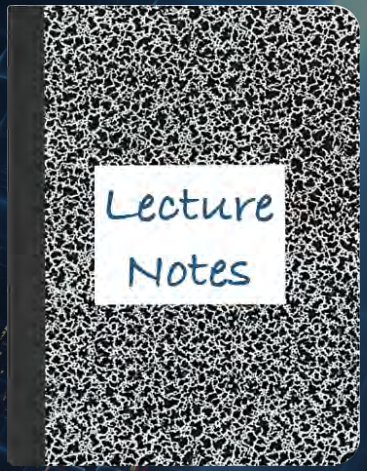
CS 417 – DISTRIBUTED SYSTEMS

Week 2: Part 4

Web Services



Paul Krzyzanowski

From Web Browsing to Web Services

- Web browser:
  - Dominant model for user interaction on the Internet
- Not good for programmatic access to data or manipulating data
  - UI is a major component of the content
  - *Site scraping* is a pain!
- We wanted
  - Remotely hosted services – that programs can use
  - Machine-to-machine communication

Web Services

- We wanted
  - Remotely hosted services – that programs can use
  - Machine-to-machine communication
- Problems
  - Web pages are presentation-focused
  - Traditional RPC solutions usually used a range of ports
    - And we need more than just RPC sometimes
  - Many RPC systems didn't work well across languages
  - Firewalls restrict ports & may inspect the protocol
  - No support for load balancing

RPC Had Problems

Distributed objects mostly ended up in intranets
of homogenous systems and low latency networks

- Interoperability – different languages, OSes, hardware
- Transparency – not really there
  - Memory access, partial failure
- Firewalls – dynamic ports
- State – load balancing, resources
- No group communication – no replication
- No asynchronous messaging
  - Large streaming responses not possible
  - Notifications of delays not possibly
  - No subscribe-publish models

Web Services

Set of protocols by which services can be published, discovered, and used in a technology neutral form

- Language & architecture independent

- Applications will typically invoke multiple remote services
  - **Service Oriented Architecture (SOA)**

Service Oriented Architecture (SOA)

SOA = Programming model

- App is integration of network-accessible services (components)
- Each service has a well-defined interface
- Components are unassociated & loosely coupled

Neither service depends on the other: all are mutually independent

Neither service needs to know about the internal structure of the others

Benefits of SOA

- **Autonomous modules**
  - Each module does one thing well
  - Supports reuse of modules across applications
- **Loose coupling**
  - Requires minimal knowledge – don't need to know implementation
  - **Migration**: Services can be located and relocated on any servers
  - **Scalability**: new services can be added/removed on demand
    … and on different servers – or load balanced
  - **Updates**: Individual services can be replaced without interruption

General Principles of Web Services

- Coarse-grained
  - Usually few operations & large messages
- Platform neutral
  - Messages don't rely on the underlying language, OS, or hardware
  - Standardized protocols & data formats
  - Payloads are text (XML or JSON)
- Message-oriented
  - Communicate by exchanging messages
- HTTP often used for transport
  - Use existing infrastructure: web servers, authentication, encryption, firewalls, load-balancers

Web Services vs. Distributed Objects

Web Services	Distributed Objects
• **Document Oriented** – Exchange documents	• **Object Oriented** – Instantiate remote objects – Request operations on a remote object – Receive results – … – Eventually release the object
• **Document design is the key** – Interfaces are just a way to pass documents	• **Interface design is the key** – Data structures just package data
• **Stateless computing** – State is contained within the documents that are exchanged (e.g., customer ID)	• **Stateful computing** – Remote object maintains state

XML RPC

Origins

- Born: early 1998
- Data marshaled into XML messages
  - All request and responses are human-readable XML
- Explicit typing
- Transport over HTTP protocol
  - Solves firewall issues
- No IDL compiler support for most languages
  - Lots of support libraries for other languages
  - Great support in some languages – those that support introspection (Python, Perl)
- Example: WordPress uses XML-RPC

XML-RPC example

```
<methodCall>
    <methodName>
        sample.sumAndDifference
    </methodName>
    <params>
        <param><value><int> 5 </int></value></param>
        <param><value><int> 3 </int></value></param>
    </params>
</methodCall>
```

XML-RPC data types

- int
- string
- boolean
- double
- dateTime.iso8601
- base64
- array
- struct

Assessment

- Simple (spec about 7 pages)
- Humble goals
- Good language support
  - Little/no function call transparency for some languages
- No garbage collection, remote object references, etc.
  - Focus is on data messaging over HTTP transport
- Little industry support (Apple, Microsoft, Oracle, …)
  - Mostly grassroots and open source

SOAP

SOAP origins

(Simple) (Object) Access Protocol

- Since 1998 (latest: v1.2 April 2007)
- Started with strong Microsoft & IBM support
- Continues where XML-RPC left off:
  - XML-RPC is a 1998 simplified subset of SOAP
  - user defined data types
  - ability to specify the recipient
  - message specific processing control
  - and more …

SOAP

- Stateless messaging model
- Basic facility is used to build other interaction models
  - Request-response (RPC)
  - Request-multiple response
  - Asynchronous notification
- Objects marshaled and unmarshaled to SOAP-format XML
  - Usually sent over HTTP
- Like XML-RPC, SOAP is a messaging format
  - No garbage collection or object references
  - Does not define transport
  - Does not define stub generation

SOAP Web Services and WSDL

- Web Services Description Language
  - Analogous to an IDL

- A WSDL document describes a set of services
  - Name, operations, parameters, where to send requests
  - Goal is that organizations will exchange WSDL documents
    - If you get WSDL document, you can feed it to a program that will generate software to send and receive SOAP messages

WSDL Structure

<definitions>

<types>
data type used by web service: defined via XML Schema syntax
</types>

<message>
describes data elements of operations: parameters
</message>

<portType>
describes service: operations and messages involved
</portType>

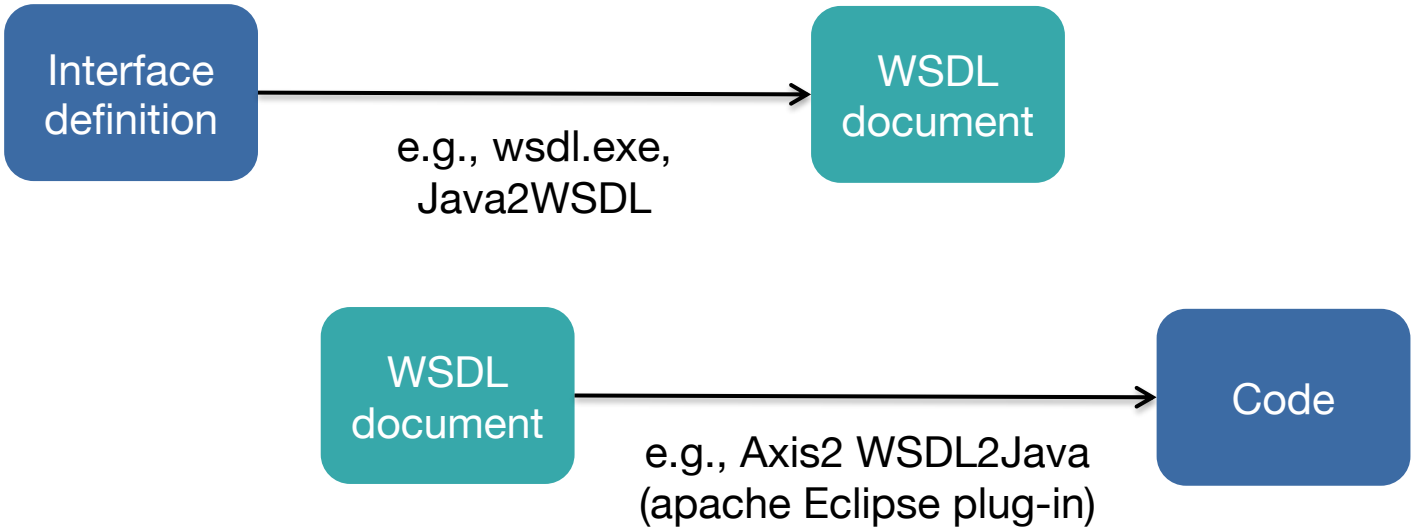
<binding>
defines message format & protocol details for each port
</binding>

</definitions>

Java Web Services

What do we do with WSDL?

It's an IDL – not meant for human consumption



JAX-WS: Java API for XML Web Services

- Lots of them! We'll look at one

- JAX-WS (evolved from earlier JAX-RPC)
  - Java API for XML-based Web-Service messaging & RPCs
  - Invoke a Java-based web service using Java RMI
  - Interoperability is a goal
    - Use SOAP & WSDL
    - Java not required on the other side (client or server)

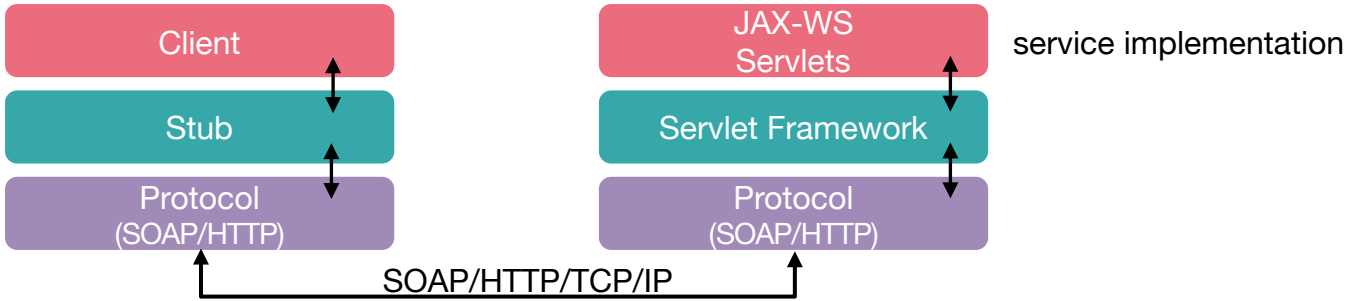
- Service
  - Defined to clients via a WSDL document

JAX-WS: Creating an RPC Endpoint

- Server
  - Define an interface (Java interface)
  - Implement the service
  - Create a publisher
    - Creates an instance of the service and publishes it with a name
- Client
  - Create a proxy (client-side stub)
    - *wsimport* command takes a WSDL document and creates a stub
  - Write a client that creates an instance of the service and invokes methods on it (calling the stub)

JAX-RPC Execution Steps

1. Java client calls a method on a stub
2. The stub creates marshals the request into a SOAP message for the web service
3. The request is sent to the server.
4. Server gets the call and directs it to the framework
5. Framework calls the implementation
6. The implementation returns results to the framework
7. The framework marshals the results into a SOAP message
8. The sends the results back to the client stub
9. The client stub returns the information to the caller



The future of SOAP?

- Still used but…
  - Required a framework – you won't create & parse messages yourself
  - Language support not always great
  - Hard to understand & hard to use in many cases
  - Allegedly complex because "*we want our tools to read it, not people*"
    - *unnamed Microsoft employee*
  - Heavyweight: XML + verbose messaging strutcture

- Dropped by Google APIs in 2006
- Still used in many places, including Microsoft APIs
- But we wanted something lighter and easier

REST

REST

REpresentational State Transfer

- Stay with the principles of the web
  - Four HTTP commands let you operate on data (a resource):
    - **PUT** (create)
    - **GET** (read)
    - **POST** (update)
    - **DELETE** (delete)

    } CRUD: *Create, Read, Update, Delete*
  - And a fifth one:
    - **OPTIONS** (query) - determine options associated with a resource
      - Rarely used … but it's there
- The URL identifies the resource
- Messages contain representation of data (contents)

Resource-oriented services

Blog example

- Get a user's blogroll – a list of blogs subscribed by a user

```
HTTP GET //myblogs.org/listsubs?user=paul
```

- To get info about a specific blog (id = 12345):

```
HTTP GET http://myblogs.org/bloginfo?id=12345
```

Resource-oriented services

- Get parts info

```
HTTP GET //www.parts-depot.com/parts
```

- Returns a document containing a list of parts

```
<?xml version="1.0"?>
<p:Parts xmlns:p="http://www.parts-depot.com"
         xmlns:xlink="http://www.w3.org/1999/xlink">
      <Part id="00345" xlink:href="http://www.parts-depot.com/parts/00345"/>
      <Part id="00346" xlink:href="http://www.parts-depot.com/parts/00346"/>
      <Part id="00347" xlink:href="http://www.parts-depot.com/parts/00347"/>
      <Part id="00348" xlink:href="http://www.parts-depot.com/parts/00348"/>
</p:Parts>
```

Resource-oriented services

- Get detailed parts info:

```
HTTP GET //www.parts-depot.com/parts/00345
```

- Returns a document with information about a specific part

```
?xml version="1.0"?>
<p:Part xmlns:p="http://www.parts-depot.com"
xmlns:xlink="http://www.w3.org/1999/xlink">
      <Part-ID>00345</Part-ID>
      <Name>Widget-A</Name>
      <Description>This part is used within the frap assembly</Description>
      <Specification
xlink:href="http://www.parts-depot.com/parts/00345/specification"/>
      <UnitCost currency="USD">0.10</UnitCost>
      <Quantity>10</Quantity>
</p:Part>
```

REST vs. RPC Interface Paradigms

Example from wikipedia:

RPC

getUser(), addUser(), removeUser(), updateUser(), getLocation(), AddLocation(), removeLocation()

```
exampleObject = new ExampleApp("example.com:1234");
exampleObject.getUser();
```

REST

http://example.com/users

http://example.com/users/{user}

http://example.com/locations

```
userResource =
new Resource("http://example.com/users/001");
userResource.get();
```

Examples of REST services

- Various Amazon & Microsoft APIs
- Facebook Graph API
- Yahoo! Search APIs
- Flickr
- Twitter
- Open Zing Services – Sirius radio

  svc://Radio/ChannelList

  svc://Radio/ChannelInfo?sid=001-siriushits1&ts=2007091103205

- Tesla Cars

  POST https://owner-api.teslamotors.com/api/1/vehicles/vehicle_id/command/flash_lights

AJAX

Web Clients: AJAX

- Asynchronous JavaScript And XML
  - Bring web services to web clients (JavaScript)
- Asynchronous
  - Client not blocked while waiting for result
- JavaScript
  - Request can be invoked from JavaScript (using `XMLHTTPRequest`)
  - JavaScript may also modify the Document Object Model (DOM): the elements of the page: content, attributes, styles, events
- XML
  - Data sent & received as XML … but JSON encodings can also be used

AJAX & XMLHttpRequest

- Allow Javascript to make HTTP requests and process results (change page without refresh)

```
var ajax = new XMLHttpRequest();
ajax.onreadystatechange = function() {
    // stuff to do when the request is ready
}
ajax.open("GET", "http://poopybrain.com/stuff.txt", true);
ajax.send();
```

- Tell object:
  - Type of request you're making, URL to request
  - Function to call when request is made

AJAX on the Web

- AJAX ushered in Web 2.0 – responsive web pages

- Early high-profile AJAX sites:
  - Microsoft Outlook Web Access, Gmail, Google Maps, Writely (Google Docs), …

The End