AtomsMasher: Personal Reactive Automation for the Web

Max Van Kleek[1], Paul André[2], Mikko Perttunen[3],
Michael Bernstein[1], David Karger[1], Rob Miller[1] and m.c. schraefel[2]

[1]CSAIL, MIT
32 Vassar St.
Cambridge, MA, 02139, USA
max@mit.edu

[2]Electronics and Computer Science
University of Southampton
SO17 1BJ, UK
pa2@ecs.soton.ac.uk

[3]Dept. of Electrical
and Information Engineering
University of Oulu, Finland
mikko.perttunen@ee.oulu.fi

ABSTRACT

The rise of "Web 2.0" has seen an explosion of web sites for the social sharing of personal information. To enable users to make valuable use of the rich yet fragmented sea of public, social, and personal information, data mashups emerged to provide a means for combining and filtering such information into coherent feeds and visualizations. In this paper we present AtomsMasher (AM), a new framework which extends data mashups into the realm of context-aware reactive behaviors. Reactive scripts in AM can be made to trigger automatically in response to changes in its world model derived from multiple web-based data feeds. By exposing a simple state-model abstraction and query language abstractions of data derived from heterogeneous web feeds through a simulation-based interactive script debugging environment, AM greatly simplifies the process of creating such automation in a way that is flexible, predictable, scalable and within the reach of everyday Web programmers.

Keywords: toolkit, programming language, end user automation, rdf, context aware, mashup, reactive behaviors.

INTRODUCTION

Despite the ever-increasing quantity of potentially valuable information brought to us by the Web and other channels of digital communication, our limited time and energy necessitates that some be neglected. In this paper, we explore the potential for *web-based personal reactive behaviors* to help us both cope with this information, and utilize it, saving us time and effort. These behaviors let users specify simple actions for responding to incoming information, leveraging web information sources for increased adaptivity. Some examples are as follows:

- *contextual* information such as location – remind me to call my mother when I get home;
- bridging channels of communication – translating texts I send from my phone to Facebook status updates;
- *make my apps schedule aware* – filter my e-mails and documents pertaining to my current meeting;
- queries across multiple sources –find out if any bands I like are playing tonight, and which of my friends that like similar music are free to come.

While programmers could write custom applications to realize each of these behaviors, doing so would require repeatedly solving the same problems a number of times from scratch. Specifically: the transformation of raw data from web APIs, the identification of items of interest within that data, and articulating conditions for action.

AtomsMasher addresses these needs by providing a single, consolidated representation of data aggregated from arbitrary sources on the web, an object relation mapper (ORM) and query language to simplify access to this representation, and a rule engine for efficiently determining when behaviors should be run. This tool is aimed at a similar audience to that of most mashups and end-user automation, the "*growing groups of web designers and developers familiar with scripting languages*"[1], reducing the barrier to let users "jump in" and specify their behaviors in as simple and efficient a manner as possible, interactively experimenting and debugging as they go along.

SYSTEM DESCRIPTION

AtomsMasher consists of data management components, a programming language interface, and a graphical interface for behavior development and simulation/debugging. In order to simplify installation and access by web developers, AM is deployed as a self-contained Firefox plug-in that can be pulled up with a hotkey as a sidebar or full screen interface (see figure 1). The role of each of the components is described.

Data modeling and rule engine

The "engine" of AM consists of code for data acquisition (fetching, filtering and storage) and behavior management (a rule chainer). The former is responsible for pulling

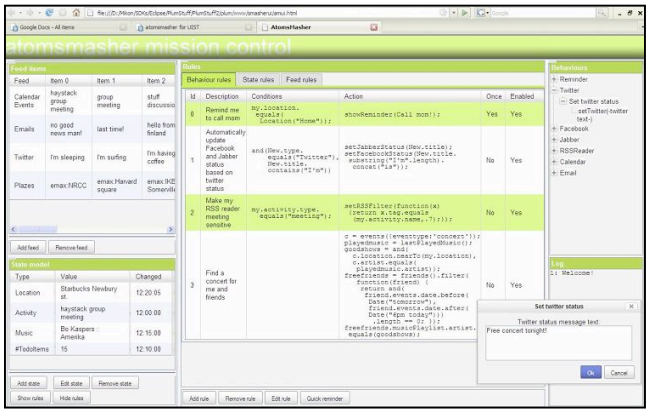

Figure 1. The fullscreen AtomsMasher interface, displaying: rules, query and state variables, splitters, feed rules.

structured information from web data sources (i.e., Atom or RSS feeds and REST-based APIs), and transforming this information into instances of a common, open ontology that can be extended by users. Due to the highly varied information returned in often inconsistent form from Web APIs, entries processed by AM's generic feed parser are handed over to hand-written feed-specific transformation plug-ins called *atom splitters*, which are responsible for distilling elements from obtained form into the AM ontology. Once ontology elements have been derived from new feed items, they are persisted in a knowledge base (KB) that is kept locally in the user's Firefox profile. This KB acts as the world model for evaluating behavior antecedents and actions. Since behaviors can only access instances in the KB, the KB serves the important role of decoupling data sources from use. To reduce the need for users to write splitters, we have constructed a repository where authored splitters can be uploaded for others to use; when new data sources are added, AM checks this repository for new feeds prior to asking the user for help.

AM's rule engine is modeled on a simple forward-chainer, with AM-specific extensions. To speed up trigger identification, antecedents are compiled into SQL queries that get pushed down into the underlying SQLite engine of the triple-store DB. The chainer keeps track of its history of firings to avoid repeated triggerings with previously fired bindings.

Programming language interface

To maintain familiarity to web programmers, AM extends JavaScript (JS) to provide mechanisms for easily specifying rules and referring to stored entities in the KB. To this end, we combined the roles of a object relation mapper (ORM) with that of a query language like jQuery (jQuery.com) to create AM query variables (AMQVs) representing entities in the KB as JS objects, i.e., query sets. Applications of operators to AMQVs evaluate to new AMQVs with expanded or restricted sets of values, representing all *non-false* values resulting from mapping the operator over each of the items in the original set. The end result is that queries to the KB resemble plain expressions involving JS objects.

Besides query variables, AM maintains a global JS object that functions as a persistent state model that can be driven by new incoming data items. This state model is used to make convenient the tracking of aspects of the user's state, such as their location or current activity. The state model is updated whenever a behavior that assigns to the state variable object is fired.

Rule simulation and debugging environment

AM's UI provides two facilities to help users debug their behaviors. The first, inspired by the Pig pen [3], is behavior simulation, in which AM generates a set of example "situations" for which a particular behavior antecedent would fire, and the resulting action it would take. The second is an easy-to-inspect history of firings for debugging past actions.

SCENARIOS

We provide an implementation of the last scenario from the introduction (friends and music preferences), to illustrate how suitable feed or state rules, and behavior or query rules can be written in AM syntax. Implementations for the other scenarios can be found at http://snipurl.com/2wto4. This illustrates an ambitious use of AM to query across potentially hundreds of data sources—i.e., our friend's calendars. In this rule, triggered remotely from our phones via a text message, we isolate a set of concerts the user may wish to attend, by finding the intersection between concerts in the area and artists on the user's recently played (Last.fm) list. The script then selects friends of the user who have no appointments scheduled, and who have recently listened to any of the artists that are in concert. The list of such friends and concerts are returned in a reply.

```
// run if we get an incoming text message from
// ourselves with command "goodshows"
when(New.type==Message.Text&&New.contents=="goodshows") {
 cs = events({type:'concert', start:Now.day()});
 playedmusic = recentlyPlayedMusic();
 freefriends = friends().filter(function(friend) {
  return friend.cal.freebetween(Today.hour(18),Tomorrow);
 });
 goodshows = cs.filter(function(c) {
  return c.location.nearTo(my.location,miles(2))&&
         playedmusic.artist.eq(c.artist);
 }); // now reply to original message
 reply(New, freefriends.
            musicPlaylist.artist.eq(goodshows.artist));
}
```

CONCLUSIONS AND FUTURE WORK

By combining heterogeneous sources of personal and social information available on the web, AtomsMasher enables the construction of rich, context-aware reactive behaviors. In particular, AM demonstrates the simplicity and expressive power gained in deriving a unified data representation from heterogeneous web feeds, and the use of simple language extensions to make these representations natural to web developers already familiar with Javascript.

Ongoing work spans two main areas: language design and user interface design. We will identify the most useful types of rules and use them to reconsider rule syntax. Our user interface will draw on work in visual programming and PBD to simplify the initiation, understanding, and scrutability of actions. In addition, we will be investigating the sharing of behaviors (similar to the Co-Scripter wiki [2]) as well as privacy and security implications of our work. We are currently planning an in-depth evaluation into the utility and unexpected applications of AM, as well as the usability of our rules and predicates, with a public release in the fall.

REFERENCES

1. Hartmann, B., Wu, L., Collins, K., Klemmer, S.R. Programming by a sample: rapidly creating web applications with d.mix. UIST'07.

2. Leshed, G., Haber, E., Lau, T., Cypher, A. Co-Scripter: Sharing 'How-To' Knowledge in the Enterprise. GROUP'07

3. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins A., "Pig Latin: A not-so-foreign language for data processing", ACM SIGMOD 2008.