Mashing up Visual Languages and Web Mash-ups

M. Cameron Jones, Elizabeth F. Churchill
Yahoo! Research
{mcjones, elizabeth.churchill}@yahoo-inc.com

Michael B. Twidale
University of Illinois at Urbana-Champaign
twidale@uiuc.edu

Abstract

Research on web mashups and visual languages share an interest in human-centered computing. Both research communities are concerned with supporting programming by everyday, technically inexpert users. Visual programming environments have been a focus for both communities, and we believe that there is much to be gained by further discussion between these research communities. In this paper we explore some connections between web mashups and visual languages, and try to identify what each might be able to learn from the other. Our goal is to establish a framework for a dialog between the communities, and to promote the exchange of ideas and our respective understandings of human-centered computing.

1. Introduction

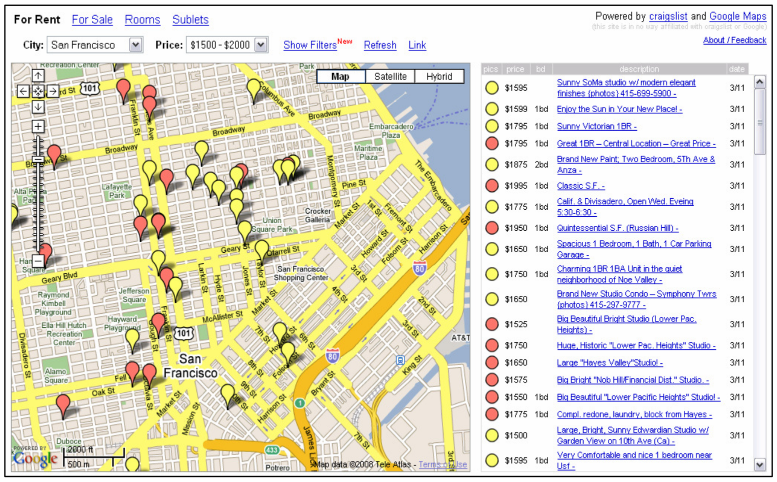
The recent popularity of the "programmable web" reflects a shift in the ways in which people think about information technologies, computing, and programming. There are several trends embedded in this notion of a programmable web which are coming together to encourage new modes of technology production. One trend is the shift away from the desktop computing model where software is installed locally on your machine which contains your data, towards web applications where personal and public data and services coexist on remote servers distributed across the web. Layering these new applications on top of existing web technologies has the consequence of everything being addressable with URLs, accessible via HTTP, and presented in a structured or semi-structured format. This opens up access to these data sources and services, either through formal or de facto application programming interfaces (APIs), allowing people to computationally access them, and remix data to create new applications called web mashups.

Mashups offer great potential as extremely rapid development environments, allowing skilled programmers to leverage existing systems and services to quickly and easily compose an application. However, in practice mashups are complex to program, involving arcane calls, complex data transformations and the challenges of integrating applications developed without regard to reuse and lacking consistency in their architectures and communication protocols. Various attempts are being made to address the problems that are caused for end-users as a result of these complexities – for example the development of new languages and programming tools, many of which offer manipulable, visual representations for coding and debugging. This paper is an exploration of ideas originating from the mashing up of concepts from visual languages with web mashups. We examine cognitive dimensions [4] as a possible common framework for identifying and discussing issues of human-centered computing which may enable a mutual dialog between these communities.

2. Background

The landscape of mashup programming is not well charted. In contrast to typically closed environments of domain-specific visual programming languages (VPLs) where technologies can be controlled and homogenized, mashups exist in an open ecosystem where the technologies involved are constantly shifting, evolving, and being replaced and regenerated. Furthermore, it is unclear who the mashup developers are, with little known about their particular development practices, and motivations [11]. What is known is that the mashup community includes professional and "hobbyist", end-user programmers. Researchers have studied students learning to program [3], and workers in enterprise organizations [1], but little is known about the motivations and practices of hobbyist programmers and "everyday programmers" [9].

For many programmers an attraction to mashups is the apparent quick payoff that they offer in terms of bootstrapping development. That is, developers may use a pre-existing data source, extract and manipulate the data they want and then use a pre-existing visualization to display the results in an interesting way. The developer "just" has to write the code to "glue" the services together. Achieving this can take relatively few



pics	price	bd	description	date
🟡	$1595		Sunny SoMa studio w/ modern elegant finishes {photos} 415-699-5900 -	3/11
🟡	$1599	1bd	Enjoy the Sun in Your New Place! -	3/11
🟡	$1795	1bd	Sunny Victorian 1BR -	3/11
🔴	$1795	1bd	Great 1BR – Central Location – Great Price -	3/11
🟡	$1875	2bd	Brand New Paint; Two Bedroom, 5Th Ave & Anza -	3/11
🔴	$1995	1bd	Classic S.F. -	3/11
🟡	$1775	1bd	Calif. & Divisadero, Open Wed. Eveing 5:30-6:30 -	3/11
🔴	$1950	1bd	Quintessential S.F. (Russian Hill) -	3/11
🟡	$1650	1bd	Spacious 1 Bedroom, 1 Bath, 1 Car Parking Garage -	3/11
🟡	$1750	1bd	Charming 1BR 1BA Unit in the quiet neighborhood of Noe Valley -	3/11
🟡	$1650		Brand New Studio Condo – Symphony Twrs {photos} 415-297-9777 -	3/11
🔴	$1525		Big Beautiful Bright Studio (Lower Pac. Heights) -	3/11
🔴	$1750		Huge, Historic "Lower Pac. Heights" Studio -	3/11
🔴	$1650		Large "Hayes Valley"Studio! -	3/11
🔴	$1575		Big Bright "Nob Hill/Financial Dist." Studio. -	3/11
🔴	$1550	1bd	Big Beautiful "Lower Pacific Heights" Studio! -	3/11
🔴	$1775	1bd	Compl. redone, laundry, block from Hayes -	3/11
🟡	$1500		Large, Bright, Sunny Edwardian Studio w/ Garden View on 10th Ave (Ca) -	3/11
🟡	$1595	1bd	Very Comfortable and nice 1 bedroom near Usf -	3/11

Visible Components	
Page	HTML & JS controlling interactions between property listings and map
Listings	Fetched via RSS feed from Craigslist
Map panel	JS library from Google Maps
Filters	JS which filters the current data set to display only relevant properties.
Non-visible Components	
Proxy	Proxies and caches the RSS feed from Craigslist to bypass browser security restrictions. Transforms data to proprietary text format.
Entities	
Google Maps	Map Libraries
Craigslist	Property Listings in RSS
HousingMaps	The HTML page; RSS proxy and caching module; JS for filtering data; JS for controlling interaction

Figure 1. Decomposing a mashup - HousingMaps.com

lines of code – often astonishingly less than building the same functionality from scratch, and without the need to manage seed data. However, effective glue code snippets can be very complex to write or discover so that the word "just" becomes highly ironic.

2.1 Decomposing a Mashup

HousingMaps.com (HM) is an example of a simple mashup which combines real-estate listings from Craigslist with a Google Map allowing users to browse properties spatially (see Figure 1). The basic functionality of HM involves searching for listings in Craigslist, and displaying them in a map. Search results are displayed in a map-view and a matrix-view, side-by-side. Of the elements of the HM mashup described in Figure 1, the page itself, the JavaScript for filtering search results, the RSS proxy, and other pieces of code for controlling the presentation of listings and the interactions between the matrix and map views, are all glue code for building the mashup.

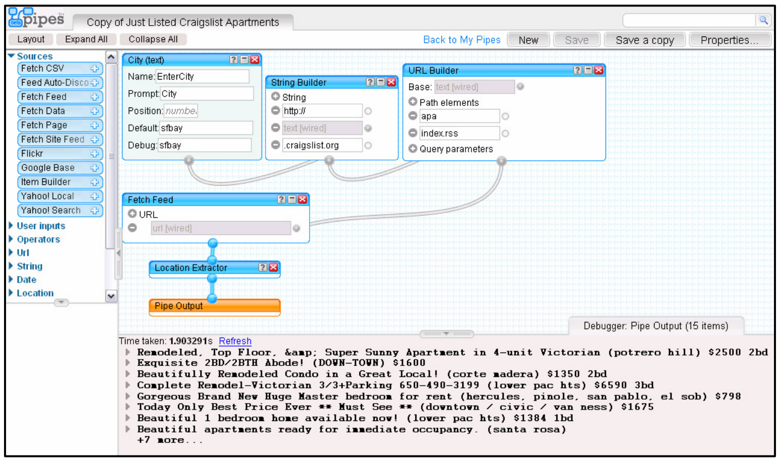
Additionally, the RSS feed from Craigslist does not contain the geo-coded GPS coordinates of the real-estate listings, only street addresses. However, viewing the HTTP traffic shows that the data being pulled into the mashup is geo-coded. Therefore, HM must be geo-coding the addresses after it fetches them from Craigslist. It is most likely using a web service to look up GPS coordinates for each street address when it fetches and caches the RSS data (it should be noted that recent versions of the Google Maps API include geo-coding, allowing addresses to be mapped directly without GPS coordinates). Thus, what appears to be a simple data and visual combination, on deeper analysis, turns out to be a complex and multi-faceted challenge for those hoping to build a mental model of the data flow, integration, and representation. Often these complexities only become apparent when the mashup "fails".

2.2 Visual Tools for Mashups

Concepts and techniques from visual programming have been applied to improving mashup development [7], resulting in numerous visual mashup development environments (MDEs). MDEs come in a variety of forms, including wikis (e.g., [1], [5], and QEDWiki), spreadsheets (e.g., EditGrid), and dataflows (e.g., [10], Yahoo! Pipes, and Microsoft Popfly). Figure 2 shows the visual source code of a Yahoo! Pipes application which approximates the functionality of the HousingMaps mashup. Yahoo! Pipes affords certain kinds of programming interactions, making some operations easier and others more complex. For example, over half of the source code of this pipe (3 of 5 modules) is devoted to constructing the URL for the RSS feed from Craigslist. Fetching and geo-coding the data are executed in the last two modules (there is an "output" module which is used to demarcate what data is to be returned after the pipe has been executed). Mapping the data is handled outside the code of the pipe itself.

3. Cognitive Dimensions

Cognitive dimensions (CDs) have been used as one way of analyzing VPLs and evaluating interface and interaction designs. Following the approach of Burnett [2], we have applied CDs to both MDEs and the broader mashup ecosystem. Our motivation was not only to reflect on what CDs tell us about mashup programming, but also to see the ways in which mashups challenge the definitions and boundaries of current CDs. The CDs originally defined by [4] are: abstraction gradient, closeness of mapping, consistency, diffuseness, error-proneness, hidden dependencies, premature commitment, progressive evaluation, role-expressiveness, secondary notation, viscosity, and visibility. We provide a cursory



Name: EnterCity
Prompt: City
Position: number
Default: sfbay
Debug: sfbay

String
http://
text [wired]
.craigslist.org

Base: text [wired]
Path elements
apa
index.rss
Query parameters

URL
url [wired]

```
Time taken: 1.903291s
 Remodeled, Top Floor, &amp; Super Sunny Apartment in 4-unit Victorian (potrero hill) $2500 2bd
 Exquisite 2BD/2BTH Abode! (DOWN-TOWN) $1600
 Beautifully Remodeled Condo in a Great Local! (corte madera) $1350 2bd
 Complete Remodel-Victorian 3/3+Parking 650-490-3199 (lower pac hts) $6590 3bd
 Gorgeous Brand New Huge Master bedroom for rent (hercules, pinole, san pablo, el sob) $798
 Today Only Best Price Ever ** Must See ** (downtown / civic / van ness) $1675
 Beautiful 1 bedroom home available now! (lower pac hts) $1384 1bd
 Beautiful apartments ready for immediate occupancy. (santa rosa)
 +7 more...
```

Figure 2. HousingMaps programmed in Yahoo! Pipes

analysis of mashups with respect to the dimensions we feel are most relevant.

Abstraction gradient

Green and Petre [4] define an abstraction as a grouping of multiple elements into a single entity. The abstraction gradient is the difference between the minimum and maximum levels of abstraction. Mashups have a large abstraction gradient, touching multiple layers of the programming stack, from data and file systems, to the graphical user interface. Interactions with these multiple layers are often blended in a single script (or even single lines of code – see Figure 3). Working across layers is often required when combining different services or sources which operate at different levels of abstraction. For example, in the HousingMaps mashup, Google Maps is abstracted into a high-level entity, freeing the user from the many low-level details of fetching and rendering the map data. However, Craigslist lacks an API library, and so the mashup developer must write code to request and marshal data at a fairly low-level. This alignment happens across multiple programming languages, in both the server and client-side contexts.

Consistency

Consistency, the degree to which new knowledge can be inferred given what is already known, is very low in mashups. Little knowledge about how new services will function can be inferred by analogy to other services. Each mashup service provides its own API, data formats, and invocation patterns. Standards provide some consistency in the technological infrastructure which enables mashups to be possible; however, many are merely serialization standards, and the data structures being serialized and passed back and forth are unique to each service, requiring custom marshalling.

Hidden dependencies

The mashup ecosystem has many hidden dependencies. What happens within the black-boxes of the remote server calls is unknown, and more importantly, unknowable. The black-boxes of remote APIs reflect the *intentional hiding* of what is inside. This is a very different kind of black-box than the abstractions used in VPLs, which are used as scaffolds to aid development.

Furthermore, the dependencies between services are not necessarily explicit. A service may require data of a particular type, or formatted in a particular manner. Type and format may not be the same as that used by other services, nor correspond to the developer's conception of the data (e.g., the geo-coding transformation in the HM mashup). Data transformations raise the complexity of the overall mashed-up system, and impose additional dependencies between the services.

The browser programming environment likewise imposes dependencies which are not always visible or obvious. In the HousingMaps example presented earlier, the need for a proxy service to fetch and cache the RSS feeds is required to bypass the JavaScript "same origin" security restriction.

Progressive evaluation

The all-or-nothing remote call to a remote service does not easily afford debugging. When services fail to respond, or respond with unexpected results, the developer can only debug to a certain level before reaching the atomicity of the black-box service call.

Hard mental operations

Remembering multiple, inconsistent syntaxes for different API calls, sometimes involving two or more languages that are embedded in each other is hard; this imposes an overhead of keeping track of what is being done and why. These "hard mental operations" are often a result of the limitations imposed by other CDs, e.g., large abstraction gradients, with low consistency can combine to create mentally hard operations which require working across several levels of abstraction, in multiple languages, simultaneously. For example, in the snippet shown in Figure 3, a PHP array reference (`$item`) is embedded in a line of JavaScript code (`var asin...`), which is in an HTML document, being

```
<?php
  //...
?><html><head>
  <script type="text/javascript">
  var asin = "<?=$item['Asin']?>"
  ...
<?php
  //...
?>
```

Figure 3. Messy mashup code from [8]

generated by a PHP script.

3.1 Refocusing Cognitive Dimensions

Analyzing the cognitive dimensions of mashups not only helps formalize an understanding of mashups, but also surfaces aspects of CDs which have not been extensively discussed.

Stability

Stability is the measure of resistance to change. It is similar to *viscosity*, however viscosity is a measure of the effort required for the user to perform a change and stability is about the resistance of the ecosystem as a whole to changes over time. When proposed, the cognitive dimensions framework was not used to evaluate usability over time, merely the usability of a given instance. However, mashups exist in a highly dynamic, open ecosystem, with rapidly evolving and changing components. This can lead to data, functions, and APIs being deprecated, made redundant or obsolete, or simply disappearing. Thus increasing the cognitive burden on users, as programming know-how is short-lived in such an environment.

Robustness

Another cognitive dimension foregrounded by mashups is robustness. Domain-specific VPLs may be able to make assumptions about the quality and reliability of data, but mashup data is often messy and sources fail sporadically. How well do end-users and VPLs cope with incomplete, missing, or contradictory data? Unlike *error-proneness* which measures how easy it is for users to make mistakes, robustness is about how the *system* responds to errors. Trapping and tolerating errors can obscure their sources, blurring the distinction between syntax errors, logical bugs, and bugs in the underlying infrastructure. This can complicate problem identification and resolution.

Sharability

One mechanism mashup developers use to cope with the complexities, and dependencies of mashup programming is sharing, and reusing shared code [6]. The ability to share and reuse code and code snippets affords cognitive off-loading, allowing users to reduce the individual cognitive overhead of having to reconstruct code from scratch or establish shared context by other means. The most successful MDEs have facilities for viewing and sharing code with others. The conversations developers have with, and around code serve as informal documentation with reusable examples, and provide human-language descriptive contexts for searching and understanding.

4. Conclusion

Clearly, we believe there is much purchase to be gained from analyzing similarities between mashup programming and visual language programming from an end-user perspective. The cognitive dimensions framework provides a rich vocabulary for analyzing not only visual programming languages, but also the mashup ecosystem. Analyzing mashups has produced additional insights into the cognitive dimensions of programming heterogeneous applications. Cognitive dimensions is just one framework for analysis; others may shed light on other aspects of the mashup ecosystem previously understudied, or highlight other ways in which mashups may inform our understanding of human-centered computing.

5. References

[1] C. Anslow & D. Riehle (2008) Towards End-User Programming with Wikis. In Proceedings of WEUSE 4.

[2] M. Burnett (1999) Visual Programming. In Wiley Encyclopedia of Electrical and Electronics Engineering (J. Webster, ed.) John Wiley & Sons, Inc.

[3] I. R. Floyd, M. C. Jones, D. Rathi & M. B. Twidale (2007) Web Mash-ups and Patchwork Prototyping: User-driven technological innovation with Web 2.0 and Open Source Software. In Proceedings of HICSS'07.

[4] T. R. G. Green & M. Petre (1996) Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework. Journal of Visual Languages and Computing 7, 131-174.

[5] B. Hartmann, L. Wu, K. Collins & S. R. Klemmer. (2007) Programming by a Sample: Rapidly Prototyping Web Applications with d.mix. In Proceedings of UIST'07.

[6] M. C. Jones (2007) Web Mashups: Technological Appropriation in Web 2.0. In Proceedings of 4S 2007.

[7] S. C. Lim, S. Lowe, J. Koempel, (2007) Application of Visual Programming to Web Mash Up Development. In Proceedings of HCII'07.

[8] M. C. Jones & M. B. Twidale (2006) Mashups and CSCW: opportunities and issues. CSCW'06 Workshop.

[9] M. B. Rosson, J. Ballin, & H. Nash (2004) Everyday Programming: Challenges and Opportunities for Informal Web Development. In Proceedings of VLHCC'04 IEEE.

[10] J. Wong & J. Hong (2007) Making Mashups with Marmite: Towards End-User Programming for the Web. In Proceedings of CHI'07.

[11] N. Zang, M. B. Rosson, V. Nasser. (2008). Mashups: Who? What? Why? In CHI'08 Extended Abstracts.