UVA HPC & BIG DATA COURSE

Service oriented Architecture and Web services

Adam Belloum
Content

• Service Oriented Architecture
• Web services
  – SOAP Based Web services
  – Web Service Reference Framework (WSRF)
  – RESTful Web service
Web services are everywhere

• “If you hit Amazon.com gateway page, the application calls more than 100 Web services to collect data and construct the page for you”
  
  Werner Vogels, CTO Amazon, ACM Queue 2006

• “More than 1 6 000 000 job submission were processed using Web Services during 2006 (30% of all jobs run at the EBI)”
  
  European Bioniformatics institute
The case for developing Service Oriented Architecture

• Level of Software complexity continues to increase, and traditional architectures seem to be reaching the limit of their ability

• Need to respond quickly to new requirements of the Application

• Need to continually reduce the cost of IT for the application

Ability to absorb and integrate new partners, new users and applications
Problems: The joy of porting...

- It is hard to get insight into the inner workings of a Fortran application that has been worked on for 25 years!
  - 20% of Parallel Ocean Program to CUDA is hard
  - In addition, a lot of 'boilerplate code' is required to connect the Fortran and CUDA parts. We need more powerful tools!

Source: Ben van Werkhoven
Service Oriented Architecture

• Leverage **existing assets** (services).
  – Existing systems can not be thrown away, because they contain within them great value to the enterprise/search group.

• Support all required types of **integration**.
  – User Interaction
  – Application Connectivity
  – Process Integration
  – Information Integration
  – **Build to Integrate**
Service Oriented Architecture
--New computing models

• Allow for **incremental** implementations & **migration** of assets
  → Include a development environment that will be built
  • around a **standard component** framework,
  • promote better **reuse** of **modules** and **systems**,
  • allow legacy assets to be **migrated** to the framework,
  • allow for the **timely** implementation of new technologies.

Allow implementation of new **computing models**; specifically, new **portal-based client models**, **Grid computing**, and **on-demand computing (cloud)**
A service-oriented architecture
-- not just Web services

• First, it must be understood that *Web services* does not equal *service-oriented architecture*.

• Web services is a *collection of technologies*, including HTTP, XML, SOAP, WSDL, and UDDI,

Service Oriented Architecture is "an application architecture within which all functions are defined as *independent* services with well defined *invocable interfaces* which can be called in defined sequences to form business processes".
A service-oriented architecture
-- services are independent

• All functions are defined as **services**.

• All **services** are **independent**
  – Operate as "**black boxes**";
    • external components neither know nor care how boxes are executed

• The **interfaces** are **invocable**
  – it is irrelevant whether the service are **local** or **remote**
    • what **interconnect scheme** or **protocol** is used to effect the invocation,
    • what **infrastructure components** are required to make the connection.
A service-oriented architecture
-- service Interface

- **Interface** is the key, & the focus of the calling application.
  - It defines the **required parameters** and the **nature** of the **result**

- It is the **system's responsibility** to **effect** and **manage** the invocation of the service,

- This allows two critical characteristics to be realized:
  
  - Services are **truly independent**,
  - They can be **managed**: Security, Deployment, Logging, Dynamic rerouting, and Maintenance
**The Nature of a Service**

- **In a business environment**
  - Service means *business functions & transactions, and system services*.

- **In a research environment**
  - Service means *application functions, and system services*.

- **The difference in the types of services.**
  - Business functions are from *the application's perspective, non-system functions*.
  - Services might be *low-level* or complex *high-level* (fine-grained or course grained) functions.
A service-oriented architecture
-- Model

• A service provider
  – provides a service interface for a software asset that manages a specific set of tasks.

• A service requester
  – discovers and invokes other software services to provide a business solution.

• A service broker
A service-oriented architecture
-- Model: Service Requester

• Simple client software

• Aggregator:
  - Content Aggregation
    • Activity where an entity interacts with a variety of content providers to process/reproduce such content in the desired presentation format of its customers.

  - Service Aggregation
    • Activity where an entity interacts with a variety of service providers to rebrand, host, or offer a composite of services to its customers.
A service-oriented architecture
-- Model: Service Provider

Independent software vendors are prime examples of potential service providers.

- They own and maintain a software asset that performs tasks.
- Software assets could be made available as an aggregation of services or broken down into distinctly separate software service.

Processes that are proven/generalized for a diverse set of applications would be good candidates for service providers.

For example, a bank feels that its business process for loan processing was a strong enough asset to be made publicly available and was willing to support it as a business offering, then that bank could view itself as a loan processing service provider.
A service-oriented architecture  
-- Model: Service Registry

• Is an entity that collects and catalogs data about other entity and then providing that data to others (a form of Requirement for **Service Oriented Architecture** Broker.)

• Usually, a registry would collect data such as
  – Entity name,
  – Description, and contact information.

In UDDI terms, this Registry role is often referred to as the **White Pages**.
sketchy example of how a Web Service works

• The clients (the PCs at the store)
  – contact the Web Service on remote server
  – send a service request asking for the catalog
  – The server returns the catalog through a service response.
A Typical Web Service Invocation

1. First step will be to find a Web Service that meets our requirements: contact a registry.
2. The registry will reply, telling what servers can provide the service required.
3. The location of a Web Service is now known, but the actually invocation method is still unknown.
   The second step is to ask the Web Service to describe itself
4. The Web Service replies using WSDL.
5. The Web Service is located and invocation method is known.
   The invocation is done using SOAP.
6. The Web Service will reply with a SOAP response which includes the information we asked for, or an error message.
12 Steps to implement a Service Oriented Architecture

1. Understand the functional objectives and define success.
2. Define your problem domain.
3. Understand all application semantics in your domain.
4. Understand all services available in your domain.
5. Understand all information sources and sinks available in your domain.
6. Understand all processes in your domain.
12 Steps to implement a Service Oriented Architecture

7. Identify and catalog all interfaces outside of the domain you must leverage (services and simple information).

8. Define new services/information bound to the services.

9. Define new processes, services, and information bound to the processes.

10. Select your technology set.

11. Implement & Deploy SOA technology.

12. Test and evaluate
Content

- Service Oriented Architecture
- Web services
  - SOAP Based Web services
  - RESTful Web service
- Example of usage of Web Service in Scientific applications
Web Services have certain advantages over other technologies

what makes Web Services special?

• Web Services are **platform-independent** and **language-independent**

• Web Services use **HTTP** for transmitting.
Enabling technologies

• XML: The Extensible Markup Language

• SOAP:
  – Simple Object Access Protocol is an XML-based lightweight protocol for the exchange of information in a decentralized,

• WSDL:
  – The Web Services Description Language is an XML vocabulary that provides a standard way of describing service IDLs.

• UDDI:
  – The Universal Description, Discovery, and Integration specification provides a common set of SOAP APIs that enable the implementation of a service broker.
Web Services also have some disadvantages

- **Overhead.** Transmitting all data in XML is not as efficient as using a proprietary binary code.
  - What you win in *portability*, you lose in *efficiency*.
  - This overhead is usually acceptable for most applications, but you will probably never find a critical real-time application that uses Web Services.

- **Lack of versatility.** Web Services are not very versatile, since they allow for *very basic forms* of service invocation.
  - Do not offer a standardized support for *persistency, notifications, lifecycle management, transactions*, etc.
One important characteristic that distinguishes Web Services

• While technologies such as CORBA and EJB are oriented toward *highly coupled distributed systems*, where the client and the server are very dependent on each other

Web Services are oriented towards *loosely coupled systems*, where the client has no prior knowledge of the Web Service until it actually invokes it.
Web Services Addressing

• The registry (UDDI) tells the client *where* the Web Service is located. But, how exactly are Web Services addressed?
  – The answer is simple: just like web pages. simple URIs. For example, the UDDI registry might have replied with the following URI:
    
    http://webservices.mysite.com/weather/us/WeatherService

• Note:
  – Web Services are always used by software (never directly by humans). When you have a Web Service URI, you will need to feed the URI to a program.
WS-Addressing

Invoking a Web Service

Dr. Daniel Sabbah “Bringing Grid & Web Services Together”, IBM Software Group
What a Web Service Application Looks Like

1. Client application invoke the Web Service, by calling the client stub.
   • The client stub will turn this 'local invocation' into a proper SOAP request.
2. The SOAP request is sent over a network using the HTTP protocol.
   • WS container receives the SOAP request & hands it to the server stub.
   • The server stub converts the SOAP request into something the service implementation can understand.
3. The service implementation receives the request from the service stub, and carries out the work it has been asked to do.
4. The result of the requested operation is handed to the server stub, which turns it into a SOAP response.
5. The SOAP response is sent over a network using the HTTP protocol.
   • The client stub receives the SOAP response and turns it into something the client application can understand.
6. The application receives the result of the Web Service invocation.
What a Web Service Application Looks Like

1. Client application **invoke** the Web Service, by calling the client stub.
   - The client stub will **turn** this 'local invocation' **into** a proper SOAP request.

2. The SOAP request is sent over a network **using the HTTP protocol**.
   - WS container **receives the SOAP requests** & hands it to the server stub.
   - The server stub converts the **SOAP request** into something the service implementation can understand

3. The service implementation **receives the request** from the service stub, and carries out the work it has been asked to do.

4. The result of the requested operation is handed to the server stub, which **turns** it into a **SOAP response**.

5. The SOAP response is sent over a network **using the HTTP protocol**.
   - The client stub receives the **SOAP response** and **turns** it into something the client application can understand.

6. The application receives the result of the Web Service invocation
What do we need to create a Web service?

• Web Services programmers usually never write a single line of SOAP.
  
  – Once we've reached a point where the client application needs to invoke a Web Service, we delegate that task on a piece of software called a stub.
  
  – There are plenty of tools available that will generate stubs automatically, based on the WSDL description of the Web Service.

• A Web Services client doesn't usually do all those steps in a single invocation. A more correct sequence of events would be the following:

  1. We locate a Web Service that meets our requirements through UDDI.
  2. We obtain that Web Service's WSDL description.
  3. We generate the stubs once, and include them in our application.
  4. The application uses the stubs each time it needs to invoke the Web Service.
• Service Oriented Architecture
• Web services
  – SOAP Based Web services
  – RESTful Web service
• Usage of Web Service in Scientific applications
Web Service and State (WSRF)

- A WS-Resource is defined as the composition of a Web service and a S-Resource
  - Expressed as an association of an XML document with defined type with a Web services portType
  - Addressed and accessed according to the conventional use of WS-Addressing endpoint references

S-Resource identifier is encapsulated in an endpoint reference to identify the S-Resource to be used in the execution of a Web service message exchange.
What is REST

• REST stands for **Representational State Transfer**.

• REST was coined by Roy Fielding in 2000 in his Ph.D. dissertation to describe a **design pattern for implementing networked systems**.

In many ways, the World Wide Web itself, based on HTTP, can be viewed as a REST-based architecture.

Roger I. Costello, Timothy D. kehoe
Why is it called "Representational State Transfer?"

The Client references a Web resource using a URL.
- A representation of the resource is returned (in this case as an HTML).
- The representation (e.g., Boeing747.html) places the client in a new state.

When the client selects a hyperlink in Boeing747.html, it accesses another resource. The new representation places the client application into yet another state.

Thus, the client application transfers state with each resource representation.

Roger I. Costello, Timothy D. kehoe
Representational State Transfer

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (the next state of the application) being transferred to the user and rendered for their use."

- Roy Fielding
REST - Not a Standard

• REST is **NOT** a standard
  – You will not see the W3C putting out a REST specification.
  – You will not see IBM or Microsoft or Sun selling a REST developer's toolkit.

• REST is just a **design pattern**
  – You can only understand it and design your Web services to it.

• REST does prescribe the **use** of standards:
  – HTTP
  – URL
  – XML/HTML/GIF/JPEG/etc. (**Resource Representations**)
  – text/xml, text/html, image/gif, image/jpeg, etc. (**Resource Types, MIME Types**)
REST Fundamentals

1. Create a resource for every service.
2. Identify each resource using a URL.
3. The data that a Web service returns should link to other data.

Thus, design your data as a network of information.
Parts Depot Web Services

- Parts Depot, Inc has deployed some web services to enable its customers to:
  - get a list of parts
  - get detailed information about a particular part
  - submit a Purchase Order (PO)
The REST way of Designing the Web Services

HTTP GET request → URL 1 → Parts List

Response (HTML/XML doc) → HTTP response

HTTP GET request → URL 2 → Part Data

Response (HTML/XML doc) → HTTP response

HTTP POST

PO (HTML/XML) → URL 3

URL to submitted PO → HTTP response
Web Service for Clients to Retrieve a List of Parts

Service: Get a list of parts

– The web service makes available a URL to a parts list resource. A client uses this URL to get the parts list:

  http://www.parts-depot.com/parts

Note that how the web service generates the parts list is completely transparent to the client. This is loose coupling.

Roger I. Costello, Timothy D. kehoe
Data Returned - Parts List

<?xml version="1.0"?>
<Parts>
    <Part id="00345" href="http://www.parts-depot.com/parts/00345"/>
    <Part id="00346" href="http://www.parts-depot.com/parts/00346"/>
    <Part id="00347" href="http://www.parts-depot.com/parts/00347"/>
    <Part id="00348" href="http://www.parts-depot.com/parts/00348"/>
</Parts>

Note that the parts list has links to get detailed info about each part.

→ This is a key feature of the REST design pattern. The client transfers from one state to the next by examining and choosing from among the alternative URLs in the response document.

Roger I. Costello, Timothy D. kehoe
The REST Design Pattern (cont.)

All interactions between a client and a web service are done with simple operations.

• Most web interactions are done using HTTP and just four operations:
  – retrieve information (HTTP GET)
  – create information (HTTP PUT)
  – update information (HTTP POST)
  – delete information (HTTP DELETE)
The REST Design Pattern (cont.)

- **Web components** (firewalls, routers, caches) make their decisions based upon information in the **HTTP Header**.
- Consequently, the **destination URL** **MUST** be placed in the HTTP header for Web components to operate effectively.
  - Conversely, it is anti-REST if the HTTP header just identifies an intermediate destination and the payload identifies the final destination.

Roger I. Costello, Timothy D. kehoe
Question

What if I have a complex query?

For example:
Show me all parts whose unit cost is under $0.50 and for which the quantity is less than 10

How would you do that with a simple URL?
Answer

For complex queries, Parts Depot will provide a service (resource) for a client to retrieve a form that the client then fills in.

When the client hits "Submit" the browser will gather up the client’s responses and generate a URL based on the responses.

Thus, oftentimes the client doesn't generate the URL (think about using Amazon - you start by entering the URL to amazon.com; from then on you simply fill in forms, and the URLs are automatically created for you).
Real REST Examples

• Twitter has a REST API
  – https://dev.twitter.com/docs/api

• Flickr
  – http://www.flickr.com/services/api/

• Amazon.com offer several REST services, e.g., for their S3 storage solution

http://rest.elkstein.org/2008/02/real-rest-examples.html
## Twitter has a REST API

### Tweets
Tweets are the atomic building blocks of Twitter, 140-character status updates with additional associated metadata. People tweet for a variety of reasons about a multitude of topics.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET statuses/retweets/:id</td>
<td>Returns up to 100 of the first retweets of a given tweet.</td>
</tr>
<tr>
<td>GET statuses/show/:id</td>
<td>Returns a single Tweet, specified by the id parameter. The Tweet’s author will also be embedded within the tweet. See Embeddable Timelines, Embeddable Tweets, and GET statuses/oembed for tools to render Tweets according to Display Requirements.</td>
</tr>
<tr>
<td>POST statuses/destroy/:id</td>
<td>Destroys the status specified by the required ID parameter. The authenticating user must be the author of the specified status. Returns the destroyed status if successful.</td>
</tr>
<tr>
<td>POST statuses/update</td>
<td>Updates the authenticating user’s current status, also known as tweeting. To upload an image to accompany the tweet, use POST statuses/update_with_media. For each update attempt, the update text is compared with the authenticating user’s recent tweets. Any attempt that would result in duplication...</td>
</tr>
<tr>
<td>POST statuses/retweet/:id</td>
<td>Retweets a tweet. Returns the original tweet with retweet details embedded.</td>
</tr>
<tr>
<td>POST statuses/update_with_media</td>
<td>Updates the authenticating user’s current status and attaches media for upload. In other words, it creates a Tweet with a picture attached. Unlike POST statuses/update, this method expects raw multipart data. Your POST request’s Content-Type should be set to multipart/form-data with the media[].</td>
</tr>
<tr>
<td>GET statuses/oembed</td>
<td>Returns information allowing the creation of an embedded representation of a Tweet on third party sites. See the oEmbed specification for information about the response format. While this endpoint allows a bit of customization for the final appearance of the embedded Tweet, be aware that the...</td>
</tr>
</tbody>
</table>
# SOAP vs REST

**Strengths and weaknesses for both SOAP (above) and REST (below).**

<table>
<thead>
<tr>
<th></th>
<th>SOAP</th>
<th>Weaknesses (cons)</th>
</tr>
</thead>
</table>
| **Strengths (pros)** | + Handle distributed computing environments  
                      + Built-in error handling  
                      + Extensibility  
                      + Language, platform, and transport agnostic  
                      + Prevailing standard for web services  
                      + Support from other standards (WSDL, WS-*) | - More verbose  
                      - Harder to develop, requires tools  
                      - Conceptually more difficult, more ”heavy-weight” than REST |
| **Strengths (pros)** | + Language and platform agnostic  
                      + Much simpler to develop than SOAP  
                      + Small learning curve, less reliance on tools  
                      + Concise, no need for additional messaging layer  
                      + Closer in design and philosophy to the Web | - Assumes a point-to-point communication model  
                      - Not usable for distributed computing environment  
                      - Lack of standards support for security, etc.  
                      - Tied to the HTTP transport model |
Content

• Service Oriented Architecture
• Web services
  – SOAP Based Web services
  – RESTful Web service
• Example of usage of Web Service in Scientific applications
Usage of Web Services in e-science

• Constraints:
  – Service orchestration, all data is passed to the workflow engine before delivered to a consuming WS
  – Data transfers are made through SOAP, which is unfit for large data transfers

Enabling web services to consume and produce large distributed datasets Spiros Koulouzis, Reginald Cushing, Konstantinos Karasavvas, Adam Belloum, Marian Bubak to be published JAN/FEB, IEEE Internet Computing, 2012
ProxyWS

- uses multitude of protocols to transport large data
  - used as an interface for developing WSs able to stream data.
  - Or as enabler for legacy web services to stretch their current potential by referencing data that would otherwise be delivered via SOAP

*Enabling web services to consume and produce large distributed datasets* Spiros Koulouzis, Reginald Cushing, Konstantinos Karasavvas, Adam Belloum, Marian Bubak to be published JAN/FEB, IEEE Internet Computing, 2012
Indexing Name Entry Recognition

- AIDA provides a set of components which enable the indexing of text documents in various formats.
- AIDA's Indexer component, called IndexerWS is a WS able to index document with the use of the Streaming library.
Enabling web services to consume and produce large distributed datasets Spiros Koulouzis, Reginald Cushing, Konstantinos Karasavvas, Adam Belloum, Marian Bubak to be published JAN/FEB, IEEE Internet Computing, 2012