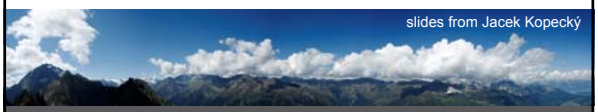
STI · INNSBRUCK

Semantic Web Services

Web 2.0 and RESTful Web Services

Lecture V – 2nd April 2009

Dieter Fensel



Where are we?

STI · INNSBRUCK

#	Date	Title
1	5th March	Introduction
2	12th March	Web Science
3	19th March	Service Science
4	26th March	Web Services (WSDL, SOAP, UDDI, XML)
5	2nd April	**Web 2.0 and RESTful services**
6	23rd April	WSMO
7	30th April	WSML
8	7th May	WSMX
9	14th May	OWL-S and others
10	28th May	WSMO-Lite, MicroWSMO
11	4th June	SWS Use Cases
12	18th June	seekda: the business point of view
13	25th June	Mobile services
14	2nd July	Exam Preparation

Overview

STI · INNSBRUCK

- Motivation
- Technical solutions
  - Structure of RESTful services
  - Structure of current, "REST-like" services
  - Overview of REST, HTTP, JSON
  - Authentication considerations
  - Describing RESTful services
    - Text, WADL

STI · INNSBRUCK

Motivation

Web 2.0 & RESTful Services

- Early Web was about information
- Then it grew with commerce
- Now it becomes interactive
  - AJAX user interfaces
  - Web application APIs
- Machine-oriented Web resources
  - Web of Services

What is Web 2.0?

Depends who you ask, but mainly:

1. Read-write Web
2. Democratized/Social Web
3. Mature Web
4. **Programmable Web**

Why Programmable Web 1

- Read-write Web needs user participation
  - Web sites need the data from users
  - E.g. flickr, del.icio.us, facebook
- Make it easy for users to contribute
  - Many applications for uploading pictures to flickr
  - Bookmarklets to add a del.icio.us bookmark
  - Notifying facebook of blog posts, new pictures, tweets

Why Programmable Web 2

- Democratized and Social Web
- Web sites don't treat users as consumers
  - Prosumer = producer + consumer
  - Pushing boundaries: privacy, trust, IPR
- The users own their data
  - Export APIs for application sync, backup, migration
  - E.g. calendar in vCal, friends in FOAF
  - Without screen scraping

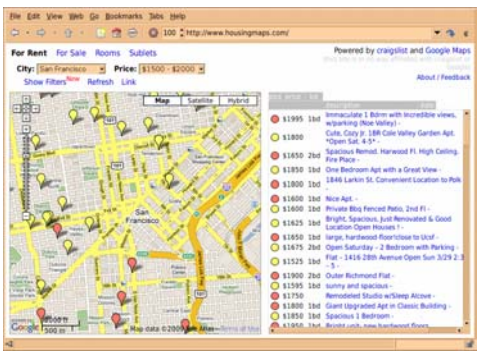
Why Programmable Web 3

- Mature Web: things really work now
  - AJAX, fast pipes
  - E.g. Google maps: interactive vast images
- AJAX makes the browser smarter
  - The AJAX code accesses the Web server as a **service**

Programmable Web

- Javascript can do more
  - Update a part of a page on the fly, without full reload
- Desktop apps can work with Web apps
  - Flickr uploadr, calendar update/sync
- Web apps can work with others
  - LinkedIn can import your Facebook friends
  - Facebook can import your Dopplr trips
- Mashups: easy service composition

Mashup: Housingmaps.com



Web APIs & Services

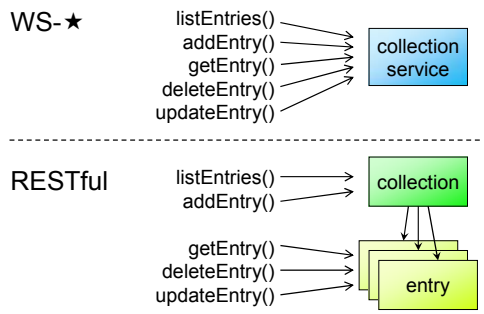
- Data providers
  - Google maps, Geonames, phone location…
  - Microformats (vcard, calendar, review…)
  - Data feeds
- Functionality
  - Publishing, messaging, payment…
- Reverse APIs: Human computation
  - Amazon Mechanical Turk, ESP game…
- Web as a Platform

Technical Solutions

RESTful WS Definition

- A RESTful Web service is:
  - A set of Web resources
  - Interlinked
  - Data centric, not functionality centric
  - Machine oriented
- Like Web applications, but for machines
- Like WS-*, but with more Web resources

WS-★ vs REST: Collection



Technologies

- **REST:** the architectural style of the Web
- **HTTP:** the basis
- XML, JSON, Microformats for data exchange
- Atom/RSS, AtomPub
  - Feeds, publishing, syndication
- Javascript programming the browser, AJAX

REST Overview

- Architectural style
  - Set of architectural constraints
  - Not a concrete architecture
  - An architecture may adopt REST constraints
- HTTP the main implementation of REST
  - Should not be confused

WWW Requirements

- Simplicity, low barrier of entry
- Extensibility
  - To allow growth past simplicity
- Distributed hypermedia
- Anarchic scalability
- Independent deployment
  - Coexistence of old and new
- Human-oriented optimizations
  - Latency, usability

REST Ingredients (1)

- Client-server
  - Separation of concerns
  - Networking
  - Independent evolution
- Layering
  - Composability
- Stateless communication
  - Scalability, reliability
  - Resources are stateful (stateless = no session)

REST Ingredients (2)

- Uniform Interface
  - Simplicity (vs. efficiency)
  - Large-grained hypermedia data transfer
  - Example: Create, Retrieve, Update, Delete
- Caching
  - Efficiency, scalability
  - Consistency issues
- Code-on-demand
  - Extending client functionality

RESTfulness

- Criterion for Web applications:

1. Using the uniform interface
   - Using safe interactions
2. Hypertext drives the application
3. Cacheability
4. Statelessness
   - Often over- generalized, misunderstood

HTTP Quick Overview

- Addressing resources with http:// URIs
- Request/response, four methods:
  - GET – information retrieval (*safe*)
  - POST – adding data for processing
  - PUT – replacing resource content
  - DELETE – removing resource
- Commonly, a response contains links to further resources

Safe Interactions

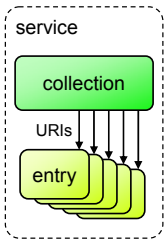
- No obligations incurred
- Safe → idempotent (not vice versa)
- HTTP GET is safe
- Can be used opportunistically
  - E.g. link pre- fetching, crawling
- Unsafe does not mean dangerous
- Applications with unsafe GETs may get undesired effects

Example RESTful Services

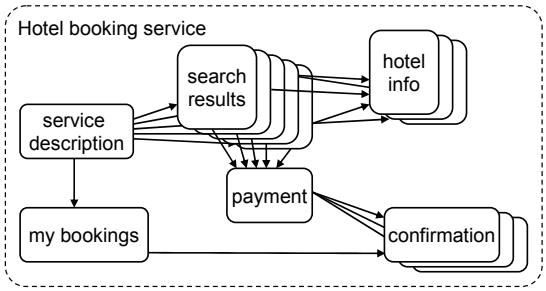
- Atom Publishing Protocol
- Hypothetical hotel booking service
- Flickr API

Example: AtomPub Protocol

- Atom: a collection (feed) of entries
- Collection resource
  - GET: list entries
  - POST: add an entry
- Entry resource(s)
  - GET: retrieve
  - PUT: replace/update
  - DELETE: delete



Example: Hotel Booking



Hotel Service Workflow

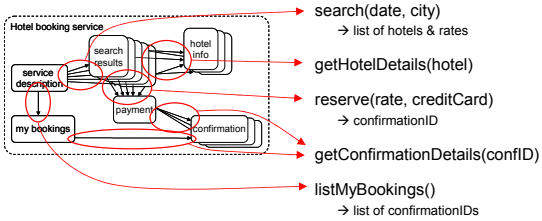
1. Retrieve service description
2. Submit search criteria according to description
3. Retrieve linked details of interesting hotels
4. Submit payment details according to selected rate description
5. Retrieve confirmation of booking

2b. Retrieve list of user's bookings

Hypermedia

- Links between resources
  - Not only <a href="…">, also forms
    - Also applies to machine-oriented data: XLink, XForms
  - Allow further actions
    - That means all of GET, POST, PUT, DELETE
  - GETs can be bookmarked
- Client navigates the graph
  - May discover new links at every step
  - Can always do GET retrieval without worries
  - Can go back in browsing/state history
    - Which does **not** undo the action(s)

Hypermedia → Operations



nouns vs. verbs

Hypermedia → Operations

- Hypermedia is a UI paradigm
- In REST, hypermedia is the "engine of application state"
- Programmatic clients work with operations
- Operation available when the client has the link

Example: Flickr

- Example operations (methods):
  - flickr.photos.addTags
  - flickr.photos.delete
  - flickr.contacts.getList
  - flickr.photos.comments.editComment
  - …
- HTTP GET or POST
  - @ http://api.flickr.com/services/rest/?method=*method*&parameters
- Special authentication method

Flickr API Authentication

- Application needs an API key
  - API key requested by application developer
  - Application has a shared secret with Flickr
- Every method needs API key
- Application lets user log in, gets auth token
- Authenticated methods need auth token and signature
  - Signature uses shared secret and all parameters

More at http://www.flickr.com/services/api/misc.userauth.html

Authentication in General

- Most "REST-like APIs" have proprietary authentication mechanisms
- Plus mandatory API keys for applications
  - But a user with a browser needs no API key
  - Web sites cautious of overload, abuse
  - Conflict between openness and control

Flickr Not RESTful

- The client must know all methods beforehand
  - Out-of-band information drives the interaction instead of hypertext
- Non-standard authentication, tokens in URI
  - More out-of-band information on forming URIs
- URIs with method name identify verbs (operations), not nouns (data)
  - Instead of "DELETE URI <photo>" it has "POST *nothing* to URI <delete/photo>"

Not RESTful, So What?

Disadvantages of not being RESTful:

1. Client specific to an application
   - A different photo app → client rewrite
2. No generic client can use the API
   - Cannot navigate with your browser from a different Web site directly to add a contact
   - Preventing seredipitous reuse

Simply, the API not "on the Web"

Describing RESTful Services

- Machine readable descriptions useful
  - Tool support for developing clients
  - Service discovery
- There's no WSDL for Web apps
  - APIs described mostly in text
  - AtomPub has RFC, Service Documents
  - Resources behind AJAX stay undescribed
- Some machine readable descriptions:
  - HTML links, forms
  - AtomPub service document
  - WADL

Textual Descriptions

- "There's usually an HTML page"
  - E.g. the Flickr documentation
- Such text can be annotated
  - hRESTS, MicroWSMO (Class 10)

Example Description

flickr.contacts.getList

Get a list of contacts for the calling user.

Authentication

- This method requires authentication with 'read' permission.

Arguments

- api_key (Required)
  - Your API application key.
- filter (Optional)
  - An optional filter of the results.

from http://www.flickr.com/services/api/flickr.contacts.getList.html

WADL

- Web Application Description Language
  - Proposed by Marc Hadley (SUN), no real uptake (yet?)
- *Application* (= our Web service)
  - Has *resources*
  - Resources have HTTP *methods*
  - Inputs and outputs can contain liks to resources
- WADL focuses on resources and hypertext
  - As opposed to operations (WSDL)

JSON

- JavaScript Object Notation
  - RFC 4627, json.org
- Simple alternative to XML
  - Used by a growing number of services
  - Especially Javascript- friendly, good for AJAX
- A serialized object or array
  - No namespaces, attributes etc.
  - No schema language (for description)

JSON Example


```
{
   "Image": {
      "Width":  800,
      "Height": 600,
      "Title":  "View from 15th Floor",
      "Thumbnail": {
         "Url":    "http://www.example.com/image/481989943",
         "Height": 125,
         "Width":  "100"
      },
      "IDs": [116, 943, 234, 38793]
   }
}
```


RESTful Services Summary

- Natural development of the Web
  - Rich AJAX UIs need back-end services
  - Sites make it easy for users to contribute (using external tools)
- Real RESTful services integrate well with the Web
  - Not many well-known APIs actually RESTful
- Using HTTP, XML, JSON

Further Developments

- Describing RESTful services
  - hRESTS like a simplified WSDL
  - MicroWSMO adds semantic annotations
- Analyzing Javascript in AJAX sites
  - Could extract information about the services
- Expect more openness

References

- Web 2.0: http://en.wikipedia.org/wiki/Web_2.0
- REST: http://en.wikipedia.org/wiki/REST
  - Includes RESTful Web services
- HTTP: http://tools.ietf.org/html/rfc2616
- JavaScript: http://en.wikipedia.org/wiki/Javascript
- AJAX: http://en.wikipedia.org/wiki/AJAX
- JSON: http://en.wikipedia.org/wiki/JSON
- Atom: http://en.wikipedia.org/wiki/Atom_(standard)
- Mashups: http://en.wikipedia.org/wiki/Mashup_(web_application_hybrid)

Next Lecture

#	Date	Title
1	5th March	Introduction
2	12th March	Web Science
3	19th March	Service Science
4	26th March	Web Services (WSDL. SOAP, UDDI, XML)
5	2nd April	Web 2.0 and RESTful services
6	**23rd April**	**WSMO**
7	30th April	WSML
8	7th May	WSMX
9	14th May	OWL-S and others
10	28th May	WSMO-Lite, MicroWSMO
11	4th June	SWS Use Cases
12	18th June	seekda: the business point of view
13	25th June	Mobile services
14	2nd July	Exam Preparation

Questions?