A SEMANTIC APPROACH FOR WEB WIDGET MASHUP

Jinan Fiaidhi, Adam McLellan and Sabah Mohammed
Department of Computer Science, Lakehead University
955 Oliver Road, Thunder Bay, Ontario P7B 5E1, Canada

Keywords: Gadgets, Mashup, Semantic web, Widgets.

Abstract: The current status of the most popular web widget formats was examined with regard to composition of existing widgets into more complex mashups. A prototype was created demonstrating the creation of a mashup in web widget format using other web widgets as components. A search tool was developed which crawls and indexes in semantic format the metadata of web widgets found in a public repository. This tool provides a web widget interface to find other web widgets, and serves as the first pre-requisite tool for further work in this area. A likely path towards further results in this area is discussed.

1 INTRODUCTION

Web Widgets/gadgets are serving an important role in the modern web. For example, content producers seek to syndicate their content in order to reach a broader audience, as well as attracting visitors to their main sites. Portal providers seek to provide a variety of interesting content to make their website a valuable destination for their visitors. These goals can be achieved through the use of web widgets.

A web widget is a portable, self-contained program module which can be inserted into and used on any standard webpage, without special installation requirements. It is essentially a small program designed for syndication on the web. Web widgets are most commonly found on the websites of portal providers and other content aggregators, including popular social networking sites, since they provide a modular way to provide a wide variety of user-customizable content.

Mashups are composite applications that combine elements of existing applications to produce a result which is more than the sum of its parts (Hoyer et al., 2008). Mashups typically provide increased usability and/or functionality compared to using the components separately (Yu et al., 2008). Although mashups are not always formed from web widgets as components, it is those mashups which shall be a focus in this paper.

Although the popularity of web widgets and web mashups has been growing rapidly, both with users and developers, the development methodology has had little time to mature as yet.

There remains a great deal of duplicated effort in web widget development, with tens of thousands of widgets available which independently implement common functionality. New tools which could provide a more modular approach to web widget development would be invaluable, both in reducing costs of new development, and in speeding new ideas' transition to implementation and widespread use.

This paper reviews currently popular web widget standards, tests the applicability of the proposed development advances, and provides the first tools needed on the path to simpler, modular development of web widget mashups.

2 WEB WIDGET STANDARDS

2.1 Criteria of Comparison

In order to develop advancements in web widget creation and mashup techniques, it is first necessary to examine and compare the various available standards for web widgets. Several key criteria were considered in this comparison.

Capability for mashup is the foremost consideration, since a standard unable to support even rudimentary re-use of its widgets would be unsuitable for the purposes of this paper. A standard may provide varying levels of support for this possibility depending upon its API and security model. Since the goal is a modular standard of development, ideally the standard should provide the capability for multiple widgets to be instantiated and to communicate with each other.

Finally, the level of openness of the standard and its associated development tools must be considered. Creation of mashups using web widgets as components is most useful with a large body of fresh and innovative component widgets to build upon. Thus the possibilities for mashup are best served when the standard is easily accessible to the widest possible range of developers who can provide these components.

2.2 Rich Internet Application Frameworks

Rich internet application frameworks are widely known on the web and are certainly put to many uses, including web widgets. The most popular rich internet application (RIA) frameworks are Adobe® Flash®, Microsoft® Silverlight™, and Oracle® JavaFX™. Each of these is well capable of producing a wide variety of powerful internet applications, including web widgets.

However, none are intended specifically as a web widget standard; these frameworks are designed as general-purpose programming environments. Cross-widget communication could be implemented via some form of shared client-server networking, but it is not provided out of the box. There is no centralized repository of widget-like applications with detailed and standardized metadata.

Although any of these frameworks could be used as a basis for further work, they would require re-implementation of features already provided by existing widget standards intended for that purpose.

2.3 Web Widget Standards

There are many web widget standards; only those considered the most widely used are examined.

2.3.1 Widgetbox™

Widgetbox provides a very large repository of web widgets, well over 200,000. Most widgets provide a visually-attractive array of user customization options. Also provides tools to allow end-users to easily create certain types of widgets from templates with no programming knowledge.

Table 1: Summary of Popular Web Widget Standards.

Standard	Cross-Widget Communicatio n Support	Licensing
Widgetbox	No	Commercial
SpringWidge	No	Free1
Yahoo Konfabulator Widgets	Yes	Open
Microsoft Windows Live Web Gadgets	Unknown2	Unknown2
Google Gadgets (OpenSocial)	Yes, Multiple Types	Open

2.3.2 SpringWidgets™

SpringWidgets provides a large repository of approximately 50,000 widgets. Some metadata is available for each widget. Internally, the widget engine provides a number of Flash methods; widgets using this standard must be implemented using Adobe Flash.

2.3.3 Yahoo!® Widgets

Widgets are defined using an XML-based format and can provide various items of metadata. The widget engine provides an extensive API exposing numerous features to widget developers, including some items with potential to be used for cross-widget communication. Approximately 6,000 widgets are listed in the Yahoo! Widgets repository.

2.3.4 Microsoft® Windows Live™ Web Gadgets

Currently, it is difficult to find developer documentation on Microsoft's Web Gadgets. However, like Yahoo! Widgets, this standard appears to lack popular support, with approximately 5,000 gadgets listed in the Live Web Gadgets repository.

2.3.5 Google™ Gadgets (OpenSocial)

Google Gadgets were originally developed as a standard for Google's portal site, iGoogle. However, the standard has since been opened and used as the basis for the OpenSocial gadget standard. OpenSocial is now a widely-accepted format accepted on other well-known sites including Yahoo!, MySpace, orkut, hi5, and LinkedIn. The repository of Google Gadgets available for syndication lists over 200,000 gadgets.

3 SELECTION OF A STANDARD

After considering the criteria of comparison, the Google Gadget standard was selected as the most appropriate for continuing this research. The Google Gadget standard is an open standard, widely-accepted across popular websites, and able to be embedded in those without native support. It is easily accessible to developers at no cost, and is well-supported with a base body of over 200,000 gadgets to work from as potential components. Perhaps most critically, the standard provides multiple methods of cross-gadget communication. These communication channels can be utilized to make possible the use of Google Gadgets as components in a mashup gadget.

The Google Gadget standard requires use of a gadget server which can render the XML-based gadget specification into the appropriate HTML and JavaScript content for a web browser's use. Google's iGoogle portal site provides one publicly available option. The Apache Shindig project also provides a free, open-source implementation of a gadget server according to the OpenSocial specification. For the purposes of this research, most development and testing was done using WSO2 Gadget Server, a much expanded open-source project which builds upon Apache Shindig as a base, but also provides a portal interface much like the iGoogle site. This will be addressed further when discussing the gadget search tool.

4 GADGET IMPLEMENTATION

4.1 Implementation Language - Criteria of Comparison

The implementation option must be capable of supporting the Google Gadget JavaScript API. This API provides important features which were a major criterion of selecting the Google Gadget standard. In fact, this criterion is so critical that an implementation option was not considered unless it was capable of interfacing with JavaScript to utilize the Google Gadget API.

User accessibility is another primary criterion. Finally, developer accessibility was considered. This includes factors such as whether the developments tools are free or commercial products, as well as whether both client and server code can be developed in one IDE.

4.2 Implementation Options

There are many options including

1- Adobe Flash which is the most widely installed RIA framework on the web. It is available for a wide variety of browsers and operating systems. Notably unable to view Flash content are Apple's popular iOS devices: the iPhone, iPad, and iPod Touch.

2- Microsoft® Silverlight ™ that has the lowest install base of the major three RIA frameworks (Flash, Java, and Silverlight). Additionally, Silverlight is not supported under Linux or iOS based platforms.

3- Google® Web Toolkit (GWT) is yet again different from all other options. This free, open-source project by Google consists of a compiler and supporting libraries which transform developer-written Java code into JavaScript. As developers write code in Java, the software is strongly-typed, permits use of most Java as well as JavaScript libraries, and can be seamlessly integrated with server-side code using the same shared object definitions. When the client-side code is compiled, multiple permutations are generated for different browser and platform combinations, and a small section of bootstrap JavaScript selects and launches the most compatible version at runtime according to the user's environment.

4.3 Selection of Implementation Language

After careful consideration, GWT was selected due to providing many features of full Java, such as strong typing and support for many Java libraries, yet not requiring framework installation by the end-user.

5 PROTOTYPE MASHUP

5.1 Google Gadget Structure and Environment

Google Gadgets are defined using an XML specification file including three primary sections: the Module Preferences section, container standard gadget metadata, such as details of any special features that will be requested of the server, the User Preferences Section, which defines user-specific gadget data the server will need to store, and the Content section, which provides the actual implementation of the gadget.

Google Gadgets are parsed and then rendered into their final form by a gadget server. Originally the only gadget server was the iGoogle portal server, but as the Google Gadgets standard is open, the Apache Shindig project was soon started to provide an implementation which can be deployed anywhere. For the purposes of this research, the focus will be upon the base Google Gadget standard, since the additional features provided by the OpenSocial specification are not needed for the items implemented. The Google Gadget API provides two primary methods to communicate with other content on the same web page: The Publisher/Subscriber (Pubsub) feature, and the Gadgets RPC feature.

Gadget containers typically render gadgets inside IFRAME elements for reasons of security. By using an IFRAME, the container can take advantage of browsers' security features to avoid problems with cross-site scripting attacks, which could compromise data from other gadgets or non-gadget content. All cross-gadget communication desired by this paper was found to be perfectly achievable using the two communication methods in combination. See Figure 1.

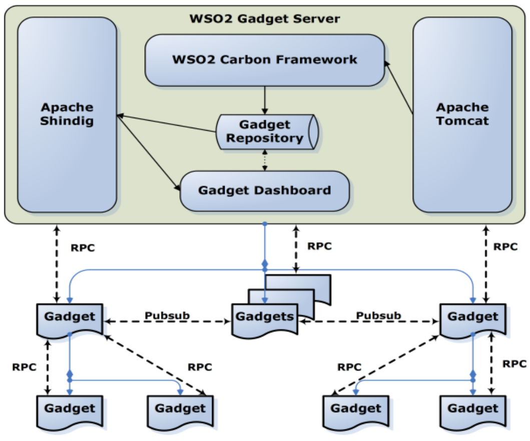


Figure 1: A graphical display of which cross-gadget communication methods are functional in which circumstances. This is shown as tested using WSO2 Gadget Server 1.10.

The Pubsub communication feature can be used to send text messages between sibling gadgets, under the condition that these sibling gadgets are found at the root level, contained only inside the gadget container itself. The Gadget RPC communication feature can be used to asynchronously invoke arbitrary JavaScript methods found either one level up, in the running gadget's container, or one level down, in a gadget rendered inside the currently running gadget. To invoke a method of a child gadget, the child's IFRAME's ID attribute must be passed as the target parameter into the Google Gadgets API call. To invoke a method of the parent, the page containing the currently gadget, the target parameter is omitted. Through careful combination of both methods of gadget communication, it is in fact possible to pass messages between any two gadgets.

The Gadget RPC feature's capability of allowing a parent gadget to communicate with those gadgets it contains fulfills most use cases being considered for this project. This will be the focus of our prototype mashup gadget using gadgets and components.

5.2 Component Gadgets

The purpose behind such focus on cross-gadget communication is to enable the possibility of using gadgets as components in designing new, more powerful gadgets. Currently gadget developers may utilize some shared libraries, but this imposes restrictions such as using the same implementation language as the library or developing a wrapper. Additionally, unlike gadgets there is no central, searchable repository for shared libraries, and few libraries provide user interfaces.

Gadgets as re-usable components show a great deal of potential. Due to the Google Gadgets standardized specification, component gadgets can be written using entirely different implementation languages, with the Google Gadgets API providing a wrapper. Gadgets can be listed in a central repository to make them easier to find, and gadgets can provide complete user-facing interfaces to their functionality.

There are two primary limitations to keep in mind when developing mashups using gadget components. First, to use a gadget as a component, it must expose functionality to Gadget RPC, which requires action on the part of the original developer. The second limitation is that it quickly becomes non-trivial to keep track of the various mini-APIs that component gadgets can provide via Gadget RPC. This problem can be alleviated by standardizing component gadget interfaces, a matter which is discussed in more detail in the Future Work section of this paper. The prototype mashup developed here serves as a proof-of-concept, and is built using gadgets intended as suitable components.

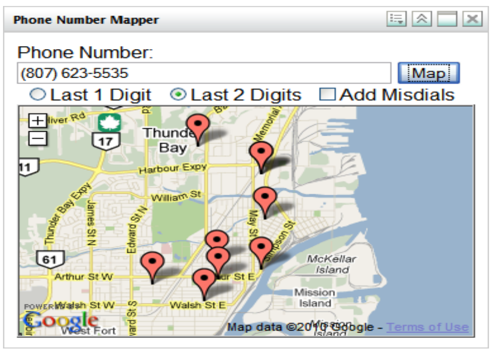
5.3 Mashup Design

The mashup prototype developed is based upon three components. The first component gadget is a tool which accepts a telephone number as input, and produces neighbouring telephone numbers as output. Optionally, users can choose to include possible common misdials of the number which are not numerically adjacent.

The second component gadget is a tool which takes a telephone number as input, and performs a reverse lookup to produce the publically listed associated address, if any.

The third component gadget is a tool which marks any set of addresses on a map. The resulting mashup gadget, which coordinates the layout of these components and passes messages between them, is a tool which takes a phone number as input, and produces a map of the locations of neighbouring phone numbers.

This demonstrates the capability of this style of gadget development to take advantage of existing component gadgets to produce a more useful composite, while minimizing development time. A screenshot of this gadget can be seen in Figure 2, below.



Phone Number: (807) 623-5535

○ Last 1 Digit ◉ Last 2 Digits ☐ Add Misdials

Figure 2: A screenshot of the developed prototype mashup in action.

A pair of diagrams showing the internal structure of the mashup follows in Figure 3.

6 GADGET SEARCH

6.1 Purpose of Gadget Search

The next stage of this paper was to develop tools necessary to search for and find gadgets. As Google already provides basic search functionality as part of their public gadget directory, it may not at first be obvious why further search capability is

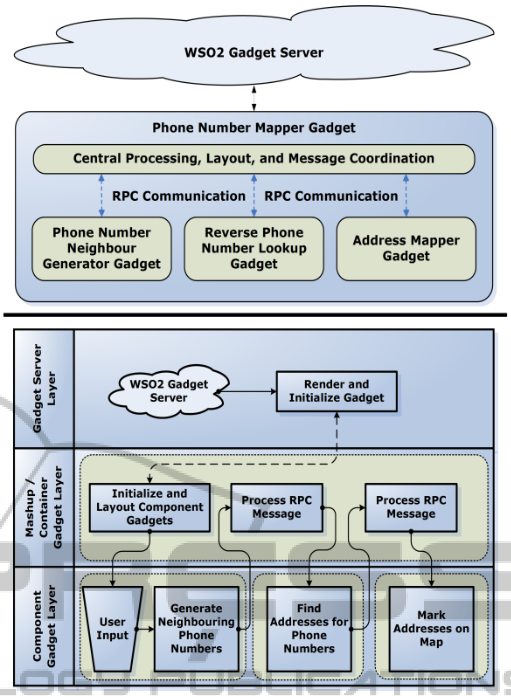

WSO2 Gadget Server
Phone Number Mapper Gadget
Central Processing, Layout, and Message Coordination
RPC Communication
RPC Communication
Phone Number Neighbour Generator Gadget
Reverse Phone Number Lookup Gadget
Address Mapper Gadget
Gadget Server Layer
WSO2 Gadget Server
Render and Initialize Gadget
Mashup / Container Gadget Layer
Initialize and Layout Component Gadgets
Process RPC Message
Process RPC Message
Component Gadget Layer
User Input
Generate Neighbouring Phone Numbers
Find Addresses for Phone Numbers
Mark Addresses on Map

Figure 3: Above: A generalized diagram of the architecture of the Phone Number Mapper Gadget. Below: An execution flow diagram shows how the program control moves between the different software layers and the components within them.

necessary. There are two primary reasons for developing search tools: intranet gadget search, and semantic feature search.

The first reason, intranet gadget search, is to provide search capabilities to organizations which cannot publically publish some or all of their internally developed gadgets. Although Google provides the ability to search their public gadget directory, it can only return results which have been published for all on the internet to see. For numerous reasons, organizations may want to design and develop gadgets for internal use which cannot be made available to the general public, and perhaps which cannot even be made accessible outside that organization's own secured intranet.

The second reason for new search tools is to provide more precise search capabilities than what Google can provide as part of their public directory. As an example, if one searches for "map" on the Google Gadget public directory, the top results include a simple graphics editor, games, weather listings, and other miscellaneous gadgets which mention the term "map" only in passing. Many of these results are not actually gadgets providing the user with a map. This is the expected behaviour of a general-purpose search tool; however for the purpose of locating appropriate component gadgets to use in a mashup, far more precision is required. By indexing all available metadata in a semantic format, it will be possible to locate suitable candidates using very specific criteria.

6.2 Indexing Gadget Metadata

The first step in providing search results must be to identify and index the data which will provide such results. Google provides a directory of gadgets available for public syndication purposes. The primary item of interest for each item in this directory is the web address where the gadget's specification file can be located; this is where the gadget's full metadata and details can be found.

Also it provides access to its gadget directory in two formats: the standard, graphical interface for end users, and an RSS feed more suitable for programmatic access. It is this second which is used for retrieving the list of gadget specification file URLs. Since both this RSS feed and the listed gadget specification files meet specific XML schema requirements, Java's JAXB (Java Architecture for XML Binding) was used to access the target data elements within the files.

As threads run to retrieve these lists of gadget specification file URLs, 100 results at a time, other threads run to request those XML files from the various hosts across the internet where their authors uploaded them. Using the official XSD (XML Schema Document) provided as the standard by Google, JAXB is used to validate those specification files. Some of the files are now missing from their web host, contain syntax errors, or for various other reasons do not meet the Google Gadget standard; those gadgets are discarded before proceeding further.

Those gadgets meeting the standard are then processed. The Jena library is used to store all gadgets' metadata in a semantic format, with TDB providing the high-speed storage backend. For each gadget, each item of metadata is treated as a 3-tuple (triple), with the XML specification file's URL as the subject, the name of the metadata item as the predicate, and the gadget author's provided data as the object.

Once all gadgets have been processed, this database of metadata of publically listed gadgets is exported into RDF format. While not strictly necessary for the indexing phase, this allows the data to potentially be used in other ways in the future.

With this summary of all gadget metadata to work from, an index suitable for search engine use is built next. Lucene is a powerful search engine library, and SIREn provides a semantic extension which preserves the semantic nature of the data rather than requiring a specific, structured set of fields. Lucene organizes data in a different manner than a triple-based system such as that provided by Jena; any item which the user wishes to retrieve as a result must be a "document". Thus the data is indexed by considering each gadget XML specification file as a document, with all of the associated metadata statements being filed as contents of that document.

At this point, with all of the metadata having been indexed by Lucene through the semantic filter of SIREn, the search pre-processing is complete. It must be noted that this crawling and indexing process is distinct from the gadget and associated servlet that the user accesses to make use of this data. Additionally, this entire process must be re-run periodically to take into account new entries into the public gadget directory.

7 CONCLUSIONS

The creation of composite web widgets – mashups – using other web widgets and components shows great promise. Though most web widgets have not been developed with the intent of re-use as a component, there are an enormous number of web widgets available which serve a multitude of purposes. Component use of web widgets can result in highly useful end products while taking advantage of the benefits of a modular design.

The prerequisite tools developed as part of this paper provide a crucial stepping stone towards further work in this area.

ACKNOWLEDGEMENTS

The first author would like to thank NSERC for funding this project as well as funding Mr. Adam McLellan to work on this project under the first author supervision.

REFERENCES

Hoyer, V., Stanoevska-Slabeva, K., Janner, T., and Schroth, C. 2008. Enterprise Mashups: Design Principles towards the Long Tail of User Needs. In

IEEE International Conference on Service Computing (SCC'08). Volume 2, 601-602.

Yu, J., Benatallah, B., Casati, F., Daniel, F. 2008. Understanding Mashup Development. IEEE Internet Computing, 12(5), 44-52.

APPENDIX A

THIRD-PARTY LIBRARIES AND TOOLS

Here follows the list of the primary third-party libraries and tools used for this paper.

Software	Website
Java JDK	http://www.oracle.com/technetwork/java/javase/downloads/index.html
Eclipse	http://www.eclipse.org/
Apache Ant	http://ant.apache.org/
Google Web Toolkit	http://code.google.com/webtoolkit/
GWT-Gadgets	http://code.google.com/p/gwt-google-apis/wiki/GadgetsGettingStarted
WSO2 Gadget Server	http://wso2.com/products/gadget-server/
Apache Shindig	http://shindig.apache.org/
Apache Tomcat	http://tomcat.apache.org/
SIREn	http://siren.sindice.com/
Apache Lucene	http://lucene.apache.org/
Jena	http://jena.sourceforge.net/
TDB	http://openjena.org/TDB/
GWT-Maps	http://code.google.com/p/gwt-google-apis/wiki/MapsGettingStarted
Firebug	http://getfirebug.com/