V22.0480-004
Web Services Architecture and Programming

Lecture 12
Web Services Using Visual Studio.NET (cont’d)
SOAP
Announcements

• Lab 3 due date extended to November 2\textsuperscript{nd} (Sunday), 11:59pm
  – Additional helper code available from
    \texttt{D:\VSDev\Public\vijayk\Lab3-Helper.zip}
  – Please see the TA’s if you continue to have difficulty

• Lab 4 out on October 30\textsuperscript{th} (Thursday)
  – due back November 12\textsuperscript{th} (Wednesday)
(Review) SOAP, WSDL, and UDDI

• SOAP: Simple Object Access Protocol
  – XML-RPC-like request/response protocol +
  – Support for asynchronous invocations
  – Encoding of additional information in the message
    • Security tokens for authentication/encryption,
    • Message route information, …

• WSDL: Web Services Description Language
  – RPC, Distributed Objects-like common structs/interface +
  – Support for asynchronous invocations
  – Possibility of language-neutral (and automatic) interpretation
    • Web-services tools use WSDL description of a service to automatically generate a SOAP-capable proxy

• UDDI: Universal Description, Discovery, and Integration
  – Defines ways of mapping service “characteristics” to service providers
  – “characteristics” generalize “names”
Web Services in .NET using Visual Studio

[ Code walkthrough using Remote Desktop ]

Building and deploying a simple web service
• Setting up the service class and methods
  – [WebMethod] and [WebService] attributes
• Invoking its functionality
• Inspecting its description and discovery interfaces

Writing a web services client
• Adding a “web reference”
• Inspecting automatically generated code for proxy
• Instantiating and using the proxy

Understanding the implementation
• SOAP message exchange
• IIS mapping of service URL
SOAP

History

• SOAP 1.0 (1997): An XML-based protocol for accessing objects
• XML-RPC (1998): A subset of SOAP 1.0
• SOAP 1.1 (2000): Widely supported, de facto standard
• SOAP 1.2 (2003): W3C “Recommendation” as of June 2003
  – Standard, will soon replace SOAP 1.1 implementations

• Originally an interoperable protocol for accessing “objects”
• Current versions focus on a generalized XML messaging framework
What is SOAP?

SOAP is a lightweight protocol intended for exchanging structured information in a decentralized environment. SOAP uses XML technologies to define an extensible messaging framework, which provides a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation-specific semantics.

- SOAP is an XML-based messaging framework
  - Defines a way to move XML messages from point A to point B
- It is extensible
- It is usable over a variety of underlying networking protocols
- It is independent of programming models
SOAP Features

• Extensibility
  – Allows addition of features as layered extensions
  – Basis for IBM/Microsoft Global XML Web Services Architecture (GXA)
    • WS-Security, WS-Routing, WS-Referral, ….  
    • More about this in a few weeks
  – Contrast with XML-RPC specification

• Usable over a variety of transport protocols
  – TCP, HTTP, SMTP, MSMQ (message queues)
  – Standard protocol bindings need to be defined that specify how a SOAP message is encoded in each protocol

• Support for a variety of programming models
  – RPC-like request-response
  – One-way messaging
  – Several other Message Exchange Protocols (MEPs)
Elements of the SOAP Specification

- **Messaging framework**
  - Defines a suite of XML elements for “packaging” arbitrary XML messages for transport between systems
  - i.e., what constitutes a “SOAP message”

- **Processing model**
  - Rules for processing a SOAP message as it travels from a SOAP sender to a SOAP receiver
  - Permits multiple intermediary nodes that can act upon message

- **Protocol bindings (an explicit one for HTTP)**
  - Defines transmission of SOAP messages using a given transport protocol

- **RPC encoding**
  - Standard way for mapping RPC calls to SOAP messages
SOAP Messaging Framework

- Core XML elements: *Envelope*, *Header*, *Body*, and *Fault*
  - Defined in a version-specific XML namespace
    - SOAP 1.1: http://schemas.xmlsoap.org/soap/envelope
    - SOAP 1.2: http://www.w3.org/2003/05/soap-envelope

- Structure of a SOAP message

```xml
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header> <!-- optional -->
    <!-- header blocks go here … -->
  </soap:Header>
  <soap:Body>
    <!-- payload or Fault element goes here … -->
  </soap:Body>
</soap:Envelope>
```
XML Schemas

- XML Schema specifies the structure of an XML element
  - What are its sub-elements?
  - What are their “types”, other restrictions (if any)?

- Example:

```xml
<schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://netserver1.pdsg.cs.nyu.edu/vijayk"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <complexType name="Book">
    <element type="Title"/>
    <element type="Author"/>
    <element type="Copyright"/>
  </complexType>
  <simpleType name="Title" xsi:type="string"/>
  <simpleType name="Author" xsi:type="string"/>
  <simpleType name="Copyright" xsi:type="integer"/>
</schema>
```
XML Namespaces

- Denoted by the `xmlns:` tag
- Define a set of unique names within a given context
  - Context identified by a URI
    - Unlike a URL, need not have a physical resource associated with it
    - Performs same function as in C++, Java, C#, …
    - Permits reuse of names
- In the SOAP messaging framework, namespaces serve two functions
  - They help distinguish between different versions of SOAP
  - The associated schema defines the structure of the SOAP elements: Envelope, Header, Body, and Fault
    - This can then be checked by a parser/validator

```xml
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/”>
```
Examples of SOAP Messages

• Client-to-StringReverser:

  <soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body>
      <Reverse
        xmlns="http://netserver1.pdsg.cs.nyu.edu/vijayk">
        <arg>hi there</arg>
      </Reverse>
    </soap:Body>
  </soap:Envelope>

• StringReverser-to-Client

  <soap:Body>
    <ReverseResponse
      xmlns="http://netserver1.pdsg.cs.nyu.edu/vijayk">
      <ReverseResult>ereht ih</ReverseResult>
    </ReverseResponse>
  </soap:Body>
SOAP Fault Element

- Like XML-RPC, faults communicated back to the receiver as part of the response message

```xml
<soap:Body>
  <soap:Fault>
    <faultcode>soap:Server</faultcode>
    <faultstring>Insufficient funds</faultstring>
    <detail>
      <x:TransferError xmlns:x="urn:examples-org:banking">
        <sourceAccount>22-342439</sourceAccount>
        <transferAmount>100.00</transferAmount>
        <currentBalance>89.23</currentBalance>
      </x:TransferError>
    </detail>
  </soap:Fault>
</soap:Body>
```

- Fault codes: VersionMismatch, MustUnderstand, Client, Server,
 SOAP Header Element

- Header element is optional
- Can contain one or more sub-elements (called header blocks)
  - Each header block can be an element from some namespace
  - `mustUnderstand="1"` indicates receiver must understand this header block (mandatory)
  - Else return a Fault element
- Primary source for extensibility
  - Security tokens, routing information, processing instructions, …

- `<soap:Header>
  <!-- security credentials -->
  <s:credentials xmlns:s="urn:examples-org:security" soap:mustUnderstand="1">
    <username>dave</username>
    <password>evad</password>
  </s:credentials>
</soap:Header>`
SOAP Processing Model

• Three kinds of SOAP nodes
  – Initial sender, an intermediary, or ultimate receiver

• When processing a message, a SOAP node assumes one or more roles
  – Roles determine how headers are processed
    • Headers target specific roles using the global actor attribute (role in SOAP 1.2)
      – SOAP 1.1 defines only one role: http://schemas.xmlsoap.org/soap/actor/next
  – SOAP 1.1 defines only one role: http://schemas.xmlsoap.org/soap/actor/next

• A node first processes mandatory headers (mustUnderstand="1"), then others
SOAP Processing Model (cont’d)

• Example

```xml
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <wsrp:path xmlns:wsrp="http://schemas.xmlsoap.org/rp"
      soap:actor="http://schemas.xmlsoap.org/soap/actor/next"
      soap:mustUnderstand="1" > ... 
  </soap:Header>
</soap:Envelope>
```

• Fault element generated if node does not understand header
• Successful processing of a header removes it from the message
  • Can reinsert the header, but now treated as relationship between the intermediary node and the downstream node

• Ultimate receiver also responsible for processing the Body element
SOAP Protocol Bindings (HTTP)

- SOAP request/response mapped to HTTP Post/Reply model

```xml
POST /path/bank.asmx HTTP/1.1
Content-Type: text/xml
SOAPAction: "urn:banking:transfer"
Content-Length: nnnn
<soap:Envelope...)
```

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn
<soap:Envelope...
```

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml
Content-Length: nnnn
<soap:Envelope...
```
SOAP RPC Encoding

- Defines how to encapsulate RPC info within the SOAP body
  - Endpoint location (URI), method name, parameter names/values

- Method invocation modeled as a struct named after the method
  - Named fields for each in or in/out parameter

```xml
<soap:Body>
  <Reverse
    xmlns="http://netserver1.pdsg.cs.nyu.edu/vijayk">
    <arg>hi there</arg>
  </Reverse>
</soap:Body>
```

- Method response also modeled as a struct, named `<Method>Response`

```xml
<soap:Body>
  <ReverseResponse
    xmlns="http://netserver1.pdsg.cs.nyu.edu/vijayk">
    <ReverseResult>ereht ih</ReverseResult>
  </ReverseResponse>
</soap:Body>
```