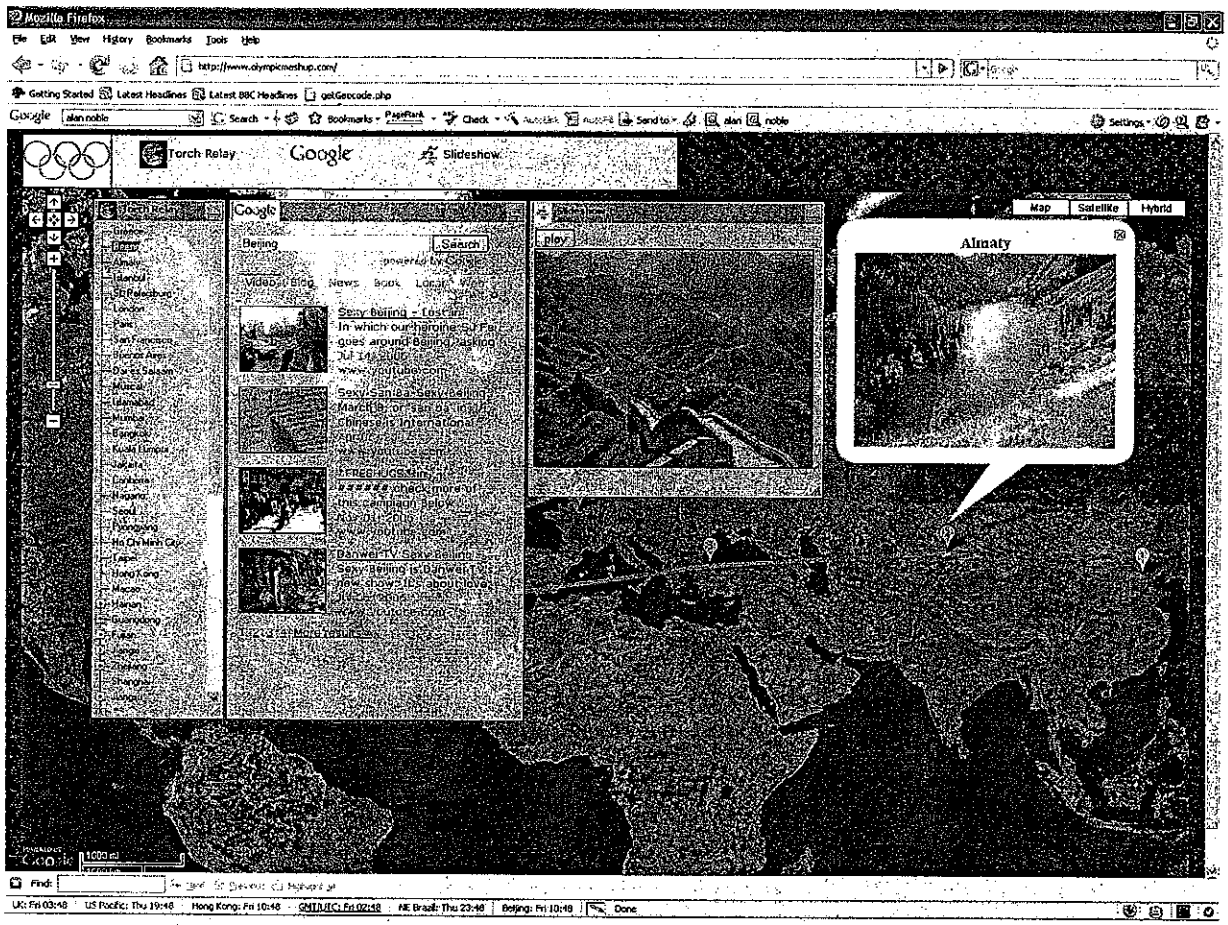

Fig 1. The Olympic Mashup prototype showing information about torch relay destinations

The Google Maps API (http://code.google.com/apis/maps/) allows the integration of Google's interactive maps with data from other sources. The Olympic Mashup application uses the Google Maps API to display the torch relay route, the Google Ajax Search API to display search results and the Dojo Javascript framework to display a slideshow of images.

2.2 Data Integration Design

As illustrated in Figure 2, the Olympic Mashup 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 images are retrieved from an external image repository.

In general, the following factors need to be considered when addressing the data integration 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

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 and result in "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 listings and latitude and longitude values for the various locations) avoids the need to continuously refresh this information. The caching of latitude and longitude values in a local (MySQL) database was an important factor in optimizing the performance of the Olympic Mashup prototype. Real-time geocode resolution would incur a significant overhead, increasing the latency and the number of geocode requests sent over the network during map rendering.

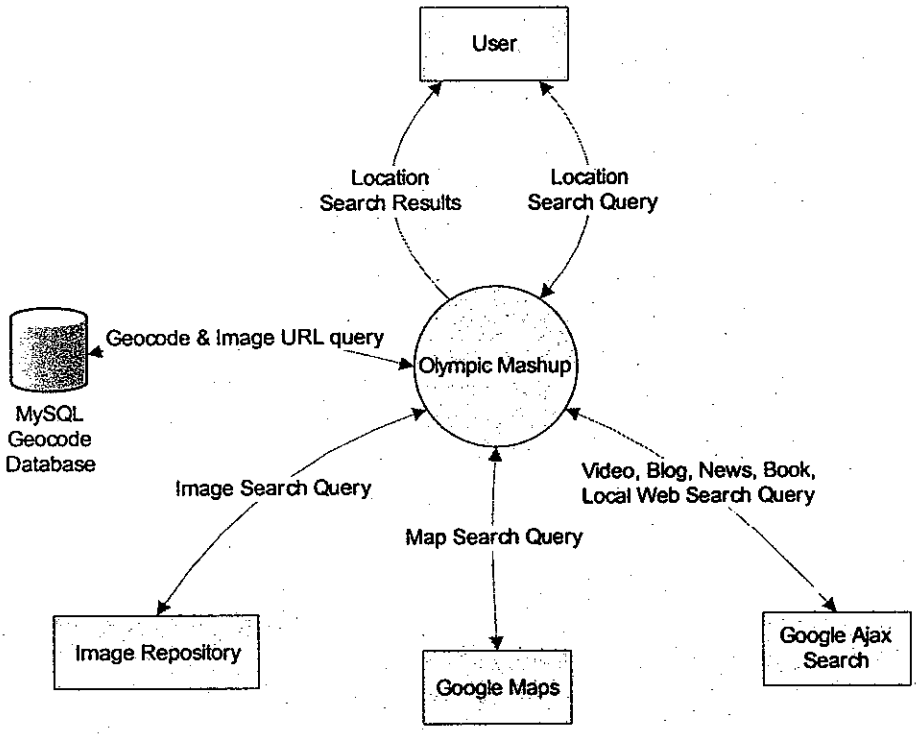

Fig 2. Olympic Mashup (proof-of-concept) Context Diagram

To avoid such overheads, the URLs for the images are pre-fetched from the Olympic Torch relay website using a PHP script and stored in a local MySQL database. The PHP script uses regular expressions to extract the image URLs from the HTML files on the Olympic Torch Relay site. While the screen-scraping of third-party websites has legal implications, the purpose of the Olympic Mashup prototype application was to explore the possibilities and demonstrate the capabilities of mashup applications; this feature was not used in the final LOP application.

2.3 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 considered during the development and deployment of mashup applications. For example, the initial loading of the Olympic Mashup application was significantly impacted by the time taken to load the Dojo widgets. This latency could be attributed to dependency resolution in the Dojo framework. Delaying the loading of these widgets to later on in the application execution, and loading the widgets on-demand (i.e. in response to user interactions) reduces the time taken for system initialization. Performance of Dojo widgets could be improved 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 transmission and load times can be achieved by configuring the web server to cache static content such as Javascript files without requesting status information [7].

2.4 Security

The widespread adoption of AJAX has resulted in security vulnerabilities such as exposure to Cross-site scripting (XSS) [8], Cross-Site Request Forgery (CSRF) [9] and phishing [10], [11]. The increasing trend towards the use of JSON as a transport mechanism instead of XML makes an attack against the data transport mechanism (e.g. Javascript Hijacking) possible [12] 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 [12]. The typical way of using JSON responses from the server, by invoking the eval() function, can expose the application to attacks. A popular method of addressing this issue is to enclose the JSON response with Javascript comment characters (/* */) on the server-side and removing these comment characters before evaluating the response on the client-side. Security is however an ongoing concern for mashup applications and extensive effort is being made to address security of AJAX applications.

3. Lenovo Olympic Podium Case Study

Following the implementation of the Olympic Mashup prototype and identifying the various issues discussed in the previous section, we have applied more advanced programming techniques to the construction of the final LOP application. Web 2.0 technologies are advancing at a rapid rate and the methods employed in the Olympic Mashup proof-of-concept prototype application had to be significantly revised to allow the use of more advanced programming techniques.

3.1 Google Gadgets

An important component of advanced mashup applications are portable and reusable frameworks such as Google Gadgets [13]. Google Gadgets are light-weight portable Web application components based on standard technologies (i.e. HTML, XML, CSS and Javascript). Google Gadgets use XML to encapsulate the HTML elements used within the application, and can be deployed on personalized iGoogle home pages (http://www.google.com/ig), as well as third-party websites using Gadget Syndication [14]. Google Gadgets support user interface features such as tabs through the Google Gadgets Javascript API [15]. Using this interface allows the display of different types of information including images, videos and text.

3.2 User Interface Design

Since the LOP application features Google Gadgets that are hosted on a Personalised Start Page (PSP) using Google Applications, the user interface design of the main page was significantly influenced by the three column layout provided by the PSP. While users have the option of adding new Gadgets and customizing the location of the Gadgets within the page, the width of the Gadgets was limited to a third of the browser window size. The look and feel of the "mini-application" that executed within the Gadgets themselves was entirely customizable.

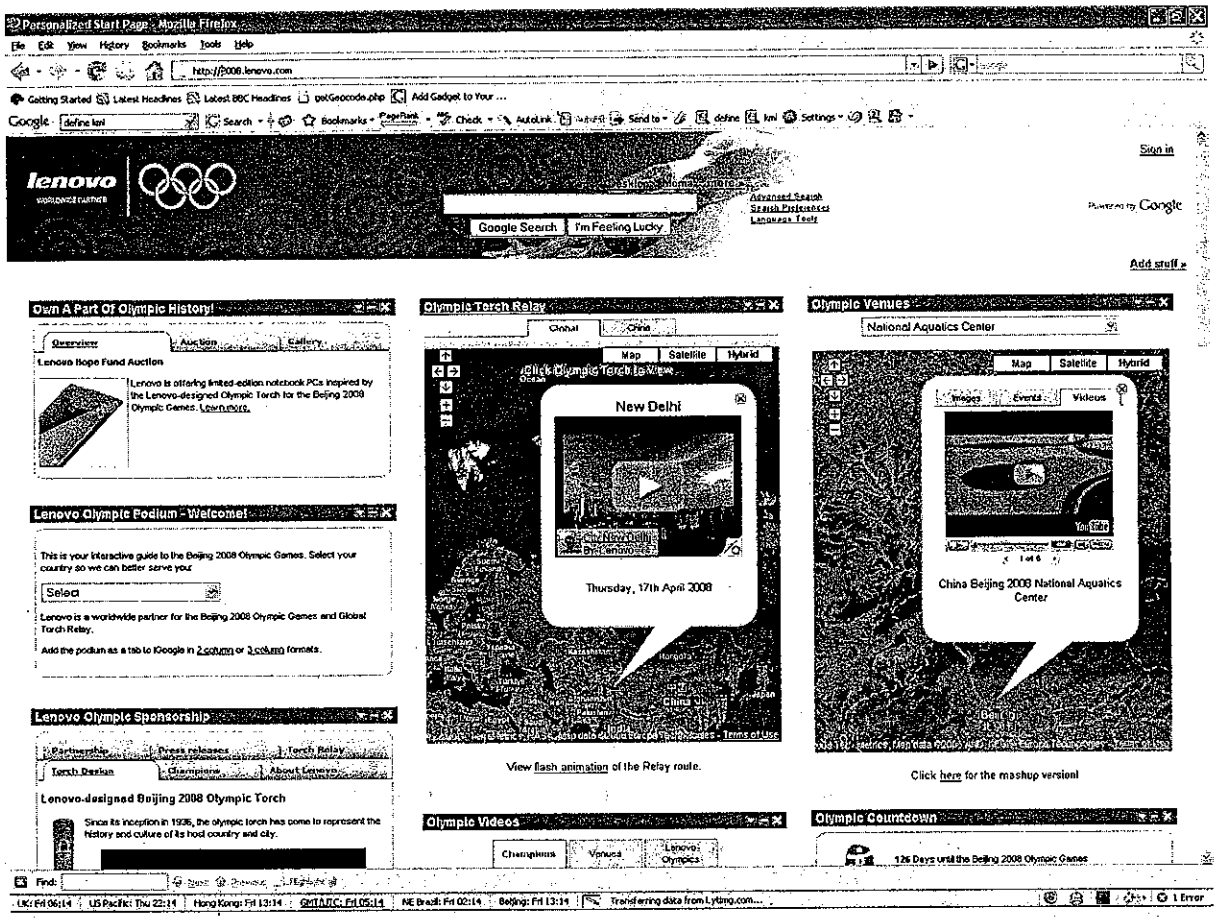

Fig 3. Lenovo Olympic Podium

As shown above, the Olympic Venues Gadget displays the locations of all Olympic venues in Beijing on a Google Map. When users select a venue, an information window opens displaying images from Picasa, videos from YouTube and the Olympic event schedule for the selected location. While, most mapping applications display static content such as text, hyperlinks and images on the map, combining Google Gadgets with Google Maps allows dynamic functionality within a map (i.e. users can interact with individual markers on the map using the Google Maps API).

3.3 Data Integration Design

While the Olympic Mashup proof-of-concept application used a local database to store the latitude and longitude values of the Olympic torch relay destinations, the Lenovo Olympic Podium took a different approach by storing data in JSON format in Javascript files. The JSON files used by the application were pre-generated, removing the need for real-time access to a database. Given that Gadgets and Mapplets [16] are light-weight, portable application components hosted on the google.com domain, real-time database access was no longer needed and would have significantly degraded application performance.

As shown in Figure 4, the LOP application integrates content from Google Maps, Picasa, YouTube, Blogger, News, Calendar and other custom feeds such as the Olympic Event Schedule.

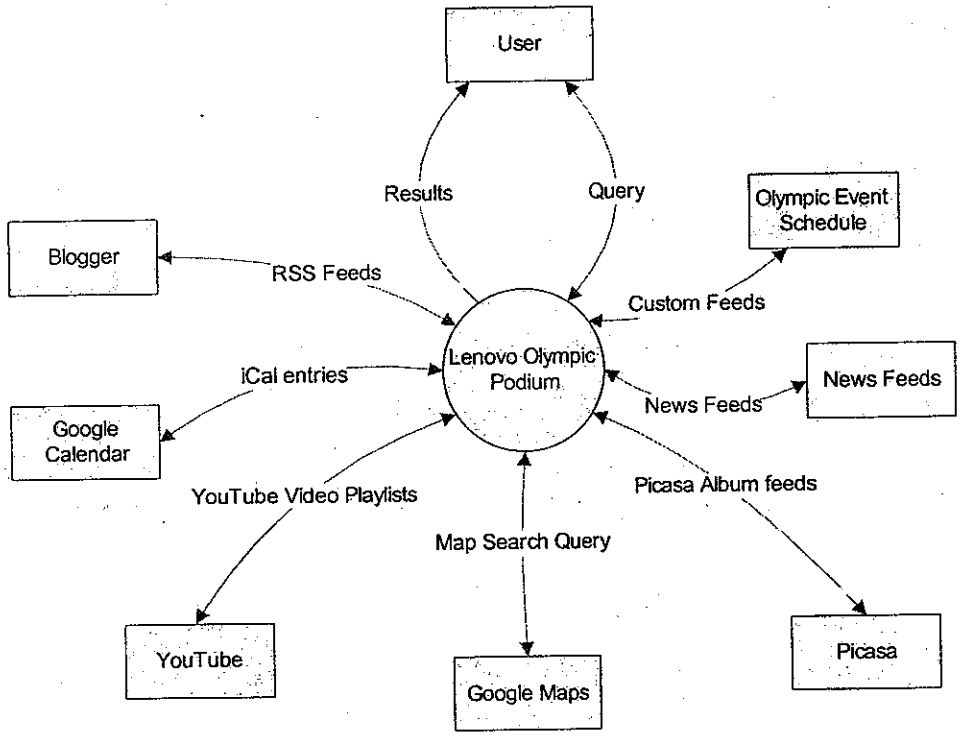

Fig 4. Context Diagram of the Lenovo Olympic Portal

As discussed in more detail below, the Beijing 2008 Olympic Venues Gadget uses KML to display all the Olympic venues on the map. When a user clicks on a place marker on the map, the application displays videos from YouTube, a slideshow of images from Picasa and the Olympic competition schedule within the information window using a tabbed interface. The application displayed within the information window is a syndicated Google Gadget embedded within a pre-generated KML file.

3.4 Keyhole Markup Language (KML)

KML is an XML file format used to plot and describe locations on a map. The KML file format is specific to Google Maps and Google Earth. Features such as placemarks, polygons, images, textual descriptions and 3D models can be displayed in Google Earth, Google Maps and Mobile using the KML file format. Display of a large numbers of place markers on the map, through iteration using the Google Maps API, could cause a considerable delay in the loading of the map. For instance, iterating through 400 markers using Javascript took more than 1 minute and in some cases, caused the browser to crash. A significant re-design of the application was needed to address this issue. While the dynamic display of markers, based on the current viewport, resulting from zooming and panning events was considered, it soon became apparent that this technique required significant programming effort. A more effective option was to generate KML files for the Olympic torch relay destinations and Olympic venues and to add the KML file as a layer to the map. This can be accomplished through one asynchronous call to the Google Maps API "addOverlay()" method in Javascript.

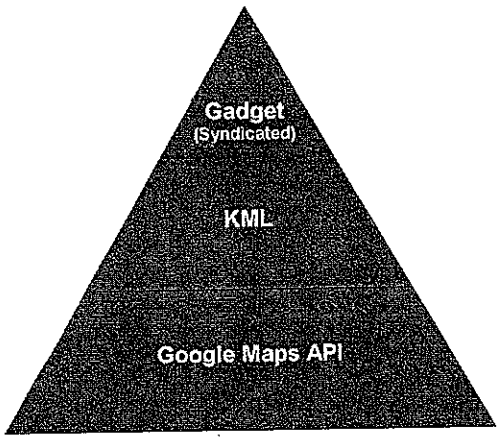

Fig 5. Gadget Syndication using KML

The second challenge with using KML related to displaying dynamic functionality within the information window on the map. The Google Maps KML server restricts what can be displayed within the information window using KML; while Google Gadgets can be displayed within the information window, generic HTML Iframes cannot be shown. For this reason, we decided to create an Olympic Venue Gadget and "syndicate" this Gadget for inclusion in the KML file. Google Gadget syndication allows developers to display Gadgets on third-party websites as light-weight portable applications.

3.5 YouTube and Picasa Integration

Individual playlists were created in YouTube for the various video categories. The Olympic Videos Gadget accessed these playlists using the YouTube GData API [17].

Fig 6. Olympic Videos (YouTube) and Olympic Images (Picasa) Gadgets

The Olympic Images Gadget displays a slideshow of images from Picasa using the Picasa slideshow widget, which accesses the images from their respective Picasa web albums. The Gadgets also feature

a drop-down list-box that allows users to select a video or image category by name. Users can navigate through the videos using the arrow buttons displayed below the video. Although this Gadget is fairly simple in functionality, it is a good example of how information (including rich video content) can be accessed using an open API and data feeds to produce a compelling end user experience without much programming effort and with minimal hosting requirements.

4. Conclusion

In this paper we have described the implementation of a sophisticated mashup application that uses portable and reusable components (Google Gadgets), and advanced programming techniques. We have contrasted these advanced techniques with the techniques used in the development of a mashup prototype application. Such *first generation mashups* are simple to design and can be implemented without much programming effort, but suffer from a number of limitations that make these types of applications difficult to maintain. As new, more advanced programming techniques emerge, the design of mashup applications needs to be revised to take advantage of technologies such as Google Gadgets. These *second generation* mashups are based on reusable design patterns and enable the implementation of enterprise-level Web 2.0 applications that use a large number of data sources, reusable components, and deliver real business value to organizations.

The LOP application has been highly successful and was well received by the users who commented on the wide range of functionality and the highly dynamic style of the user interface. The following is a quote from David Churbuck, the Vice-President of Global Web Marketing at Lenovo [18]:

"Starting in August we began discussions at the highest levels about using Google's iGoogle platform to build a sophisticated Olympic platform of our own. It is live, it is http://2008.lenovo.com. It, like iGoogle, is a collection of gadgets – content modules that draw on feeds to present a dynamic stream of customized information. We call it the Lenovo Olympic Podium and thanks to Google's devoted engineers and passion for these sorts of things, we gained the capability to not only build and host this Podium, but also to develop the most important content stream in the history of the modern Olympic Games."

Further research is needed to gain a better understanding of the design patterns and implementation techniques required to develop advanced mashup applications. Of particular interest are design patterns for data integration involved in the aggregation of different types of data from a range of heterogenous sources.

References

1. Maximilien, E.M., Ranabahu., A. (2007), *The Programmable Web: Agile, Social, and Grassroot Computing*. in the Proceedings of 1st IEEE International Conference on Semantic Computing special session on The World of Semantics beyond the Semantic Web. Irvine, CA, USA, September, 2007, pp 477-481, ISBN

2. O'Reilly, T. *What Is Web 2.0.* 2007 [cited 20 December 2007]; Available from: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html.

3. ProgrammableWeb. *ProgrammableWeb - Mashups, APIs, and the Web as Platform.* 2008 [cited 16 April 2008]; Available from: http://www.programmableweb.com/.

4. Doernhoefer, M., (2006) *JavaScript.* SIGSOFT Softw. Eng. Notes, Vol 31 (4): p. 16-24, ISSN 0163-5948

5. Govardhan, S., Feuerlicht, G. (2007), *Itinerary Planner: A Mashup Case Study*. in the Proceedings of First International Mashups'07 Workshop, ICSOC 2007. Vienna, Austria. , September 17, 2007, pp ISBN

6. dojotoolkit.org. *Home | The Dojo Toolkit*. 2008 [cited 21 April 2008]; Available from: http://dojotoolkit.org/.

7. Dojo. *Improving performance of Dojo-based web applications*. 2007 [cited; Available from: http://lazutkin.com/blog/2007/feb/1/improving-performance/.

8. Neville-Neil, G., (2005) *Vicious XSS*. Queue, Vol 3 (10): p. 12-15, ISSN 1542-7730

9. Johnson, D., A. White, and A. Charland, *Enterprise Ajax*. 1 ed. 2008: Prentice Hall. ISBN 978-0-13-224206-0

10. 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.

11. 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.

12. Chess, B., Y. Tsipenyuk O'Neil, and J. West, *JavaScript Hijacking*. 2007.

13. Google. *Google Gadgets*. 2008 [cited 24 April 2008]; Available from: http://www.google.com/webmasters/gadgets/.

14. Google. *Publishing Your Gadget - Google Gadgets - Google Code*. 2008 [cited 24 April 2008]; Available from: http://code.google.com/apis/gadgets/docs/publish.html.

15. Google. *Creating a User Interface - Google Gadgets - Google Code*. 2008 [cited 24 April 2008]; Available from: http://code.google.com/apis/gadgets/docs/ui.html#Tabs).

16. Google. *Google Mapplets API - Google Code*. 2008 [cited 24 April 2008]; Available from: http://code.google.com/apis/maps/documentation/mapplets/.

17. Google. *YouTube APIs and Tools - Google Code*. 2008 [cited 24 April 2008]; Available from: http://code.google.com/apis/youtube/overview.html.

18. Churbuck, D. *Churbuck.com*. 2008 [cited 16 April 2008]; Available from: http://www.churbuck.com/wordpress/.