

# *Distributed Software Development*

## *Web Services*

Chris Brooks

Department of Computer Science

University of San Francisco

## 10-2: Service-Oriented Computing

---

- Services are the hot new model for providing access to data or functionality in a distributed system.
- Like objects, the definition of what a service is depends on who you ask.
- Microsoft: A piece of business logic available via the Internet using open standards.
- DestiCorp: Encapsulated, loosely coupled contracted software functions offered via standard protocols over the Web.
- Gartner: Loosely coupled software components that interact with one another dynamically via standard Internet technologies.
- W3C: A software application identified by a URI, whose interfaces and binding are capable of being defined, described, and discovered by XML artifacts and supports direct interactions with other software applications using XML-based messages via Internet-based technologies.

## 10-3: Service-Oriented Computing

---

- Common threads:
  - Network-available
  - Open standards/protocols
  - Loose coupling
  - Dynamic composition
  - Publically available interface

## 10-4: Services

---

- We can think of a service as (yet another) computing metaphor.
- modular functionality
- Can be composed, extended and combined to provide more complex or extensive services.
  - This ability to compose and repurpose web services is extremely important.
  - Example: Google Maps.
- Defined in terms of an interface

## 10-5: A step back: building on the Web

---

- Currently, the Web consists primarily of client-server interactions.
- Web pages are the predominant objects exchanged.
- No central authority.
- Heterogenous, autonomous components.

## 10-6: Next steps: the Semantic Web

---

- Today's Web is structured for use by humans.
  - HTML focuses on presentation; semantics are inferred through background knowledge.
  - Example: Find the price on an Amazon page.
- Program-program communication on the Web requires more explicit markup.
- A program needs to know the meaning (or semantics) of an element.
- Separation of structure and layout from information.

## 10-7: Next steps: Integrating Processes

---

- Currently, most web-based interactions are stateless.
- Client makes a request and returns a response.
- *Processes* may require long-lived multi-part interactions.
  - Construction of supply chains, complex orders
- Services can be composed into processes.
- Currently, this is done manually.

## 10-8: Next steps: The Pragmatic Web

---

- These combine to yield what Singh and Huhns call *the pragmatic web*.
- Information is annotated according to its meaning and its place in a larger process.
- In other words, information can have annotations that describe *meaning and context*.
- This provides the basis for automated negotiation.
- This is only one possible way in which the Web may evolve, but it is one that will allow for rich, complex, dynamic interactions between software components.

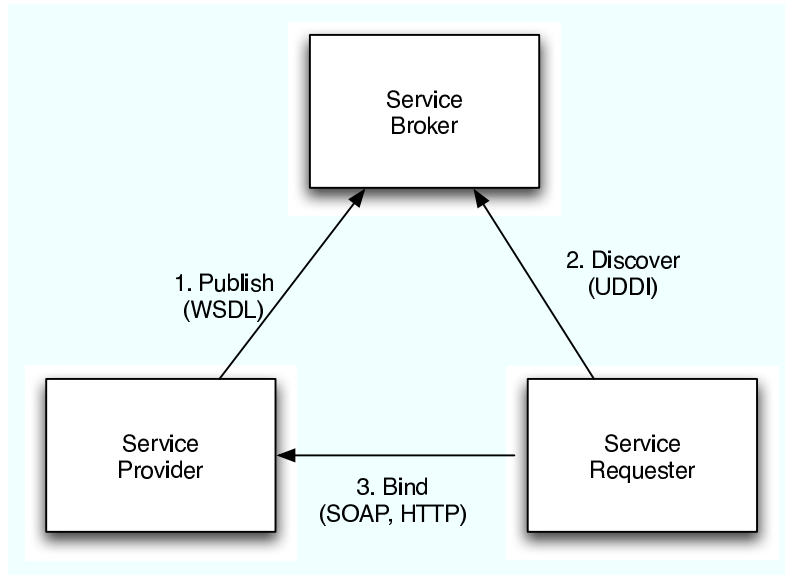
## 10-9: Returning to Services

---

- Processes are being developed, often in an ad hoc way.
- Pragmatics are still primarily a research area.
- Basic standards for automated Web services are being quickly developed.
- First step in incorporating programs as consumers of information, rather than just producers.

## 10-10: Web Services Architectural Model

---



- A Web service architecture consists of three types of participants:
  - Service providers
  - Service brokers
  - Service requesters
- XML is used as the basic 'glue' language.

## 10-11: Basic components of Web Services

---

- HTTP
- SOAP
- XML
- XML Schema
- UDDI

## 10-12: Design concepts

---

- Language and platform independence
- Use existing, scalable technology
- HTTP as transport mechanism
  - Simple request-response semantics
  - Avoids firewall issues.

## 10-13: SOAP

---

- SOAP (Simple Object Access Protocol) is a protocol for exchanging XML messages via HTTP.
- Evolved out of XML-RPC, a protocol for performing remote procedure calls using XML as a message-encoding language.
- SOAP consists of an *envelope* that encodes the message, and a *body* that contains the details of the service being invoked.
- On a request, the body will contain the method being invoked and any parameters.
- On a reply, the body will contain the result of the method's execution.

## 10-14: SOAP request example

---

```
POST /temp HTTP/1.1
Host: www.socweather.com
Content-Type: text/xml; charset="utf-8"
Content-Length: xxx
SOAPAction: "http://www.socweather.com/temp"
<!-- Above: HTTP headers and blank line. This line and below: an XML document
<?xml version=1.0?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  <env:Body>
    <m:GetTemp xmlns:m="http://www.socweather.com/temp.xsd">
      <m:City>Honolulu</m:City>
      <m:When>now</m:When>
    </m:GetTemp>
  </env:Body>
</env:Envelope>
```

## 10-15: SOAP reply example

---

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: xxx
SOAPAction: "http://www.socweather.com/temp"
<?xml version=1.0?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  <env:Body>
    <m:GetTempResponse xmlns:m="http://www.socweather.com/temp.xsd">
      <m:DegreesCelsius>30</m:DegreesCelsius>
    </m:GetTempResponse>
  </env:Body>
</env:Envelope>
```

## 10-16: Limitations of SOAP

---

- Character-oriented
  - Easy to encrypt/decrypt
  - Human readable
  - Large messages, particularly headers
- Stateless, request-response
  - No support for bidirectional or multiparty messages.
  - Content can be added to messages to keep track of state.

## 10-17: Describing Services

---

- If services are to be used in an autonomous fashion by programs, there must be an automated way of describing the service.
  - Service name
  - Operations provided by the service
  - Input parameters and types
  - Output parameters and types
  - Types of errors

## 10-18: WSDL

---

- WSDL (Web Services Description Language) is an XML-based language for describing a programmatic interface to a Web Service.
- Includes:
  - Valid data types
  - a *port type*, which specifies The order in which messages are sent and received.
  - a *binding*, which specifies a document style, an encoding and a transport mechanism
  - a service, which consists of documentation and ports, which bind a service to a URI.

## 10-19: WSDL Example pt 1

---

```
<?xml version="1.0"?>
<!-- the root element defines a set of related services -->
<wsdl:definitions
  name="Temperature"
  targetNamespace="http://www.socweather.com/schema"
  xmlns:ts="http://www.socweather.com/TempSvc.wsdl"
  xmlns:tsxsd="http://schemas.socweather.com/TempSvc.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wSDL="http://schemas.xmlsoap.org/wsdl/">
```

- Set up namespaces

## 10-20: WSDL Example pt 2

---

```
<!-- wsdl:types encapsulates schema definitions of communication types -->
<wsdl:types>
<!-- all type declarations are expressed in xsd -->
  <xsd:schema targetNamespace="http://namespaces.socweather.com"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<!-- xsd def: GetTemp [City string, When string] -->
  <xsd:element name="GetTemp">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="City" type="string"/>
        <xsd:element name="When" type="string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

- Declare types, just as in a schema

## 10-21: WSDL Example, pt 3

---

```
<!-- xsd def: GetTempResponse [DegreesCelsius integer] -->
  <xsd:element name="GetTempResponse">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name="DegreesCelsius" type="integer"/>
      </xsd:all>
    </xsd:complexType>
  </xsd:element>
<!-- xsd def: GetTempFault [errorMessage string] -->
  <xsd:element name="GetTempFault">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name="errorMessage" type="string"/>
      </xsd:all>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
</wsdl:types>
```

## 10-22: WSDL Example, pt 4

---

```
<!-- request GetTempRequest is of type GetTemp -->  
<wsdl:message name="GetTempRequest">  
  <wsdl:part name="body" element="tsxsd:GetTemp"/>  
</wsdl:message>
```

```
<!-- response GetTempResponse is of type GetTempResponse -->  
<wsdl:message name="GetTempResponse">  
  <wsdl:part name="body" element="tsxsd:GetTempResponse"/>  
</wsdl:message>
```

- Describe messages

## 10-23: WSDL Example, pt 5

---

```
<!-- wsdl:portType describes messages in an operation -->
  <wsdl:portType name="GetTempPortType">
<!-- wsdl:operation describes the entire protocol from -->
<!-- input to output or fault -->
  <wsdl:operation name="GetTemp">
<!-- The order of input and output is significant; input -->
<!-- preceding output indicates the request-response -->
<!-- operation type -->
  <wsdl:input message="ts:GetTempRequest"/>
  <wsdl:output message="ts:GetTempResponse"/>
  <wsdl:fault message="ts:GetTempFault"/>
</wsdl:operation>
</wsdl:portType>
```

- Defined the order in which messages are exchanged.

## 10-24: WSDL Example, pt 6

---

```
<wsdl:binding name="TempSvcSoapBinding" type="ts:GetTempPortType">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/
<!-- semi-opaque container of network transport details -->
  <wsdl:operation name="GetTemp">
    <soap:operation soapAction="http://www.socweather.com/TempSvc"/>
<!-- further specify that the messages in the -->
<!-- wsdl:operation "GetTemp" use SOAP? @@@ -->
    <wsdl:input>
      <soap:body use="literal" namespace="http://schemas.socweather.com/Temp
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" namespace="http://schemas.socweather.com/Temp
    </wsdl:output>
    <wsdl:fault>
      <soap:body use="literal" namespace="http://schemas.socweather.com/Temp
    </wsdl:fault>
  </wsdl:operation>
</wsdl:binding>
```

- Bind ports to URIs

## 10-25: WSDL Example, pt 7

---

```
<!-- wsdl:service names a new service "TemperatureService" -->
<wsdl:service name="TemperatureService">
  <wsdl:documentation>socweather.com temperature service
</wsdl:documentation>
<!-- connect it to the binding "TempSvcSoapBinding" above -->
  <wsdl:port name="GetTempPort" binding="ts:TempSvcSoapBinding">
<!-- give the binding a network address -->
    <soap:address location="http://www.socweather.com/TempSvc" />
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

- Declare the service and bind it to the port.

## 10-26: Using WSDL

---

- WSDLs provide a way to automatically construct messages with the same semantics
  - Agreement of the types of elements
  - Agreement on the order of messages
  - Agreement on the types of faults
  - Agreement on the location of services
- Typically generated automatically from a higher-level specification.

## 10-27: Discovery

---

- A fundamental challenge in open distributed systems is the ability to locate services.
- This is done via a *directory service*
- Stores the location and description of services.
  - White pages - index by name
  - Yellow pages - index by service
- Currently searching for services is typically done using keywords, rather than semantic descriptions.

## 10-28: UDDI

---

- UDDI (Uniform Description, Discovery and Integration) describes a mechanism for registering and locating Web services.
- UDDI is in fact a SOAP-based web service.
- Registers:
  - Business Entities - descriptions of all the information surrounding a service provider
  - tModels - descriptions of a particular service

## 10-29: UDDI

---

- UDDI provides two APIs:
  - Inquiry API - allows a service requester to find a service.
  - Publishing API - allows a service provider to register a service.

## 10-30: Tools for generating SOAP and WSDL

---

- SOAP and WSDL are typically not generated by hand.
- Instead, a tool is used to automatically construct the messages or description from some higher-level representation
  - Often a programmatic interface.
- Apache Axis, .NET, JAX-RPC are a few of the tools that can do this.

## 10-31: REST

---

- A criticism of SOAP/WSDL is the low-level procedural focus.
- This is very different from the model that has been successful for the web:
  - A URL/URI refers to a unique resource which can be retrieved via HTTP.
- Focus is on retrieving data referenced with a particular name, as opposed to executing a sequence of operations.
- Rather than specifying how a client should interact with a service, we specify a reference to a data object in the form of a URI.
- Web as a shared information space, rather than as a medium for transporting messages between hosts.

## 10-32: REST

---

- REST stands for Representational State Transfer
  - Idea: Applications work with a representation of a resource (i.e. an XML representation of a song)
  - These representations are shared, or transferred between components.
  - These representations allow client applications to manage their state.
- Data-centric: all services and resources can be referenced with URIs.
- Servers respond to a request by providing a representation of an object.

## 10-33: REST

---

- REST is really more of an architectural model than a protocol.
  - A recipe for building web-scale applications
- In practice, it refers to:
  - encoding requests within an URI
  - using HTTP to deliver them
  - returning results via XML.

## 10-34: REST Philosophy

---

- Current success stories for the Web are: URLs/URIs, HTTP, XML.
- A successful Web services architecture will be built on these.
- The Web should be seen as a distributed, universally indexable, shared information source.

## 10-35: REST vs SOAP

---

- REST sees Web problems as ones of accessing information.
  - HTTP GETs to the most zealous.
  - Providing URIs to access everything allows one to link Web services directly into the rest of the Web. (for example, a Web service can be referred to in an RDF document as an `rdf:resource`)
- SOAP and XML-RPC see Web problems as client-server distributed applications.
  - Users should be able to send and receive complex data
  - REST approach may not fit as nicely into apps that need to change state on the server.

## 10-36: REST vs SOAP

---

- “vs” is not necessarily fair.
  - SOAP is a complex protocol specification.
  - REST is more of an approach.
- We could implement a REST interface on top of SOAP.

## 10-37: REST vs SOAP

---

- There have recently been several “pure REST” services developed.
  - Amazon, Google Maps, Yahoo!, Technorati, ...
- No WSDL or formal protocol specification
- Works very well for quickly developing services.
- Particularly well-suited for querying specific data from a large repository.
- All request information is contained in URL
  - No data typing
  - Encryption must be done at the network level if security is a concern.

## 10-38: REST vs SOAP

---

- SOAP provides the ability to automatically generate code conforming to a WSDL.
- Messages and faults may be validated.
- More complex to develop, but potentially easier to add more complex semantics on top.
  - Transactional semantics
  - Reliable delivery
  - Forwarding of requests

## 10-39: Summary

---

- Web services provide a methodology for discovering, describing, and invoking services.
- Take advantage of existing technology: HTTP, XML, URIs
- SOAP provides a RPC-style mechanism for invoking services.
- WSDL provides a way to formally describe services
- UDDI provides a way to describe and locate services
- REST provides a higher-level URI-based interface to Web Services.