Swashup: Situational Web Applications Mashups

E. Michael Maximilien
IBM Almaden Research Center
San Jose, CA USA
maxim@us.ibm.com

Ajith Ranabahu
Wright State University
Dayton, OH USA
ajith.ranabahu@gmail.com

Stefan Tai
IBM T. J. Watson Research Center
Hawthorne, NY USA
stai@us.ibm.com

Abstract
Distributed programming has shifted from private networks to the Internet using heterogeneous Web APIs. This enables the creation of situational applications of composed services exposing user interfaces, i.e., mashups. However, this programmable Web lacks unified models that can facilitate mashup creation, reuse, and deployments. This poster demonstrates a platform to facilitate Web 2.0 mashups.

Categories and Subject Descriptors D. Software [D.3 PROGRAMMING LANGUAGES]: D.3.3 Language Constructs and Features

General Terms Languages, Experimentation

Keywords DSLs, Mashups, Ruby on Rails, Web APIs

1. Introduction
There are two paradigm shifts occurring on the Web that are changing the way software is developed. The first is the increasing availability of Web APIs (or Web services) in the form of Representational State Transfer (REST), Atom Publishing Protocol (APP), and SOAP services, as well as RSS and Atom data services. Indeed, the proliferation of these Web APIs have resulted in various composed services with UIs, or mashups, which provide solutions to very specific and narrow problems. The second paradigm shift is a movement to increasingly program Web applications using dynamic programming languages and frameworks, e.g., JavaScript with AJAX, Ruby with Ruby on Rails (RoR), Smalltalk with Seaside, and PHP with Zend framework.

In many ways these two paradigm shifts are complementary since they essentially help realize the vision of a programmable Web or the Web as programming substrate. While many available frameworks are facilitating aspects of Web programming, none are focused directly on facilitating services mashups and helping deal with the common distributed systems issues that arise.

In this poster we present a DSL and platform for services mashups that can alleviate some of these issues. In particular, our DSL (1) allows for a common interface representation among service types, (2) gives a uniform model for service data and service operations' interactions, and (3) a common high-level for mashups. We demonstrate our DSL and platform's implementation using Ruby and the RoR framework.

2. Architecture Overview
Our architecture extends the RoR architecture with a new DSL, supporting libraries, as well as associated platform models, services, and end-user applications. Using the Swashup Web UI tools, end users create, edit, and deploy a Swashup project which contains the necessary information for describing the services to be mashed up as well as the mashup information. Users can also share these mashups for reuse and extensions. Using the Swashup platform services, a project is deployed as a complete RoR Web application with all necessary service proxies, models, and initial views for each mashup. A user typically uses the Swashup tools to iterate over the mashups and to refine and customize the UI components. Swashup enables agile situational Web applications and mashups creation, sharing, deployment, and management. Figure 1 illustrates the architecture as well as the main components and applications.

3. Swashup DSL
Before introducing our language we give a general high-level definition of DSLs and briefly note their value.

3.1 What are DSLs?
A domain-specific language (DSL) is a ‘mini’ language built on top of a hosting language that provides a common syntax and semantics to represent concepts and behaviors in a particular domain. In general, using or designing a DSL helps achieve the following goals: (1) abstraction by enabling programming at a higher level; (2) terse code; (3) simple and natural syntax; (4) ease of programming; and (4) code generation by translating the DSL statements at runtime.
3.2 Language Overview

Briefly, for us a mashup is a composition of services (REST, APP, SOAP, RSS, or Atom) that contains information on how to represent the service API and data and how to combine the services and present a user interface. Our language includes primitives for:

1. Defining the service’s data and api and enabling a uniform model and representation of various types of services.
2. Data mediation, which represents data manipulations such as conversions, transformations, and combinations.
3. Service protocol mediations or wiring, specifies the choreography of the services.

In the Swashup DSL [1], we directly represent in the syntax, the concepts necessary to cover the three main components of our conceptual model for mashups: (1) data and mediation; (2) service APIs, their protocols, and choreography; and (3) a means to create customized UIs for the resulting mashups. Views are added by generating and tailoring RoR’s RHTML templates. Our language comprises:

- **data** describes a data element used in a service. A data element corresponds to an XML schema complex type.
- **api** gives a complete description of a service’s interface including operation names, parameters, and data types.
- **mediation** describes the transformation of one or multiple data elements to create a new one.
- **service** binds a service api with a concrete service by indicating the service’s type.
- **recipe** constitutes a collection of services and mashups. A recipe also includes views for each of the mashup’s

**wiring.** Some views are automatically generated and others are customized by the user.

- **mashup** is a composition of one or multiple services. It comprises a collection of mediates and wirings.
- **mediate** invokes a mediation declaration with instances of the data elements to mediate.
- **wiring** which comprises two levels of granularities of connecting the services that are part of a mashup. (1) :protocol is a top-level structure of a mashup. It represents one or multiple operation wirings and steps invocations; and (2) :operation is the wiring of one or multiple services’ operations. Operation wiring includes the ability to invoke services’ operations in an asynchronous fashion by automatically setting up callbacks.
- **step** constitutes one atomic step in a protocol mediation. A step can be invoked multiple times as part of a protocol wiring. A step is invoked by the step’s name as a method call.

- **desc** to provide short English descriptions for constructs (enables searching).
- **tag** and **tags** allow users to annotate the various constructs.

3.3 Value of DSL

Our DSL enables mashup programming at a higher-level of abstraction than frameworks supporting Web application programming. With our DSL, we explicitly represent the main activities that a mashup programmer needs to address as specified in our conceptual model. Uniform design for mashups and integration into RoR, which enables adding rich Web views including AJAX views.

4. Conclusion

Services mashups are an essential part of the current paradigm shift occurring on the Web. One of the main consequences is a change in the way Web programming is approached and achieved. Our Swashup platform is at the forefront of this shift. The mashup DSL and platform presented in this paper is an important aspect of this platform. The DSL is essentially the glue code that enables composition of Web services while also giving some structure to the tasks of a mashup designer. Our platform’s implementation uses the RoR framework which gives us a rich substrate to enable rapid and sophisticated mashups.

References