The Mashup as a Lens on End-User Programming for the Web

Jeffrey Wong, Jason I. Hong
Human-Computer Interaction
Carnegie Mellon University
Pittsburgh, PA 15217
jeffwong@cmu.edu, jasonh@cs.cmu.edu

ABSTRACT
Our past work in Web End-User Programming (WebEUP) has focused mainly on users trying to build or replicate functionality found in mashups. In this paper however, we discuss how this is perspective is simply one of several perspectives on WebEUP. We review different characteristics of WebEUP tools, and how these may be combined with characteristics of actual mashups, mashup-like artifacts, and tasks to form a model that may give insight into how to organize the work on WebEUP

Author Keywords
End-user programming, mashups, web macros, automation, data integration

ACM Classification Keywords
H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION
Mashups and mashup construction tools have recently proliferated as users and developers have recognized that data and functionality found on websites can be used as computational resources. Although our work has focused on mashups [*b] and enabling end-users to replicate mashup functionality using higher-level tools [*a], we have noticed that there are patterns to common approaches, common problems, and interesting future challenges to end-user programming for the Web (WebEUP).

MOVING FROM ON FROM TRADITIONAL EUP
There has been much research end-user programming systems [*c] and tools that are widespread in daily use [*d]. Spreadsheets can be thought of as numerical models or interconnected formulae. Programming by demonstration [*f] can be used help in text manipulation[*e], construction of user interfaces [*g], and many other domains. Is end-user programming for the web simply another domain? The web is a programmable domain in large part due to a few fortunate factors:

1) data seen in the interface of a website is extractable from the textual representation of data,

2) data on many web pages tend to follow structural and semantic patterns [*k],

3) and data on web sites mostly all accessible from the same network.

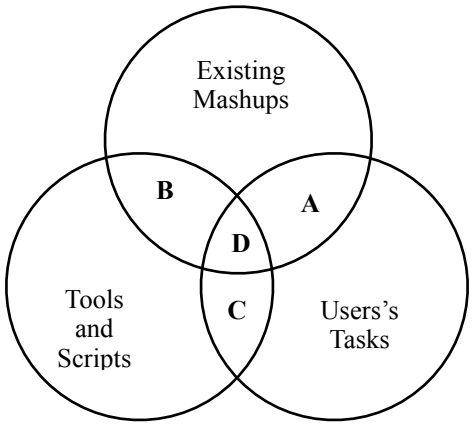
But is WebEUP a single domain? Our work has focused on a subset of WebEUP commonly referred to as "mashups". The definition of what a mashup is very fuzzy, when based on the intentions of mashup creators, providers of mashup resources, and users. In surveying mashups on popular mashup directories [*b], we found that there were certain patterns, for example *aggregation of data sets* (when a website can be considered an interface) or providing focused views on a data set. Mashups that aggregate data might be distinguished by whether they resolve multiple schemas or whether they cross-reference attributes between data sets [*i, *j]. In our survey of mashup patterns [*b], we found that the only necessary and sufficient condition that allows something to be called a mashup is that it re-uses a web resource in a novel way; combination of multiple web resources is not necessary.

Being a website might be a necessary property of being a mashup, but many WebEUP tools enable users to "create mashups" (a synonym for the mashing up activity), without necessarily having an website as an end result. For example, "within-page" programming languages and component sets such Greasemonkey, Chickenfoot [*l], and Intel's Mashmaker [*m] allow users to modify or augment data on pages while they are viewing them. As far as some users are concerned, this can be equivalent to what a comparable (but hypothetical) mashup might achieve, but without the overhead of searching for a site that fits a particular requirements in a particular moment, which may also evolve over the course of a task. Categories are blurred further when we consider simple automation scripts on one or more websites from tools such as CoScripter [*h] and emergent properties that arise from component sharing.

The web as an EUP problem might be considered a single domain or a collection of multiple domains, or it can be both at the same time. We propose that there are 3 main perspectives that will help describe WebEUP:

1. perspectives taken by existing tools to create mashups and mashup-like artifacts such as the within-page programming and automation languages
2. perspectives embodied by existing mashups
3. and example use cases from empirical data recorded potential users of WebEUP tools.

A helpful framework for characterizing WebEUP that unifies this perspectives is to see how different tools, existing mashups, scripts, and tasks overlap in Figure 1.



Region A: Mashups that users can complete their tasks with. They just need to be found.

Region B: Mashups that can be built with mashup tools but have no clear task fit (perhaps experimental).

Region C: Tasks that can be solved with mashup tool but are not full-featured websites. "One-time mashups" that are thrown away.

Region D: Mashups that can be built with mashup tools that resemble existing mashup websites. Reused by many and adaptable to support many tasks.

Figure 1. Proposed Model of WebEUP space.

In Figure 1, there are tools that can create mashups which are of use to no one (Region B) while there are also mashups that have been constructed for fun or experimentation that do not fulfill a specific task (or none that has emerged yet). Region C represents tools that can help fulfill some WebEUP tasks but cannot create fully reusable mashups, as either public websites or sharable objects that can be quickly understood and adapted to another person's task. Region D represents represents tools that are adaptable to many tasks and can also be generalized or abstracted from an initial concrete or prototyped object to something reusable, available, and reliable.

MASHUP TOOLS

This section gives a brief summary of popular mashup construction tools, grouped roughly by their framing of what WebEUP programming should be.

Microsoft Popfly, Marmite, Yahoo Pipes

These tools conceive of mashups as data-flows between operations. Operations are input-output blocks that the user must select from a palette. Operations in Popfly and Marmite[*a] are constructed using at the Javascript programming level with some textual programming interface. They become available to the higher-level flow design. Yahoo Pipes operations are standard low-level operations such as counting, building up strings, and filtering for strings. The primary unit of data is the tuple. Marmite shows the state of the data at each step, while Pipes must be explicitly put in debug mode to see the data as it flows through the program. Each of these tools construes WebEUP as the construction of a program. Yahoo Pipes emphasizes saving and sharing so that others can learn from examples.

Operations in Marmite and Popfly are intended to be easy-to-use wrappers around web services

d.mix

d.mix[*o] is a system that attempts to bridge the gap between elements as seen by the user on a page, and web services calls that produce similar results. It is functionally a system that appends user contributed annotations that show how certain elements on web pages can be replicated with web services calls. Like Mashmaker, these annotations are canonicalized into a shared repository where users can discover API calls by using the normal website and later, write small scripts to assemble their own web pages using component web services that they have discovered while browsing.

MASHUP-LIKE ARTIFACTS

CoScripter

CoScripter (aka Koala) [*n] is a demonstrative recording system where users record a sequence of steps that they want to repeat in the future or describe processes they want to share with others [*h]. Script actions are recorded in English-like statements so they can be previewed by other users who may consider using the script. Koala supports some parameterization of scripts with "personal variables" and mixed-initiative interaction, where the human can intervene for in steps where the script no longer works perfectly, a human needs to verify an action, or to make judgements which may not be computable (e.g. "select the nicest dress in the list").

Karma, Dontcheva's Relational Cards

Karma [*i] and Dontcheva *et al.*'s relational cards [*j] address the problem of cross-referencing entities and retrieving attributes of entities from multiple databases. Disambiguation of entities and matching in foreign datasets is a challenge but demonstrational transformations have been presented as solutions. WebEUP as conceived by these systems is a very data-centric activity that focuses on how the user can transform the data by enriching them and consolidating them into useful condensed interfaces that remove content from a page (although the original page context is still accessible).

Miro, Sifter

Miro and Sifter[*k] are semantically-oriented data detectors that try to detect object of interest on a page (with some demonstration), and present the user with semantically relevant operations or attribute filters. Sifter in particular is one of the few tools that attempts to use page structure and semantics to deal with the fact that many listings found on the web are paginated across multiple pages. Useful

elements are extracted into a persistent personal store which can be reused when the user visits other pages.

Intel Mashmaker, Chickenfoot, Greasemonkey

Greasemonkey allows user-created Javascript scripts to modify pages and can be triggered to automatically activate when particular types of pages are visited. Chickenfoot [*l] is a higher-level dialect of Greasemonkey that allows the use of plain English terms to refer to page elements. These tools are backed by community-generated repositories of scripts. Data is represented as text nodes on a page but can be converted into variables internal to the Javascript environment surrounding those tools.

Similarly, Intel's Mashmaker [*m] consists of components that are user-generated but divides the problem of "mashing up" into the separate problems of extraction of information and modification or presentation of information. Any web page can be associated with a canonical detector/extractor that type of web page. Incentives for fixing a canonical extractor are similar to those for fixing articles on Wikipedia. Extractor authors fix things because they are broken but also help the community at the same time Also, since extractors auto-detect what pages they are applicable to, users don't have to search a repository for appropriate extractors. Re-presentation and transformation of data found on a page is done using widgets that a created by more advanced users who can program in a Javascript. Data are represented as RDF tuples that can be adapted to semantically relevant output widgets.

All of these tools can communicate with web services on behalf of the user.

MASHUPS IN THE WORLD

There are now thousands of publicly mashups with are websites and many more that are hidden within the ecosystems of their own tools. Although looking at mashups that have conceivable uses can yield patterns, whether these patterns are representative of the kinds of WebEUP that users need is questionable[*b]. There is little value to creating tools that will land users in Region B (see Figure 1). On the other hand, more detailed or automated examination of objects such as GreaseMonkey scripts or Firefox plugins (such as in [*q]) may reveal mashup needs that exist but did not warrant constructing an entire site for. Also, there is probably a bevy of unfulfilled WebEUP needs that cannot be fulfilled because learning curve of existing tools is simply too high.

WEB EUP TASKS FOUND IN THE REAL WORLD

There has been work in taking an ethnographic methods to task discovered such as contextual inquiry and observation as well as recording the tasks that researchers noticed in their own daily lives from the perspective of web macros [*r]. We have also kept a list of our own tasks but from a mashup perspective. However, there may a greater diversity of tasks that have not been discovered because any users in the moment must be simultaneously aware of the work context they are in, the capabilities of available tools, and the depth and quality of data sources to be combined. A diary study with a pre-briefing of how tools might work and layman's guides to both popular and unusual API's might uncover such tasks. This study or even an aggregation and classification of the tasks we already have may help us understand Region C of Figure 1 in better detail. It may be the case that elements of computational thinking [*s] is a prerequisite to developing the necessary awareness of what problems can be solved with WebEUP.

COMMON THREADS IN EXISTING TOOLS

The brief survey of tools in the earlier section shows that there are several common characteristics of tools:

- Social sharing: occurs in the form of sharing examples of previous work, experts creating widgets or high-level components for novices, and searching archives for an existing solution supported by a particular tool.
- Program construction vs. data manipulation focus: Some tools orient the user around the construction of a program artifact (Pipes, Popfly, Marmite, CoScripter) while others focus primarily on the data and supports manual culling of a personal store (Dontcheva *et al.*'s Relational Cards and Sifter). There is no explicit program because interaction establishes the relationships and places the data in the foreground.

Creating a social ecosystem is a helpful way provide a starting ground for novices with examples [*d]. However, the design and maintenance of such an ecosystem is as important if not equally important as the contributions of the tools themselves.

Although one can find a script or program written by someone else, making use of it still incurs search costs that may be different from that of typical information scent theory. Is evaluating a mashup, script, or other shared software solution in light of one's task requirements possible without training? Perhaps more metadata is needed to make reusable code objects sharable. Or can search be eliminated when the opportunities for mashing up appear while the user browses the web? Furthermore, if WebEUP is intended to address the far end of the long tail, if a user finds themselves on a website where no others have gone (or cared to automate), can WebEUP systems rely on community?

Furthermore, the proliferation of WebEUP tools means that users who think of WebEUP as a solution to their problems, also face an attention investment [*t] dilemma because they must assess which tool will solve their problems. Different tools will solve certain aspects of their problems. However, given that most WebEUP tools only attempt to address a subset of the problem space, any task which crosses the boundaries of tool capabilities may be hard to complete. Perhaps a mashup protocol for WebEUP tools is necessary.

FUTURE CHALLENGES FOR WEB EUP

The Web is evolving quickly and designers of WebEUP tools are designing for a moving target. Will dynamic AJAX or Flash oriented websites spell the end of methods that rely on screen-scraping? Are web services APIs more stable than user interface elements or is the reverse true?

In our future work, we will be moving towards the data-centric design approach instead of focusing on program

construction as our previous system had [*a]. And we will attempt to create a tool where novices does rely the existence of an expert community to program widgets for a higher-level layer.

CONCLUSION

In this workshop, we hope to contribute discussion on WebEUP along the lines of what has been discussed in this paper. We would further like to be exposed to alternative ways of framing the WebEUP space, categorizing the tool design space, and categorizing the task space. Since no single tool is unlikely to be best at all aspects of WebEUP, we would like to discuss how interoperability may be possible. Finally, we hope that, on top of being a compendium of current research, the resulting book that can also be abridged into an edition for the lay person trying to select tools to program the Web.

REFERENCES

[*t] Blackwell, A. F. (2002). First steps in programming: A rationale for attention investment models. HCC, 2-10.

[*l] Bolin, M., Webber, M., Rha, P., Wilson, T., and Miller, R. C. 2005. Automation and customization of rendered web pages. In Proc of *UIST '05*.

[*f] Cypher, A. (1993). Watch what I do: Programming by demonstration. Cambridge, Mass: MIT Press

[*j] Dontcheva, M., Drucker, S. M., Salesin, D., and Cohen, M. F. (2007). Relations, cards, and search templates: user-guided web data integration and layout. In Proceedings of *UIST '07*, 61-70.

[*m] Ennals, R. J. and Garofalakis, M. N. (2007). MashMaker: mashups for the masses. In Proc of SIGMOD '07.

[*p] Faaborg, A. and Lieberman, H. (2006). A goal-oriented web browser. In Proc of CHI '06.

[*o] Hartmann, B., Wu, L., Collins, K., and Klemmer, S. R. (2007) Programming by a sample: rapidly creating web applications with d.mix. In Proc of UIST '07.

[*k] Huynh, D. F., Miller, R. C., and Karger, D. R. (2006) Enabling web browsers to augment web sites' filtering and sorting functionalities. In Proc of UIST '06.

[*c] Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Comput. Surv., 37(2), 83-137.

[*e] Lau, T., Wolfman, S., Domingos, P., & Weld, D. S. (2000). Learning repetitive text-editing procedures with smartedit. In H. Lieberman (Ed.), Your Wish is My Command. Morgan Kaufmann.

[*h] Leshed, G., Haber, E. M., Matthews, T., and Lau, T. (2008) CoScripter: automating & sharing how-to knowledge in the enterprise. In Proc of *CHI '08*. New York, NY, 1719-1728.

[*n] Little, G., Lau, T. A., Cypher, A., Lin, J., Haber, E. M., & Kandogan, E. (2007). Koala: Capture, share, automate, personalize business processes on the web. Proc of CHI 2007.

[*g] Myers, B., & McDaniel, R. (2000). Demonstrational interfaces: Sometimes you need a little intelligence; sometimesyou need a lot. In H. Lieberman (Ed.), Your Wish is My Command. Morgan Kaufmann.

[*d] Nardi, B. A. (1993). A small matter of programming. Cambridge: MIT Press.

[*q] C. Scaffidi, A. Cypher, S. Elbaum, A. Koesnandar, and B. Myers. (2008) Using Scenario-Based Requirements to Direct Research on Web Macro Tools. Journal of Visual Languages and Computing, Vol. 19, No. 4, Aug 2008, 485-498.

[*r] C. Scaffidi, A. Cypher, S. Elbaum, A. Koesnandar, and B. Myers. Scenario-Based Requirements for Web Macro Tools. Proc of VL/HCC 2007,

[*i] Tuchinda, R., Szekely, P., & Knoblock, C. A. (2008). Building mashups by example. *Proceedings of IUI 2008*.

[*s] Wing, J. M. 2006. Computational thinking. *Commun. ACM* 49, 3 (Mar. 2006), 33-35.

[*a] Wong, J., & Hong, J.I. (2007). Marmite: Towards end-user programming for the web. *Proc of CHI '07.*

[*b] Wong, J., & Hong, J.I. (2008). What Do We “Mashup” When We Make Mashups?. *Workshop on End-User Software Engineering IV* (WEUSE IV) in *Proc. of ICSE 2008.*