WEB APPLICATION INTEGRATION USING MASHUPS

by

RONDA HILTON

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2009

ACKNOWLEDGEMENTS

I wish to acknowledge the support of the Frost Bank which provided tuition assistance when I worked for them in Arlington, Texas in 2004.

Thanks most of all to Dr. Leonidas Fegaras; I could never have completed this effort without his ideas and steady patience. Thanks also for the inspiration from lectures given by UTA professor Dr. Matthew Wright. Thanks to Mr. David Levine of the UTA CSE faculty for generous help and advice. Also I wish to acknowledge the kind encouragement of TCU professor Dr. Michael Meckna, UTA professor Dr. Roger Walker, and CSE Graduate Advisor Mike O'Dell who went above and beyond.

In memoriam, I would like to express my deep gratitude to exemplary UTA GTA John Stephen "Steve" Schuchman for advising me to study computer science engineering.

November 23, 2009

ABSTRACT

WEB APPLICATION INTEGRATION USING MASHUPS

Ronda Hilton, M.S.

The University of Texas at Arlington, 2009

Supervising Professor: Leonidas Fegaras

The HTML DOM is the W3C standard data model for HTML documents. A web page may be viewed as a tree structure with data nodes at each level, according to the HTML DOM, to enable web applications to access it dynamically. If a web page mashes up more than one web application, the data from one web application may flow as input into another. The user may manually transfer such data piecemeal by using the mouse to cut and paste the displayed text. Instead of this tedious and error-prone repetitious method of data transfer, the mashup may contain a software agent which can dynamically integrate web applications so that data flows from one to another automatically. The user indicates the source web application, e.g. by clicking one text item in a list. The agent traverses the source web application's internal HTML DOM structure, finding all data nodes at the same level as the user-indicated data node, in an attempt to discern the selection intentions of the user. A GUI displays the agent's selected nodes in an array structure, which is part of the mashup. The user may indicate from the array

structure which array element to transfer into the target web application. As proof of concept, we have built a mashup of two *iframe*s that uses an agent which selects text and transfers it from one *iframe* into the form input fields of the other *iframe*.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

Human users integrate data from the web by using search engines to find web sites, and by extracting data for their own purposes from the displayed information using human interface devices such as a computer monitor or mouse to cut and paste, save to disk, etc. A parallel activity would be humans hunting and gathering food from the animals and plants of a forest. Just as humans use agriculture tools in an infrastructure of farms and fisheries to yield a more precise and reliable food supply, human users of the web may use software tools in the infrastructure of the programmable web, sometimes called the Web 2.0, to make more thorough searches and integrate the pertinent data. These software tools can gather information from web services, RSS feeds, and information retrieval which require application programming to access. Development of more efficient data integration tools and techniques may be necessary to access the ever-expanding volume of information available on web sites, web services, blogs, social networking sites, and web databases.

A mashup is defined by [Maximilien et al., 2007, p. 14] as “a Web application that aggregates multiple services to achieve a new purpose.” However, juxtaposing web applications for visual comparison provides little more functionality than browsing to each web page in sequence. Furthermore, mashups are instantiated only after communication to each embedded web page's server. The more complex the mashup, the longer the end user must wait for the mashup to load in his or her browser.

The web site “Programmable Web” (at http://www.programmableweb.com/) maintains a description with links of many mashups. It is disappointing that so many of these mashups are not functional, possibly due to the revokation of their development platforms, as explained in

section 3.2. Mashups tend to be limited by the design skill of the programmer, a limitation which is also common to static pages on the web. Programming mashup functionality requires a skill set of several web languages and scripting skills, unlike static pages which can be created using only HTML. The majority of mashups fall short of the mark in design, functionality, or robustness. Yet programmers continue to attempt to relate web data using mashups, which promote ideals of scalability and adaptability by re-using existing web sites in a new application.

Many web sites make data available in XML format through an interface accessed by web applications, which are said to consume the data. This is providing a web service, also called an application programming interface (API). Common API's provide data such as schedules, geographical information, photographs or images, and items for sale. An example API is the public API offered by Amazon.com called Product Advertising API, often referred to under its old name AWS [Mueller, 343], a classic e-commerce web service. Web services furnish critical dynamic information to most mashups, but they do require software programming to access, or some software tool which can generate code. Web services are the bar which is raised before mashup designers, who may necessarily be mashup developers.

Our proof of concept must necessarily be a mashup itself, and it must have the capability of querying a web service. Within our mashup we wish to establish a workflow to connect data between web applications which the end user may choose dynamically.

CHAPTER 2

DATA INTEGRATION

2.1 Dynamic choice of web applications

Advanced mashups have programmed interaction between chosen web applications, e.g. “About Airport Parking” (http://www.aboutairportparking.com/). “About Airport Parking” has been online since Nov. 14, 2006 and possibly earlier. When for example a list of commercial parking lots near Dallas-Fort Worth airport is displayed, mousing over any list item causes the Google map on the same page to rotate to show the geographical location of the indicated parking lot, and pops up a label with information about prices and address. Such generally useful mashups are popular. However, the choice of web pages or web applications is static, being pre-determined by the programmer. [Maximilien et al., 2007, p. 14] poses the problem of dynamic choice of the web services: “These services mashups are typically point solutions to specific (specialized) problems; however, what is missing is a programming model that facilitates and accelerates creation and deployment of mashups of diverse services.”

To allow the end user to choose the web applications, previous research has followed the direction of inventing software tools to enable people without programming skills to create individual point solution mashups of web pages. However, using the software tools still requires an understanding of the concepts of programming, and there remains the hurdle of programming always required for dynamic web services. Instead of following previous research in this direction, our efforts are aimed toward web programmers.

2.2 Utilizing the DOM

The HTML DOM is the W3C standard data model for HTML documents. A web page may be viewed as a tree structure with data nodes at each level, according to the HTML DOM

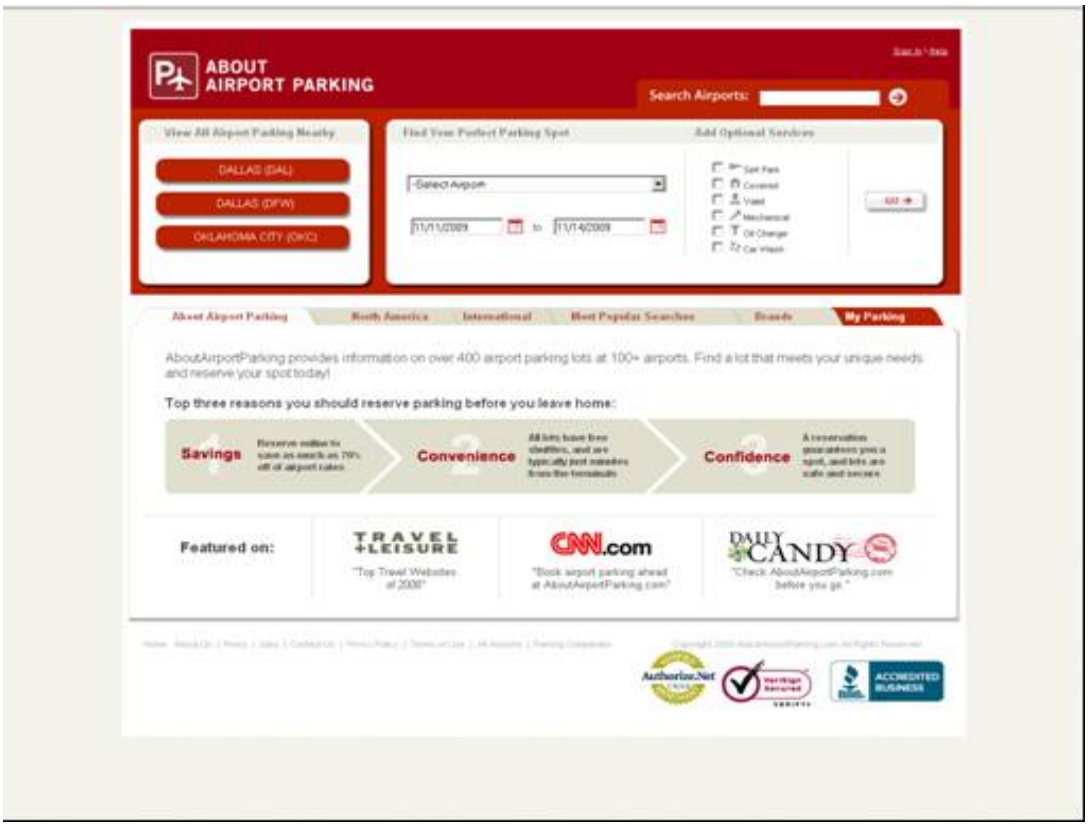

Figure 1. Mashup example from http://www.aboutairportparking.com/ accessed November 11, 2009.

(see Figure 1) to enable web applications to access it dynamically. Web applications may be traditional static HTML documents, or any web site with dynamic content such as wikis, blogs, widgets, API's, or RSS feeds. If a web page mashes up more than one web application, the data from one web application may serve as iterative input into another. Rather than the user

manually transferring such data piecemeal, a mashup contains an interface agent to "facilitate the interaction between the user and various computational resources (including other interface agents)" [Shoham and Leyton-Brown, p. xvii]. There are many mashups which integrate two or more specific data sources as programmed, but a more flexible objective is to dynamically integrate web applications within a mashup so that data flows from one to another. The user indicates the source web application, e.g. by clicking one text item in a list. The GUI traverses the source web application's HTML DOM, finding all data nodes at the same level and access path as the user-indicated data node, in an attempt to discern the selection intentions of the user. As proof of concept, we have built a mashup of two *iframe*s that transfers text from one into the form input fields of the other.

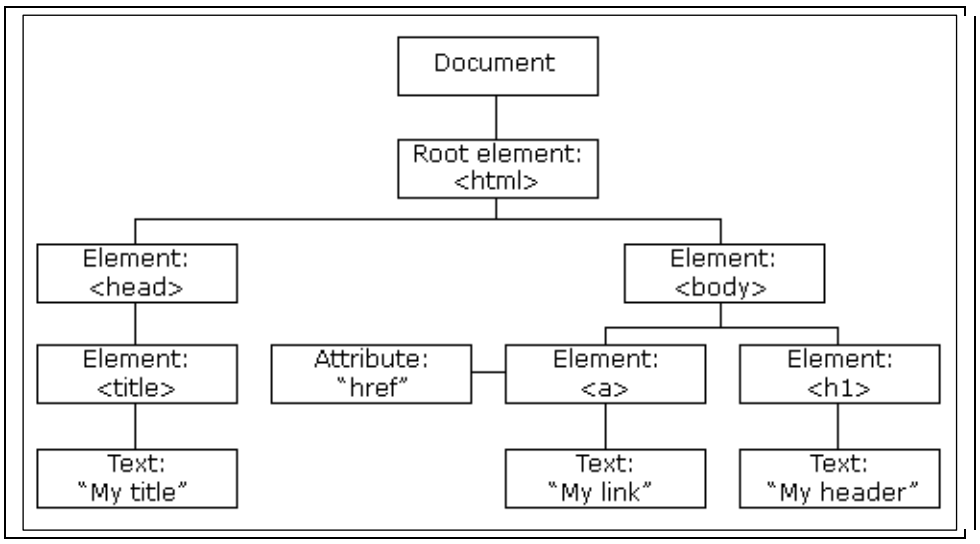

Figure 2. HTML DOM Tree example from http://www.w3schools.com/HTMLDOM/ accessed October 21, 2009.

2.3 Virtual Web Service

The proof of concept application fits the definition of a software agent, and the resulting mashup is a Virtual Web Service (VWS), as described by [Rykowski et al.]. Since the web application concept is XML-based, this software agent is in fact a private agent (PA) which carries text from one section of a mashup to another. [Rykowski et al.] make a point that agent code is less efficient because it is interpreted rather than compiled. However, the JavaScript web application running in the browser may seem sufficiently efficient to the end user because he or she does not experience the latency in communicating with the server.

The agent should possess mobility by being runnable generally in different mashups or VWS's. This is accomplished by interposing a separate file for the variables, specific to the mashup with its *id* attribute for each element, mapped to the same variable names. In order to be manipulated programmatically by our JavaScript functions, the variable names must be associated with *id* attributes of structures in the mashup HTML. For example, the library variable *rightiframe* is mapped to the right-side *iframe* using the JavaScript document method *getElementById( )* as in Figure 3. The variable names do not change, but the quoted *id* attributes may be anything. Such a custom "variable file" can connect any mashup HTML source to our JavaScript library.

```
var rightiframe = document.getElementById('r_browser');
```

Figure 3. Example of JavaScript "variable file" source.

CHAPTER 3

PROOF OF CONCEPT

3.1 Server-side vs. client-side

In order to bypass browser cross-domain security in the proof of concept application, we coded a proxy. Online documentation at Sun Developer Network refers to a mashup with a proxy as "server-side". The mere presence of a proxy source file on the server, however, does not redefine the proof of concept application which is designed deliberately to be client-side. The distinction made by Ed Ort, Sean Brydon, and Mark Basler, August 2007 at the URL http://java.sun.com/developer/technicalArticles/J2EE/mashup_1/ includes this explanation (referenced on October 12, 2009):

> The service or content integration takes place in the client, typically a web browser. In general, a proxy is a component that acts as an intermediary between two parties. A significant reason for using the proxy style is to overcome the constraints of the browser security sandbox and make XMLHttpRequests across domains.

There is an important configuration step which must follow the establishment of a proxy in Mozilla Firefox browser. In the browser address bar, enter the URL *about:config* and filter the results on the initial string *signed*. Toggle the *signed.applets.codebase_principle_support* entry *true.* This enables *UniversalBrowserRead* privileges. This is explained in more detail on the site http://bytes.com/topic/javascript/answers/845717-cross-domain-ajax.

It is important to use client-side software technology as much as possible in a mashup in order to reduce the wait time the end user experiences for the server communication required for each and every web application mashed up. More powerful technology such as Java Server Pages (JSP) can more effectively traverse domains. We did consider JSP for the proof of concept application, and it offers many advantages, but the tradeoff is more wait time for the

end user while the server manages all communications and then finally delivers the finished page to the user's browser.

3.2 Mashup tools

Mashup design and construction remain in the purview of web programmers, although software tools have been introduced with the purpose of enabling non-programmers to design mashups and automatically generate the code behind. [Di Lorenzo, 2009] surveys these tools using examples considered to be popular. Commercial example tools include Damia, Apatar and MashMaker, which leverage data on an enterprise intranet.

Among the open source examples in [Di Lorenzo, 2009], Google Mashup Editor (GME) was discontinued by Google on August 12, 2009 along with its entire distribution environment: all the mashups which had been built using GME were no longer available after this date. Another open source example, Popfly, was discontinued by Microsoft on August 24, 2009, rendering all mashups unavailable which had utilized this tool.

Extant tools include Greasemonkey, Yahoo Pipes, and the database API FreeBase. Nevertheless it is necessary in any mashup to utilize some method or methods of screen scraping. There is no other way to access static web pages. The proof of concept application uses an XMLHttpRequest( ) object to get the *iframe*'s source document. At first we attempted to get this object in XML mime type, in order to send the node structures as parameters through our library functions. However, even the cleanest HTML often returns an error "not well-formed" when accessed as XML. For a robust application, we overrode the mime time and requested the source document as HTML text. This override command is possible in the Mozilla Firefox browser but not in all others.

3.3 JavaScript

JavaScript often emerges as the language of choice for developing dynamic content in web pages, and even more often in mashups. Entire frameworks in JavaScript have gained notable popularity among developers. Figure 4, an AJAX Programming lecture slide from the

"Web Programming" course taught in 2007 by Marty Stepp at the University of Washington, lists some common JavaScript frameworks.

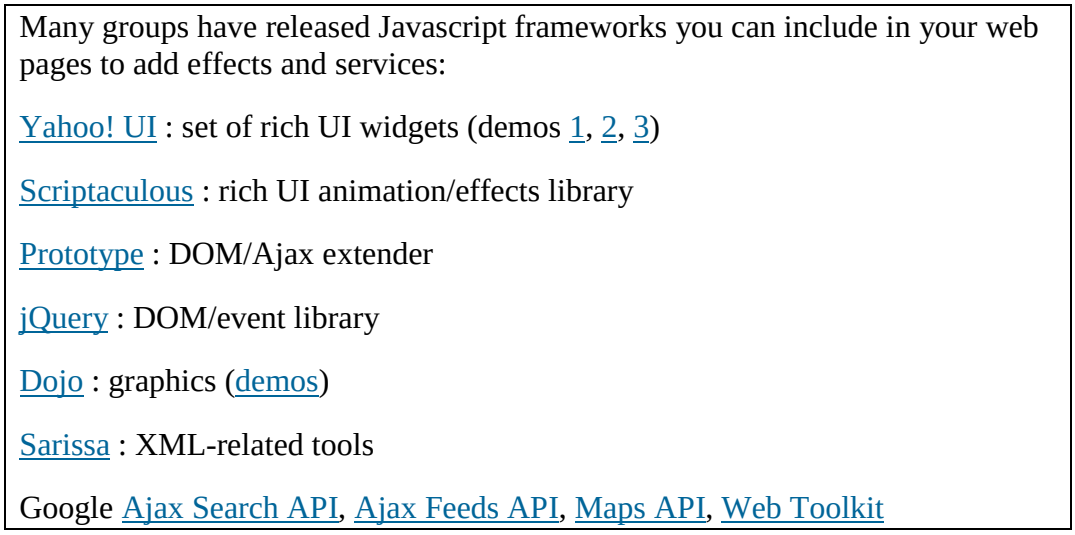

Figure 4. JavaScript frameworks lecture slide.

The entire lecture, the entire course, and many more can be accessed at the Google Code University, http://code.google.com/edu/curriculumsearch/ .

The goal of our thesis is to develop a proof of concept web application in JavaScript. Its architecture category is that of a web-based mashup, which dynamically arranges the mashup in the client's browser, as opposed to server-based mashups which put everything together and then deliver the final product to the browser. [Bolin 2005, 22-23] Client-side script runs in the user's web browser, after the page has been loaded. JavaScript possesses the advantage of avoiding the latency in communicating with the server. However, there are disadvantages. JavaScript is subject to problems inherent in open source products. It is less robust than commercially developed platforms, and there is only a poor set of debugging options. It is

challenging to organize source code in any form other than a single long file. Recursion causes intermittent timing problems. Alternatively the proof of concept application could have been developed in JSP or .NET with tradeoffs for latency, capability, and cross-browser security restrictions.

Mashups display web content within one page utilizing many kinds of structures. For example, HTML *div* tags provide an open-ended section division for styles or web applications. There are three types of text fields defined by tags in HTML (Wenz 2007, 125):

1. form input field on a single line
2. textarea containing wrapped lines
3. password input fields, which are not relevant to this thesis and need not be mentioned again.

The mashup GUI must transfer text into form input or *textarea* fields without requiring the user to repetitively select, cut and paste between frames in the mashup. The proof of concept application uses *iframe*s, which embed one web page inside another. The embedded page's programmability is restricted to the same domain in every browser, but *iframes* present powerful navigation possibilities for the user. Browsers are designed to limit certain information, such as the history of the pages that have been loaded in an *iframe*. [Holdener, p. 323] points out that *iframes* are deprecated and their functionality will be assumed and expanded with something called *XFrames*.

An *iframe* contains an embedded web browser within an HTML document. In a mashup's *iframe*, a user may navigate by clicking links displayed in the *iframe* "src" attribute page. The new page will be loaded into the *iframe* but the "src" attribute will not change. The new page's URL will be visible in the status bar of the browser window unless specifically coded to be different. Also the browser's "Back" button will not change the mashup page, but instead loads the previous document in the *iframe*.

Onclick events coded into the mashup will not fire. Instead, the *iframe*'s loaded document onclick events will fire, if it has any. The mashup developer might like to capture the

information of what the user has clicked within an *iframe*, but this capability may require either browser design changes or software more powerful and flexible than applications in JavaScript.

Within the mashup we wish to have a GUI application with which the disparate data sources can interact, and also the user, so that the paradigm for interaction is many-to-one. As first presented to the user, we have a JavaScript application to extract text from DOM nodes in the left *iframe* and display it in an array of buttons. Clicking any button causes the application to do two things: (1) compare DOM search paths and eliminate all buttons with text elements at a different level than the one clicked, and (2) transfers the clicked text to the right *iframe* form entry.

Mozilla Firefox browser supports XSLT function *transformToFragment*. For the "not well-formed" error, we wish to continue requesting the document as HTML, and then create nodes using an XSLT stylesheet. The Mozilla function *transformToFragment* returns a new HTML DOM DocumentFragment node. This fragment node can belong to the overall document instead of the *iframe* “src”, thus getting past all cross-domain issues. We send this fragment node as a parameter through the same JavaScript subroutines developed for same-domain processing.

Cross-browser compatibility and accessibility are the two most critical web development issues, according to [Harris, p. 34]. Our proof of concept application is limited to its development only in Mozilla Firefox; making it compatible with other browsers would be another thesis in itself. Some standards of accessibility are maintained by separation of code content, style, and behavior in separate source files.

3.4 Separation of content, style, and behavior

In mashups, a barrier to flexible, dynamic information integration is commonly the lack of separation between components of the application code. [Tatemura et al., 2007, p. 1128] states the issue: “Mashups are usually written in procedural programming languages such as

JavaScript, and the code rarely separates the user interface layer (dynamic HTML) and the data integration layer well."

The proof of concept application JavaScript application consists of several source files, organized by functionality. One "variable file" declares variables necessary to access the hard-coded HTML source id attributes, which are DOM nodes. Variables are necessary for the JavaScript to associate events to each *iframe*, form, button, and other structures in the HTML mashup page source. Each external Javascript source file is loaded by creating a *script* element and attaching it as the last node of the mashup *body* or *frame* element.

Placement of a JavaScript source element within the HTML source affects the event properties. If the source is defined in the *head* element instead of the *body* or *frame* element, no event will fire except the *onload* event, if there is one; and then only after the delay of the entire page load, which may be considerably slow. In order to immediately have event sensitivity, the JavaScript must be coded at the end of the page, within the closing tag of the *body* or *frameset* element. For example, the proof of concept application divides the page into two *div* sections with id attributes *pal_page* and *jsdiv*. The latter contains code which appends all the external JavaScript source files. Having a *div* tag just for JavaScript allows the application to append specifically to the end of the *body* or *frame* element. Other than the "variable file", the JavaScript source listings are generic and might be appended to any mashup.

Separation of content from style and from behavior is a worthy standard in web applications, though not easy to achieve. For mashups an Ajax-influenced standard is proposed in [Edwards and Adams, p. 320]: the mashup content is defined by the HTML source listings, the style by the CSS, and the behavior by the JavaScript. The JavaScript may serve more than one purpose in a mashup, however. [Maximilien et al., 2007, p.16] compares "three primary components" of mashups to the Model-View-Controller (MVC) aspects of a typical web application. In mashups the parallel components, which might all be coded in JavaScript, are listed by [Maximilien et al., 2007, p.15] as:

1. Data mediation,
2. Process mediation, and
3. User interface customization.

Our effort presents further work in the first component, coded in JavaScript, though other languages and development platforms might serve this purpose with advantages and drawbacks, including notably Ruby on Rails used successfully by [Maximilien et al., 2007].

CHAPTER 4

PROBLEMS ENCOUNTERED

4.1 Web services discovery

Locating available, functioning public web services presents a challenge. Starting in 2000 there were three public registries of web services, called UDDI's, maintained by IBM, Microsoft, and SAP which attempted to be comprehensive. Almost any entity wishing to expose a web service could register with these UDDI's. After five years, these three comprehensive public UDDI servers were all shut down in January, 2006 by their respective corporate owners. Developers have since tried many methods to find available web services. One method, by [Wu and Chang, 2007], searches for web sites having WSDL documents. If found, they parse the WSDL file text, attempting to gain enough information to categorize and access the web service which might reside on that site. Discovery of web services has also been approached in [Liu and Chao, 2006] by combining information from such public and private UDDI's as may still be found, and storing the information in a searchable graph.

Most web services provide only limited functionality, such as the ISBN Database (at isbndb.com) where you can look up books by ISBN but not by author name. The code behind web services may not be robust, or the information unreliable. For example at the Aonaware Dictionary service (at aonaware.com) when the query word "bore" is submitted to look up, it returns the definition of "bear". The proof of concept application utilizes rather minimal web pages to expose both these web services as typical examples in a mashup.

4.2 Site security

According to Rykowski et al. JavaScript web applications have insufficient inherent security features. The supporting example given in [Rykowski] is that an infinite loop will actually run unrestricted by any compiler or run-time checking mechanism in the JavaScript

interpreter. Furthermore interpreted code is relatively easy to change by an attacker. In fact the proof of concept application web site was hacked in April 2009. All scripts and HTML files on the *fwago.org* site had JavaScript code appended which instantiated a Trojan horse whenever the page was loaded in a browser. It is an outstanding security problem that any subsequent re-definition of the same variable in JavaScript overwrites the previous definition.

McAfee blacklisted the proxy folder on the proof of concept application site as an unverified anonymizer in September 2009. The proof of concept application site's reputation was later reinstated at our request. Its status may be checked on McAfee's TrustedSource site: http://www.trustedsource.org/en/feedback/url?action=checksingle .

4.3 Cross-domain browser restriction

Any search engine query seeking a way to overcome browser cross-domain security restrictions results in a plethora of wrong-headed advice and wishful suggestions. It is enough to make one consider seriously the problem of misinformation on the web. Even on otherwise reliable sites, there is nonsense such as this quote from http://developer.yahoo.com/javascript/howto-proxy.html, accessed on October 27, 2009: "Internet Explorer will allow cross-domain requests if the option has been enabled in the preferences." After reading this, we followed the path given in Figure 5. No matter what Tools settings become Enabled, finally instead of crossing domains, the result is a JavaScript alert: "Access is denied."

One may bypass browser cross-domain security in several ways. The lazy way: wait patiently until it becomes allowed. There is indeed a draft standard to allow cross-domain XMLHttpRequests in the upcoming W3C Web API WG draft on Cross-Domain XMLHttpRequest (CS-XHR). A thorough discussion of this and all the extant security issues may be seen in a technical article titled "Client-side Cross-domain Security" on Microsoft's MSDN web site at http://msdn.microsoft.com/en-us/library/cc709423(VS.85).aspx

```
Tools
Internet Options
Security tab
click on the Internet globe icon
security level for this zone: Custom Level button
under Miscellaneous category
Access data sources across domains: Enable bullet
Navigate sub-frames across different domains: Enable bullet
also Tools / Pop-up Blocker / Turn off Pop-up Blocker
```

Figure 5. Futile actions taken in Internet Explorer 6 browser to enable cross-domain access.

CHAPTER 5

RELATED WORK

5.1 Data integration

The research by [Di Lorenzo et al., 2009] titled “Data Integration in Mashups” introduces a set of criteria for analyzing and comparing various software tools which are intended to facilitate the development of mashups. The first criterion is how well the tool extracts data from API’s or web services, which are not accessed in a browser but instead structured in XML, CSV, or other common data format. Speaking to this criterion, the proof of concept application’s proxy can access either HTML or XML using the JavaScript XMLHttpRequest (XHR), but other tools may be better equipped than the proof of concept application to integrate data other than HTML.

The second criterion in [Di Lorenzo et al.] is the quality of the internal data model, either graph-based or object-based. This criterion applies only loosely to the private agent of this paper, because the agent may be placed arbitrarily in a mashup among its data sources. It would be possible even to instantiate the agent completely in the background of a web application, displaying only the necessary controls for user indication of source, destination, and iteration.

The basic problem addressed by our paper actually comes from the third criterion in [Di Lorenzo et al.]. Data mapping is defined as the mashup tool’s strategy for specifying the correspondence between the internal data model of the previous paragraph, and the data retrieved in the mashup. Data mapping strategies are divided into three categories:

1. Manual, i.e. all mappings are specified by the user
2. Semi-automatic, i.e. hints about possible mappings are given
3. Automatic, wherein the correspondences are generated without user intervention.

The conclusion reached by [Di Lorenzo et al.] on Automatic data mapping is that more research would be necessary: “This is a challenging issue in the data integration area. Since the Mashup area is in its ‘early stage’, this type of mapping is not supported by any Mashup tool.” Our effort may begin to address their conclusion.

Other criteria presented in [Di Lorenzo et al.] include data flow operators, data refresh, mashup output, extensibility, and sharing.

5.2 Mashup Feeds

[Tatemura et al.] present a mashup query model with data flows in database terminology in designing their application named Mashup Feeds. Database terms are utilized to model the data flow between elements in the mashup, enhancing data integration. The concept of “binding patterns” in a web service model is explained:

> A web service $\mathcal{WS} : \mathcal{O}_{in} \rightarrow \mathcal{C} (\mathcal{O}_{out})$ is modeled as a virtual collection of composite objects: $\mathcal{C}_{WS} : \mathcal{C} (< \mathcal{O}_{in} , \mathcal{O}_{out} >)$ where $< \mathcal{O}_{in} , \mathcal{O}_{out} >$ denotes the type of a pair of complex objects $\mathcal{O}_{in}$ and $\mathcal{O}_{out}$. A web service call is a selection $\sigma \mathcal{O}_{in}$ over this collection.

The collection is bounded by the parameters placed on the attributes of the web service request. An example of the first type of complex object $\mathcal{O}_{in}$ is Figure 6 which is the slightly altered text of a web service request on the ISBN database. The bounds, or parameters, specify the third page of tuples in the “books” table having Ramez Elmasri as author or editor.

```
http://isbndb.com/api/books.xml?access_key=***&index=person_id&value=elmasri_ramez&page=3
```

Figure 6. Example bound attributes complex object web service request.

An example of the second type of complex object $\mathcal{O}_{out}$ is Figure 7 which is the XML document returned as the web service response from the example request in Figure 6. Responses possess free attributes. This document contains four tuples, each representing a different book having Ramez Elmasri as author or editor.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <ISBNdb server_time="2009-11-25T13:37:48Z">
- <BookList total_results="24" page_size="10" page_number="3" shown_results="4">
- <BookData book_id="fundamentals_of_database_systems_3_e_with_oracle_programming"
    isbn="0201612593" isbn13="9780201612592">
  <Title>Fundamentals of Database Systems 3/e with Oracle Programming</Title>
  <TitleLong />
  <AuthorsText>Ramez Elmasri, Shamkant B. Navathe, Navathe,</AuthorsText>
  <PublisherText publisher_id="addison_wesley_a01">Addison Wesley</PublisherText>
  </BookData>
- <BookData book_id="computer_organization_and_architecture_designing_for_per_a03"
    isbn="0582849632" isbn13="9780582849631">
  <Title>Computer Organization and Architecture: Designing for Performance</Title>
  <TitleLong>Computer Organization and Architecture: Designing for Performance (IE) with Objects
    First with Java - A Practical Introduction Using BlueJ with Introduction ... with Fundamentals of
    Database Systems (IE)</TitleLong>
  <AuthorsText>Rick F.Van Der Lans, Ramez Elmasri, William Stallings,</AuthorsText>
  <PublisherText publisher_id="prentice_hall">Prentice Hall</PublisherText>
  </BookData>
- <BookData book_id="fundamentals_of_database_systems_a08" isbn="1405811420"
    isbn13="9781405811422">
  <Title>Fundamentals of Database Systems</Title>
  <TitleLong>Fundamentals of Database Systems: AND Introduction to SQL Mastering the Structured
    Query Language</TitleLong>
  <AuthorsText>Ramez Elmasri, Rick Van Der Lans,</AuthorsText>
  <PublisherText publisher_id="addison_wesley_a01">Addison Wesley</PublisherText>
  </BookData>
- <BookData book_id="structure_based_xml_indexing" isbn="0542226871" isbn13="9780542226878">
  <Title>Structure based XML indexing</Title>
  <TitleLong />
  <AuthorsText />
  <PublisherText publisher_id="" />
  </BookData>
  </BookList>
  </ISBNdb>
```

Figure 7. Example free attributes complex object XML document web service response.

Our application would use the binding patterns model to depict a workflow which begins with a collection of text objects $\mathcal{T}$ selected (for having the same DOM path) from another structure in the mashup such that $\mathcal{T} : \mathcal{O}_{in} \rightarrow \mathcal{C}$ ($\mathcal{O}_{out}$). This is called "stateless" in [Tatemura et al., p. 1129] because the input collection is taken at only one point in time, for example text extracted from a web page in an *iframe*. Our web application's workflow model could be stated in terms similar to the Mashup Query Model described in [Tatemura et al., p. 1129] as $\mathcal{T} \rightarrow \mathcal{WS}(\mathcal{P})$ where the output collection of text objects serves as a set of parameters $\mathcal{P}$ sent to a web service, or to another type of web application requiring text input.

The Mashup Feeds application maintains four databases with information enabling serialization of continuous communication over time from web services, RSS feeds, and other dynamic web applications.

5.3 Karma

Previous research by [Tuchinda et al.] delineates five basic issues in complex mashups:

Data Retrieval
Source Modeling
Data Cleaning
Data Integration
Data Visualization

This paper's topic belongs in the Data Integration category, which may be defined as the methods used to combine data from more than one source. The proof of concept application is constructed using some of the approaches invented by Tuchinda et al. for their mashup tool named Karma.

The mashup presentation of an embedded web browser to the user's left, and a table containing user-selected text to its right, is original with Karma but utilized here with some differences. The user is not expected to highlight, drag, and drop text as in Karma. Instead a single mouse button click is enough to allow the proof of concept application to make a decision

and extract all text strings into the table. In Karma the table, actually a spreadsheet, is the end point of the data integration. However the proof of concept application really has no endpoint, as the data may be integrated from one embedded browser to another in either direction, within the mashup.

The proof of concept application uses the DOM for data retrieval in essentially the same way: identifying an XPath to the user-selected value, then searching the DOM tree for all elements at the same level in the tree. This concept was first researched at MIT and published as the mashup tool Simile (Huynh) in 2005.

Karma maintains a repository of data, which like Mashmaker adds an information retrieval element to the mashup. In another difference from our approach, Karma is intended for users with no programming background.

APPENDIX A

ACRONYMS

API	Application programming interface
AWS	Amazon Web Service
BSME	Bachelor of Science in Mechanical Engineering
CSE	Computer Science Engineering
CSV	Comma-separated values
DFW	Dallas-Fort Worth
DOM	Document Object Model
FORTRAN	Formula Translation
FTP	File Transfer Protocol
GTA	Graduate Teaching Assistant
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol
ICSOC	International Conference on Service-Oriented Computing
IE	Internet Explorer
IEEE	Institute of Electrical and Electronics Engineers
JSP	Java Server Pages
LNCS	Lecture Notes in Computer Science
NPR	National Public Radio
PA	Private agent
RDF	Resource Description Framework, an element of the semantic web
REST	Representational State Transfer
RPC	Remote Procedure Calling
SIGMOD	Special Interest Group on Management of Data
SOAP	Simple Object Access Protocol
TCU	Texas Christian University
UDDI	Universal Description Discovery and Integration
URL	Uniform Resource Locator
UTA	University of Texas at Arlington
VWS	Virtual Web Service
W3C	World Wide Web Consortium
WSDL	Web Services Definition Language [Jacobs, p. 86]
XHR	XMLHttpRequest
XML	Extensible Markup Language
XML-RPC	RPC using HTTP as the transport and XML as the encoding

REFERENCES

(1) Bolin, Michael. 2005. End-user programming for the Web. Master's thesis, Massachusetts Institute of Technology.

(2) Di Lorenzo, Giusy, Hakim Hacid, and Hye-young Paik. 2009. Data Integration in Mashups. SIGMOD, Association for Computing Machinery's Special Interest Group on Management of Data, March 2009, Vol. 38, No.1.

(3) Edwards, James, and Cameron Adams. 2006. *The JavaScript Anthology: 101 Essential Tips, Tricks & Hacks*. Melbourne: SitePoint.

(4) Harris, Ray. *Murach's JavaScript and DOM Scripting*. 2009. Fresno: Mike Murach & Associates.

(5) Holdener, Anthony T., III. *Ajax: The Definitive Guide*. 2008. Sebastopol: O'Reilly.

(6) Huynh, David, Stefano Mazzocchi, and David Karger. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. ISWC'05, 4th International Semantic Web Conference, November 6-10, 2005, Galway, Ireland.

(7) Jacobs, Sas. *Beginning XML with DOM and Ajax:From Novice to Professional*. 2006. Berkeley: Apress.

(8) Liu, Jianxun and Lian Chao. Web services as a Graph and Its Application for Service Discovery. 2006. IEEE Proceedings of the Fifth International Conference on Grid and Cooperative Computing, GCC'06. 21-23 October 2006, Hunan, China.

(9) Maximilien, E. Michael, Hernan Wilkinson, Nirmit Desai, and Stefan Tai. 2007. A Domain-Specific Language for Web APIs and Services Mashups. Krämer, Bernd; Lin, Kwei-Jay; Narasimhan, Priya (Eds.): ICSOC 2007, Fifth International Conference on Service-Oriented Computing, Vienna, Austria, September 17-20, 2007. Proceedings.

Lecture Notes in Computer Science LNCS 4749, pp. 13-26. Berlin Heidelberg: Springer-Verlag.

(10) Mueller, John Paul. *Mining Amazon Web Services: Building Applications with the Amazon API*. 2004. San Francisco: Sybex.

(11) Rykowski, Jarogniew, and Wojciech Cellary. 2004. ICEC'04, Sixth International Conference on Electronic Commerce, September 1-3, 2004, Eindhoven, The Netherlands.

(12) Shoham, Yoav, and Kevin Leyton-Brown. 2009. *Multiagent Systems*. New York: Cambridge University Press.

(13) Tatemura, Junichi, and Arsany Sawires, Oliver Po, Songting Chen, K. Selcuk Candan, Divyakant Agrawal, and Maria Goveas. Mashup Feeds: Continuous Queries over Web Services. SIGMOD '07, June 12-14, 2007, Beijing, China.

(14) Tuchinda, Rattapoom, Pedro Szekely, and Craig A. Knoblock. 2008. Building Mashups By Example. IUI'08, International Conference on Intelligent User Interfaces, January 13-16, 2008, Maspalomas, Gran Canaria, Spain.

(15) Wenz, Christian. 2007. *JavaScript Phrasebook*. Indianapolis: Sams Publishing.

(16) Wu, Chen and Chang, Elizabeth. 2007. Searching services "on the web": A public web services discovery approach, in Andres, Frederic and William Grosky (ed), Third International Conference on Signal-Image Technology & Internet-based Systems, 16 Dec 2007, pp. 297-304. Shanghai, China: IEEE.

(17) Yee, Raymond. 2008. *Pro Web 2.0 Mashups: Remixing Data and Web Services*. New York: Springer-Verlag.

BIOGRAPHICAL INFORMATION

Ronda Hilton first coded FORTRAN programs as a student employee of NPR station WSUI/KSUI in Iowa City, Iowa. She graduated from UTA in 1989 with the B.S. degree in Mathematics with Computer Science Engineering option. Her son, Elijah Gay, graduated from UTA in May 2009 with his own B.S.C.S. degree. Miss Hilton has worked as a software engineer since 1990, currently at Raytheon Company in McKinney. She is also an accomplished professional musician, holding Bachelor's and Master's performance degrees from the University of Iowa and Texas Christian University, respectively. She plays a repertoire of masterworks on pipe organs in Tarrant County, and serves the Fort Worth chapter of the American Guild of Organists as Webmaster. Miss Hilton plans to pursue research in information security, information retrieval, and web databases.