Distributed Systems - SOA & Web Services

P-O Östberg

2007-09-11

1. Service Oriented Architectures
   - Loose Coupling
2. Web Services
   - WSDL
   - SOAP
   - Related Technologies
   - Best Practices

Service Oriented Architectures (SOA)

- A style of building distributed systems where functionality is provided by modular services
- Focuses on *loose coupling* between interacting services (i.e., minimizing formal knowledge between components)
- Services are *virtualized* as much as possible (i.e., focus is placed on interfaces, not implementations)
- Usually built on Web Services (today)

SOA Characteristics

- Logical view - No implementation details are revealed
- Coarse-grained - few operations, large messages
- Platform- (and language-) neutral
- Wide-spread technology base (XML, HTTP, TCP/IP)

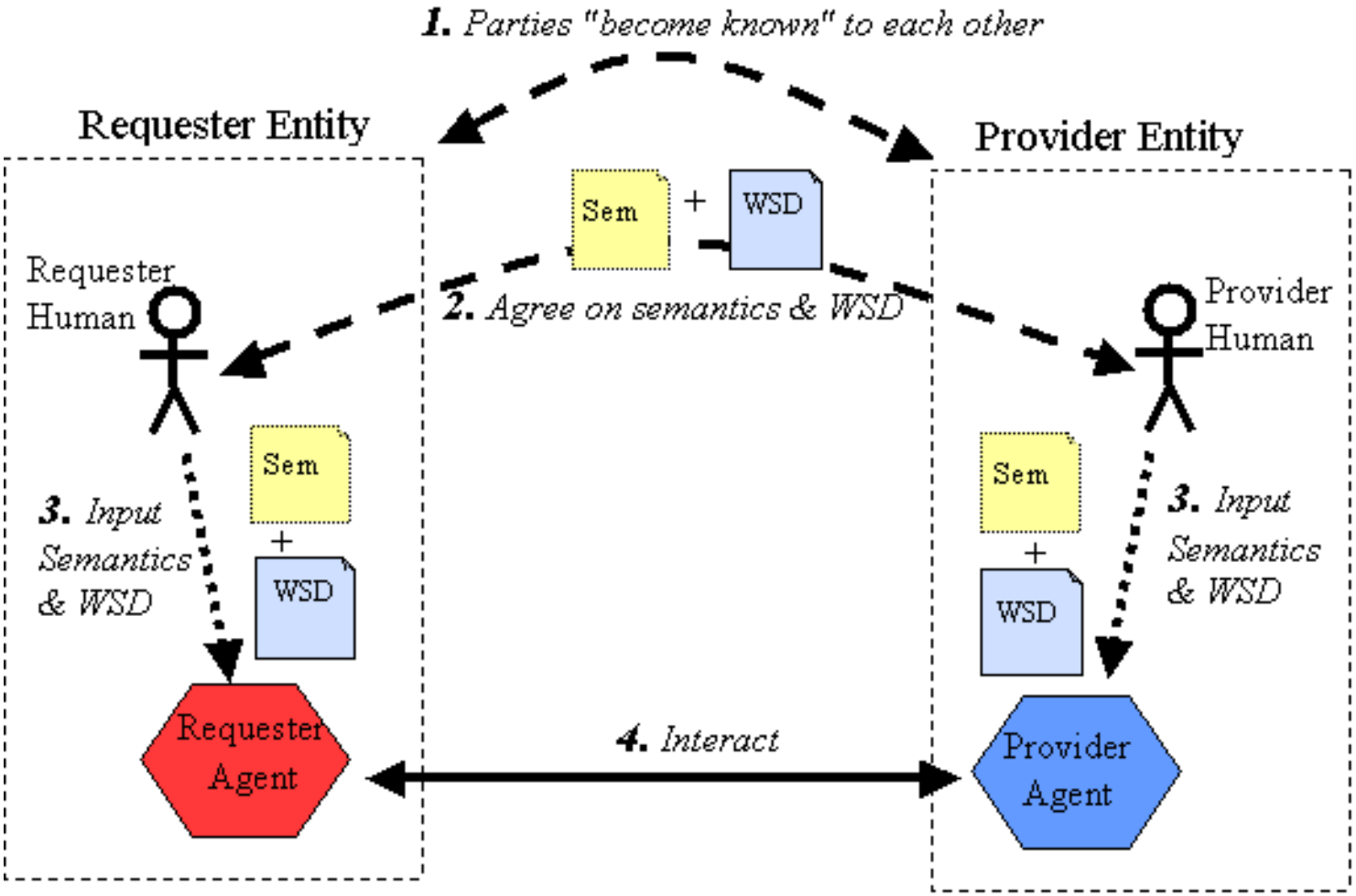
SOA Service Characteristics

- Message-oriented - communicate by exchanging messages
  - abstract - interface defined in terms of messages
  - encapsulated - implementation details hidden
  - technology independent (platform, OS, API etc)
- Self-describing: provides machine-readable metadata (advertises capabilities, service interface, protocols etc)
- Discoverable: dynamic "on-demand" service discovery (includes service location, service interface, protocols etc)

SOA Service Characteristics

- Modular: solves one well-defined task
  - used individually (by different services / applications)
  - can be composed (by other services)
  - facilitates reusability
  - self-contained or dependent on other services / resources
- Interoperable: standardized service access
  - standardized protocols
  - standardized data formats

Interactions



Loose Coupling

- Components minimize built-in knowledge of each other (focus placed on interfaces, not implementations)
- Services are dynamically discovered when needed (includes interfaces, supported protocols, location etc)
- Ideal: zero-coupling ("frictionless") (services used without providing any information)

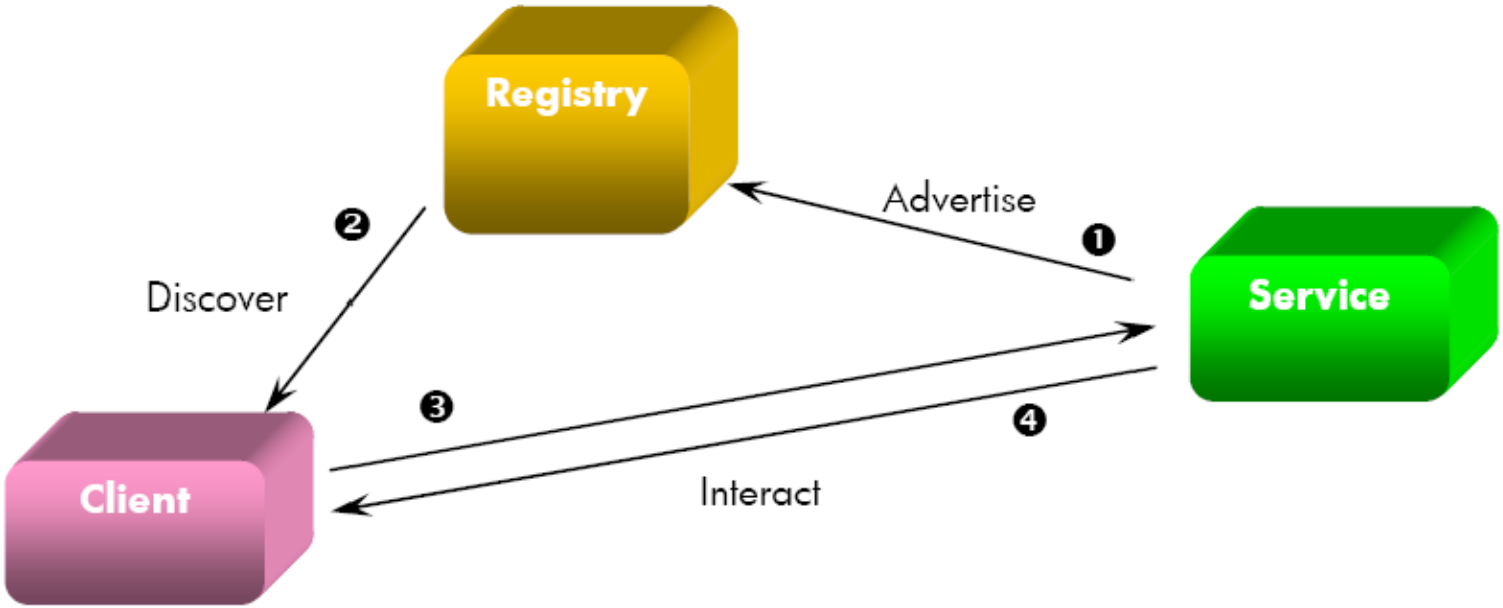
Benefits Of Loose Coupling

- Flexibility: services can be (re)located on any server
- Scalability: services can be added / removed on demand (load balancing)
- Replacability: service implementations can be replaced (without user disruptions)
- Fault tolerance: upon failures, clients can query registries for alternative services offering the same functionality

Publish, Find, Bind

1. Advertisement: service publishes information in a registry
2. Discovery: client queries registry for services
3. Connection establishment: client contacts service
4. Interaction: client and service interact

Publish, Find, Bind



SOA vs Distributed Object Systems

- Distributed object systems (e.g. CORBA, JavaRMI) typically characterized by:
  - objects maintaining a fairly complex internal state
  - fine-grained or "chatty" interaction
  - shared type system and interface hierarchy
  - special-purpose protocols
- Service Oriented Architecture (SOA) typically characterized by:
  - logical view: no implementation details are revealed
  - coarse-grained: few operations, large messages
  - platform and language-neutral
  - widespread technology base (XML, HTTP, TCP/IP)

Web Service

W3C Definition:

"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

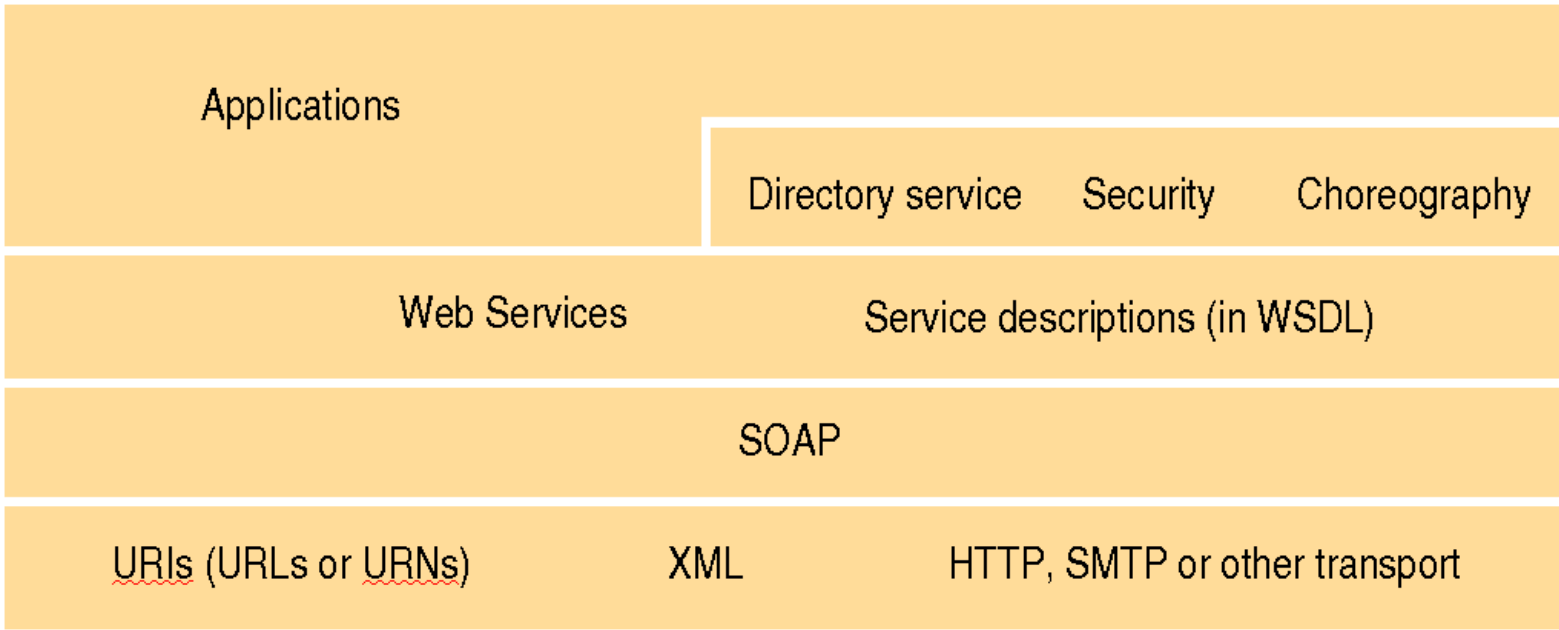
Web Services

- Characterized by:
  - interoperable application-to-application communication
  - abstract interface: implementation details are hidden
  - platform and language neutral, wrapping technology
  - use of widespread and open standards / technology bases (e.g. XML, HTTP, TCP/IP)
  - facilitates loose coupling (particularly relevant for Grid)
- Key specifications (all based on XML):
  - standard means of representing data (XML)
  - standard means of defining service interfaces (WSDL)
  - standard means of invoking services (SOAP)
  - standard means of discovering services (e.g. UDDI)

Web Service

- *Service*: A software component accessed *over a network* that provides functionality to a service requester
- *Web Service*: A service which publishes a service interface in WSDL and uses a message-driven protocol (usually via SOAP / HTTP)
- Built on a host of XML-based technologies
  - XML (data representation)
  - XML Schema (data validation)
  - SOAP (XML-serialized data transfer protocol)
  - WSDL (Web Service interface description, XML Schema)
- Uses a *deployment descriptor* to configure service (XML-based configuration file for the service container)

Web Service Infrastructure



Developing Web Services

- Two main approaches
  - generate WSDL from code
  - generate code (stubs) from WSDL
- Generated WSDL tend to be platform / tool-dependent (quick and easy, but incompatibility issues may arise)
- Generating stubs from WSDL ensures compatibility (but require more work from all parties involved)
- **GOAL: interoperability** (favor the WSDL approach)

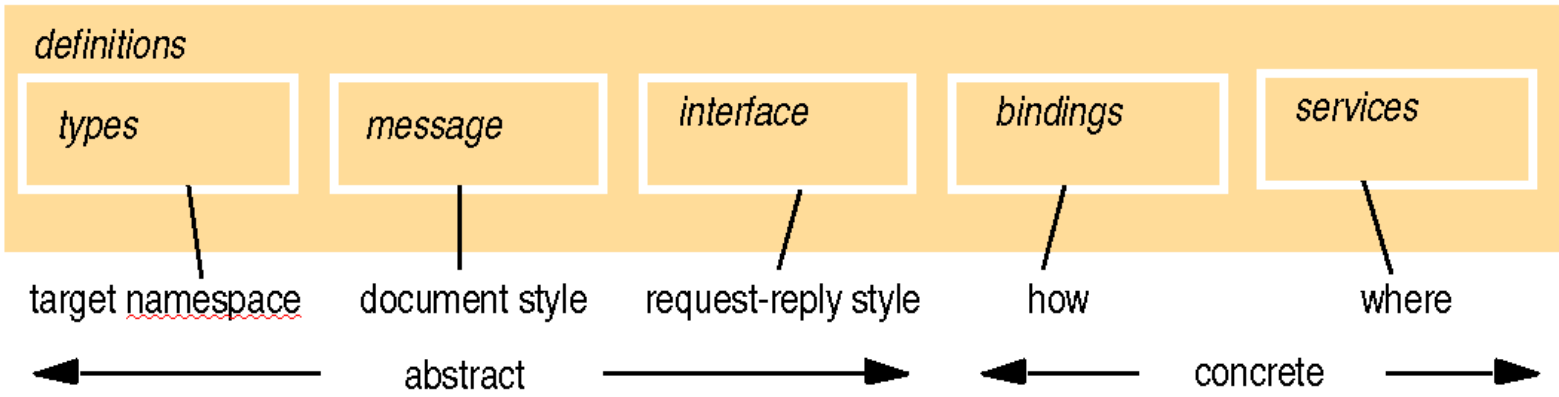
Calling a Web Service

1. Locate Web Service (discovery)
2. Obtain WSDL description
3. Generate stubs from WSDL description
4. Use stubs to invoke Web Service methods

WSDL

- XML Schema-based language for describing Web Services
- Completely describes the Web Service interface
- Constitutes a "contract" between the client and the service
- Can be generated from code, or vice versa
- Two major parts
  - abstract: interface (types, operations and messages)
  - concrete: deployment (encodings, protocols, bindings)

WSDL Structure



WSDL

```
<definitions name="CounterService"
    targetNamespace="http://course.example/Counter"
    xmlns:counter="http://course.example/Counter"
    xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    ...
  </types>

  <message>
    ...
  </message>

  <portType>
    <operation> ... </operation>
  </portType>

</definitions>
```

WSDL Types

```
<types>
  <schema targetNamespace="http://course.example/Counter"
          xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="IncrementRequest">
      <complexType>
        <sequence>
          <element name="Value" type="int"
                   minOccurs="1" maxOccurs="unbounded"/>
        </sequence>
      </complexType>
    </element>
    <element name="IncrementResponse">
      <complexType/>
    </element>

    <element name="GetValueRequest">
      <complexType/>
    </element>
    <element name="GetValueResponse">
      <complexType>
        <sequence>
          <element name="Value" type="int"/>
        </sequence>
      </complexType>
    </element>
  </schema>
</types>
```

WSDL Messages

```
<!-- Message definitions for Increment -->
<message name="IncrementRequestMessage">
  <part name="parameter" element="counter:IncrementRequest"/>
</message>
<message name="IncrementResponseMessage">
  <part name="parameter" element="counter:IncrementResponse"/>
</message>

<!-- Message definitions for GetValue -->
<message name="GetValueRequestMessage">
  <part name="parameter" element="counter:GetValueRequest"/>
</message>
<message name="GetValueResponseMessage">
  <part name="parameter" element="counter:GetValueResponse"/>
</message>
```

WSDL portTypes (interfaces)

```
<portType name="Counter">

  <operation name="Increment">
    <input  message="counter:IncrementRequestMessage"/>
    <output message="counter:IncrementResponseMessage"/>
  </operation>

  <operation name="GetValue">
    <input  message="counter:GetValueRequestMessage"/>
    <output message="counter:GetValueResponseMessage"/>
  </operation>

</portType>
```

SOAP

- Formerly known as *Simple Object Access Protocol*
- XML-based protocol to invoke Web Services (XML-serializes web service requests / responses)
- Usually transported via HTTP (in HTTP body)
- Can send messages
  - point-to-point (directly)
  - via intermediaries (in chains of actors)

SOAP Messages

- Outer layer (e.g., HTTP data)
- Envelope (message root element)
- Header (optional)
  - factorization
  - different recipients (actors)
- Body
  - application specific data (message payload)
  - XML elements
  - Faults (error messages)

SOAP Message

```
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <soap:Body>
    <w:Greeting xmlns:w="http://www.wrox.com/helloworld/">
      <w:message>Hello world!</w:message>
    </w:Greeting>
  </soap:Body>
</soap:Envelope>
```

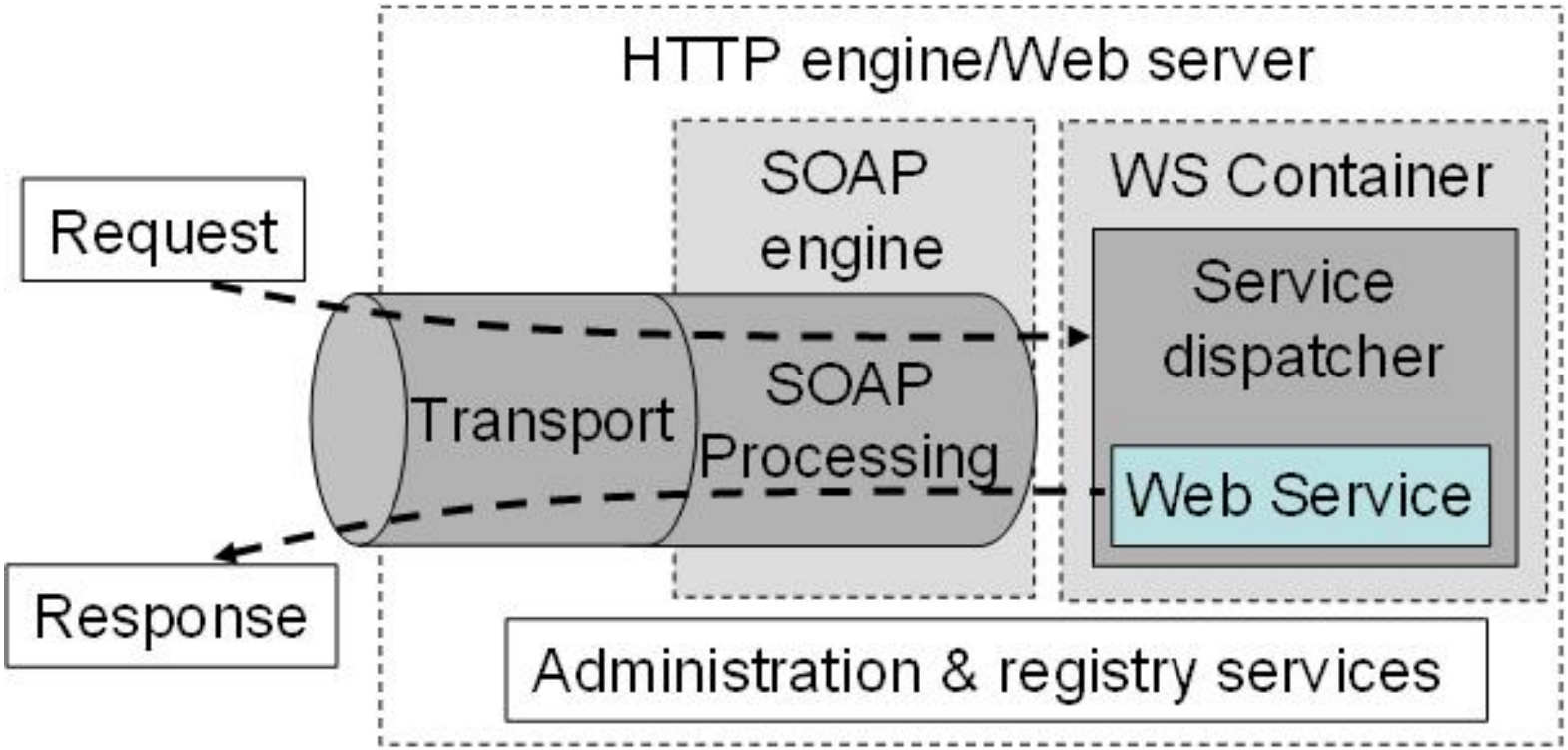
SOAP Faults

- Faults reported in SOAP message body
- Error messages
- Comparable to exceptions in Java
- Fault information
  - `faultcode`: error identifier
  - `faultstring`: human readable identifier
  - `faultactor`: origin of error
  - `detail`: additional fault information

SOAP Fault

```
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Server</faultcode>
      <faultstring>Insufficient funds</faultstring>
      <detail>
        <t:TransferError xmlns:t="http://course.example/transaction">
          <sourceAccount>accountX</sourceAccount>
          <transferAmount>1000.00</transferAmount>
          <currentBalance>910.50</currentBalance>
        </t:TransferError>
      </detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

SOAP Processing



Representational State Transfer (REST)

- Alternative to SOAP for invoking Web Services
- Calls conveyed directly in HTTP bodies
- No extra encoding layers
- Simpler than SOAP
- Less versatile than SOAP

Web Service Resource Framework (WSRF)

- Framework to enable development of stateful Web Services
- Focuses on representations of state: *resources*
- Contains a whole host of specifications
- Provides
  - resource discovery
  - resource addressing
  - resource lifetime management
  - notification (publish / subscribe based state updates)
  - renewable references
  - service groups
  - base fault representations

Comparison

	Java RMI	CORBA	Web Services
Multilingual	No	Yes	Yes
External Data Representation	Object Serialization	CDR	XML
Data format	Binary	Binary	Text (XML/SOAP)
IDL	Java Interfaces	CORBA IDL	WSDL
Type Support	Objects	Primitive & Aggregated	XML Schema
Distributed Garbage Collection	Yes	No	N/A
Binder	RMIRegistry	CORBA Naming Service	UDDI
Bootstrapping	Registry Look-Ups	IOR / Registry Look-Ups	Address / Registry Loo
Call Semantics	At-most-once	At-most-once / Maybe	Configurable

Know When To Use Web Services

- Is really a web service a good solution?
- Would COM/DCOM, Corba, Java RMI etc be a better choice?
- Are there application requirements which are incompatible with web service characteristics?

When the only tool you have is a hammer...

Design In Terms Of Interfaces

- Start by considering what it is the service provides
- Consider the user / client perspective
- Design SOAs in terms of interfaces (not implementations)
- Avoid cross-interface dependencies
- Separate interface and implementation
  (a web service is merely an interface to a software component)

Favor Single-Purpose Services

- A single-purpose software component is..
- Less error-prone
- Easier to develop
- Easier to maintain
- Easier to understand and use
- Most object oriented software design principles are applicable to distributed object models and web services...

Consider Security Implications

- XML is "human readable" / genericly parseable - an eavesdropper can determine whether your data is interesting or not without having to implement a protocol handler
- Web service calls are slow and sometimes computationally intensive (ergo susceptible to DOS attacks)

"Always encrypt everything"

Provide Error Information

- When writing networked services, emphasize robustness
- Provide typed error information (SOAP faults)
  - allows clients to handle errors
- Document everything, especially error behaviors

Provide Version Information

- Providing a unique namespace for each version of the WSDL
  - yields a built-in way to handle the distributed system versioning problem
  - easily done by including a date in the URI for the web service namespace

targetNamespace="http://example.com/2007/09/11/myservice.wsdl"

Enforce Type Checking

- The WSDL type schema provides type checking for your service
- Strict type checking
  - catches client errors early
  - simplifies service error handling
- Well defined WSDL schemas provides information about the intended use of a service

Publish Service WSDL

- ...with the web service
- ...in the service documentation

Offer A Client API

- Allows users without web service experience to use your components
- Demonstrates intended use
- Provides a natural way to group services

Avoid "Chatty Interaction"

- Web service calls can be slow
  - connection establishment
  - transport level encryption
  - message encryption / decryption
  - SOAP serialization
  - XML validation
  - XML parsing
  - ...and the actual Web service logic

Avoid Huge Messages

- Increases server load
- Increases service response time
- Can cause socket timeouts
- Can cause out-of-memory errors (message size * X in parsing)
- Makes for unintelligible interfaces

"Ask not what you can do for WSDL, but what WSDL can do for you"

Summary

- Web Services are
  - accessible over networks
  - technology and platform-independent
  - hosted in service containers (e.g., Apache Axis)
  - accessed through generated stubs or APIs
  - not very efficient
  - very versatile
- Service Oriented Architectures draw up guidelines for (large-scale) deployment of Web Services

Next Time

- Security and PKI