Murdoch
UNIVERSITY

Web Services + Rest (Intermediate) and SOAP

Lecture 9 (A)

Assignment Two

- You will be required to parse and process XML AND JSON documents
- Full details of the requirements are explained in the assignment question
- Each student should submit his/her assignment to the LMS according to the instructions in the assignment question sheet
- AND also has an identical version residing under his/her web home directory on Eris server. **No change is permitted to this version after submission!**

Murdoch UNIVERSITY

Learning Objectives

- Learn about the concept of Web Services, and why it is an important development in Internet technologies
- Learn about the basic set-up for Web Services

Learning Objectives

- In the scheme of what we are doing in this unit:
  - We are studying how to use XML / JSON as important Internet technologies for solutions in different areas
  - It is likely that any work in this industry will involve the use of Web Services
  - This lecture is aimed at learning about the concepts

Lecture Outline

- What are Web Services?
- Components of Web Services
- The Role of XML

From Manual to Automated

- *"Today, the principal use of the World Wide Web is for interactive access to documents and applications. In almost all cases, such access is by human users, typically working through Web browsers, audio players, or other interactive front-end systems. The Web can grow significantly in power and scope if it is extended to support communication between applications from one program to another."*

W3C XML Protocol Working Group Charter.

Murdoch UNIVERSITY

What is a "Web Service"?

- *"A web service is any service that is available over the Internet, uses a standardized XML messaging system, and is not tied to any one operating system or programming language."*

What is a "Web Service"?

- Note, the previous definition states ″a standardized XML messaging system″
  - XML is not the only language used in standardized messaging systems
  - Nowadays, JSON is also commonly used in standardized messaging systems

What is a "Web Service"?

- The options for a "messaging system" are:
  - XML-RPC (Remote Procedure Call)
  - SOAP
  - REST
- We will talk more about SOAP shortly, and cover REST next week

Difference from "Normal" Web

- What is the difference between a Web Service and the outdated "normal" web applications?
  - Eg: dynamic PHP web-sites, shopping carts, etc.
- Answer: "normal" web applications are NOT
  - Self-Describing
  - Discoverable

Difference from "Normal" Web

- For example, with the "normal" web application
  - You can not get your web client to go and find available online applications that do a particular thing (eg: return weather data in London, convert between currencies, …)
  - When you have a new application, no existing programs will be able to use it unless you explicitly tell other programmers about it

Difference from "Normal" Web

- When an application changes its result format, other programs making use of that application cannot automatically detect what those changes are

Towards the Semantic Web

- Development in Web Technologies required a move from
  - Human-centric …
  - To Application-centric …
  - To Automated Web
- Therefore, we have moved toward the concept of the "Semantic Web"

The Semantic Web

- The ***Semantic Web*** is W3C's vision of the Web where:
  - "it becomes a place where data can be shared and processed by automated tools as well as by people."
  - "tomorrow's programs must be able to share and process data even when these programs have been designed totally independently."

The Semantic Web

- “the idea is to have data on the web defined and linked in a way that it can be used by machines not just for display purposes, but for automation, integration and re-use of data across various applications.”
- http://www.w3.org/2001/sw/

Distributed Object Technologies

- There have been past efforts in providing program-level object exchange:
  - OMG's CORBA/IIOP (www.corba.org/)
  - Microsoft's DCOM (www.microsoft.com/com/tech/dcom.asp)
  - Java RMI (java.sun.com/rmi)

Distributed Object Technologies

- Common Object Request Broker Architecture / Internet Inter-Orb Protocol (CORBA/IIOP) is a standard defined by the Object Management Group (OMG)
  - CORBA is a design specification for an Object Request Broker (ORB)
  - ORB provides the mechanism required for distributed objects to communicate with one another, locally or remotely, written in different languages, or in different locations on a network

Distributed Object Technologies

- IIOP (Internet Inter-ORB Protocol) is a protocol that makes it possible for distributed programs written in different programming languages to communicate over the Internet
- IIOP is a critical part of a strategic industry standard, the Common Object Request Broker Architecture (CORBA)

Distributed Object Technologies

- Distributed Component Object Model (DCOM) is a proprietary Microsoft technology for communication among software components distributed across networked computers

Distributed Object Technologies

- The Java Remote Method Invocation (Java RMI) is a Java API that performs remote method invocation with support for direct transfer of serialized Java classes and distributed garbage collection
  - It is the object-oriented equivalent of Remote Procedure Calls (RPC)

Distributed Object Technologies

- The problem is that all of these technologies suffered from:
  - Complex set-up and object invocation
  - Platform and language dependence
  - Lack of universal acceptance
  - Lack of extensibility to different problem areas

A Web Service

- As mentioned previously, a Web service (in very broad terms) is a method of communication between two applications or electronic devices over the World Wide Web (WWW)
- Nowadays, web services use a messaging system of two kinds:
  - Simple Object Access Protocol (SOAP) and
  - REpresentational State Transfer (REST)

A Web Service: SOAP

- SOAP defines a standard set of rules or communication protocol specifications for XML-based message exchange
- SOAP can use different transport protocols, such as HTTP and SMTP

A Web Service: SOAP

- The standard protocol HTTP makes it easier for the SOAP model to tunnel across firewalls and proxies without any modifications to the SOAP protocol
- SOAP can sometimes be slower than middleware technologies like CORBA due to its verbose XML format

A Web Service: REST

- REST describes a set of architectural principles by which data can be transmitted over a standardized interface (such as HTTP)
- REST does not contain an additional messaging layer and focuses on design rules for creating stateless services

A Web Service: REST

- A client can access a resource using the unique URI and a representation of the resource is returned
- With each new resource representation, the client is said to transfer state

A Web Service: REST

- While accessing RESTful resources with HTTP protocol, the URL of the resource serves as the resource identifier and GET, PUT, DELETE, POST and HEAD are the standard HTTP operations to be performed on that resource
- We will cover RESTful Web Services in more detail next week

Support for Web Services

- Tools for programming and supporting Web Services are appearing under all platforms and programming languages
- Currently:
  - SOAP, WSDL and UDDI are still in use today
  - Development has moved away from XML-RPC and SOAP towards REST
  - REST has become increasingly more wide-spread
  - However, it is still important to understand XML-based Web Services

Importance of XML

- The use of XML is an essential part of building SOAP-based Web Services
- The concepts of self-description and discovery is not easily implementable without wide support for XML

Advantages of XML

- Think about what XML offers:
  - Clients and servers know unambiguously where to retrieve the needed data
  - Clients and servers can determine if they are communicating the same information by looking at what namespace they are using
  - It can be determined unambiguously where the location of the service is

Advantages of XML

- What goes into requests and responses can be extended in the future, and those changes can be detected automatically by looking at the XML and Schema
- There is so much existing support for processing XML documents in different platforms and programming languages
- One can integrate the content of the messages with other XML content from other applications or domains

Security Problems

- Security for Web Services still needs a lot of attention and development
  - For example, most SOAP services binds to (i.e., use) HTTP for transport
  - That means a whole set of behaviours can be invoked that firewalls cannot currently distinguish (and appropriately block) from normal web server-client traffic

Motivations for Web Services

- Why not stick with “normal” programs as a basis for providing services?
- Answers - without Web Services:
  - Programs are not accessible over the Internet
  - Programs must be developed for a particular Operating System and particular languages
  - There is no program-to-program communication

Motivations for Web Services

- Why not stick with familiar HTTP / HTML / CGI as a basis for providing services?
- Answers - using such languages, there is:
  - No program-to-program communication
  - Services are not self-describing
  - Services are not discoverable
  - Services are not easily integrated

Web Services at W3C

- W3C's work in Web Service standards has in the past been concentrated on:
  - XML Protocol
    - Defining the various layers of Web Services, and how they fit together
  - Refinement of SOAP
    - Security extensions, Attachments, etc.

Web Services at W3C

- W3C's work in Web Services standards has also concentrated on:
  - WSDL
    - From Working Draft to full Recommendation
  - Web Service Choreography
    - Defining how to describe the interactions between service requester and provider

Web Service Developers

- Developers of Web Services were split mainly into two camps:
  - J2EE
  - Microsoft .NET
- However, organizations hedged their bets on both camps

Other Environments

- There has also been support for Web Services development under various applications platforms such as Oracle databases, IBM WebSphere, etc.

Ideal versus Reality

- The ideal of Web Services is to be platform and environment independent
  - It shouldn't matter how you developed the services, they will be able to communicate with each other once they are deployed
  - J2EE and Microsoft .NET, as well as all other development environments, would implement standard SOAP, WSDL, UDDI and XML

Ideal versus Reality

- However, the reality is that the development platform is critical due to:
  - Cost of tools
  - Ease and cost of development
  - Ease and cost of deployment
  - Skill set of available personnel
- So the ideal situation is not easily achieved in reality

Closing the Gap

- The difference in benefits between the J2EE and .NET is no longer as obvious as it was a few years ago
  - Complex interplay of many factors
  - Which one is better in **total** cost of development and deployment is disputable

Components of A Web Service

- In a crude way, the early Web had three components in the form of:
  - Clients
  - Servers
  - Search Engines
- The problem was that the CONTENTS between clients, servers, and search engines were not precisely defined
- Thus that situation was not good enough for automated Web

Components of A Web Service

- The more recent Web Service infrastructure consists of:
  - Providers
  - Requesters
  - Registries
- In some ways these relate to clients, servers, and search engines (respectively)

Web Service Protocol Stack

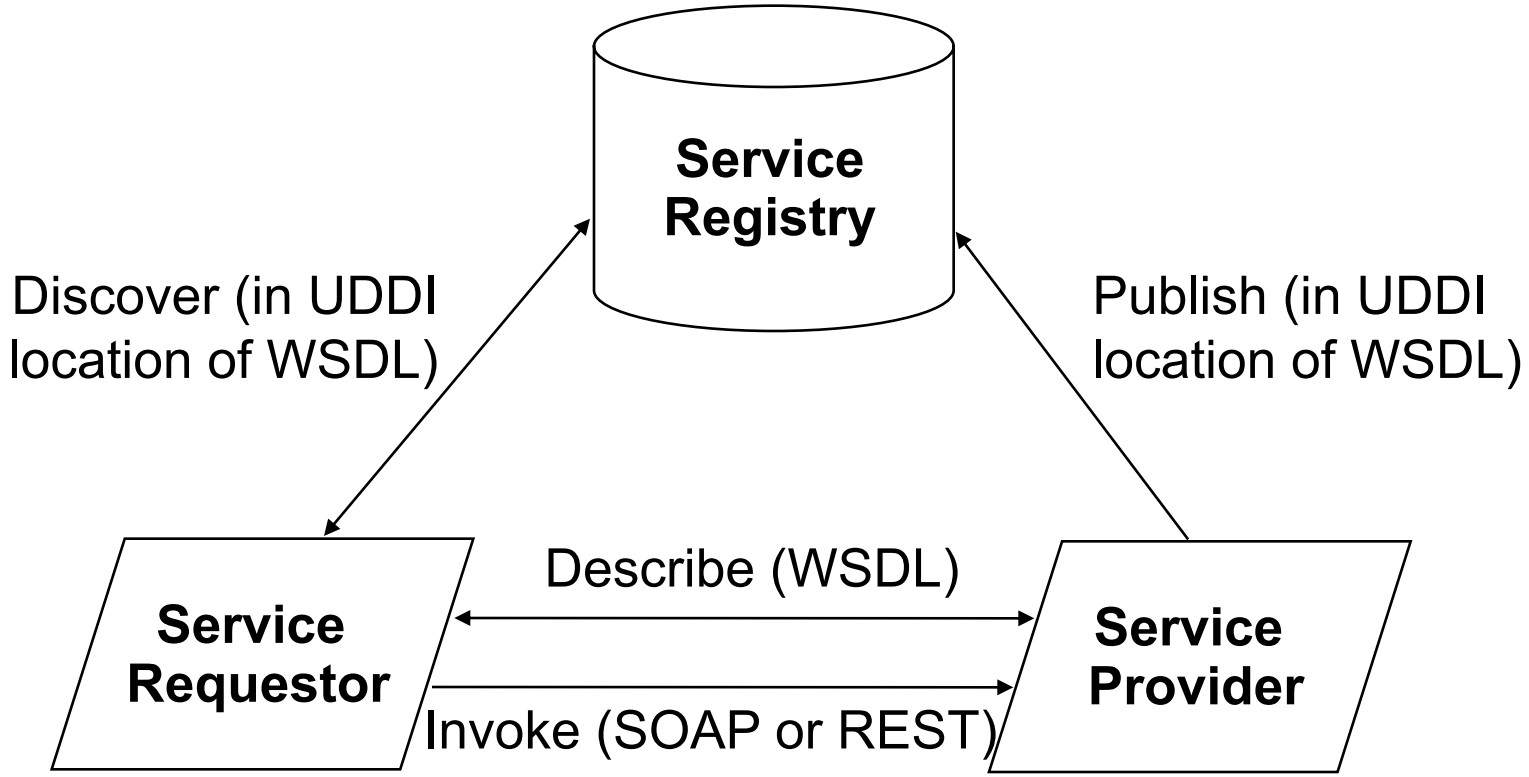
Discovery: UDDI – Universal Description Discovery and Integration

Description: WSDL – Web Service Definition Language

Messaging: XML-RPC (Remote Procedure Call)
SOAP (Simple Object Access Protocol)
REST (REpresentational State Transfer)

Transport: HTTP – Hypertext Transfer Protocol
SMTP – Simple Mail Transfer Protocol

Example Web Service Infrastructure



Example Web Service Infrastructure

- The concept of the communication is:
  - The service provider creates a WSDL description for a new (SOAP or REST) service
  - The service provider publishes the location of the WSDL description (i.e., the service providers location) in a UDDI registry
  - Service requestors find out about the existence of the service in the UDDI registry (i.e., they search the UDDI for web services of interest to them)
  - The service requestors use the location retrieved from the UDDI to get the WSDL description from the service provider; thus they discover the web service details
  - The requestors connect to the service provider using SOAP or REST and then use the service

Example WSDL for a Weather Service

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="WeatherService"
  targetNamespace="http://www.ecerami.com/wsdl/WeatherService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ecerami.com/wsdl/WeatherService.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <message name="getWeatherRequest">
    <part name="zipcode" type="xsd:string"/>
  </message>
  <message name="getWeatherResponse">
    <part name="temperature" type="xsd:int"/>
  </message>
  ...
  <service name="Weather_Service">
    <documentation>WSDL File for Weather Service</documentation>
    <port binding="tns:Weather_Binding" name="Weather_Port">
      <soap:address location="http://localhost:8080/soap/servlet/rpcrouter"/>
    </port>
  </service>
</definitions>
```

Example from section 1.4 in Web Service Essentials by Ethan Cerami, O'Reilly, 2002

Example SOAP Request

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <SOAP-ENV:Body>
      <ns1:getWeatherRequest
         xmlns:ns1="urn:examples:weatherservice"
         SOAP-ENV:encodingStyle="http://www.w3.org/2001/09/soap-
encoding/">
         <zipcode xsi:type="xsd:string">10016</zipcode>
      </ns1:getWeatherRequest>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Asking for weather data…

… for zipcode 10016

Example from section 1.3.2 in Web Service Essentials by Ethan Cerami, O'Reilly, 2002

Murdoch UNIVERSITY

Example SOAP Response

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <SOAP-ENV:Body>
      <ns1:getWeatherResponse
         xmlns:ns1="urn:examples:weatherservice"
         SOAP-ENV:encodingStyle="http://www.w3.org/2001/09/soap-
encoding/">
         <temperature xsi:type="xsd:int">65</temperature>
      </ns1:getWeatherResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Returning the response…

… with the weather data

Example from section 10.2 in Web Service Essentials by Ethan Cerami, O'Reilly, 2002

References

- Current developments in standards and protocols
  - http://www.w3.org/2002/ws/
- J2EE
  - http://java.sun.com/j2ee/
- Microsoft's .NET
  - http://www.microsoft.com/net/
  - http://www.microsoft.com/NET/Basics.aspx

Murdoch
UNIVERSITY

SOAP, WSDL, and UDDI

Lecture 9 (B)

Learning Objectives

- Learn more details of the three components of XML Web Services
- Clarify understanding of SOAP messages by looking at an example SOAP service
- Learn the basics of using the Perl module SOAP::Lite to implement a SOAP service

Lecture Outline

- Components of Web Services:
  - SOAP
  - WSDL
  - UDDI
- What is in a SOAP Message?
- A demonstration of a simple SOAP service

Components of Web Services

- In an XML-based Web Service, we
  - Describe a service using the **Web Services Description Language** (WSDL)
  - Define a way to publish and discover information about Web services using **Universal Description, Discovery and Integration** (UDDI)
  - Invoke the service using **Simple Object Access Protocol** (SOAP)

Web Service Component: SOAP

- Simple Object Access Protocol (SOAP) is an XML-based messaging and remote procedure call specification that enables the exchange of information among distributed systems
- SOAP is a public standard defined by W3C:
  - http://www.w3.org/TR/soap
  - http://www.w3.org/TR/soap12

Do not link to this page - Use dated versions of the documents

Latest SOAP versions

This page (http://www.w3.org/TR/soap) contains links to the SOAP/1.1 Note and the SOAP Version 1.2 Recommendation documents.

For information about the latest work on SOAP and a full list of SOAP specifications , please refer to the W3C XML Protocol Working Group and the list of W3C Technical Reports.

SOAP Version 1.2

Latest version of SOAP Version 1.2 specification: http://www.w3.org/TR/soap12

W3C Recommendation (Second Edition) 27 April 2007

SOAP Version 1.2 Part0: Primer
http://www.w3.org/TR/2007/REC-soap12-part0-20070427/ (errata)
SOAP Version 1.2 Part1: Messaging Framework
http://www.w3.org/TR/2007/REC-soap12-part1-20070427/ (errata)
SOAP Version 1.2 Part2: Adjuncts
http://www.w3.org/TR/2007/REC-soap12-part2-20070427/ (errata)
SOAP Version 1.2 Specification Assertions and Test Collection
http://www.w3.org/TR/2007/REC-soap12-testcollection-20070427/ (errata)

Please refer to the **errata** for these documents, which may include some normative corrections.

This document is a Recommendation of the W3C. This document has been produced by the XML Protocol Working Group, which is part of the Web Services Activity. It has been reviewed by W3C Members and other interested parties, and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

Simple Object Access Protocol (SOAP) 1.1

W3C Note 08 May 2000

Web Service Component: SOAP

- SOAP is a communication protocol designed to communicate via the Internet
- SOAP can extend HTTP for XML messaging
- SOAP provides data transport for Web services
- SOAP can exchange complete documents or call a remote procedure

Web Service Component: SOAP

- SOAP can be used for broadcasting a message
- SOAP is platform-independent and language-independent
- SOAP is the XML way of defining what information is sent and how
- SOAP enables client applications to easily connect to remote services and invoke remote methods

Web Service Component: SOAP

- Although SOAP can be used in a variety of messaging systems and can be delivered via a variety of transport protocols, the initial focus of SOAP was remote procedure calls transported via HTTP
- Frameworks including CORBA, DCOM, and Java RMI provide similar functionality to SOAP, but SOAP messages are **written entirely in XML** and are therefore platform- and language-independent

SOAP Communication

- In SOAP communication, the parties involved are called SOAP **nodes**
- Nodes send SOAP **messages** to each other, in a one-way transmission from a **sender** to a **receiver**
- There may be **intermediaries** between the sender and the receiver, because transmission occurs over the Internet

SOAP Communication

- SOAP communication combines the following underlying message transmissions to implement complex interactions, from:
  - Single message with no response, to …
  - Single request-response exchange, to …
  - Multiple back-and-forth "conversations"

SOAP Communication

- SOAP 1.1 defined 4 explicit client / server request / response-type interactions
- SOAP 1.2 moves away from this terminology, to more general message exchange models of any type

Single Request-Response

- Because a lot of SOAP services depend on HTTP, we see a lot of these services following the simple single request-response exchange
- Our exercises for lab 10 involve only services of this type
  - A client asks for something – the server sends something back - end of exchange
  - Thus, we use SOAP::Lite in our tutorial scripts; this is supported by SOAP version 1.1

Example SOAP Service

- Let's look at a hypothetical (and very simplistic) service that allows us to query information about a University unit
  - The requester sends a unit code, and the provider responds with the unit title
- What would requests and responses from a client to a server in this service look like?

Example SOAP Request (in essence)

```
<Envelope>
  <Body>
    <getUnitInfo>
      <unit_code>ICT375</unit_code>
    </getUnitInfo>
  </Body>
</Envelope>
```

Example SOAP Request

```
<?xml version='1.0' encoding='UTF-8'?>
<soap:Envelope
   xmlns:soap=http://www.w3.org/2001/09/soap-envelope
   soap:soapenc=" http://www.w3.org/2001/09/soap-encoding">
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <soap:Body>
      <getUnitInfo xmlns="urn:examples:unitinfoservice">
         <unit_code xsi:type="xsd:string">ICT375</unit_code>
      </getUnitInfo>
   </soap:Body>
</soap:Envelope>
```

Example SOAP Response (in essence)

```
<Envelope>
  <Body>
    <getUnitInfoResponse>
      <unit_name>
          Advanced Web Programming
      </unit_name>
    </getUnitInfoResponse>
  </Body>
</Envelope>
```

Example SOAP Response

```
<?xml version='1.0' encoding='UTF-8'?>
<soap:Envelope
   xmlns:soap=http://www.w3.org/2001/09/soap-envelope
   soap:soapenc="http://www.w3.org/2001/09/soap-encoding">
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <soap:Body>
      <getUnitInfoResponse
         xmlns="urn:examples:unitinfoservice">
         <unit_name xsi:type="xsd:string">
             Advanced Web Programming
         </unit_name>
      </getUnitInfoResponse>
   </soap:Body>
</soap:Envelope>
```

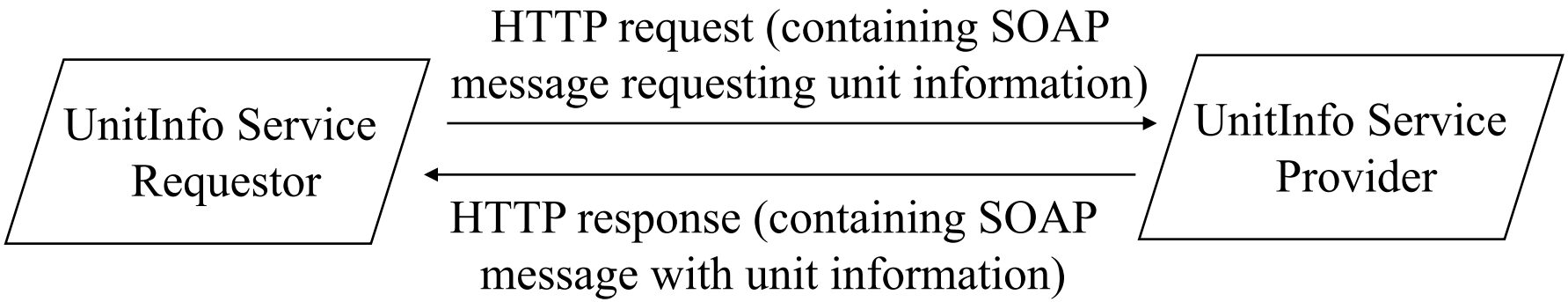
Murdoch UNIVERSITY

Binding to Different Transport Protocols

- SOAP does not define how the messages are to be transported
- SOAP must **bind** to certain messaging protocols (eg: HTTP, SMTP) to transport the messages
- It is up to the implementer of the SOAP service to decide which protocol can best support their service interaction

Example: SOAP over HTTP

- Two nodes sending request / response information to each other



Example SOAP Request In HTTP Message

HTTP Message Headers

```
POST  /ws/UnitInfoService  HTTP/1.1
Host: 134.115.64.2
Content-type: text/xml
Content-length: 300
SOAPAction: urn:examples:unitinfoservice#getUnitInfo
```

HTTP Message Body (SOAP request from previous example)

```
<?xml version='1.0' encoding='UTF-8'?>
<soap:Envelope
   xmlns:soap=http://www.w3.org/2001/09/soap-envelope
   soap:soapenc="http://www.w3.org/2001/09/soap-encoding">
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <soap:Body>
      <getUnitInfo xmlns="urn:examples:unitinfoservice"
         <unit_code xsi:type="xsd:string">ICT375</unit_code>
      </getUnitInfo>
   </soap:Body>
</soap:Envelope>
```

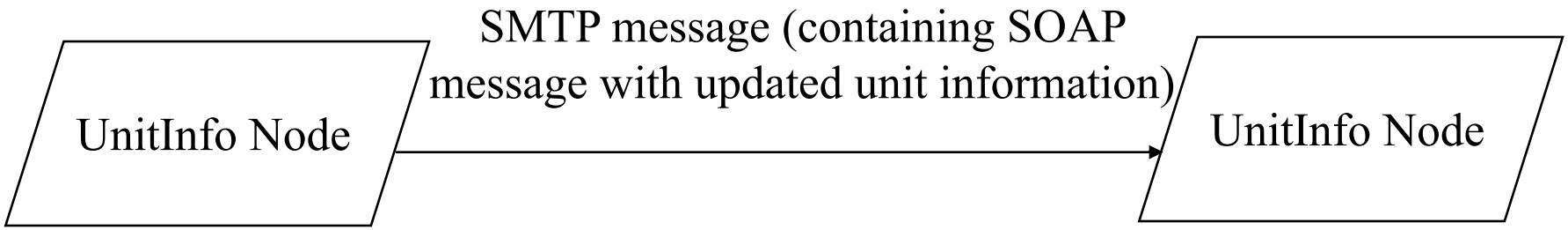
Murdoch UNIVERSITY

Binding to Other Protocols

- Although binding to HTTP is the most prevalent model, we can bind to other protocols as well
  - For example, there is a W3C Note describing SOAP binding over Email (such as SMTP)
- Sometimes it may be appropriate to bind to a protocol other than HTTP
  - For example, we may **not** want any response to the message we send to the receiver

Example: SOAP over SMTP

- One node sending update information to another node



Web Service Component: WSDL

- Web Service Description Language (WSDL) is an XML-based language to describe the operations of a web service
- Web Service servers can publish WSDL documents to enable clients to read and determine how to use the service

Example WSDL

Input/output description

Service location description

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="UnitInfoService"
   targetNamespace="http://www.pieinthesky.edu.au/UnitInfo.wsdl"
   xmlns:tns="http://www.pieinthesky.edu.au/UnitInfo.wsdl"
…>

   <message name="getUnitInfo">
      <part name="unit_code" type="xsd:string" />
   </message>
   <message name="getUnitInfoResponse">
      <part name="unit_name" type="xsd:string" />
  </message>

   <portType name="UnitInfo_PortType">
      <operation name="getUnitInfo">
            <input message="tns:getUnitInfo" />
            <output message="tns:getUnitInfoResponse" />
      </operation>
   </portType>

   <service name="UnitInfo_Service">
      <documentation>WSDL File for Unit Info Service</documentation>
      <port binding="tns:UnitInfo_Binding" name="UnitInfo_Port">
         <soap:address location="http://localhost:8080/soap/servlet/rpcrouter" />
      </port>
   </service>
  …
</definitions>
```

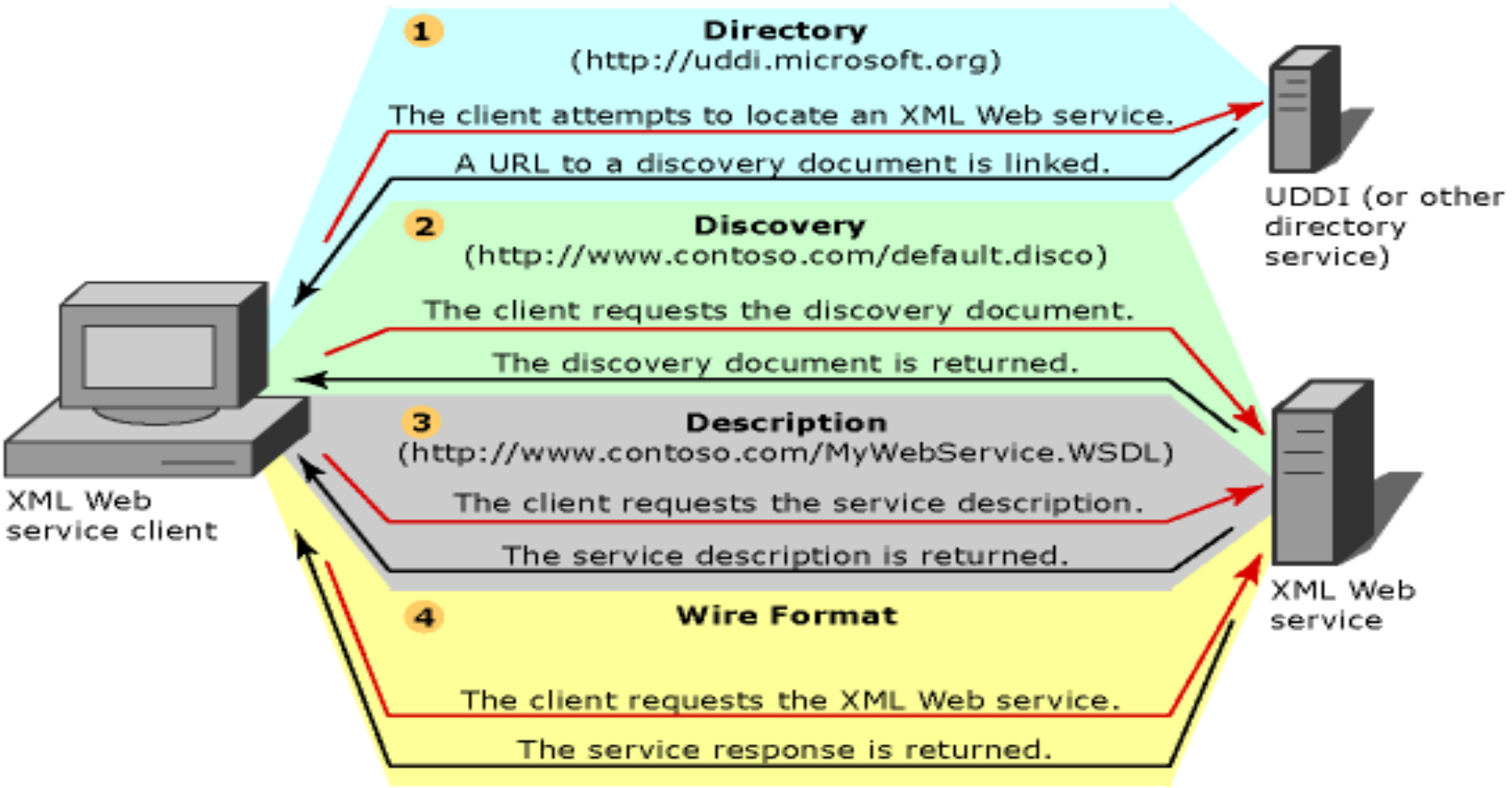
Web Service Component: UDDI

- Universal Description, Discovery and Integration (UDDI) is a platform-independent, open framework for describing and discovering businesses, and integrating business services
- Web Service companies publish themselves in UDDI **registries** (announcing they have Web Services to be discovered)

Web Service Component: UDDI

- Web Service clients (requestors) query the company to determine services
- Clients retrieve and read the WSDL document for the appropriate service (from the provider)
- Clients then invoke the service according to how it is described in the WSDL document

Web Service Example



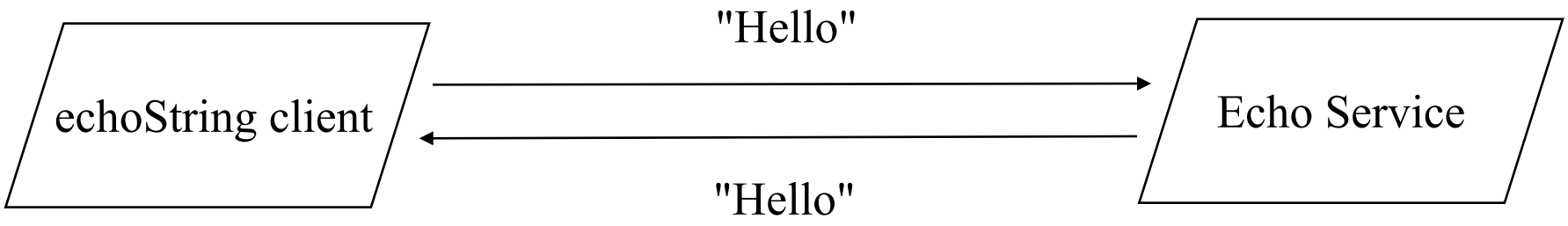
Source: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconwebservicesinfrastructure.asp

UDDI Registries

- There are UDDI registries maintained by companies like Microsoft and IBM
- UDDI registries are loosely coupled after the DNS system
  - Each registry should synchronize their data with other registries
  - Like the DNS registries, UDDI registries do not contain the service information themselves
  - They provide information on where and how to find and invoke the services

Example: A Simple SOAP Service

- A demonstration of the Echo service you will be implementing in tutorial 10



Simplified Request

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope>
  <Body>
    <echoString>
      <inputString type="string">Hello</inputString>
    <echoString>
  </Body>
</Envelope>
```

Request From echoString Client To Echo Server


```
POST http://ceto.murdoch.edu.au:8888 HTTP/1.1
Accept: application/soap
Content-Length: 478
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://soapinterop.org/"


<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <echoString xmlns="http://soapinterop.org/">
            <inputString xsi:type="xsd:string">hello</inputString>
        </echoString>
    </soap:Body>
</soap:Envelope>
```

Murdoch UNIVERSITY

Simplified Response

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope>
  <Body>
    <echoStringResponse>
      <return type="string">Hello</return>
    </echoStringResponse>
  </Body>
</Envelope>
```

Response From Echo Server To echoString Client


```
HTTP/1.1 200 OK
Server: libwww-perl-daemon/6.01
Content-Length: 484
Content-Type: text/xml; charset=utf-8
Client-Date: Wed, 28 Apr 2021 16:06:03 GMT
Client-Peer: 134.115.4.185:8888
Client-Response-Num: 1
SOAPServer: SOAP::Lite/Perl/1.27

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <echoStringResponse xmlns="http://soapinterop.org/">
            <return xsi:type="xsd:string">hello</return>
        </echoStringResponse>
    </soap:Body>
</soap:Envelope>
```

Murdoch UNIVERSITY

SOAP Envelope

- The SOAP `<Envelope>` element consist of two parts:
  - `<Header>` - optional
    - Contains control information – eg: to be used by intermediaries between sender and receiver
  - `<Body>` - mandatory
    - Contains content used for the service

SOAP Faults

- All generated errors are reported in the `<Fault>` element
  - The `<Fault>` element in the following Echo service example is one for SOAP v1.1

Simplified Fault

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope>
  <Body>
    <Fault>
      <faultcode type="string">
         soap:Client
      </faultcode>
      <faultstring type="string">
         SOAPAction should be "http://soapinterop.org/"
      </faultstring>
      <faultactor type="string">
         http://ceto.murdoch.edu.au:12345/
      </faultactor>
    </Fault>
  </Body>
</Envelope>
```

```
HTTP/1.1 500 Internal Server Error
Date: Wed, 28 Apr 2021 17:23:39 GMT
Server: libwww-perl-daemon/6.01
Content-Length: 542
Content-Type: text/xml; charset=utf-8
Client-Date: Wed, 28 Apr 2021 17:23:39 GMT
Client-Peer: 134.115.4.185:8888
Client-Response-Num: 1
SOAPServer: SOAP::Lite/Perl/1.27

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
    soap:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/
    xmlns:soapenc=http://schemas.xmlsoap.org/soap/encoding/
    xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Body>
        <soap:Fault>
            <faultcode>soap:Client</faultcode>
            <faultstring>
                SOAPAction should be http://soapinterop.org/
            </faultstring>
            <faultactor>http://ceto:8888/</faultactor>
        </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

Murdoch UNIVERSITY

Echo Server Script

```
#!/usr/bin/perl -w
use strict;
use SOAP::Lite;
use SOAP::Transport::HTTP;

my $port = shift;  # port supplied on command line when script is invoked
my $soapServer = SOAP::Transport::HTTP::Daemon
    -> new ( LocalPort => $port )
    -> dispatch_to ( qw(echoString) ) # dispatch to 'echoString'
    -> on_action ( sub {
                die "SOAPAction should be \"http://soapinterop.org/\"\n"
                unless $_[0] eq '"http://soapinterop.org/"';} );

print "Starting SOAP server on URL: ".$soapServer->url."\n";
$soapServer->handle;

sub echoString {
    # Receives a string and echoes it back
    my ($class, $inputString) = @_; # @_ catches parameters
    die "no input provided\n" if !$inputString;
    return SOAP::Data->name('return')->type('string')
                                     ->value($inputString);
}
```

Murdoch UNIVERSITY

echoString Client Script

```
#!/usr/bin/perl -w
use strict;
use SOAP::Lite;
my $url = shift; # url supplied on command line when script is invoked
$inputString="Hello";
$inputSoapParam = SOAP::Data
    -> name ('inputString')
    -> type ('string')
    -> value ($inputString);
$response = SOAP::Lite
    -> proxy($url)
    -> uri('http://soapinterop.org/')
    -> on_action( sub { '"http://soapinterop.org/"' } )
    -> echoString ($inputSoapParam); # request 'echoString' service
if ($response->fault) {
    print "SOAP Fault received.\n\n" ;
    print "Fault Code   : ".$response->faultcode."\n";
    print "Fault String : ".$response->faultstring."\n";
    print "Fault Actor  : ".$response->faultactor."\n";
    die;
}
print "Sent '".$inputString."', received '".$response->result."'\n";
```

References

- Online Book Chapter: Web Services Essentials by Ethan Cerami, O'Reilly, 2002 - Chapter 1
- Echo server and client code described in the Unit Reader chapter 6
- SOAP::Lite for Perl:
  - http://guide.soaplite.com/
- References to SOAP at W3C
  - http://www.w3.org/2002/ws/