Module One: Introduction to Service Computing and XML-RPC

Web Services Architecture

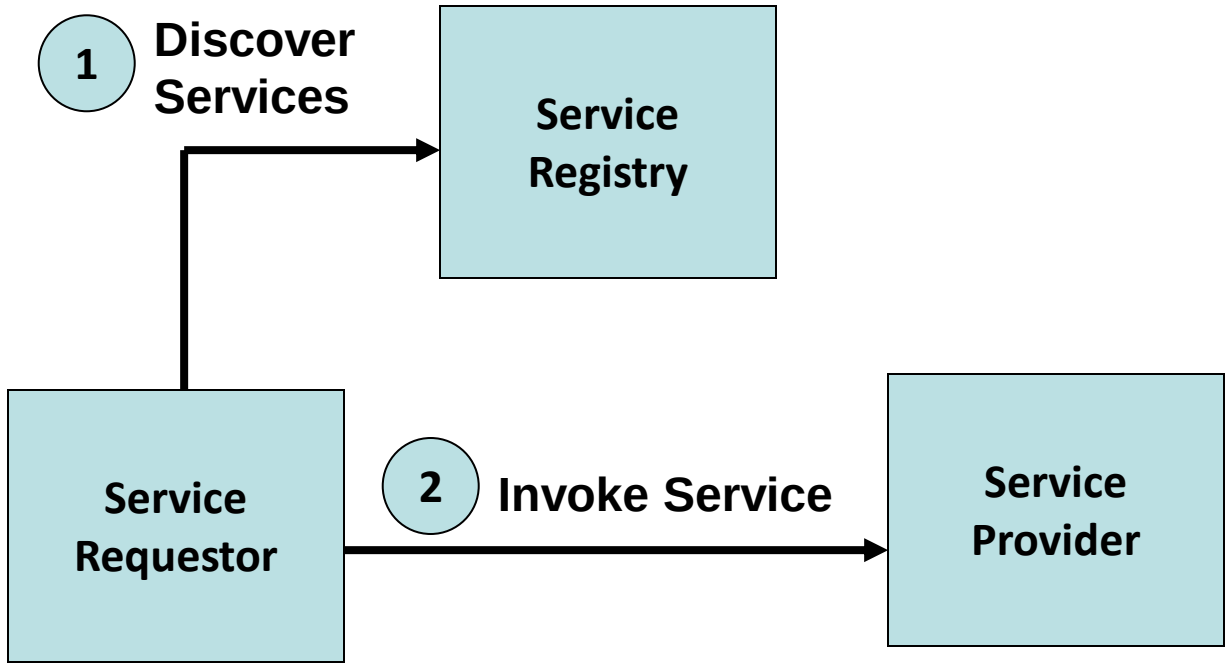
Web Service Architecture

- There are two ways to view the web service architectural framework:
  1) Examine individual roles of each web service actor
  2) Examine the emerging web service protocol stack.

Web Service Roles

- Three major roles in web services:
  - Service Provider: provider of the web service.
  - Service Requestor: any consumer of the web service.
  - Service Registry: logically centralized directory of services.

Web Service Roles



Web Service Protocol Stack

- Service transport: responsible for transporting messages.
  Examples: HTTP, BEEP

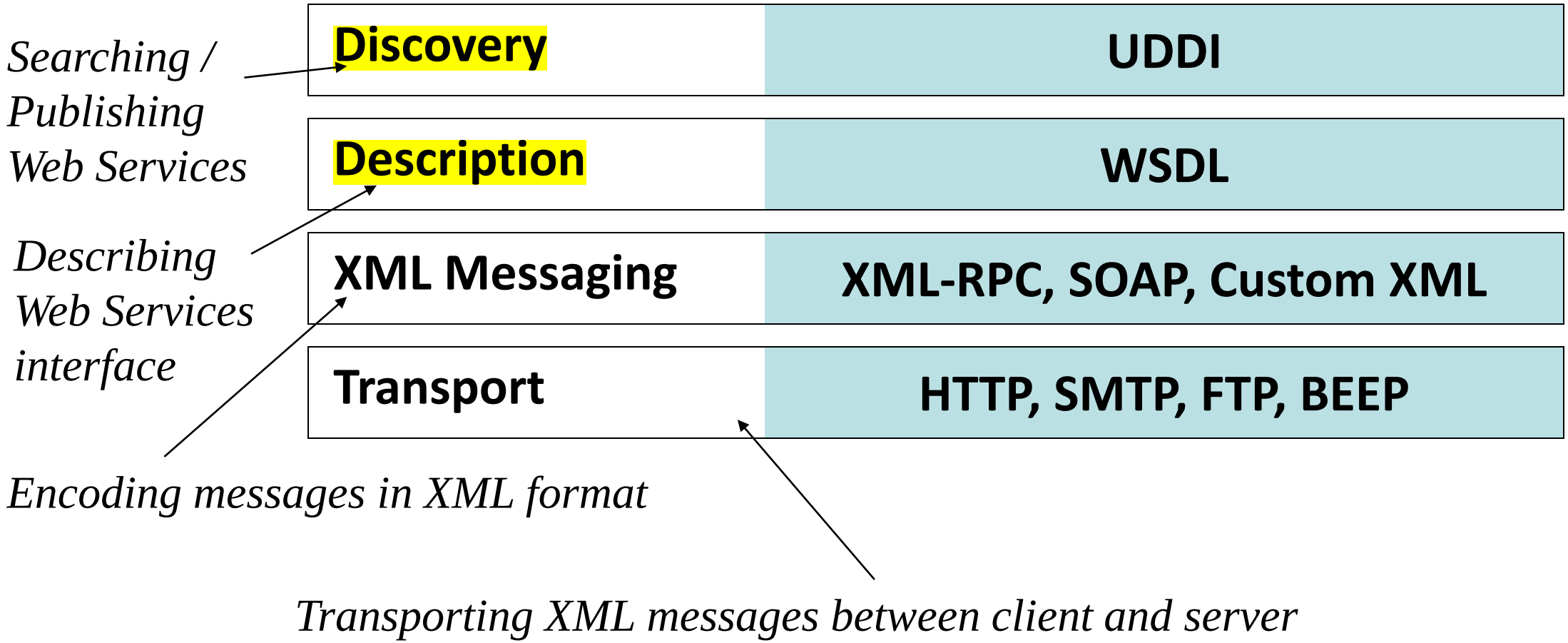
- XML messaging: responsible for encoding messages in common XML format.
  Examples: XML-RPC, SOAP

- Service Description: responsible for describing an interface to a specific web service.
  Example: WSDL

- Service Discovery: responsible for centralizing services into a common search registry.
  Example: UDDI.

standard
It is used to define, locate,implement, and make Web services interact with e ach other.

Web Service Protocol Stack

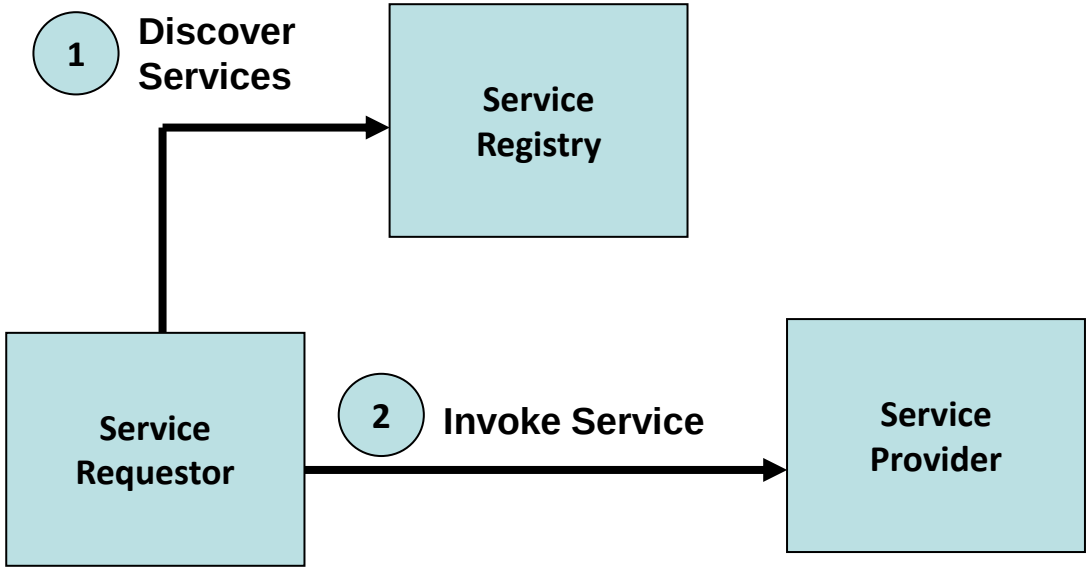


Part II:
Web Service Protocols

XML Messaging

Discovery	UDDI
Description	WSDL
XML Messaging	XML-RPC, SOAP, Custom XML
Transport	HTTP, SMTP, FTP, BEEP

Encoding messages in XML format



Option 1: XML-RPC

- XML-RPC: protocol that uses XML messages to perform Remote Procedure Calls (RPC.)
- Platform independent; diverse applications can talk to each other.
- XML-RPC is the easiest way to get started with web services.
  - Simpler than SOAP
  - Simpler data structures for transmitting data.

XML-RPC Example

- Here is a sample XML-RPC request to a weather service:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodCall>
  <methodName>weather.getWeather</methodName>
  <params>
    <param><value>10016</value></param>
  </params>
</methodCall>
```

"Give me the current weather conditions in zip code: 10016."

XML-RPC Example

- Here is a sample Weather response:

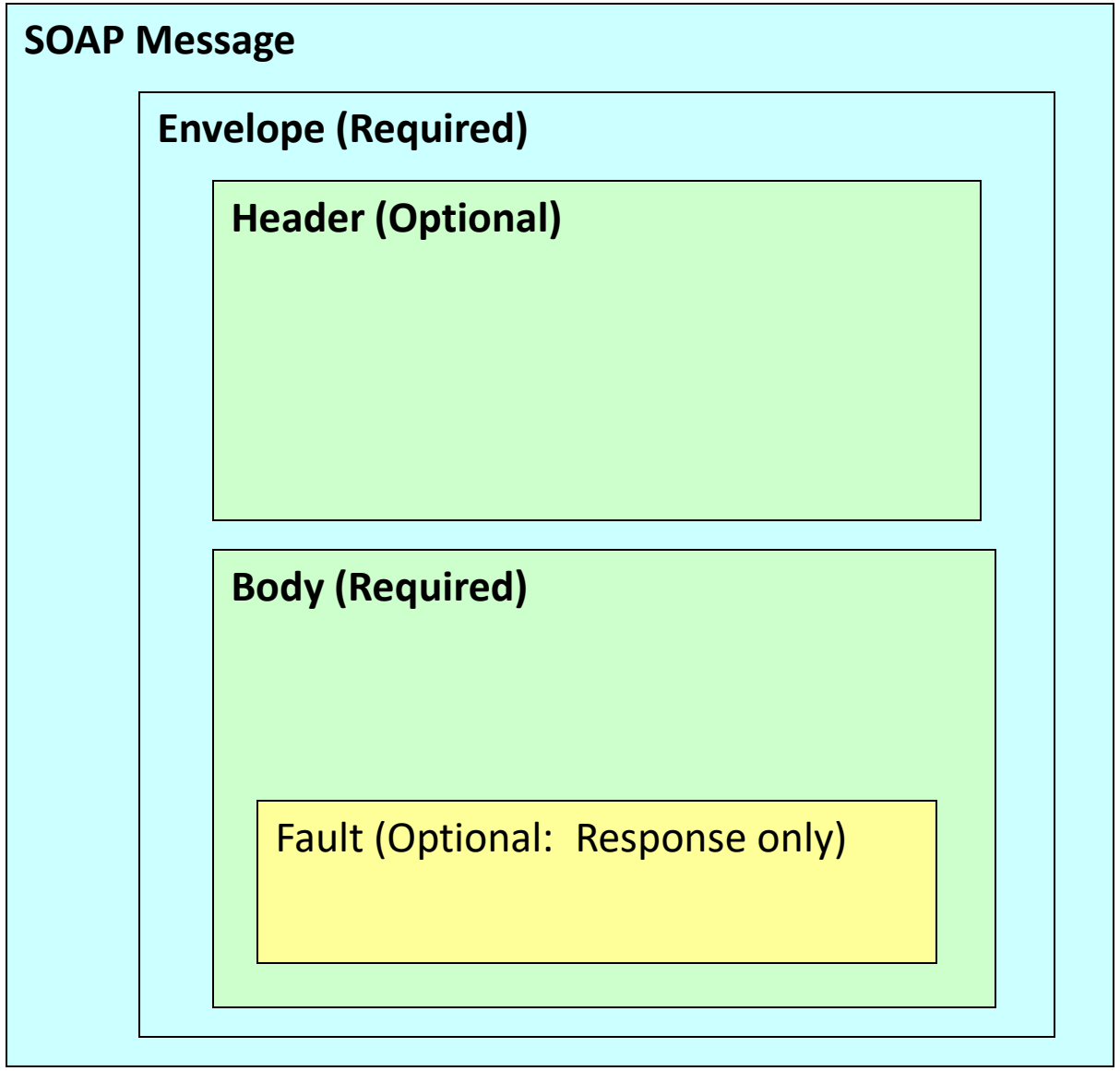
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodResponse>
  <params>
    <param>
      <value><int>65</int></value>
    </param>
  </params>
</methodResponse>
```

“Current temperature is 65 degrees”

Option 2: SOAP

- SOAP: used to stand for “Simple Object Access Protocol”
- XML-Based protocol for exchanging information between computers.
- Currently a formal recommendation of the World Wide Web Consortium (W3C.)

SOAP Message Format



SOAP 1.1 Example

- Here is a sample SOAP request to a weather service:

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

SOAP 1.1 Example:

- Here is a sample 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/">
      <return xsi:type="xsd:int">65</return>
    </ns1:getWeatherResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Weather request

XML-RPC

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodCall>
  <methodName>weather.getWeather</methodName>
  <params>
    <param><value>10016</value></param>
  </params>
</methodCall>
```

SOAP

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

Weather response

XML-RPC

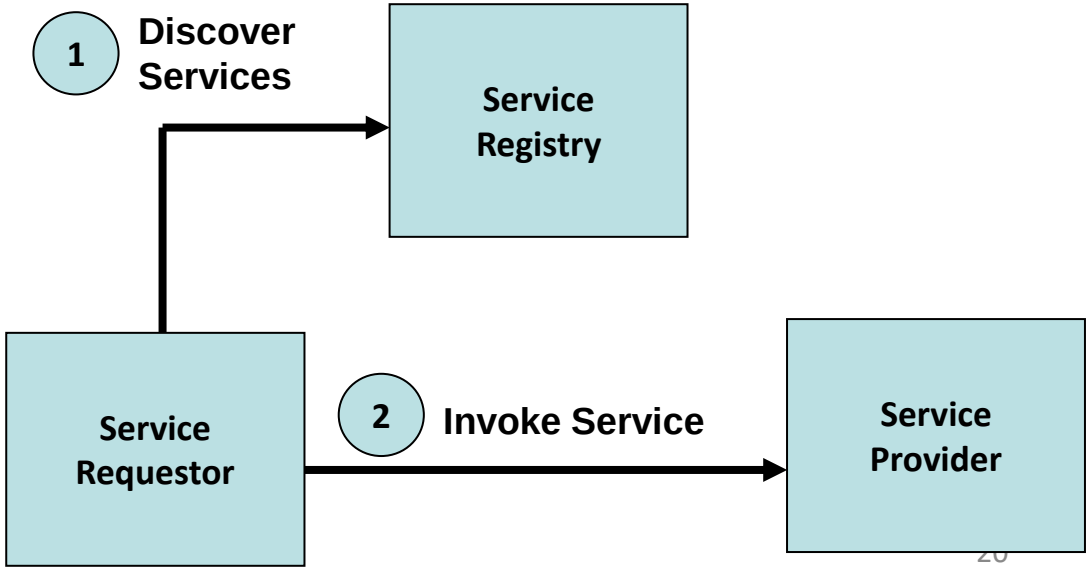
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodResponse>
  <params>
    <param>
      <value><int>65</int></value>
    </param>
  </params>
</methodResponse>
```

SOAP

```
<?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/">
      <return xsi:type="xsd:int">65</return>
    </ns1:getWeatherResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Discovery	UDDI
Description	WSDL
XML Messaging	XML-RPC, SOAP, Custom XML
Transport	HTTP, SMTP, FTP, BEEP

Describing Web Services interface



WSDL

- WSDL: Web Service Description Language.
- WSDL is an XML grammar for specifying an interface for a web service.
- Specifies
  - location of web service
  - methods that are available by the web service
  - data type information for all XML messages
- WSDL is commonly used to describe SOAP services.

WSDL In a Nutshell

<definitions>: Root WSDL Element

- **<types>: What data types will be transmitted?**
- **<message>: What messages will be transmitted?**
- **<portType>: What operations (functions) will be supported?**
- **<binding>: What SOAP specific details are there?**
- **<service>: Where is the service located?**

WSDL Excerpt: Weather Service

```
<message name="getWeatherRequest">
  <part name="zipcode" type="xsd:string"/>
</message>
<message name="getWeatherResponse">
  <part name="temperature" type="xsd:int"/>
</message>

<portType name="Weather_PortType">
  <operation name="getWeather">
    <input message="tns:getWeatherRequest"/>
    <output message="tns:getWeatherResponse"/>
  </operation>
</portType>
```

WSDL Excerpt: Weather Service

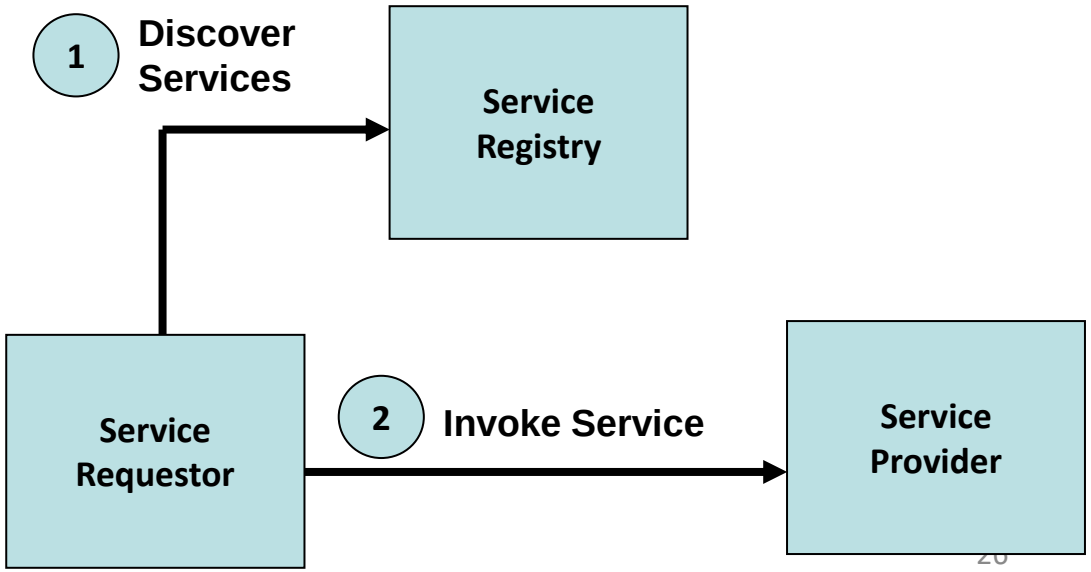
```
…
 <service name="Weather_Service">
     <documentation>WSDL File for
    Weather Service</documentation>
     <port binding="tns:Weather_Binding"
               name="Weather_Port">
               <soap:address
          location="http://ecerami.com/soap/servlet/rpcrouter"/>
     </port>
  </service>
</definitions>
```

So What?

- Given a WSDL file, a developer can immediately figure out how to connect to the web service.
- Eases overall integration process.
- Better yet, with WSDL tools, you can *automate* the integration…

Searching / Publishing Web Services

Discovery	UDDI
Description	WSDL
XML Messaging	XML-RPC, SOAP, Custom XML
Transport	HTTP, SMTP, FTP, BEEP



UDDI

- UDDI: Universal Description, Discovery and Integration.
- Currently represents the *discovery* layer in the protocol stack.
- Originally created by Microsoft, IBM and Ariba.
- Technical specification for publishing and finding businesses and web services.

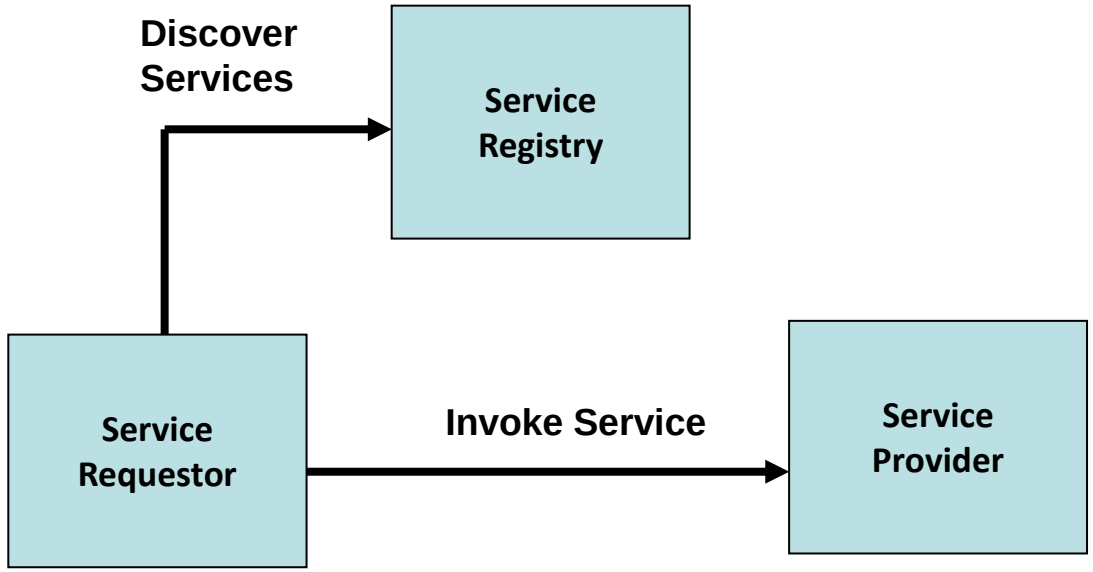
UDDI: Two Parts

- Part I: Technical specification
  - specification for building a distributed directory of businesses and services.
  - XML format for specifying businesses and services.
  - API for querying/publishing to the registry.
- Part II: Implementation
  - UDDI Business Registry, fully operational implementation of the specification.
  - Businesses can publish services here.
  - Businesses can discover services here.
  - Currently maintained by IBM, Microsoft, etc.

UDDI Data

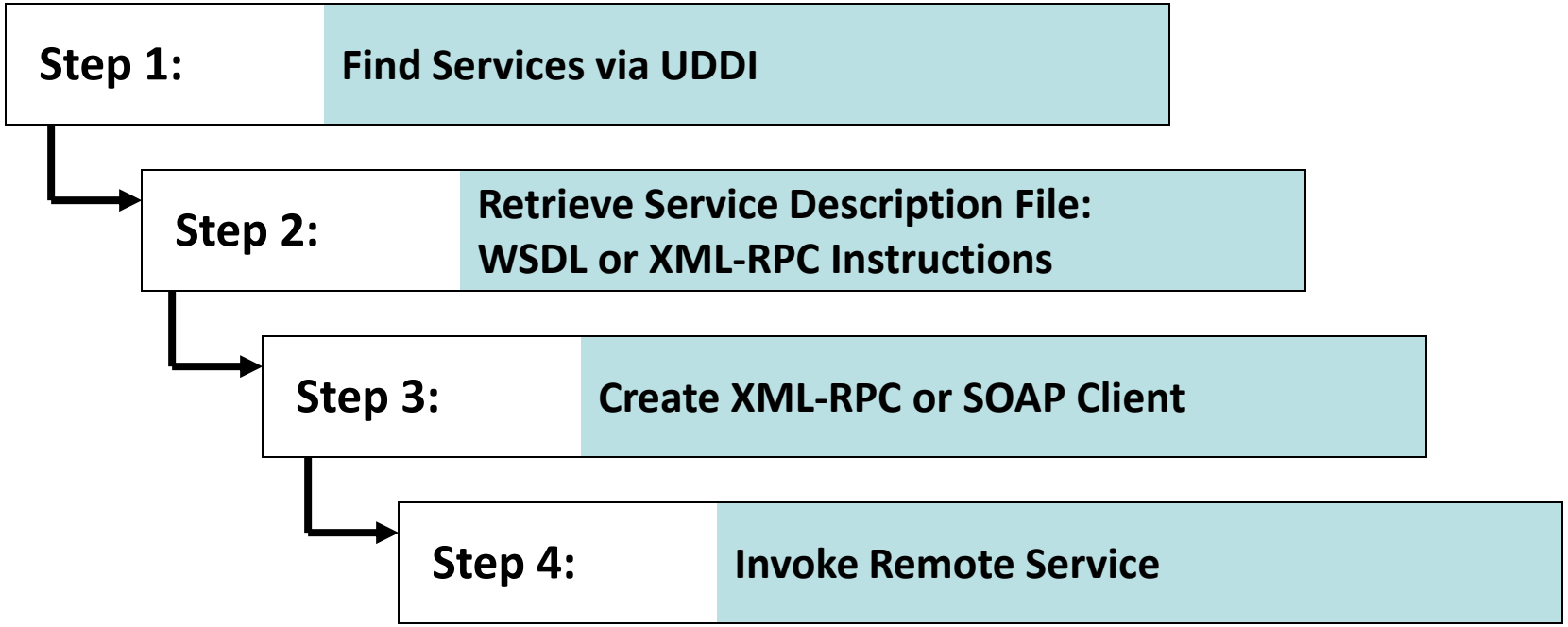
- *White Pages*
  - Information about a specific company; name description, address, etc.
- *Yellow Pages*
  - Classification data for company or service.
  - For example: industry, product or geographic codes.
- *Green Pages*
  - Technical information about specific services.
  - Pointers to WSDL Files.

All Together Now!

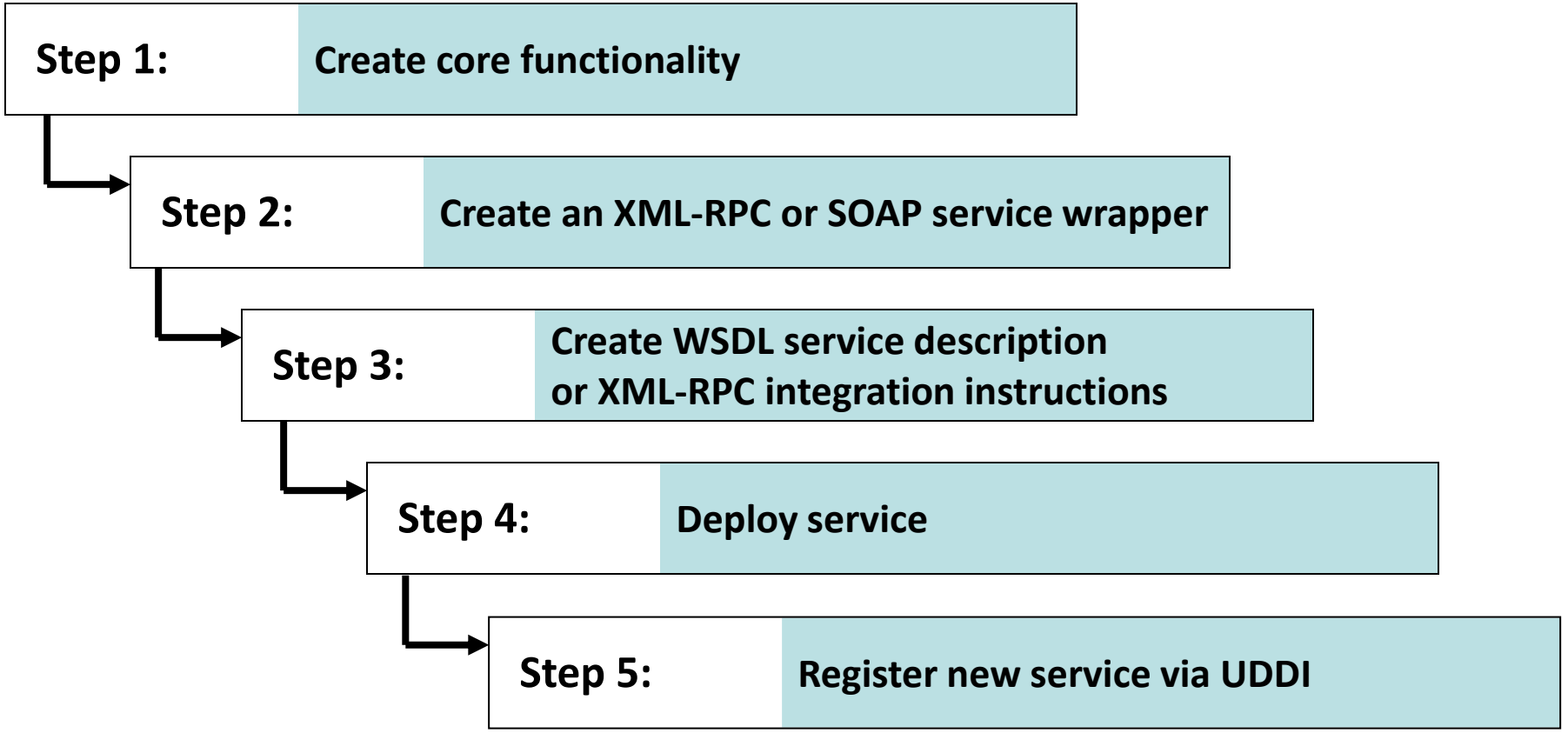


Discovery	**UDDI**
Description	**WSDL**
XML Messaging	**XML-RPC, SOAP, Custom XML**
Transport	**HTTP, SMTP, FTP, BEEP**

Using the Protocols Together – service request perspective



Using the Protocols Together – service provider perspective



Part III:
XML-RPC Essentials

XML-RPC

- XML-RPC provides an XML- and HTTP-based mechanism for making method or function calls across a network.
- XML-RPC offers a very simple, but frequently useful, set of tools for connecting disparate systems and for publishing machine-readable information.

XML-RPC advantages and use scenarios

- Simplicity - easier to integrate systems of very different types
  - XML-RPC's selection of data types is relatively small, but provides enough granularity that developers can express information in forms any programming language can use.

XML-RPC advantages and use scenarios

- Systems integrators and programmers building distributed systems often use XML-RPC as glue code, connecting disparate parts inside a private network. By using XML-RPC, developers can focus on the interfaces between systems, not the protocol used to connect those interfaces.

XML-RPC advantages and use scenarios

- Developers building public services can also use XML-RPC, defining an interface and implementing it in the language of their choice. Once that service is published to the Web, any XML-RPC capable client can connect to that service, and developers can create their own applications that use that service.

XML-RPC parts

- *XML-RPC data model*
  - A set of types for use in passing parameters, return values, and faults (error messages)
- *XML-RPC request structures*
  - An HTTP POST request containing method and parameter information
- *XML-RPC response structures*
  - An HTTP response that contains return values or fault information

XML-RPC Data Model

- The XML-RPC specification defines
  - 6 basic data types
  - 2 compound data types that represent combinations of types
- It's enough to represent many kinds of information.

Basic data types in XML-RPC

Type	Value	Examples
`int` or `i4`	32-bit integers between -2,147,483,648 and 2,147,483,647.	`<int>27</int>` `<i4>27</i4>`
`double`	64-bit floating-point numbers	`<double>27.31415</double>` `<double>-1.1465</double>`
`Boolean`	true (1) or false (0)	`<boolean>1</boolean>` `<boolean>0</boolean>`
`string`	ASCII text, though many implementations support Unicode	`<string>Hello</string>` `<string>bonkers! @</string>`
`dateTime.iso8601`	Dates in ISO8601 format: *CCYYMMDD*T*HH*:*MM*:*SS*	`<dateTime.iso8601>20021125T02:20:04 </dateTime.iso8601>` `<dateTime.iso8601>20020104T17:27:30 </dateTime.iso8601>`
`base64`	Binary information encoded as Base 64, as defined in RFC 2045	`<base64>SGVsbG8sIFdvcmxkIQ== </base64>`

Complex data types in XML-RPC

- Arrays
  - represent sequential information,
- Structs
  - represent name-value pairs, much like hashtables, associative arrays, or properties

Representing basic types

- All of the basic types are represented by simple XML elements whose content provides the value.
  - Example: to define a string whose value is "Hello World!":

```
<string>Hello World!</string>
```

Representing arrays

- Arrays are indicated by the array element, which contains a data element holding the list of values. Like other data types, the array element must be enclosed in a value element.

For example, the following `array` contains four strings:

```
<value>
  <array>
    <data>
      <value><string>This </string></value>
      <value><string>is </string></value>
      <value><string>an </string></value>
      <value><string>array.</string></value>
    </data>
  </array>
</value>
```

The following `array` contains four integers:

```
<value>
  <array>
    <data>
      <value><int>7</int></value>
      <value><int>1247</int></value>
      <value><int>-91</int></value>
      <value><int>42</int></value>
    </data>
  </array>
</value>
```

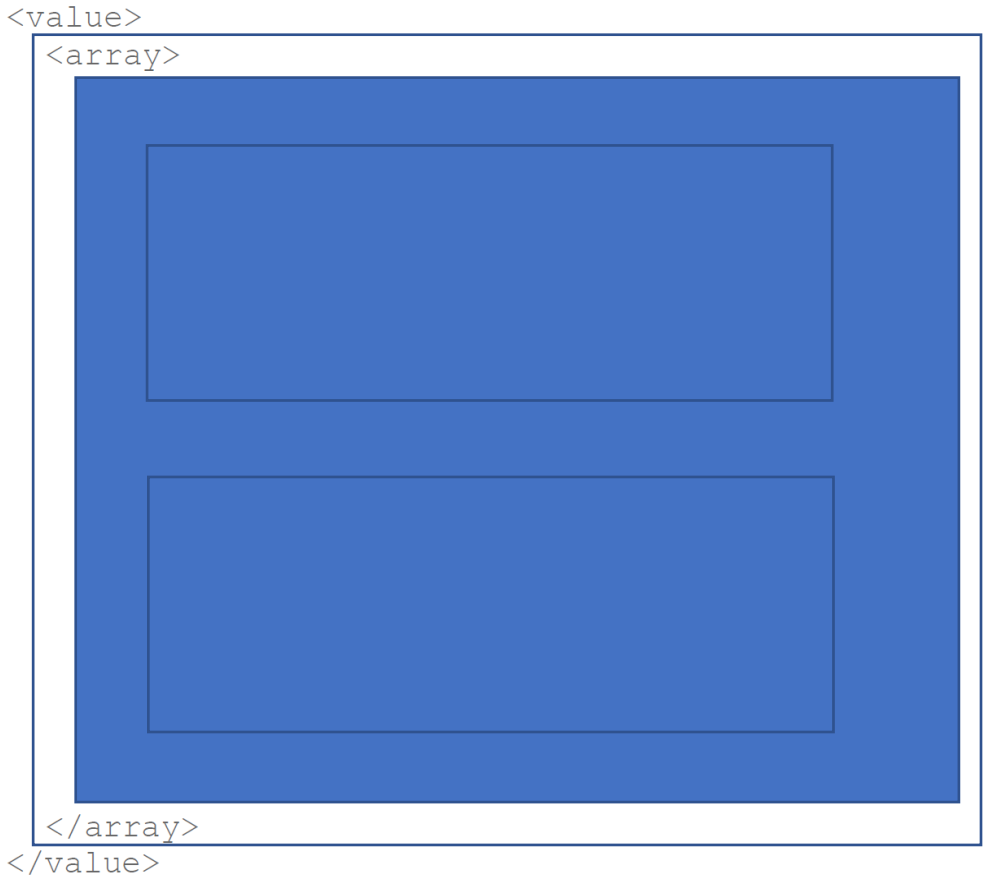
Array

Arrays can also contain mixtures of different types, as shown here:

```
<value>
  <array>
    <data>
      <value><boolean>1</boolean></value>
      <value><string>Chaotic collection, eh?</string></value>
      <value><int>-91</int></value>
      <value><double>42.14159265</double></value>
    </data>
  </array>
</value>
```

Multidimensional array

Creating multidimensional arrays is simple - just add an array inside of an array:



Representing structs

- Structs contain unordered content, identified by name.
- Names are strings, though you don't have to enclose them in string elements.
- Each struct element contains a list of member elements.
- Member elements each contain one name element and one value element.
- The order of members is not considered important.
- While the specification doesn't require names to be unique, you'll probably want to make sure they are unique for consistency.

```
<value>
  <struct>
    <member>
      <name>givenName</name>
      <value><string>Joseph</string></value>
    </member>
    <member>
      <name>familyName</name>
      <value><string>DiNardo</string></value>
    </member>
    <member>
      <name>age</name>
      <value><int>27</int></value>
    </member>
  </struct>
</value>
```

Struct

- Structs can also contain other structs, or even arrays.
- For example, this struct contains a string, a struct, and an array

```
<value>
  <struct>
    <member>
      <name>name</name>
      <value><string>a</string></value>
    </member>
    <member>
      <name>attributes</name>
      <value><struct>
          <member>
            <name>href</name>
            <value><string>http://example.com</string></value>
          </member>
          <member>
            <name>target</name>
            <value><string>_top</string></value>
          </member>
    </struct></value>
    </member>
    <member>
      <name>contents</name>
      <value><array>
        <data>
          <value><string>This </string></value>
          <value><string>is </string></value>
          <value><string>an example.</string></value>
        </data>
      </array></value>
    </member>
  </struct>
</value>
```

XML-RPC parts

- *XML-RPC data model*
  - A set of types for use in passing parameters, return values, and faults (error messages)
- *XML-RPC request structures*
  - An HTTP POST request containing method and parameter information
- *XML-RPC response structures*
  - An HTTP response that contains return values or fault information

XML-RPC Request Structure

- A combination of XML content and HTTP headers
  - The XML content uses the data typing structure to pass parameters and contains additional information identifying which procedure is being called
  - The HTTP headers provide a wrapper for passing the request over the Web

XML-RPC Request Structure

- Each request contains a single XML document
  - root element is a methodCall element
  - each methodCall element contains a methodName element and a params element.
  - the methodName element identifies the name of the procedure to be called
  - the params element contains a list of parameters and their values. Each params element includes a list of param elements which in turn contain value elements.

XML-RPC Request Structure

For example, to pass a request to a method called `circleArea` , which takes a `Double` parameter (for the radius), the XML-RPC request would look like:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>circleArea</methodName>
  <params>
    <param>
      <value><double>2.41</double></value>
    </param>
  </params>
</methodCall>
```

To pass a set of arrays to a `sortArray` procedure, the request might look like:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>sortArray</methodName>
  <params>
    <param>
      <value>
        <array>
          <data>
            <value><int>10</int></value>
            <value><int>20</int></value>
            <value><int>30</int></value>
          </data>
        </array>
      </value>
    </param>
    <param>
      <value>
        <array>
          <data>
            <value><string>A</string></value>
            <value><string>C</string></value>
            <value><string>B</string></value>
          </data>
        </array>
      </value>
    </param>
  </params>
</methodCall>
```

XML-RPC Request Structure

- The HTTP headers for these requests will reflect the senders and the content. The basic template looks like:

```
POST /target HTTP 1.0
User-Agent: Identifier
Host: host.making.request
Content-Type: text/xml
Content-Length: length of request in bytes
```

- The information in italics may change from client to client or from request to request.

XML-RPC Request Structure

- For example, if the `circleArea` method were available from an XML-RPC server listening at */xmlrpc*, the request might look like:

```
POST /xmlrpc HTTP 1.0
User-Agent: myXMLRPCClient/1.0
Host: 192.168.124.2
Content-Type: text/xml
Content-Length: 169
```

Assembled, the entire request would look like:

```
POST /xmlrpc HTTP 1.0
User-Agent: myXMLRPCClient/1.0
Host: 192.168.124.2
Content-Type: text/xml
Content-Length: 169

<?xml version="1.0"?>
<methodCall>
  <methodName>circleArea</methodName>
  <params>
    <param>
      <value><double>2.41</double></value>
    </param>
  </params>
</methodCall>
```

XML-RPC Response Structure

- If the response is successful - the procedure was found, executed correctly, and returned results - then the XML-RPC response will look much like a request, except that the `methodCall` element is replaced by a `methodResponse` element and there is no `methodName` element:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><double>18.24668429131</double></value>
    </param>
  </params>
</methodResponse>
```

XML-RPC Response Structure

- An XML-RPC response can only contain one parameter, despite the use of the enclosing `params` element. That parameter, may, of course, be an array or a struct, so it is possible to return multiple values. Even if your method isn't designed to return a value (`void` methods in C, C++, or Java, for instance) you still have to return something. A "success value" – for example a boolean set to true (1) - is a typical approach to getting around this limitation.

Fault response

- If there was a problem in processing the XML-RPC request, the `methodResponse` element will contain a `fault` element instead of a `params` element. The `fault` element, like the `params` element, has only a single value. Instead of containing a response to the request, however, that value indicates that something went wrong.

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value><string>No such method!</string></value>
  </fault>
</methodResponse>
```

Developing with XML-RPC

- Using XML-RPC in applications generally means adding an XML-RPC library and making some of the function calls through that library.
- Creating functions that will work smoothly with XML-RPC requires writing code that uses only the basic types XML-RPC supports.
- Adding XML-RPC support may require writing some wrapper code that connects the code with the library, but this generally isn't very difficult.

Module 1 Summary

- Concept of “service oriented” and the background of service computing
- Basics of XML-RPC technology

Module Two: Service Message Exchange SOAP and Service Description WSDL

Our textbook for this module:

- **Ethan Cerami, Web Services Essentials, Publisher： O’Reilly， ISBN：9780596002244，**
  - Chapter 3 - SOAP
  - Chapter 6 - WSDL
- **Liang-Jie Zhang, Services Computing, Publisher: Springer， ISBN：9783540382812 -** You can find an online version of this book for free through our library webpage.
  - Chapter 3.1
  - Chapter 3.2 (without 3.2.5)

Module 2 Learning Outcomes

- Understand the basics of the SOAP protocol
- Understand the details about the SOAP XML Message specification
- Understand the SOAP encoding rules
- Understand the basics of WSDL

Guided questions for Module 2

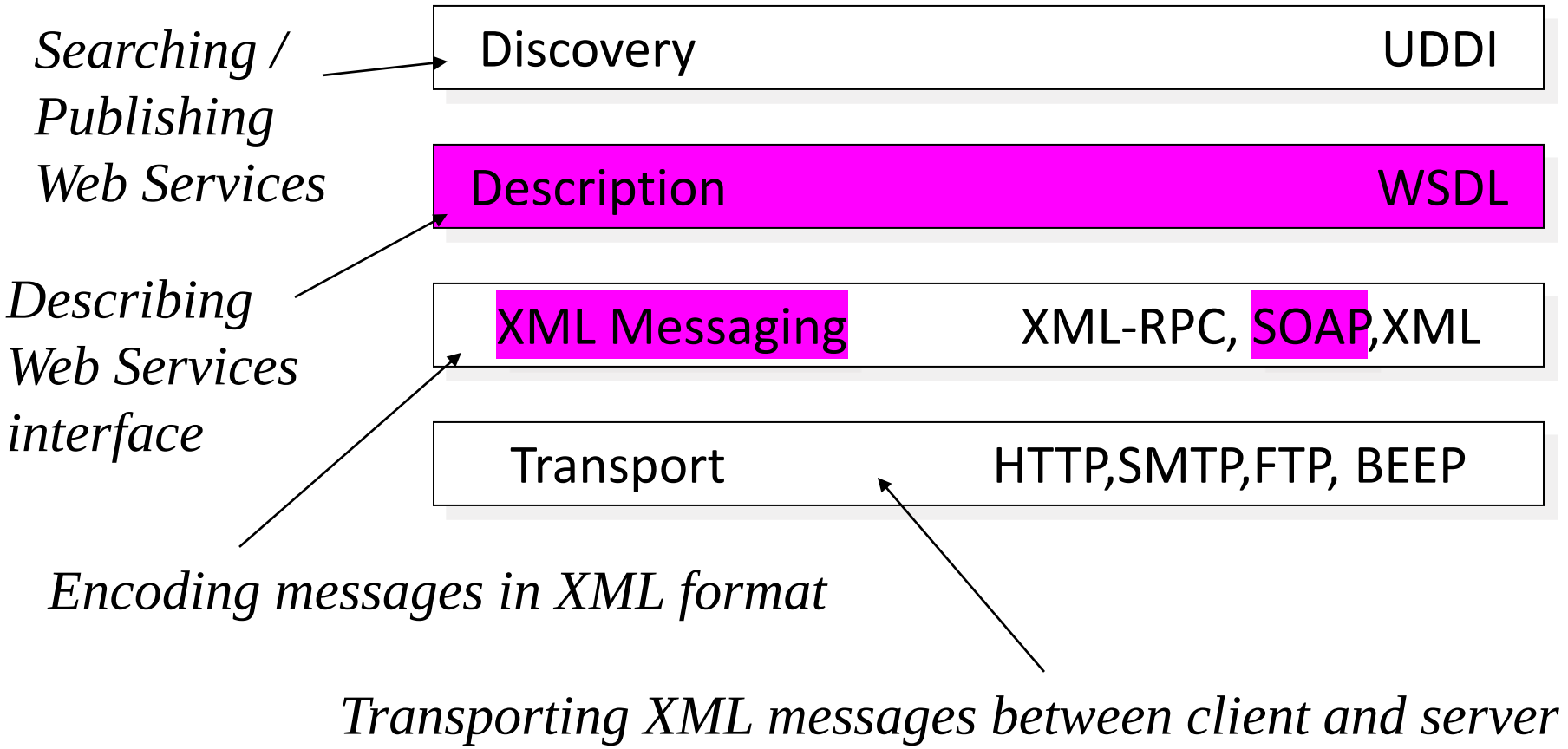
- What is XML messaging?
- What is SOAP?
- What is the relationship between XML, XML-RPC, and SOAP?
- What is advantage of SOAP over CORBA, DCOM, and Java RMI?
- What platform and language do we need to use with SOAP? (tricky question)
- What are the major parts in SOAP specification?
- What are the main SOAP encoding rules?

Guided questions for Module 2

- What is WSDL?
- What is the relationship between WSDL and service description?
- What data does WSDL describe?
- What is WSDL used for?
- What platform and language do we need to use with WSDL? (tricky question)
- What are the major elements of WSDL?
- What is the relationship between SOAP, WSDL, and UDDI?

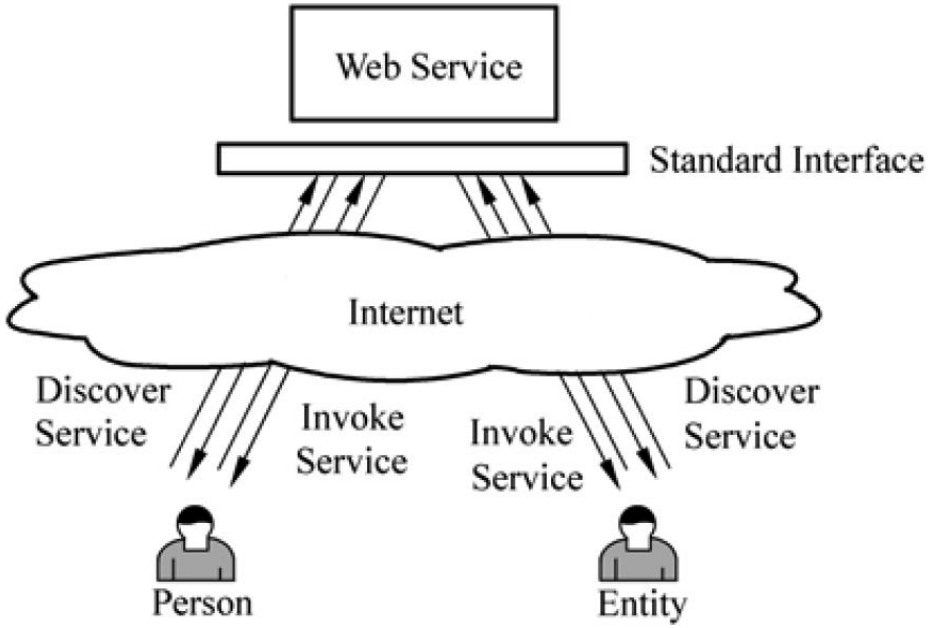
Introduction

Web Service Protocol Stack



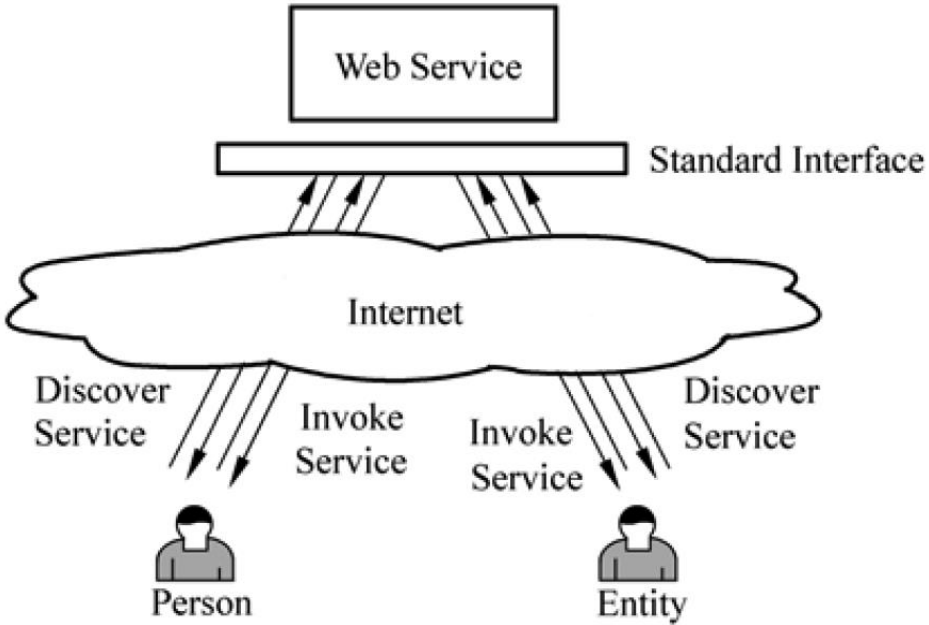
Web service basics

- An organization exposes its business applications as services on the Internet and makes them accessible via standard programmatic interfaces



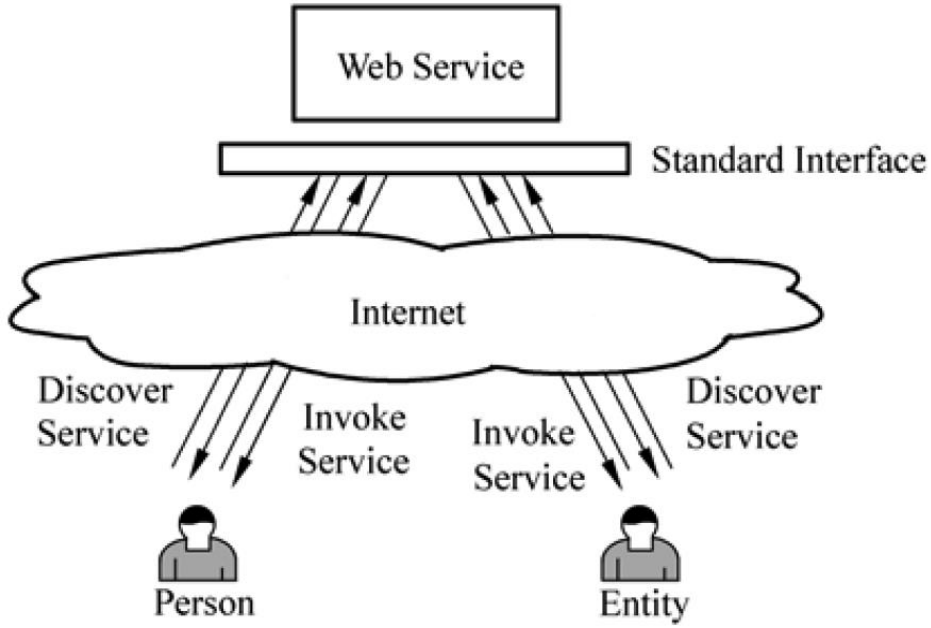
Web service basics

- The Web services technology provides a uniform and loosely coupled integration framework to increase cross-language and cross-platform interoperability for distributed computing and resource sharing over the Internet



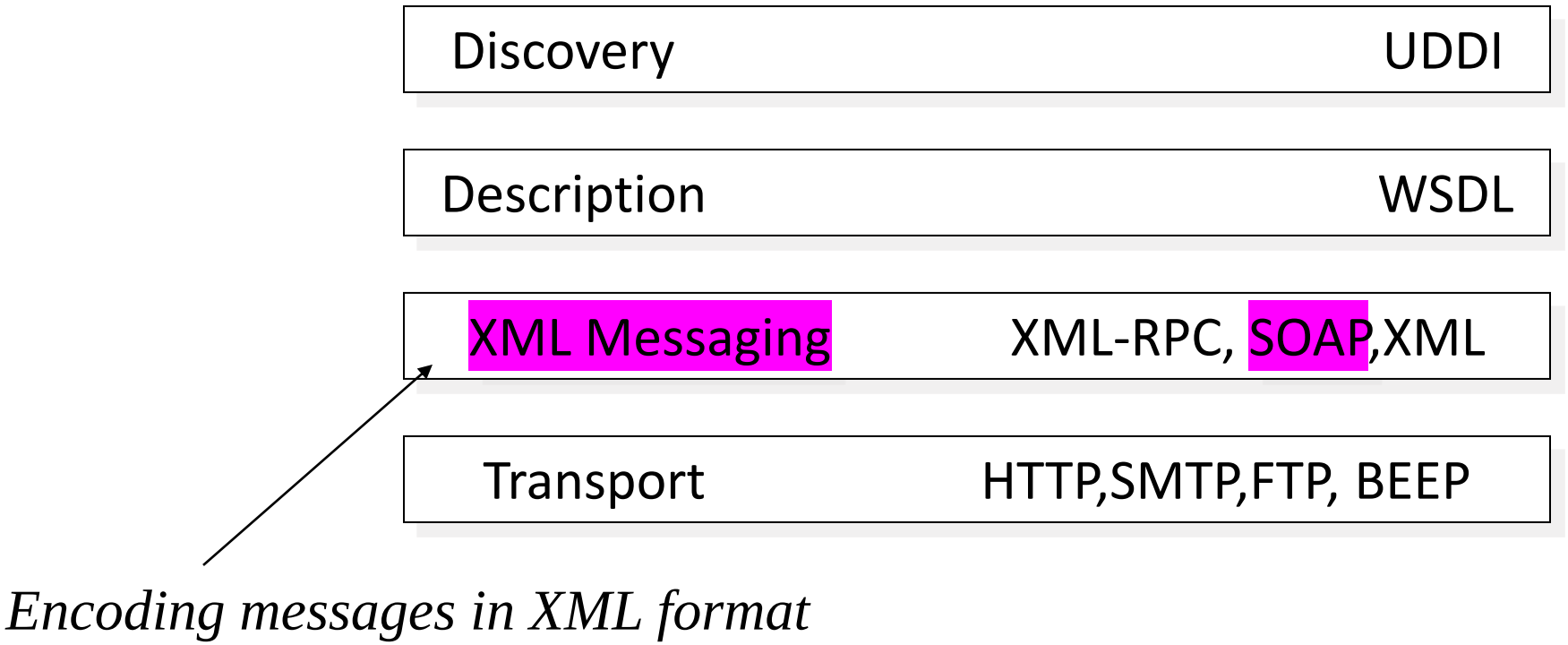
Web service basics

- The paradigm of Web services opens a new cost-effective way of engineering software to quickly develop and deploy Web applications, by dynamically integrating other independently published Web service components into new business processes

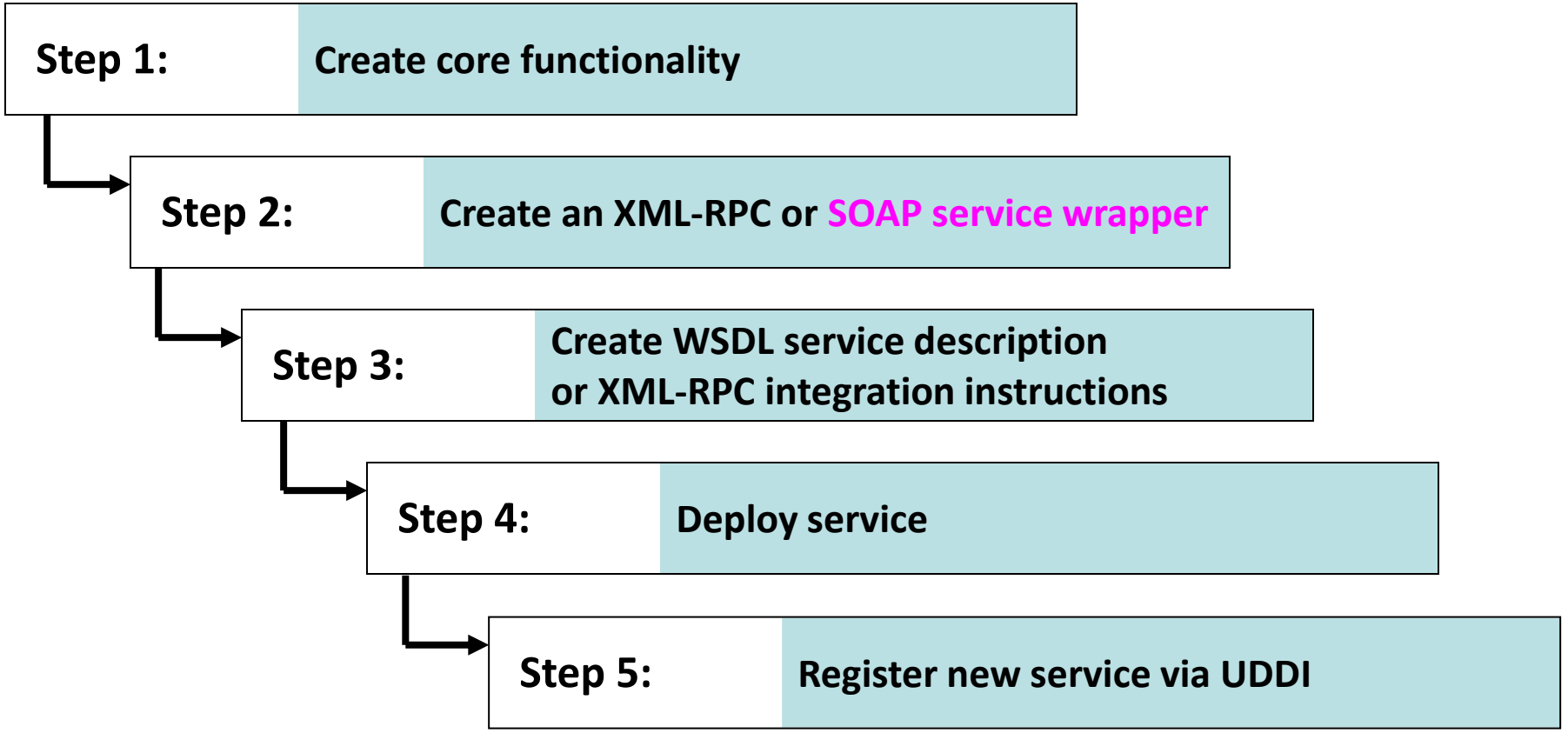


Service Message Exchange SOAP

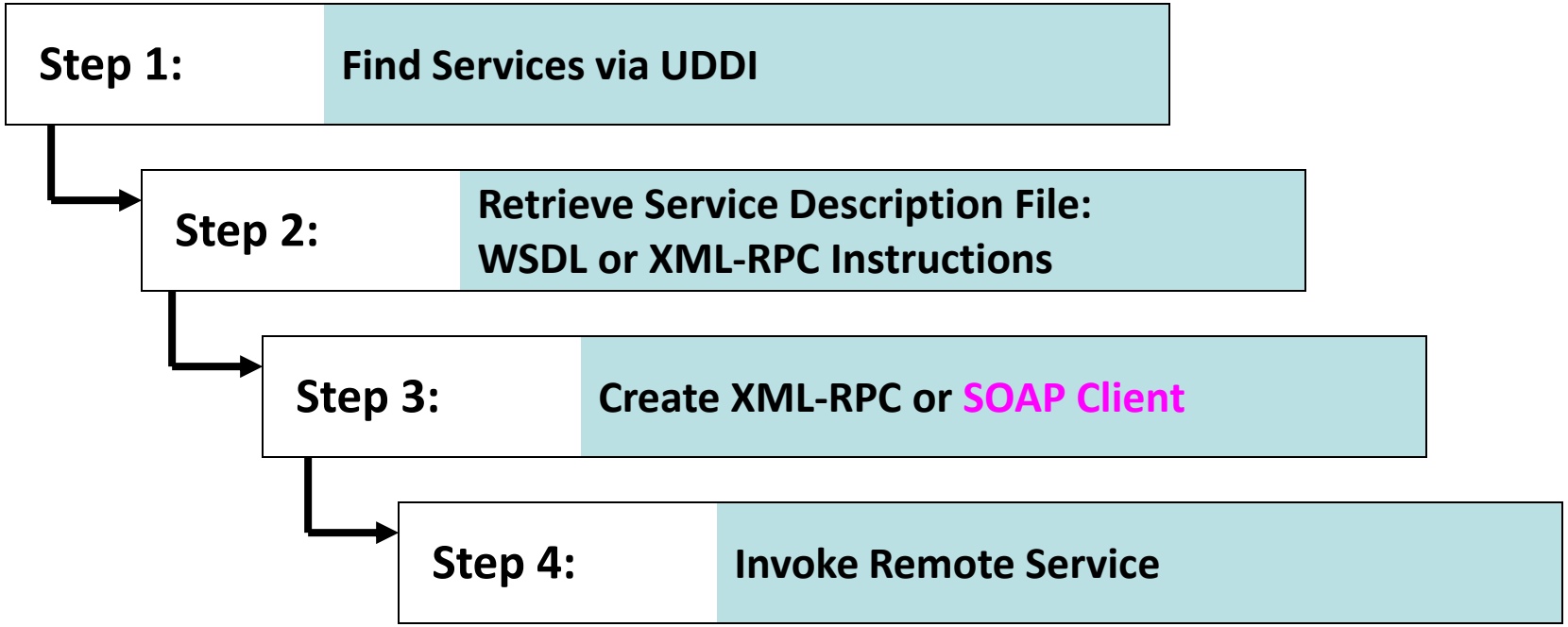
Web Service Protocol Stack



Using the Protocols Together – service provider perspective



Using the Protocols Together – service request perspective



A client program reads a WSDL document to understand what a Web service can do; then it uses SOAP to actually invoke the functions listed in the WSDL document.

SOAP

- SOAP is an XML-based protocol for exchanging information between computers.

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 is remote procedure calls transported via HTTP.
- SOAP therefore enables client applications to easily connect to remote services and invoke remote methods.
  - For example, a client application can immediately add language translation to its feature set by locating the correct SOAP service and invoking the correct method.

SOAP vs other frameworks

- Other frameworks, including CORBA, DCOM, and Java RMI, provide similar functionality to SOAP, but SOAP messages are written entirely in XML and are therefore uniquely platform- and language-independent.
- For example, a SOAP Java client running on Linux or a Perl client running on Solaris can connect to a Microsoft SOAP server running on Windows 2000.
- SOAP therefore represents a cornerstone of the web service architecture, enabling diverse applications to easily exchange services and data.

The SOAP specification

- The SOAP specification defines three major parts:
  - SOAP envelope specification
  - Data encoding rules
  - RPC conventions

SOAP envelope specification

- The SOAP XML Envelope defines specific rules for encapsulating data being transferred between computers.
  - application-specific data, such as the method name to invoke, method parameters, or return values.
  - information about who should process the envelope contents
  - in the event of failure, how to encode error messages

Data encoding rules

- To exchange data, computers must agree on rules for encoding specific data types.
- For example, two computers that process stock quotes need an agreed-upon rule for encoding float data types; likewise, two computers that process multiple stock quotes need an agreed-upon rule for encoding arrays.
- SOAP therefore includes its own set of conventions for encoding data types. Most of these conventions are based on the W3C XML Schema specification.

RPC conventions

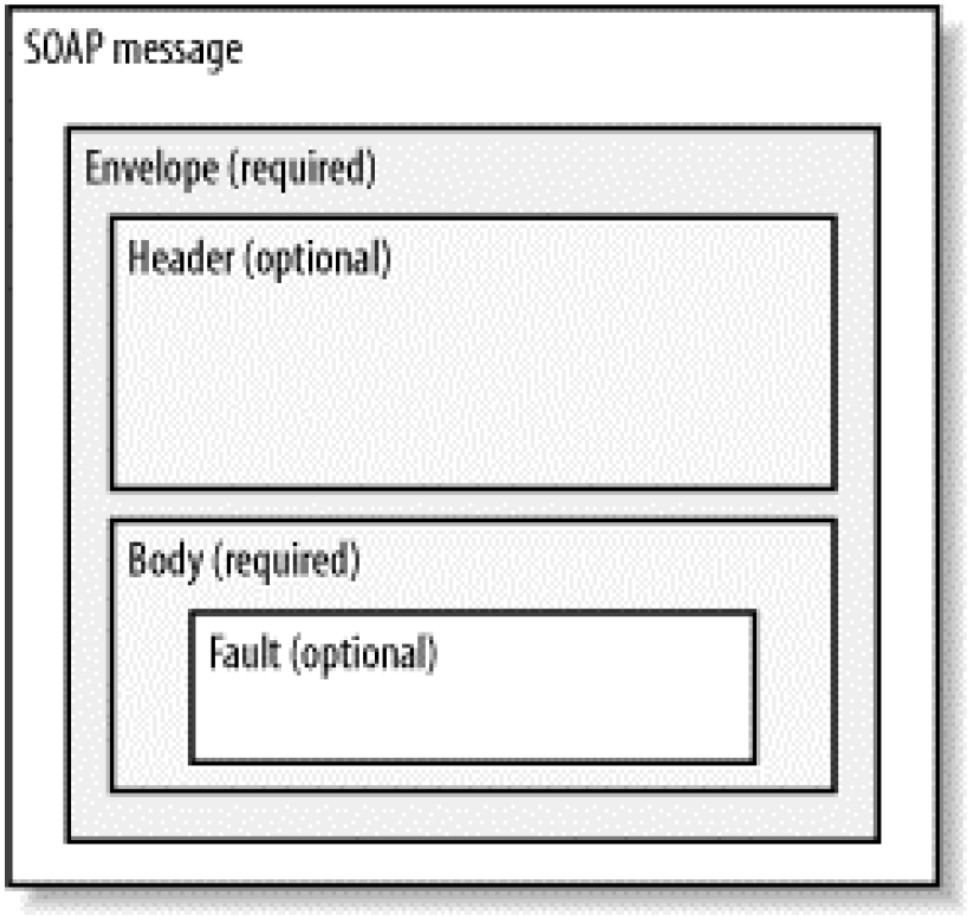
- SOAP can be used in a variety of messaging systems, including one-way and two way messaging.
- For two-way messaging, SOAP defines a simple convention for representing remote procedure calls and responses. This enables a client application to specify a remote method name, include any number of parameters, and receive a response from the server.

SOAP Message

- A one-way message
  - a request from a client
  - a response from a server

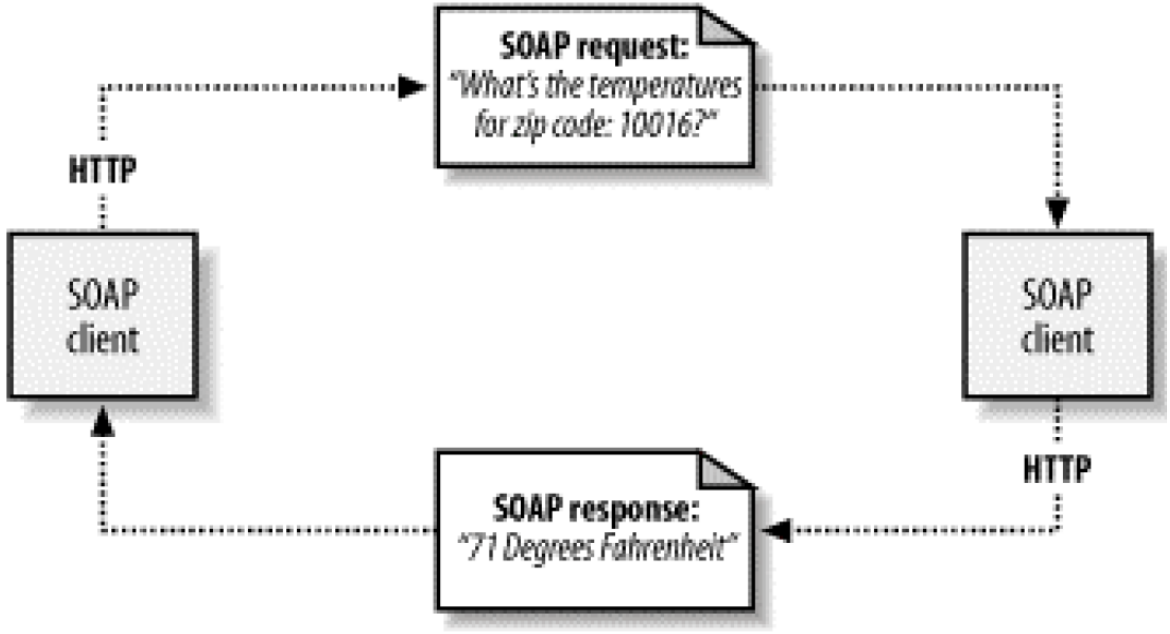
SOAP Request - elements

- Mandatory elements
  - Envelope element
    - Body element
- Optional elements
  - Header element
  - Fault element
- Each of these elements has an associated set of rules

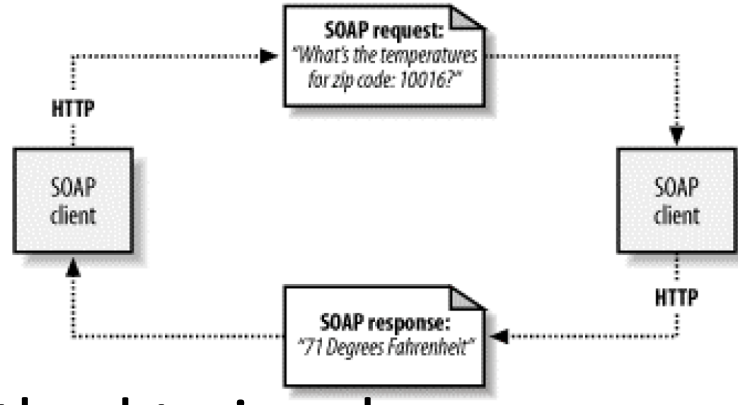


Example

- XMethods.net provides a simple weather service, listing current temperature by zip code. (See Figure 3-1.) The service method, getTemp , requires a zip code string and returns a single float value.



SOAP Request



- The client request must include the name of the method to invoke and any required parameters.

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:getTemp
      xmlns:ns1="urn:xmethods-Temperature"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <zipcode xsi:type="xsd:string">10016</zipcode>
      </ns1:getTemp>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Request - XML namespaces

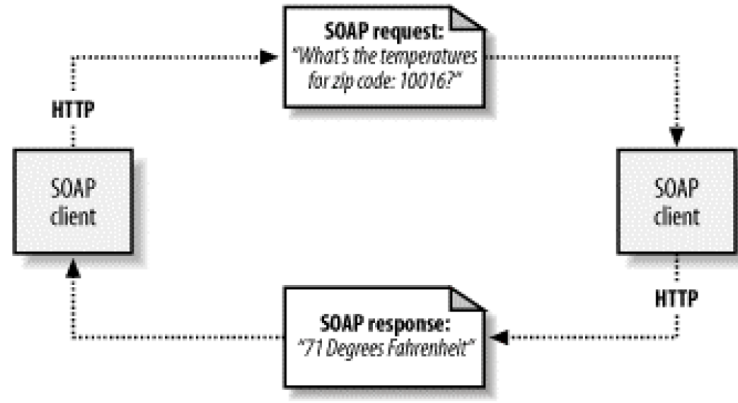
- a total of four XML namespaces are defined. Namespaces are used to disambiguate XML elements and attributes, and are often used to reference external schemas.
- In our sample SOAP request, we'll use namespaces to disambiguate identifiers associated with
  - the SOAP Envelope (*http://schemas.xmlsoap.org/soap/envelope/*)
  - data encoding via XML Schemas (*http://www.w3.org/2001/XMLSchema-instance* and *http://www.w3.org/2001/XMLSchema*)
  - application identifiers specific to Xmethods (*urn:xmethods-Temperature*)
- This enables application modularity, while also providing maximum flexibility for future changes to the specifications.

SOAP Request

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:getTemp
      xmlns:ns1="urn:xmethods-Temperature"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <zipcode xsi:type="xsd:string">10016</zipcode>
      </ns1:getTemp>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- The Body element encapsulates the main "payload" of the SOAP message.
- The only element is getTemp, which is tied to the XMethods namespace and corresponds to the remote method name.
- Each parameter to the method appears as a subelement.
  - In our example, we have a single zip code element, which is assigned to the XML Schema xsd:string data type and set to 10016.
  - If additional parameters are required, each can have its own data type.

SOAP Response




```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:getTempResponse
      xmlns:ns1="urn:xmethods-Temperature"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <return xsi:type="xsd:float">71.0</return>
      </ns1:getTempResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Response

- Just like the request, the response includes Envelope and Body elements, and the same four XML namespaces.
- This time, however, the Body element includes a single getTempResponse element, corresponding to our initial request. The response element includes a single return element, indicating an xsd:float data type.