webservices and infrastructures

- overview, pragmatic proposal, hands on example -

Volker Boehlke
University of Leipzig
webservices & infrastructures

Agenda

1. webservice technologies overview

2. building complex infrastructures

3. a pragmatic approach for a simple infrastructure

4. a small hands on example
Some facts about me

- iba Consulting Gesellschaft (programmer & consultant)
  - reporting software for energy trade
  - real estate asset management

- University of Leipzig
  - member of the scientific staff of the NLP group
  - DSpin/Clarin project (WS, Infrastructure)
webservices & infrastructures

webservices – a pragmatic view I

A webservice offers access to remote data/functions based on “internet” data transfer- and XML-technologies (interface specification and, mostly, data transfer). It is identified by an URI and it breaks the boundaries of RPC. Some overhead (for example XML parsing) is the price we have to pay for “loose coupling”.

REST – Representational State Transfer

- the „keep it simple“ approach
- stateless, uses HTTP (GET, POST, ...), URL=ressource (query)
webservices & infrastructures

web service technologies – some standards

SOAP – Simple Object Access Protocol (old acronym)
  - not simple and not only object access
  - usually people mean HTTP/TCP transfers
  - offers: XML based messaging

WSDL – Web Service Description Language
  - standardized description of webservices (functions, data types, ...)

UDDI – Universal Description Discovery and Integration
  - standardized repository service/infrastructure (uses SOAP)
  - never really “accepted”; most supporters quit about 4 years ago
The REST-ISBN-service of isbndb.com

URL: http://isbndb.com/api/books.xml?
access_key=D3SE8TXF&index1=isbn&value1=0321242653

<BookData book_id="absolute_java_a04" isbn="0321242653"
isbn13="9780321242655">
<Title>Absolute Java</Title><TitleLong/>
<AuthorsText>Walter Savitch</AuthorsText>
<PublisherText publisher_id="pearson_addison_wesley">
Boston : Pearson/Addison Wesley, 2004.</PublisherText>
</BookData>...
The REST-ISBN-service of isbndb.com II

- register an access key

- Why? The service is free, why do I need to register anyway?
The REST-ISBN-service of isbndb.com II

Managing Access Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Status</th>
<th>Usage</th>
<th>Comment</th>
<th>Created</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>D3SE8TXF</td>
<td>Active</td>
<td>5/5/0</td>
<td>my test key</td>
<td>01:41:59 03/20/2009</td>
<td>Edit</td>
</tr>
</tbody>
</table>

Mit dieser XML-Datei sind anscheinend keine Style-Informationen verknüpft. Nachfolgend wird die Baum-Ansicht des Dokumentes:

```xml
<ISBNDb server_time="2009-03-20T10:12:35Z">
  <BookList total_results="1" page_size="10" page_number="1" shown_results="1">
    <BookData book_id="absolute_java_a04" isbn="0321242653" isbn13="9780321242655">
      <Title>Absolute Java</Title>
      <TitleLong/>
      <AuthorsText>Walter Savitch</AuthorsText>
    </BookData>
  </BookList>
</ISBNDb>
```

D-Spin
webservices & infrastructures

webservices – a pragmatic view II

- UDDI not used: too complex, people found setting up an own “repository” (a webservice returning an XML with all available services and some meta-information) is good enough and easier

- WSDL/SOAP: Often too complex (for users), especially if you want to offer services to everyone. If different frameworks are used, problems will arise (non standardized functionality)
webservices & infrastructures

WSDL vs simple XML/REST

xmlns:tns1="http://datatypes.webservice.wortschatz.uni_leipzig.de"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types>
    <schema elementFormDefault="qualified" targetNamespace="urn:Sentences"
xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://datatypes.webservice.wortschatz.uni_leipzig.de"/>
      <element name="execute">
        <complexType>
          <sequence>
            <element name="objRequestParameters" type="impl:RequestParameter"/>
          </sequence>
        </complexType>
      </element>
    </schema>
  </wsdl:types>
</wsdl:definitions>
WSDL vs simple XML/REST II

- the WSDL file has 159 rows

- a REST description may look like this:
  
  - call: http://www.myhost.com/myservice/Test
  
  - answer will be all sentences from our database containing „Test“:
    
    ... <sentence>Dies ist ein Test.</sentence>
  
    <sentence>Das ist noch ein Test.</sentence> ...
  
  - otherwise than the fact, that the “query“ needs to be url-encoded (standardized), there are no further specifications needed

  - human readable; testable without effort; in many (simple) cases an example is “specification“ enough
A small list of problems:

- automatic generation of a WSDL-file from sourcecode:
  - sometimes framework specific code is added/generated
  - example: Axis2 – List<String> was not recognized (array of Objects) but an array of Strings was correctly interpreted

- you should write the WSDL-file by hand (or at least control it) and know exactly what you are doing

- many questions concerning on how to create a stub for programming language/framework xyz.

=> Ask yourself: Who is my customer?
webservice is in some way comparable to a java-method:

- URI $\leftrightarrow$ package+class
- WSDL-file / informal contract(REST) $\leftrightarrow$ method signature (parameters, returntype); java-reflection information
- calling a webservice $\leftrightarrow$ invoke a method

Do the same rules (like in java or programming in general) apply when we design complex systems/infrastructures?!
webservices & infrastructures

webservices – a pragmatic view IV

Answer: Mainly yes, but there are some specialities.

- calling a webservice is expensive:
  data transfer, request-response time, messaging (XML)
- interfaces / contracts are much more important because now there are always(!) many people involved
- ....
Building complex infrastructures

How does a complex infrastructure based on webservices look like?

Answer: A „loose“ network of simple services bound together by input-output specifications and a central repository.
Building complex infrastructures II

How do different services interact with each other? More specific: How do we realise complex process chains that are based on a variety of different services?

Example:

Service A → Service B → Service C
Building complex infrastructures III

It's all about specification! Which are the key components (businessobjects/entities) you use (for example: sentences, tokens, tags) and out of which parts do they consist?

But: Don't end up with complex/abstract formats.
What is special about webservices I

Some facts:
- you can't predict all possible usecases (people are creative)
- code is executed on the server
- the world is dynamic - services may:
  - change over time (results tommorrow may differ from today)
  - be unavailable for an unknown amount of time
  - change their input/output format

This has huge impact on how infrastructures, and every single service in it, have to be designed.
What is special about webservices II

- services need unique id's
  
  => if the service changed the id has to change too

- tools using the services can't expect every call to be successful
  
  (what to do if the service is gone/unavailable?)

- services have to protect themselves from being used in a wrong way:
  
  - unauthorized access
  
  - overload (affects all users and maybe other services running on
    the same machine!!!)

  => that's why registrations/authorization is often important

- obtaining metainformation about a service should be an infrastructure

D-Spin function
webservices & infrastructures

What is special about webservices III

Non technical aspects:

- legal issues:
  - WS allow (easy) automatic harvesting of huge data-collections
  - Who owns the content? Who is allowed to access it?
  - What are users allowed to do?
    - Build own services based on your services?
    - Sell products using your services?
  - Who pays for the infrastructure?
    - batch processing comes with high costs (CPU/Memory, Traffic)
    - who maintains the services (answer questions, ...)
Building an infrastructure the bottom up way

Some design decisions:
- we use REST-webservices
- our webservices return XML-documents
- more complex webservices read XML-documents as input
- Our definition of businessobjects stays the same in the whole infrastructure. For example: a contract looks the same (attributes, XML code) wherever it is used.

So it's the same old story again? No, not in all aspects...
Webservices & Infrastructures

Building an infrastructure the bottom up way II

Repository:

- we need a repository that holds all the information about the available services:
  - where is the service located
  - which data is needed for input (url, xml-document) - an input specification
  - which data is produced: an output specification
  - maybe some additional information (maintainer etc.)
Building an infrastructure the bottom up way II

Repository: implemented as a webservice (based on a simple DB-table):

```xml
<services>
  <service id="1">
    <maintainer>maintainer@maintainance.com</maintainer>
    <url>http://myhost.com:5555/myservice/</url>
    <input_format>myinputexample.xml</input_format>
    <output_format>myoutputexample.xml</output_format>
  </service>
  ...
</services>
```
Building an infrastructure the bottom up way III

We specify our „business objects“ and how they are “encoded” in XML:

simple example:

<text>
  <sentence>This is an example sentence.</sentence>
  <sentence>This is another example sentence.</sentence>
</text>
Building an infrastructure the bottom up way IV

Implement some services according to the specification and look out for possible process chains:

- retrieve sentences from a DB via webservice
- tokenize the sentences using another webservice
- add information of a second DB to each token (frequency information)

```xml
<sentence>
  <token freq="392">This</token>
  <token freq="6132">is</token>
  ...
</sentence>
```
Building an infrastructure the bottom up way

Start on a small scale:
- implement a few possible chains
- invest time in designing the correct "businessobject/entities"
- build a small repository (a WS backed up by a self designed DB)

Rather than investing in some fancy new technology invest into:
- faster / less memory consuming / ... algorithms
- infrastructure functionality that makes your infrastructure more stable; example: build a “ticket system”: queue jobs the right way (prefer people sending single calls to people performing batch jobs)
- the use case should dictate what to do; not the other way round
webservices & infrastructures

infrastructure proposal

- 4 layers:
  - end-user / GUI tools
  - common webservice
  - specific webservces
  - service implementations (database access etc.)

- the common webservice is configured to work with a specific ws-implementation (implemented once, used on top of every specific service)

- it offers unified infrastructure functionality like authorisation etc.
A small hands on example

We don't use any complex frameworks etc. here. Everything is out on the open to see. There is no magic!

We implement a small “Hello World!”-webservice:
- REST webservice (based on a simple java HTTP server)
- a GET request on a URL will return an XML containing:
  
  <text>Hello World!</text><url>http://www....</url>

- a POST request will result in returning the posted content
A small hands on example II

The HTTP-server loop:

```java
final ServerSocket server_socket;
try {
    server_socket = new ServerSocket(port);
    while (true) {
        Socket socket = server_socket.accept();
        try {
            RestHttpRequestHandler request = new RestHttpRequestHandler(socket, new MyRestAPI());
            Thread thread = new Thread(request);
            thread.start();
        } catch (IOException e) {
            System.err.println(e);
        }
    }
} catch (IOException e) {
    System.err.println(e);
}
```
A small hands on example III

RestHttpRequestHandler has 4 interesting methods:
- `processRequest()`: handles a request (reads HTTP header)
- `handlePostRequest()`: handles a post request (read input and send it back: `sendAnswer(String answer)`)
- `handleGetRequest(String url)`: construct the answer and call `sendAnswer(String answer)`;
- `sendAnswer(String answer)`: constructs the HTTP Header etc. and includes the answer
- the rest of the code mostly just handles connections etc.
private void processRequest() throws IOException {
    String headerLine = br.readLine();
    if (headerLine.equals(CRLF) || headerLine.equals("")) {return;
    StringTokenizer s = new StringTokenizer(headerLine);
    String requestType = s.nextToken();
    String url = s.nextToken();
    if (requestType.equals("GET")) {handleGetRequest(url);}  
    else if (requestType.equals("POST")) {handlePostRequest();} 
    else {sendAnswer(null);}
}
A small hands on example V

```java
private void handlePostRequest() throws IOException {
    StringBuffer strBuff = new StringBuffer();
    strBuff.ensureCapacity(MAX_CONTENT_LENGTH);
    while (br.ready()) {
        String line = br.readLine();
        strBuff.append(line);
        if (strBuff.length() > MAX_CONTENT_LENGTH) break;
    }
    sendAnswer(strBuff.toString());
}

private void handleGetRequest(final String url) throws IOException {
    StringBuffer answer = new StringBuffer(""");
    answer.append("<?xml version="1.0" encoding="" + ENCODING + "\" standalone="yes"?>");
    answer.append("<content><text>Hello World!</text><url>" + url + 
    "</url></content>");
    sendAnswer( answer.toString() );
}
```
webservices & infrastructures

A small hands on example VI

All in all 2 classes and 167 lines of java-code (mostly for HTTP handling):

```
<content>
  <text>Hello World!</text>
  <url>/hello</url>
</content>
```
webservices & infrastructures

A small hands on example VII

try it out yourself:

- download and install a java runtime environment:
  - http://www.java.com/de/download/

- download the JAR (and/or the java code) of the SimpleRestHttpServer:
  - http://www.informatik.uni-leipzig.de/~boehlkev/RestTutorial.zip

- open a command line window and start the server: “java -jar xyz.jar”

- start up a browser and open: http://localhost:5555/hello

- you'll see the example output from before (make sure firewalls don't block the “server”)

D-Spin
A small hands on example VIII

try it out yourself:

- start the SimpleDSpinGui: “java -jar SimpleDSpinGui.jar”
  - call an example webservice like:
    
    http://aspra18.informatik.uni-leipzig.de:8000/sentences

    using the input of query.xml (also in the zip-file)
  - SimpleDSpinGui is just a small developer-test and example tool
  - if no content is given a GET request is performed on the specified URL; otherwise the content is used in a HTTP-POST request
- feel free to use the code for your own experiments etc.

Dem Zahnmediziner aus Leipzig, der im vergangenen Oktober wieder ins NOK-Präsidium kooptiert wurde, wird vorgeworfen, er habe als Präsident des DDR-Kanuverbandes dazu beigetragen, politisch mißliebige Sportler aus der Auswahl zu entfernen.

Lange hat's gedauert, aber allmählich erobert Max Klinger, der poetische Symbolist aus Leipzig, die Welt.

Mit sechzehn hatte sich der gelernte Anlagenarbeiter aus Leipzig zum ersten Mal für ein Schauspielstudium beworben, um in Folge acht niederschmetternde Jahre regelmäßig bei der Aufnahmeverfahren durchzufallen.

Der Grund dafür war, daß zwei Frauen gegeneinander kandidierten und daher der Landtagsabgeordnete Schimoff aus Leipzig der lachende Dritte war.
webservices & infrastructures

The next steps

- implement your specific “service” code (acces to a DB, ...)
- use good interfaces while doing so; this will keep your
code agile enough to switch to different technologies
- switch to a professional tool like Apache Tomcat
- maybe implement tools that make use of your services
- implement infrastructure components like a repository (a simple
database accessed via webservice) or common functionality
TMS schedule - more about WS

24.03.2009; 15:00
A prototype infrastructure for DSpin-services based on an flexible XML data interchange format - Volker Boehlke (University of Leipzig, D-SPIN Project)

25.03.2009; 13:00
LLS – A 4 Years Summary of Providing Linguistic Web Services
Marco Büchler (University of Leipzig, eAqua Project), Gerhard Heyer (University of Leipzig)
Thanks for your attention!