What can enterprise mash-up technologies do for your?

Practical learning by looking at WebSphere sMash

Johan Eltes
johan.eltes@callistaenterprise.se
www.callistaenterprise.se
My wake-up call

• Customer requirements:
  - Web application to search for contact information on employees and locations of the enterprises sites located in the enterprise LDAP-based directory

How would an Enterprise Java Architect approach this?
Enterprise Java Architect Approach

- Follow the prescribed reference architecture
  - Presented at last years Cadec

- Business Component
  (Core Conf Mgmt Structure)

- Enterprise Archive
  (EAR)

- Composite Application
  (WAR)

- Composites
  (For re-use in composite applications)
  (JAR)

- My Gui Ear

- My Ws Ear

- Web

- Portlet

- WS

- Web Comp

- Srv

- Types

- Schemas
The customer proposed a different approach

- Supplementary requirements clarified
  - The information displayed in search result should be up-to-date with yesterday's information in LDAP master (or better)
- Proposal
  - Give every information object in LDAP (not structural nodes) an HTTP URL that produces XML output
  - Produce one - or as few as practically possible - HTML files with all these URLs, so that our search engine can index them
  - Use the search engine client to search for information in our people directory and present the output of the selected URL
- Background
  - The customer uses Websphere Omnifind to search-enable enterprise content
Some reflexions

- There are many generic tools made for the web
- Customers are used to the web architecture
- We do find information using Google
- The solution was “quick and dirty” but still produced and re-used services in very short time and with very low risk
- Enterprise Java architects may need to extend their mind- and tool-set...

Welcome to the new world of Enterprise Mash-ups!
Cornerstones for Enterprise Mash-ups

- Solution Architecture
  - Produce new web resources by consuming existing.
    - Q&D(O) over Strategic & Engineered
    - 90% Integration logic, 10% Core business logic
    - Web concepts (REST services, browser scripting)
- Programming model
  - Generic formats with open binding.
    - Content before structure
  - Agile, short roundtrips
- Solution Category
  - Primarily for information consumption
  - Situational applications
- Lifecycle
  - Lifecycle not necessarily formal. Anyone depending of mash-up services is part of the game. Web URLs usually stay sable for a couple of years...
- Tooling
  - Even the tooling is a mash-up
    - Scripting, Java, resource adapters, integration middleware...
WebSphere sMash technology

- JVM-based application server used for developing and running mash-ups
- UI development and re-use toolkit for JavaScript based on Dojo JavaScript library
- Solid component management
  - The product is componentized
  - Solutions can be componentized
- It is a very lightweight container
  - bootstraps its features from an internet repository
  - Installation: Download, unzip, start
  - Initial size is 1.8 MB. Incrementally grows to 250 MB
- It is a very agile environment
  - Develop in the container
  - Basically Zero restarts: edit, refresh, debug...
General architecture of an enterprise mash-up

**Browser URL**

- **Browser URL**

**Ajax calling REST Service**

- **Ajax calling REST Service**

**Produce HTML with ajax from template**

- **Produce HTML with ajax from template**

**REST-based mash-up service (JSON)**

- **REST-based mash-up service (JSON)**

**Assemble Flow, Groovy, PHP, Groovy Templates**

- **Assemble Flow, Groovy, PHP, Groovy Templates**

**JavaScript, Dojo-baserat ramverk**

- **JavaScript, Dojo-baserat ramverk**

**Groovy, PHP, Java**

- **Groovy, PHP, Java**

**Accessed by Connectors and Kickers (inbound events)**

- **Accessed by Connectors and Kickers (inbound events)**

**Back-end resources**

- **Back-end resources**

**Legacy with JMS**

- **Legacy with JMS**

**Existing databases with JDBC access**

- **Existing databases with JDBC access**

**CRUD interface**

- **CRUD interface**

**search-services**

- **search-services**

**JDBC connector**

- **JDBC connector**

**RESTFul services**

- **RESTFul services**

**Assemble Flow, Groovy, PHP, Groovy Templates**

- **Assemble Flow, Groovy, PHP, Groovy Templates**

**Accessed by Connectors and Kickers (inbound events)**

- **Accessed by Connectors and Kickers (inbound events)**

**Back-end resources**

- **Back-end resources**

**Legacy with JMS**

- **Legacy with JMS**

**Existing databases with JDBC access**

- **Existing databases with JDBC access**

**CRUD interface**

- **CRUD interface**
def onList() {
    // Http GET to http://localhost:8080/resources
    // Writes a list of all resources to the response (XML or JSON)
}

def onCreate() {
    // Http POST to http://localhost:8080/resources/registrations
    // Create new resource. Sets the location header of response to
    // http://localhost:8080/resources/registrations/<id of new resource>
}

def onRetrieve() {
    // Http GET to http://localhost:8080/resources/<id>
    // Writes the requested resource to the response (XML or JSON)
}

def onUpdate() {
    // Http PUT to http://localhost:8080/resources/<id>
    // No response
}

def onDelete() {
    // Http DELETE to http://localhost:8080/resources/<id>
    // No response
}
Special support for Database Resources

- Very similar top Grails
- Builds on metadata and coding-by-convention
- Basically creating a RESTful api on top of database tables
- Dojo Grid + Detail Form also by convention

```json
{  
  "fields": {  
    "name": {  
      "label": "Namn",  
      "required": true,  
      "type": "string",  
      "description": "",  
      "default_value": "",  
      "max_length": 50  
    },  
    "cadec": {  
      "label": "Cadec",  
      "required": true,  
      "type": "boolean",  
      "description": "",  
      "default_value": ""  
    }  
  }
}
```

Generates / Alters new database
Maps legacy

Zero Resource Model: Access using metadata
Demo - ZRM

- Create New Application
- Add Resource Metadata (json format)
- Add initial data fixture
- Synchronize database
- Create Groovy handler for resource

- Use RESTful api from browser
  - [http://localhost:8080/resources/registrations/1](http://localhost:8080/resources/registrations/1)
- Query-by-convention also available
ZRM – Model-driven UI

- Generate UI descriptor from ZRM model
  - Adds layout attributes, but still metadata on JSON format
- Descriptor used by UI builder
  - For CRUD user interfaces the “Rails” way, but with services
How to call a ZRM RESTful service from a Groovy Mash-up script?

Type registrations = TypeCollection.retrieve('registrations')

List<Member> allRegistrations = registrations.list()

allRegistrations.each { registration -> println registration.email}

List<Member> registrationsFromAcme = registrations.list(email__endswith: 'acme.com')

Member newRegistration = registrations.create(name:'Kalle', company:'Acme',...)
Flows – creating mash-ups graphically

- New email events should trigger creation of new resources
  - Use the POP3-kicker
Sample mash-up flow

REST-based mash-up service (JSON)

CRUD Resource

Mail-box

Database

receiveMailMessa

BuildRegistration

PostToResource
Client-side mash-ups in sMash

- **iWidgets**
  - Portal in the client
  - Same idea as collaborating portlets
  - Self-contained and packaged components
  - Interacts with server resources via ajax/JSON
  - Interacts with other iwidgets via client-side events
  - Multiple iWidgets composed and linked in webpages
- Would be like swing widgets, unless...
Supported by a solid multi-layer component model

- The sMash component model allows any application to be reused as a component
- And we yield new services as we go
Component and re-use in sMash

- All components have the same internal layout
- Dependencies are managed by Ivory

http://localhost:8080/ParseRegistration
1 inherits 2 that inherits 3
Just like a classloader, but får all resources
Summary sMash

- Developed by IBM in a way very much unlike IBM
- Full access to Project Zero development process and development products
- Based on what seems to be the coming “main stream” technologies for “Web 2.0” and Cloud Computing
- A strange mix of high-level and techie (full http protocol access)
- A slick seamless environment for small teams and situational applications
- Lack of momentum – very small community
- Coding in the browser?
  - Great debugger, no intelicense
What about Open Source?

- Few products that define themselves as Mash-up infrastructure
- WSO2 Mashup Server
  - Limited to JavaScript
  - JavaScript lacks libraries for server side integration
  - Primary focus is on accessing and publishing WebServices with JavaScript
- Several Mash-up systems in the cloud
  - Yahoo Pipes probably best known
Mash-up development skillset

- REST architecture (Resource model -> HTTP)
- JavaScript and JavaScript widget frameworks
- Server side dynamic languages (Groovy, PHP)
- Major REST-based application protocols (ATOM, RSS)
- Web Security
Conclusions

- WebSphere Smash
  - A tailored infrastructure boosts RESTFul development
  - It represents a balanced model for agility and engineering
  - Agility and integration at this level has a price: Vendor lock-in (OS may catch-up)

- WSO2 Mashup Server

- General – Enterprise Mashups
  - Q&D(O) – Probably!
  - Intersects with integration, process and information management middleware
  - Unique in its dedication to the architecture of the web