Web Services & REST

Jukka K. Nurminen

31.1.2012

Based partly on slides by Tancred Lindholm, Sasu Tarkoma and Pekka Nikander

Assignment

- I hope everybody has sent an assignment signup message to the course mailing list
- Course assistant reception time reservation system timesplayground.cs.hut.fi/kaiku
- 2 & 3 assignments today
  - Virtual machine for each pair coming soon
- General guidelines

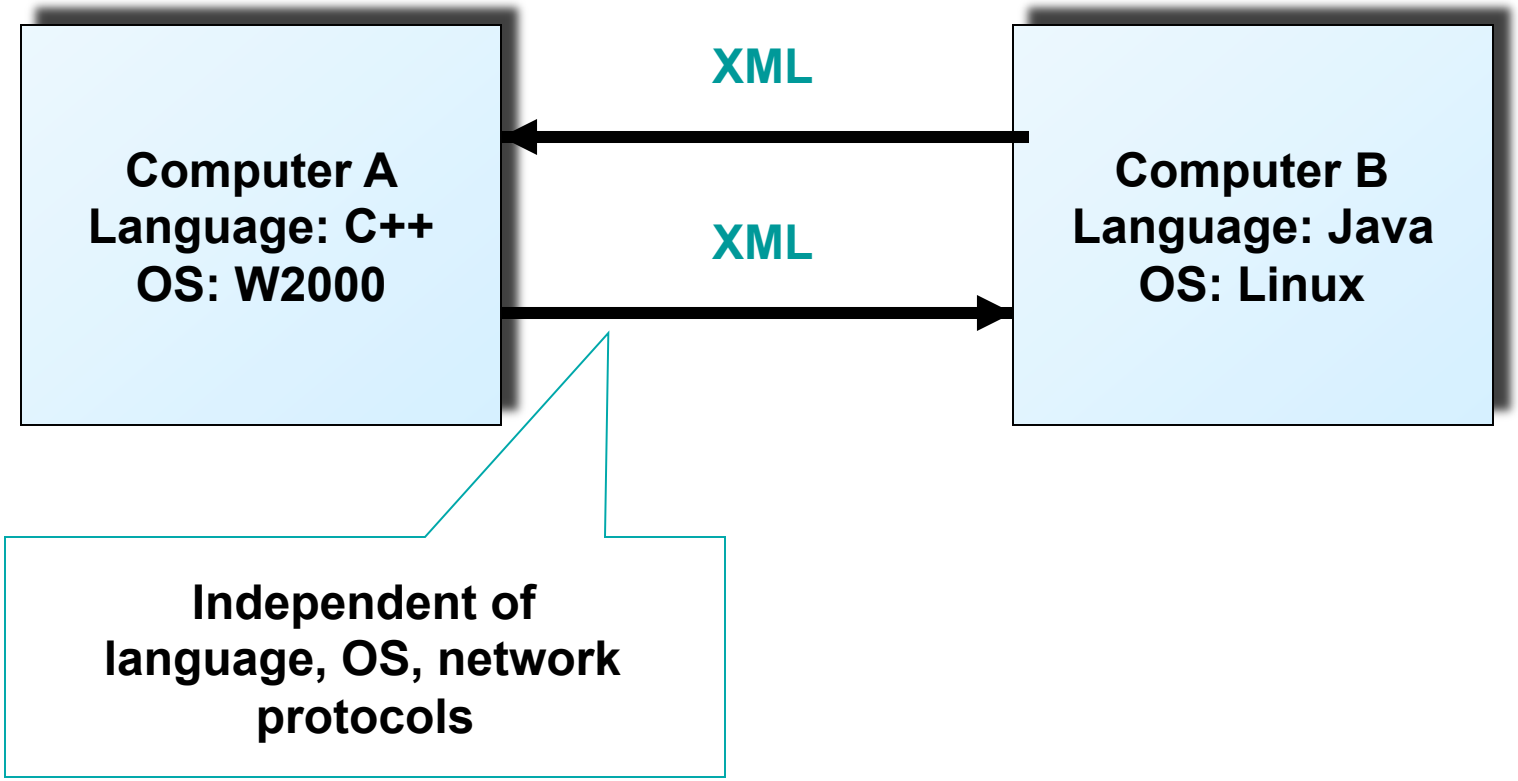
Last Time

- HTML
- XML and related languages
  - XML
  - SGML
  - DTD
  - XML Schema
  - XSLT
- JSON
- EXI
- Android lecture

Today

- Communicating between different services
- We use the presentations we discussed last week
- How do we specify what needs to be done?
- How do we find the services?
- How do we know how to call the services?
- Main approaches
  - Web Services and a set of ws-* standards
  - REST
- Application of these concepts to some services

A Basic Web Service



Web Services

- Let's make machine-callable services using web principles
- A central role is played by the description of the service's interface
- Implementation less important, avoid implementation-specifics
- Business aspects considered
  - Use across organizations
  - Multiple competing implementations

Driving forces I

- Universal data access / representation
  - Independent of OS, programming language, network protocol, …
- Move from human-centric to application-centric web
  - Applications converse with each other and use machine-related information published on the web
  - Application-areas: package tracking, card verification, shopping bots, single sign-on, calendar, email, ...

Driving forces II

- Making Web a programming interface
  - We have had servlets, CGI, CORBA for years
  - Idea is to standardise languages and protocols to have better integration
- Make service composition possible
  - Faster project throughput
  - Better utilization of global resources
  - Cope with heterogeneity
- Deferred binding
  - Discovery / broker, interpret, compose, execute
  - Many levels of deference

Additional properties

- A web service should be self-describing
  - Interface is published with the implementation
  - Minimum interface is human readable description
  - The interface can also be written in a common XML grammar (WSDL)
- A web-service should be discoverable
  - The web service is published
  - It has a life cycle
  - Interested parties can find it
- Not mandatory but desirable properties

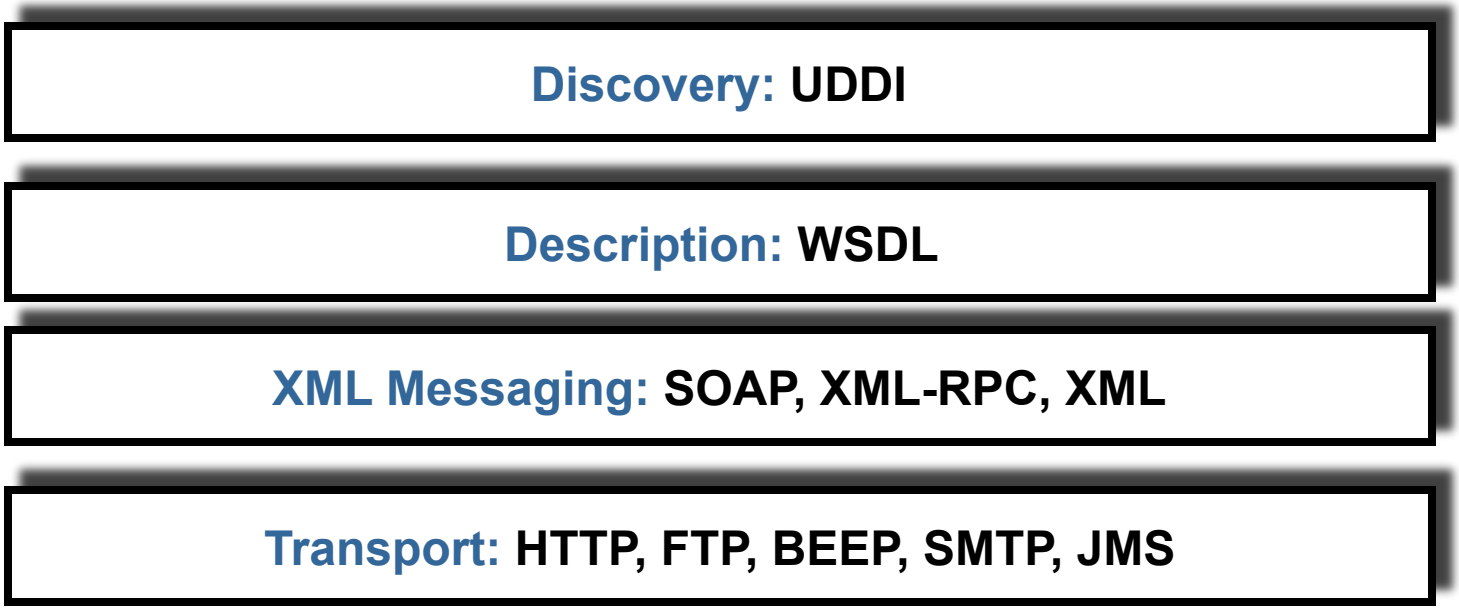
Standardization

- W3C Web Services Activity
  - XML Protocol Working Group
    - SOAP
  - Web Services Addressing Working Group
    - How to address WS entities
  - Web Services Choreography Working Group
    - Processes involving several WS, coordination
  - Web Services Description Working Group
    - WSDL
- OASIS
  - UDDI (Universal Description, Discovery and Integration)
- WS-I (Web Service Interoperability Org.)
  - Best Practices on how to use WS* standards

Web Service Architecture

- The three major roles in web services
  - Service provider
    - Provider of the WS
  - Service Requestor
    - Any consumer / client
  - Service Registry
    - logically centralized directory of services
- A protocol stack is needed to support these roles

WS Protocol Stack



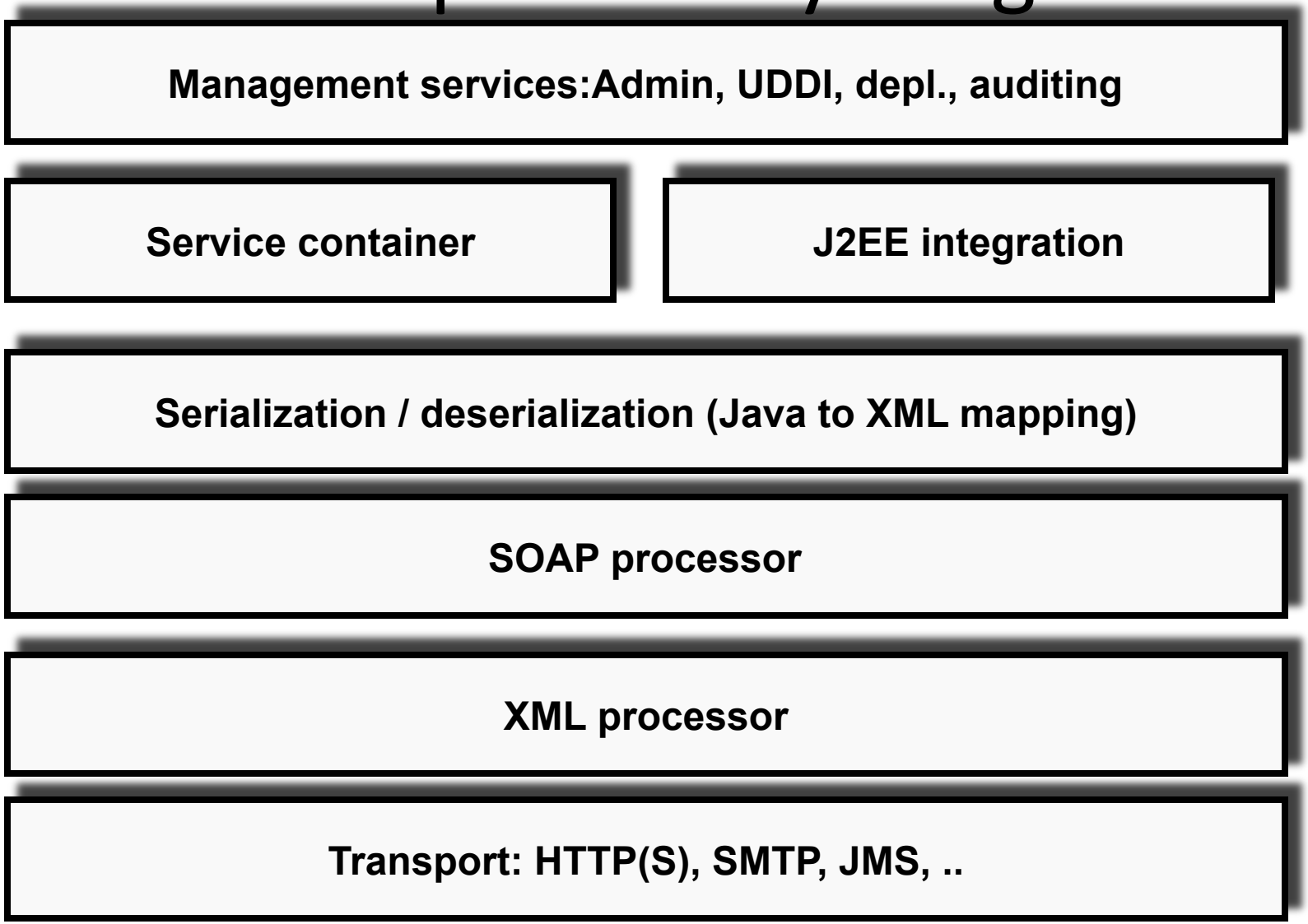
Web Services Protocol Stack

- Message Exchange
  - Responsible for transporting messages
  - HTTP, BEEP
- XML Messaging
  - Responsible for encoding messages in common XML format
  - XML-RPC, SOAP
- Service Description
  - Responsible for describing an interface to a specific web service
  - WSDL
- Service discovery
  - Responsible for service discovery and search
  - UDDI

Main components today

- XML data representation
  - XML Schema Definitions (xsd) for types
  - XML Namespaces for unambiguity
- SOAP
  - Basic transport (XML messaging)
  - Sync / async communication and RPC
- WSDL
  - Description of (SOAP) services
- UDDI
  - Universal Description Discovery and Integration
  - Service registry

Example WS layering



XML Messaging

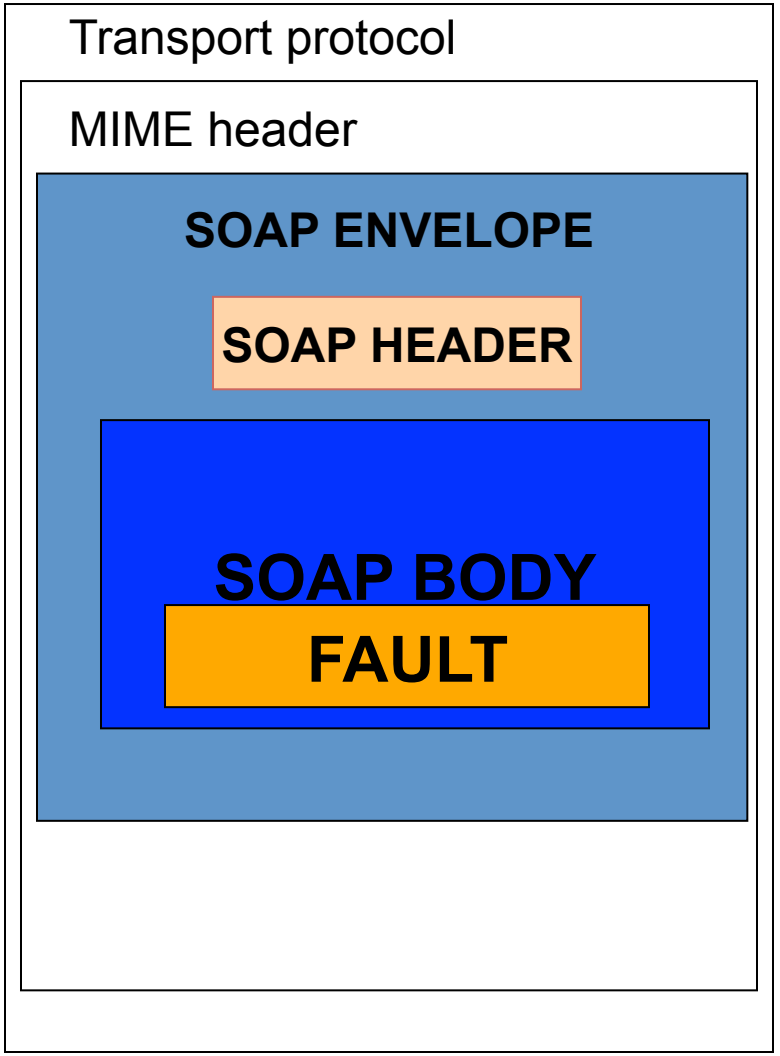
- Several alternatives for XML messaging
  - SOAP
  - XML Remote Procedure calls (XML-RPC)
  - Regular XML transported over HTTP

SOAP Version 1.2

- protocol for exchanging structured (XML) and typed information between peers
- A SOAP message is formally specified as an XML Infoset ("abstract XML")
- Infosets can have different on-the-wire representations, one common example of which is as an XML 1.0 document.
- A stateless, one-way message exchange paradigm
- Applications can create more complex interaction patterns
  - request/response, request/multiple responses

SOAP Structure

- Each SOAP message will have:
  - An `Envelope`
  - A `Header` (optional)
  - A `Body`
  - The `Body` may contain a `Fault` element



SOAP Request

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
 <?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-
   encoding">
<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>
 </soap:Envelope>
```

The actual call

SOAP Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

The actual reply

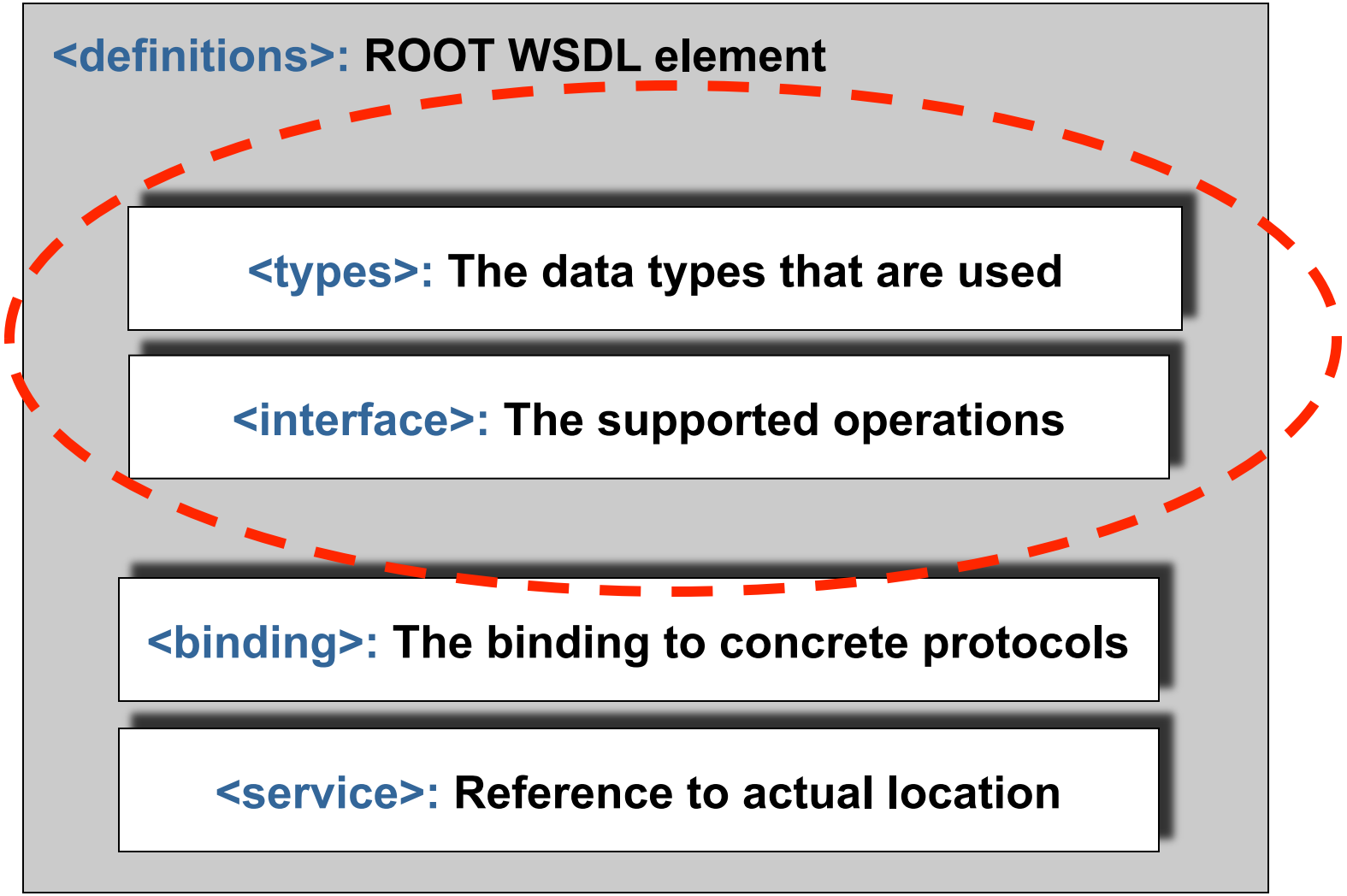
About invocation mechanisms

- WS does not define invocation/execution mechanism
- Alternatives
  - Microsoft .NET framework
  - Java-based framework
    - JAVA API for WSDL (JWSDL)
    - JAX-RPC
    - Java API for XML registries (JAXR)
    - Apache Axis
    - …

What is WSDL?

- WSDL: Web Service Description Language
- An XML language used to describe and locate web services
  - location of web service
  - methods that are available
  - data type information and XML messages
- Commonly used to describe SOAP-based services
- W3C standard (work in progress)
  - Initial input: WSDL 1.1 as W3C Note
  - Current version 2.0 (Recommendation)
  - Some differences between 1.1 and 2.0

WSDL Overview



A WSDL Document

- A WSDL document contains two parts
- Abstract part
  - Interfaces, types
- Concrete part
  - Binding to concrete protocol and encoding
- May be published separately
  - Interfaces agreed between many companies
  - Each company published their own implementation in UDDI and import the abstract interface.

Example

```
<types>
   <xsd:schema
        xmlns:xsd="http://www.w3.org/2001/XMLSchema" ...>
    <xsd:element name="TradePriceRequest" type="xsd:string"/>
    <xsd:element name="TradePrice" type="xsd:float"/>
</types>

<interface name="StockQuotePortType">
  <operation name="GetLastTradePrice"
                  pattern="http://www.w3.org/ns/wsdl/in-out ">
    <input message="In" element="TradePriceRequest" />
    <output message="Out" element="TradePrice "/>
  </operation>
</interface>
```

WSDL Document Elements (v. 2.0)

- <types> - data type definitions
- <interface> - A set of abstract operations
- <binding> - Concrete protocol and data format specifications for the operations and messages defined by a particular interface. Endpoint type.
- <endpoint> - An address for binding. Endpoint instance.
- <service> - A set of endpoints

Types

- <types> define data types used in interface declaration
- For platform neutrality, WSDL uses XML Schema syntax to define data
  - XML Schema must be supported by any vendor of WSDL conformant products
  - Other kinds of type definitions also possible
    - Possible interoperability issues
  - If the service uses only XML Schema built-in simple types, such as strings and integers, the types element is not required

WSDL Interfaces

- The <interface> element is the most important WSDL element
- The operations that can be performed
- An <endpoint> defines the connection point to a web service, an instance of <interface>
- It can be compared to a function library (or a module, or a class) in a programming language

Bindings & Services

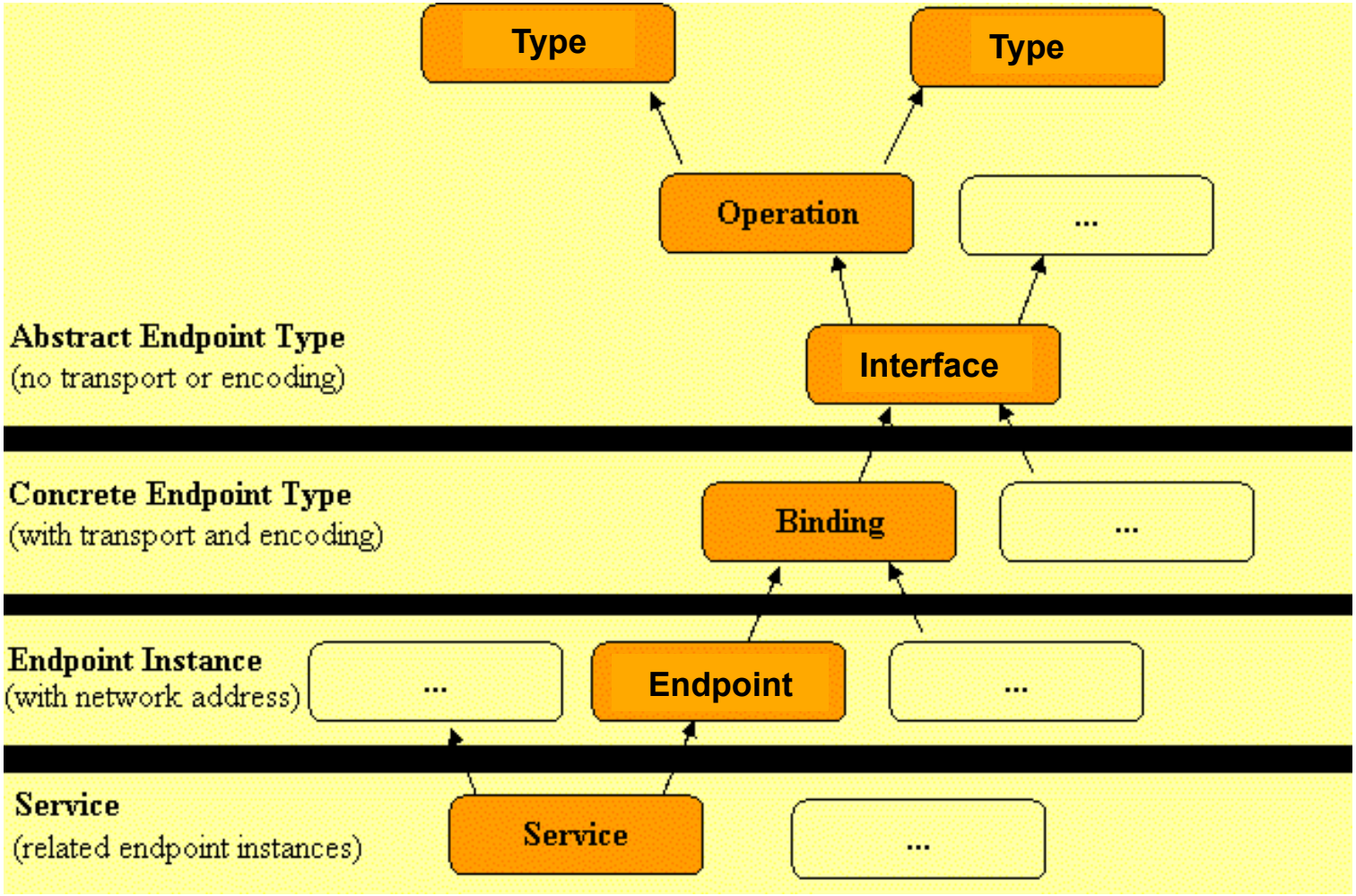
```
<wsdl:binding name="StockQuoteSoapBinding" interface="tns:StockQuoteInterface"
   type="http://www.w3.org/ns/wsdl/soap"
   wsoap:version="1.1"
   wsoap:protocol="http://www.w3.org/2006/01/soap11/bindings/HTTP/">
   <wsdl:operation ref="tns:GetLastTradePrice"
    wsoap:action="http://example.com/GetLastTradePrice"/>
 </wsdl:binding>
```

What operation is executed

```
 <wsdl:service name="StockQuoteService" interface="tns:StockQuoteInterface">
  <wsdl:documentation>My first service</wsdl:documentation>
  <wsdl:endpoint name="StockQuoteEndPoint" binding="tns:StockQuoteBinding"
  address="http://example.com/endpoint/stockquote"/>
 </wsdl:service>
```

Where the service can be found

Putting it together



Original source: http://msdn.microsoft.com/

Uses of WSDL documents

- Description of service interfaces
  - Compile-time
    - Developer uses WSDL before service deployment
  - Run-time
    - Client downloads WSDL description and uses the info it provides to execute the service
- As a side-effect
  - Developers can use WSDL to speed up the development of code
  - WSDL◊Java code
  - Java interfaces ◊ WSDL

How it could work

- 1. A standard body creates a WSDL interface definition
- 2. A service programmer implements a service according to the WSDL definition
- 3. A client programmer implements a client according to the WSDL definition
- 4. A service provider deploys the service and publishes a WSDL implementation definition, and registers it into UDDI
- 5. A client program pulls WSDL from UDDI, checks conformance, and uses SOAP for access

2. Creating server application

- Pull WSDL definition from somewhere (UDDI)
  - Only use high-level WSDL, no bindings yet
- Generate platform specific skeleton code using automated tools
- Write the actual program code

3. Creating client application

- Pull WSDL definition from somewhere (UDDI)
  - Use only high-level WSDL, no bindings yet
- Generate platform specific stub code using automated tools
- Write the actual program code

How it ~~could~~ seems to work

- 1. ~~A standard body creates a WSDL interface definition~~ a company specifies, implements, and deploys a web interface
- ~~2. A service programmer implements a service according to the WSDL definition~~
- 3. A client programmer implements a client ~~according to the WSDL definition~~ with the textual specification and tests it with the live web site
- ~~4. A service provider deploys the service and publishes a WSDL implementation definition, and registers it into UDDI~~
- ~~5. A client program pulls WSDL from UDDI, checks conformance, and uses SOAP for access~~

2. Creating server application
but it may not happen like this

- Pull WSDL definition from somewhere (UDDI)
  - Only use high-level WSDL, no bindings yet
- Generate platform specific skeleton code using automated tools
- Write the actual program code
- In many cases the server application already exists and the question is how to open an interface to the server application
- Top-down vs. bottom-up, waterfall vs. agile

3. Creating client application
why it may not happen like this

- Pull WSDL definition from somewhere (UDDI)
  - Use only high-level WSDL, no bindings yet
- Generate platform specific stub code using automated tools
- Write the actual program code
- The generated stub code may not be that useful
  - Understanding machine generated code can be harder than human written code
  - Nice applications need developer creativity e.g. for user experience
- Effort of WSDL & UDDI definition vs. effort of writing client interface

WS Criticism

- Quite heavy, lots of specs
- More suited for large well-structured organizations, rather than fast innovation?
- Is there anything new here?
- Are all abstraction layers really needed?

UDDI

- UDDI stands for Universal Description, Discovery and Integration
- XML-based standard for describing, publishing, and finding Web services

- For each service metadata + pointer to its WSDL description

REST

REST

- REpresentational State Transfer
- In 2000 by Roy Fielding in his doctoral dissertation
  - Roy was heavily involved in the specification of HTTP 1.0 and HTTP 1.1

HTTP

```
GET /path/file.html HTTP/1.1
Host: www.host1.com:80
----------
HTTP/1.1 200 OK
Date: Fri, 31 Dec 1999 23:59:59
GMTContent-Type: text/
  plainTransfer-Encoding: chunked
1a; ignore-stuff-
  hereabcdefghijklmnopqrstuvwxyz …
```

REST request

Actions specified with verbs (GET, PUT, POST, DELETE)

Typically uses HTTP

```
GET /v1/posts/recent HTTP/1.1
Host: api.del.icio.us
Authorization: Basic
  dXNlcm5hbWU6cGFzc3dvcmQ=
```

Relies on services of web protocols. E.g. authentication

REST reply

```
<?xml version='1.0' standalone='yes'?>
<posts tag="" user="username">
  <post href="http://www.foo.com/" description="foo"
    extended=""
   hash="14d59bdc067e3c1f8f792f51010ae5ac" tag="foo"
   time="2006-10-29T02:56:12Z" />
  <post href="http://amphibians.com/"
    description="Amphibian Mania"
   extended="" hash="688b7b2f2241bc54a0b267b69f438805"
    tag="frogs toads"
   time="2006-10-28T02:55:53Z" />
</posts>
```

This is just regular XML document
but is also could be JSON or something else

Key Aspects of REST

- Unique IDs for all “things”
- Links tie “things” together
- Standard methods for manipulation
- Resources with multiple representations
- Stateless communication

URIs for all things

Individual items

- `http://example.com/customers/1234`
- `http://example.com/orders/2007/10/776654`

Collections

- `http://example.com/orders/2007/11`
- `http://example.com/products?color=green`

Items linked together

```
<order self='http://example.com/
  customers/1234' >
    <amount>23</amount>
    <product ref='http://example.com/
  products/4554' />
    <customer ref='http://example.com/
  customers/1234' />
</order>
```

Standard methods

- GET (Query the state)
- POST (Modify)
- PUT (Create a resource)
- DELETE (Delete a resource)
- Other verbs are also possible
- The meanings of verbs varies between apps

- The match well with the CRUD approach (Create, Read, Update, Delete)
  - E.g. SQL

Collection URI
e.g. http://example.com/resources/

- GET
  - List the URIs and perhaps other details of the collection's members.
- PUT
  - Replace the entire collection with another collection.
- POST
  - Create a new entry in the collection. The new entry's URL is assigned automatically and is usually returned by the operation.
- DELETE
  - Delete the entire collection.

Element URI,

such as http://example.com/resources/142

- GET
  - Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.
- PUT
  - Replace the addressed member of the collection, or if it doesn't exist, create it.
- POST
  - Treat the addressed member as a collection in its own right and create a new entry in it.
- DELETE
  - Delete the addressed member of the collection

Resources with multiple representations

GET /customers/1234 HTTP/1.1

Host: example.com

Accept: application/vnd.mycompany.customer+xml

GET /customers/1234 HTTP/1.1

Host: example.com

Accept: text/x-vcard

Stateless Communication

- The server does not maintain information about a particular client
- Clients need to take care of this
  - E.g. with normal web mechanisms
- Sessions are not tied to a specific server
  - Scalability, load balancing etc.

REST styles

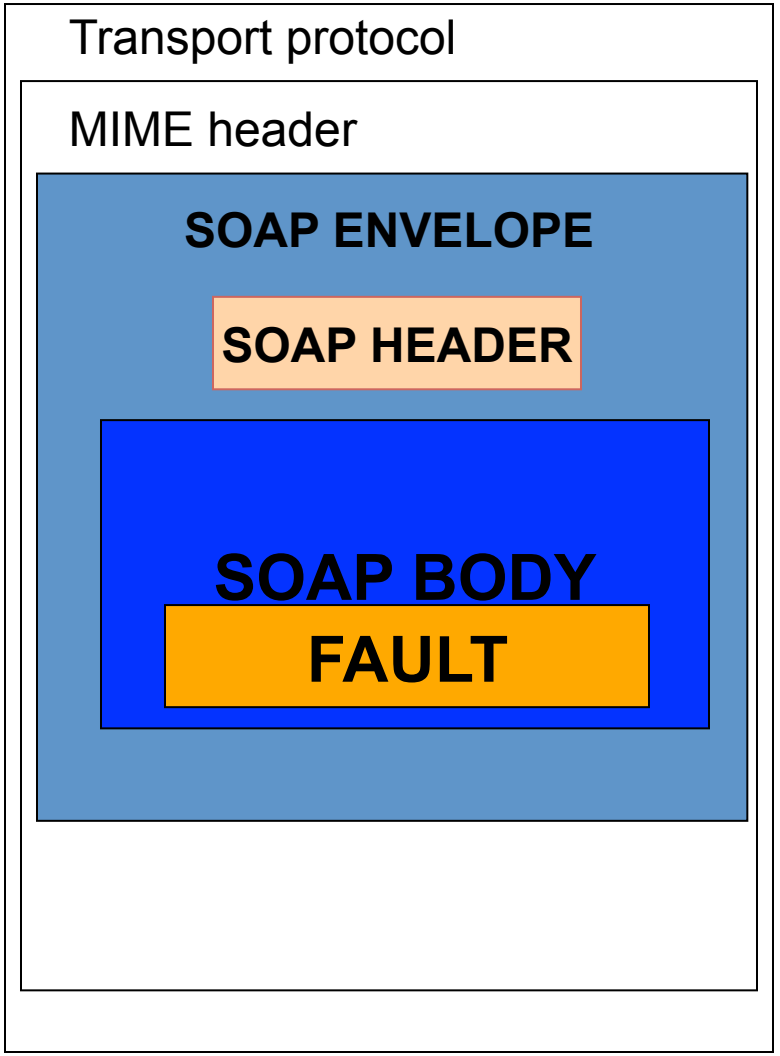
- http://flickr.com/photos/tags/penguin
  - The “proper” way
- http://api.flickr.com/services/rest/?method=flickr.photos.search&tags=penguin
  - The “function call” way
  - Falls in the middle of REST style and remote process call style
  - Not recommended by purists

REST Structure

Transport protocol (HTTP)
VERB (GET, PUT, …)
URI (…/example/item/1234)

+

All the mechanisms of HTTP and WEB are available (security, load balancing, …)



This is SOAP structure for comparison

Web Application Description Language (WADL)

- A bit similar to WSDL
- Allows services to be described in a machine processable way
  - Would allow automatic stub generation
- Major web services seem to rely on human readable documentation of the interfaces rather than on this kind of automated approaches

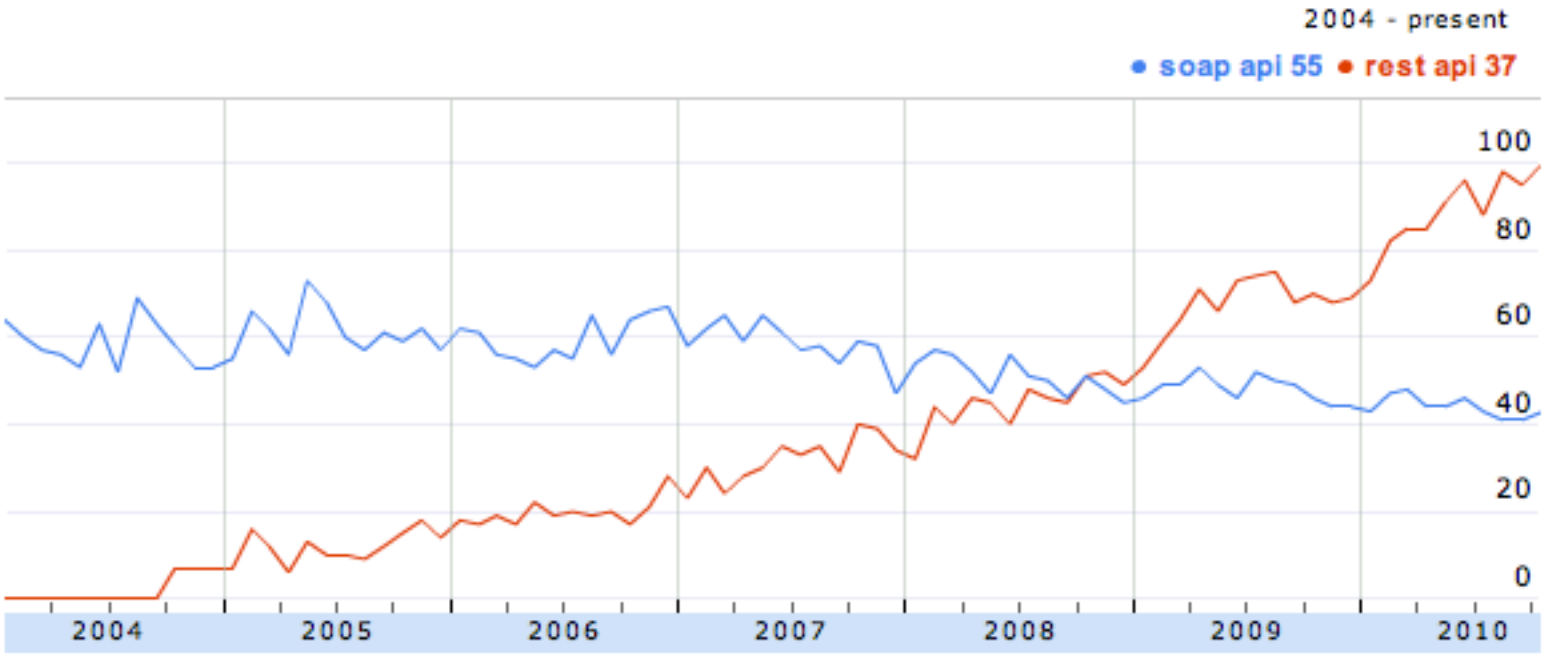
REST vs. SOAP

- Relies other web technologies (e.g. HTTP)
  - E.g. security
- Verb + term focus
- Architectural style

- Has its own mechanisms WS*
- Function call ideology
- Standard (set of)

REST is winning

- 85% clients prefer Amazon RESTful API (http://www.oreillynet.com/pub/wlg/3005)
- In 2007 Google announced it would no longer support its SOAP/WSDL API



Source: Google Insights for Search.

And REST won?

Simon Says...

Simon Phipps

The End Of The Road For Web Services

A decade of corporate politics by the big software companies is finally over as WS-I shuts up shop and gets folded in to OASIS.

Published 07:00, 11 November 10

http://blogs.computerworlduk.com/simon-says/2010/11/the-end-of-the-road-for-web-services/index.htm

- But the legacy enterprise apps will stay for a long time
- And some people are not really convinced what this means

RESTful web services

- Implemented using HTTP and the principles of REST
- A collection of resources, with three defined aspects:
  - the base URI for the web service, such as http://example.com/resources/
  - the Internet media type of the data supported by the web service, typically JSON or XML
  - the set of operations supported by the web service using HTTP methods (e.g., POST, GET, PUT or DELETE).

Summary

- Web Services: let's make machine-callable services using web principles
  - SOAP
  - WSDL
  - UDDI
  - Web Services Stack & a set of WS-* standards
- REST
  - Everything is a URI
  - Actions through verbs (GET, POST, PUT, DELETE, …)
  - Relies on on HTTP and web technologies

Questions / discussion