10-2007

Swashup: Situational Web Applications Mashups

E. Michael Maximilien

Ajith Harshana Ranabahu
Wright State University - Main Campus

Stefan Tai

Swashup: Situational Web Applications and Mashups

E. M. Maximilien[1], A. Ranabahu[2], and S. Tai[3]

[1]*IBM Almaden Research Center,* [2]*Wright State University, and* [3]*IBM T.J. Watson Research Center*

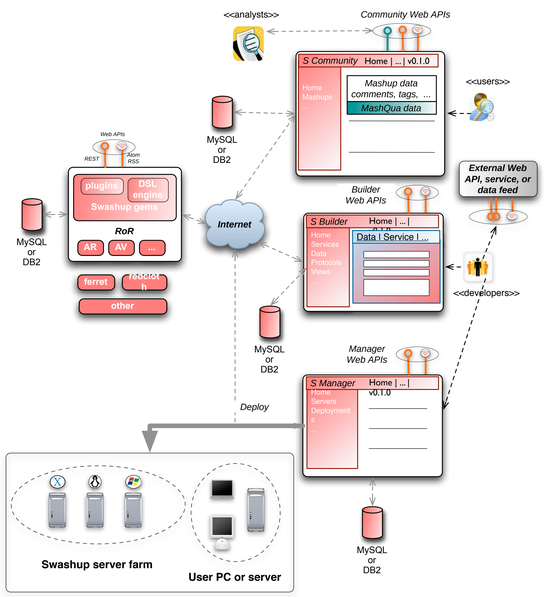
Abstract

Distributed programming has shifted from private networks to the Internet using heterogeneous Web APIs. This enables the creation of composed services exposing user interfaces, i.e., *mashups*. However, this programmable Web lacks unified models that can facilitate mashup creation and deployments. This poster demonstrates a platform to facilitate Web 2.0 mashups using a mashup domain-specific language (DSL) and a collection of Web 2.0 tools and APIs to facilitate writing, sharing, and deploying mashups created in the DSL. The platform leverages and is implemented in Ruby on Rails.

Motivation and architecture

- Key problems in building Web 2.0 mashups
  - Heterogeneous data and service types (e.g., REST, RSS, Atom)
  - Different invocation sequences requirements for each service
  - Need to cache data for performance
  - Need to invoke services asynchronously
  - Need for rich user interfaces (e.g., AJAX-style interfaces)
- Ruby on Rails (RoR) framework can be used to address issues

- Domain-Specific Languages allows
  - Define concepts at domain-level
  - High-level declarative programming
  - Reduce code to what is necessary
- Metaprogramming to translate DSL into underlying RoR code
- DSL BNF is generally useful (i.e., implemented in other languages)



Domain-specific language (DSL)

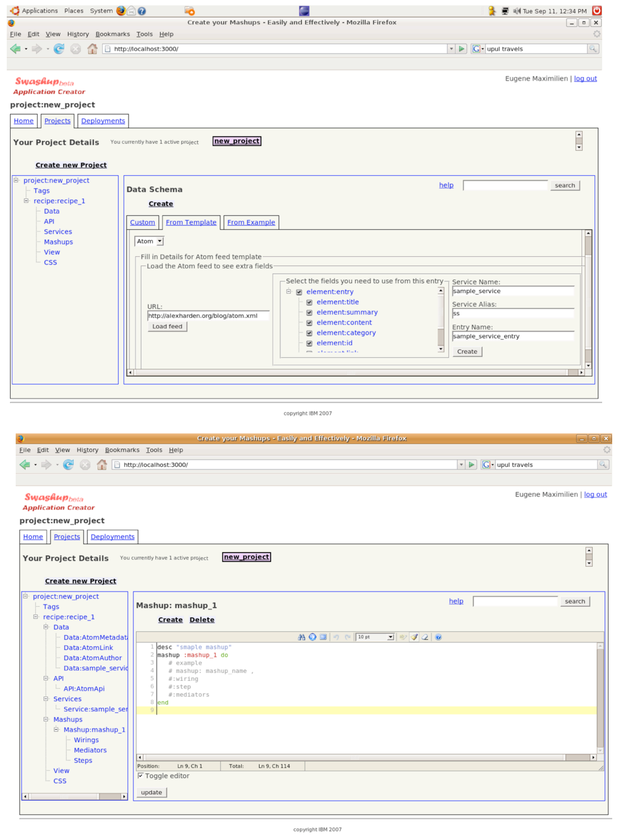
- Takes advantage of Ruby's support for creating DSLs
- Allows developers to program at higher-level of abstraction
- DSL includes primitive to represent aspects of any mashups
- Complete Backus-Naur form for the mashup DSL available
- DSL constructs are indexed for search and reuse
- Each **recipe** do
  - **data** for each API data element
  - **api** for each service (only methods used)
  - **service** to bind to endpoint
  - data **mediation**s for data transformations
  - **mashups** do
    - ★ **step**s or interactions between services
    - ★ **wiring**s or interactions with end-users
    - ★ **view**s for each wiring
  - Iterate
  - Each DSL construct supports **desc** and **tag** or **tags**
- Expose any **data** as REST, RSS, or Atom APIs
  - Deploy and share
- Examples
  - YouTube's APP service and Flickr's REST API mashup
  - Google News Atom feed with Stanford's events RSS feed
  - Many more ...

youtube_flickr_recipe.rb

```
desc "Recipe for YouTube APP and flickr.com REST mashups"
tags ['mashup', 'rest', 'app']
recipe :youtube_flickr_rest_recipe do
  desc "Flickr's error response part"
  data :FlickrErr do
  [...]
  desc "Flickr's <photo>...</photo> response part"
  data :FlickrPhoto do
    xml_name :photo
    member :id, :integer, :xml_attribute, :desc => 'unique ID for photo'
    member :server, :integer, :xml_attribute,  :desc => 'the server ID'
    member :title, :string, :xml_attribute, :desc => 'title for the photo'
    member :ispublic, :integer, :xml_attribute, :desc => 'integer encoded boolean'
    [...]
  end

  desc "Flickr's API"
  tags ['api', 'rest', 'flickr', 'yahoo']
  api :FlickrApi do

  desc "Flickr's REST service definition"
  tags ['flickr', 'rest', 'service']
  service :flickr_rest_service,
          :type => :rest, :api => :FlickrApi, :alias => :flickr,
          :endpoint => 'http://api.flickr.com/services/rest'

  desc "YouTube's Entry for an Atom or APP data feed"
  data :YouTubeAtomEntry do
    xml_name :entry
      member :id, :string, :xml_text
      member :published, :datetime, :xml_text
      member :updated, :datetime, :xml_text
      member :title, :string, :xml_text
      member :content, :string, :xml_text
      member :author, :AtomAuthor, :xml_object
      member :links, :AtomLink, :xml_object_array
  end
  [...]
  desc "Shows both YouTube top rated videos and Flickr's interesting photos on same page"
  mashup :combined_mashup do |you_tube, flickr|
    desc "Shows YouTube and Flickr along side each other"
    wiring :view do
    end
  end

  desc "Shows all current Flickr insteresting photos"
  view :flickr_interesting_photos, :mashups => [:simple_mashup],
       :content => %{<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
  <h2>Simple view for displaying the results of Flickr's flickr.interestingness.getList R
  <body>
    <% interestingness = @flickr.flickr_interestingness_getList :per_page => 15 %>
    <h3>Interesting photos (<%= interestingness.flickr_photos.total %>)</h3>
    <table>
```

Platform tools

Swashup mashup creator



Conclusion and future

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. Implementation using RoR gives us a rich substrate to enable rapid and sophisticated mashups. Additionally Ruby's excellent support for DSL results in a syntax that is natural and easy to read. Future considerations include:

- Support JSON for data construct
- SOAP/WSDL services
- Manager application to manage deployments (local or remote)
- Simple wizard for service workflows
- Simple wizard for data mediation generation
- Support for advanced API metadata via microformats
- Available on **IBM's alphaWorks** services **December 2007**

References

E. M. Maximilien, H. Wilkinson, N. Desai, and S. Tai. **A Domain Specific-Language for Web APIs and Services Mashups**. In A. Dan and S. Dustdar, editors, In Proceedings of 5th *International Conference on Service Oriented Computing (ICSOC)*, LNCS 4749, pages 13–26, Vienna, Austria, 2007. Springer-Verlag.