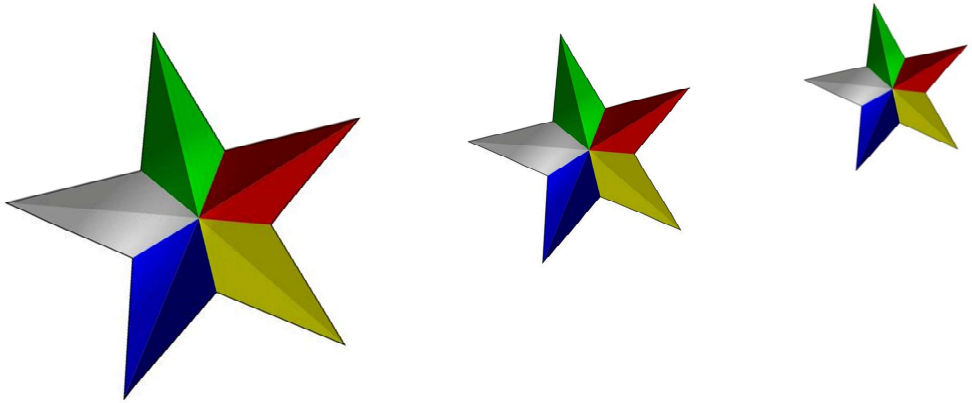
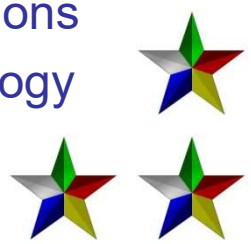


Web Services

Dr. M. Anand
Assistant Professor (Sr. G)
Dept. of Networking and Communications
SRM Institute of Science and Technology



What is Web Service?

- The term **Web services** describes a standardized way of integrating Web based applications using the **XML, SOAP, WSDL and UDDI** open standards over an Internet protocol backbone
  - XML is used to tag the data
  - SOAP is used to transfer the data
  - WSDL is used for describing the services available
  - UDDI is used for listing what services are available

What is Web Service?

- Used primarily as a means for businesses to communicate with each other and with clients, Web services allow organizations to communicate data without intimate knowledge of each other's IT systems behind the firewall

What is Web Service?

- Web services do not provide the user with a GUI
- Web services instead share business logic, data and processes through a programmatic interface across a network
- Developers can then add the Web service to a GUI (such as a Web page or an executable program) to offer specific functionality to users

How Does it Work?

- You can build a Java-based Web Service on Solaris that is accessible from your Visual Basic program that runs on Windows
- You can also use C# to build new Web Services on Windows that can be invoked from your Web application that is based on Java Server Pages (JSP) and runs on Linux

How Does it Work?

- **Example**
- Consider a simple account-management and order -processing system
- The accounting personnel use a client application built with Visual Basic or JSP to create new accounts and enter new customer orders.
- The processing logic for this system is written in Java and resides on a Solaris machine, which also interacts with a database to store the information.

How Does it Work?

The steps involved in the above example:

1. The client program bundles the account registration information into a SOAP message
2. This SOAP message is sent to the Web Service as the body of an HTTP POST request.

How Does it Work?

3. The Web Service unpacks the SOAP request and converts it into a command that the application can understand

The application processes the information as required and responds with a new unique account number for that customer.

4. Next, the Web Service packages up the response into another SOAP message, which it sends back to the client program in response to its HTTP request.

How Does it Work?

5. The client program unpacks the SOAP message to obtain the results of the account registration process

Benefits of using Web Services

- **Exposing the existing function on to network:**
  - A Web service is a unit of managed code that can be remotely invoked using HTTP requests.
  - So, Web Services allows you to expose the functionality of your existing code over the network.
  - Once it is exposed on the network, other application can use the functionality of your program.

Benefits of using Web Services

- **Connecting Different Applications ie Interoperability:**
  - Web Services allows different applications to talk to each other and share data and services among themselves.
  - Other applications can also use the services of the web services
  - For example VB or .NET application can talk to java web services and vice versa.
  - So, Web services is used to make the application platform and technology independent.

Benefits of using Web Services

- **Standardized Protocol:**
  - Web Services uses standardized industry standard protocol for the communication.
  - All the four layers (Service Transport, XML Messaging, Service Description and Service Discovery layers) uses the well defined protocol in the Web Services protocol stack
  - This standardization of protocol stack gives the business many advantages like wide range of choices, reduction in the cost due to competition and increase in the quality

Benefits of using Web Services

- **Low Cost of communication:**
  - Web Services uses **SOAP** over **HTTP** protocol for the communication
  - So you can use your existing low cost internet for implementing Web Services
  - This solution is much less costly compared to proprietary solutions like EDI/B2B
  - Beside SOAP over HTTP, Web Services can also be implemented on other reliable transport mechanisms like FTP etc.

Web Services Framework

1. Web Service Roles
2. Web Service Protocol Stack

Web Service Roles

- Three major roles within the web service architecture
  - **Service provider**
  - **Service requestor**
  - **Service registry**

Web Service Roles

- **Service provider:**
  - This is the provider of the web service
  - The service provider implements the service and makes it available on the Internet
- **Service requestor:**
  - This is any consumer of the web service
  - The requestor utilizes an existing web service by opening a network connection and sending an XML request.

Web Service Roles

- **Service registry:**
  - This is a logically centralized directory of services
  - The registry provides a central place where developers can publish new services or find existing ones.
  - It therefore serves as a centralized clearinghouse for companies and their services.

Web Service Protocol Stack

- In web service architecture, the protocol stack has four main layers
  - **Service transport**
  - **XML messaging**
  - **Service description**
  - **Service discovery**

Web Service Protocol Stack

- **Service transport**
  - This layer is responsible for transporting messages between applications
  - Currently, this layer includes hypertext transfer protocol (HTTP), Simple Mail Transfer Protocol (SMTP), file transfer protocol (FTP), and newer protocols, such as Blocks Extensible Exchange Protocol (BEEP)

Web Service Protocol Stack

- **XML messaging**
  - This layer is responsible for encoding messages in a common XML format so that messages can be understood at either end
  - Currently, this layer includes XML-RPC and SOAP

Web Service Protocol Stack

- **Service description**
  - This layer is responsible for describing the public interface to a specific web service
  - Currently, service description is handled via the **Web Service Description Language (WSDL)**

Web Service Protocol Stack

- **Service discovery**
  - This layer is responsible for centralizing services into a common registry, and providing easy publish/find functionality
  - Currently, service discovery is handled via **Universal Description, Discovery, and Integration (UDDI)**
- As web services evolve, additional layers may be added, and additional technologies may be added to each layer

SOAP

- **SOAP - Simple Object Access Protocol**
- SOAP is an XML-based protocol for exchanging information between computers

SOAP

- SOAP is a communication protocol
- SOAP is for communication between applications
- SOAP is a format for sending messages
- SOAP is designed to communicate via Internet
- SOAP is platform independent
- SOAP is language independent
- SOAP is simple and extensible
- SOAP allows you to get around firewalls
- SOAP will be developed as a W3C standard

WSDL

- **WSDL - Web Services Description Language**
- **WSDL** is the standard format for describing a web service in XML format
- **WSDL** is an XML-based language for describing Web services and how to access them

WSDL

- WSDL stands for Web Services Description Language
- WSDL is an XML based protocol for information exchange in decentralized and distributed environments
- WSDL is the standard format for describing a web service.
- WSDL definition describes how to access a web service and what operations it will perform

WSDL

- WSDL is a language for describing how to interface with XML-based services
- WSDL is an integral part of UDDI, an XML-based worldwide business registry
- WSDL is the language that UDDI uses
- WSDL was developed jointly by Microsoft and IBM

WSDL Usage

- WSDL is often used in combination with SOAP and XML Schema to provide web services over the Internet.
- A client program connecting to a web service can read the WSDL to determine what functions are available on the server.
- Any special datatypes used are embedded in the WSDL file in the form of XML Schema.
- The client can then use SOAP to actually call one of the functions listed in the WSDL

Elements of WSDL

- Three major elements of WSDL:
  - Types
  - Operations
  - Binding
- A WSDL document has various elements, but they are contained within these three main elements, which can be developed as separate documents and then they can be combined or reused to form complete WSDL files

Elements of WSDL Document

- **Definition**:
  - Element must be the root element of all WSDL documents
  - It defines the name of the web service, declares multiple namespaces used throughout the remainder of the document, and contains all the service elements described here.
- **Data types:**
  - the data types - in the form of XML schemas or possibly some other mechanism - to be used in the messages

Elements of WSDL Document

- **Message:**
  - an abstract definition of the data, in the form of a message presented either as an entire document or as arguments to be mapped to a method invocation.
- **Operation:**
  - the abstract definition of the operation for a message, such as naming a method, message queue, or business process, that will accept and process the message

Elements of WSDL Document

- **Port type:**
  - an abstract set of operations mapped to one or more end points, defining the collection of operations for a binding; the collection of operations, because it is abstract, can be mapped to multiple transports through various bindings
- **Binding:**
  - the concrete protocol and data formats for the operations and messages defined for a particular port type.

Elements of WSDL Document

- **Port:**
  - a combination of a binding and a network address, providing the target address of the service communication.
- **Service:**
  - a collection of related end points encompassing the service definitions in the file; the services map the binding to the port and include any extensibility definitions.

Elements of WSDL Document

- WSDL specification also defines the following utility elements:
- **Documentation:**
  - element is used to provide human-readable documentation and can be included inside any other WSDL element.
- **Import:**
  - element is used to import other WSDL documents or XML Schemas

The WSDL Document Structure

```
<definitions>
<types>
   definition of types.....
</types>

<message>
   definition of a message....
</message>

<portType>
      <operation>
         definition of a operation.....
      </operation>
</portType>
<binding>
   definition of a binding....
</binding>

<service>
   definition of a service....
</service>
</definitions>
```

UDDI

- **UDDI - Universal Description, Discovery and Integration**
- UDDI is an XML-based standard for describing, publishing, and finding Web services.
  - UDDI stands for Universal Description, Discovery and Integration.
  - UDDI is a specification for a distributed registry of Web services.
  - UDDI is platform independent, open framework.

UDDI

- UDDI can communicate via SOAP, CORBA, Java RMI Protocol.
- UDDI uses WSDL to describe interfaces to web services.
- UDDI is seen with SOAP and WSDL as one of the three foundation standards of web services.
- UDDI is an open industry initiative enabling businesses to discover each other and define how they interact over the Internet.

AJAX-Enabled Rich Internet Applications

Introduction

- RIAs –web applications that approximate the look, feel and usability of desktop applications.
- Two key attributes-**Performance** and **rich GUI.**
- Get **performance** from Ajax(Asynchronous JavaScript and XML),uses more responsive client-side scripting.
- Performance is achieved by separating client-side user interaction and server communication, and run them in parallel, reducing the delays of server-side processing normally experienced by the user.

- Traditional web applications use XHTML forms to build simple and thin GUIs
- RIA use Rich GUI with Ajax toolkits and with RIA environments such as Adobe's Flex, Microsoft's Silver light and Java Server Faces
- The client-side of Ajax applications is written in XHTML and CSS, and uses JavaScript to add functionality to the user interface.
- XML is used to structure the data passed between the server and the client. [use JSON (JavaScript Object Notation) also for this purpose.]

- The Ajax component that manages interaction with the server is usually implemented with JavaScript's XMLHttpRequest object — commonly abbreviated as XHR.
- The server processing can be implemented using any server-side technology, such as PHP, ASP.NET, Java Server Faces and Ruby on Rails.

Traditional Web Applications vs. Ajax Applications

Traditional Web Applications

- Step 1: User fills in the form's fields, then submits the form.
- Step 2:The browser generates a request to the server, which receives the request and processes it.
- Step 3:The server generates and sends a response containing the exact page that the browser will render.
- Step 4:browser loads the new page and temporarily makes the browser window blank. (Note that the client waits for the server to respond and reloads the entire page with the data from the response (Step 4)).

- While such a synchronous request is being processed on the server, the user cannot interact with the client web page.
- If the user interacts with and submits another form, the process begins again (Steps 5–8).

Traditional Web Application

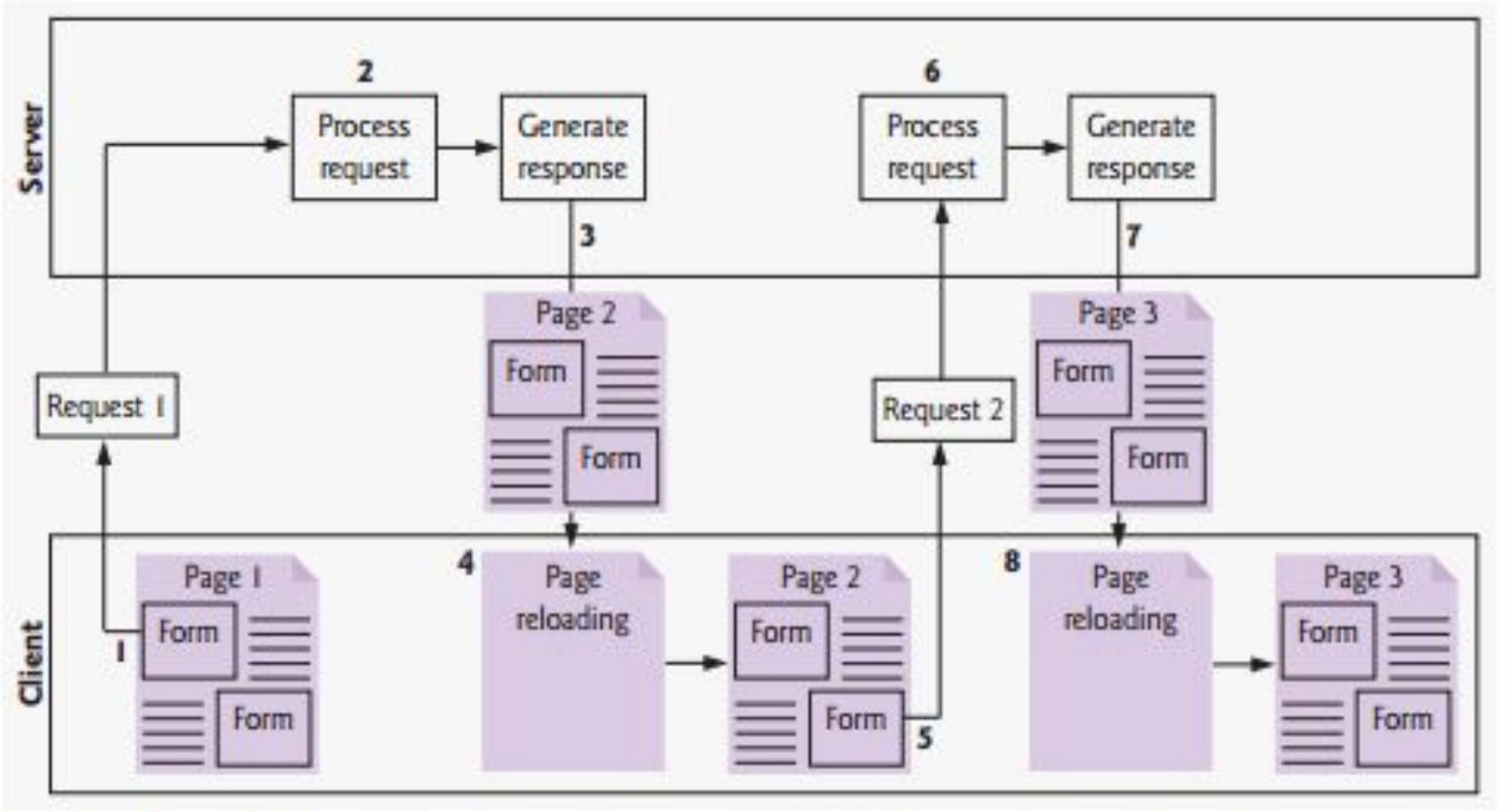


Fig. 15.1 | Classic web application reloading the page for every user interaction.

Ajax Web Applications

- Ajax applications add a layer between the client and the server to manage communication between the two
- Step 1: When the user interacts with the page, the client creates an XMLHttpRequest object to manage a request
- Step 2: The XMLHttpRequest object sends the request to the server (Step 2) and awaits the response. The requests are asynchronous, so the user can continue interacting with the application on the client-side while the server processes the earlier request concurrently.

- Steps 3 and 4:Other user interactions could result in additional requests to the server
- Once the server responds to the original request (Step 5), the *XMLHttpRequest* object which issued the request, calls a client-side function to process the data returned by the server.
- This function—known as a callback function—uses partial page updates (Step 6) to display the data in the existing web page without reloading the entire page.

- At the same time, the server may be responding to the second request (Step 7) and the client-side may be starting to do another partial page update (Step 8)
- The callback function updates only a designated part of the page. Such partial page updates help make web applications more responsive, making them feel more like desktop applications
- The non-Ajax web application does not load a new page while the user interacts with it.

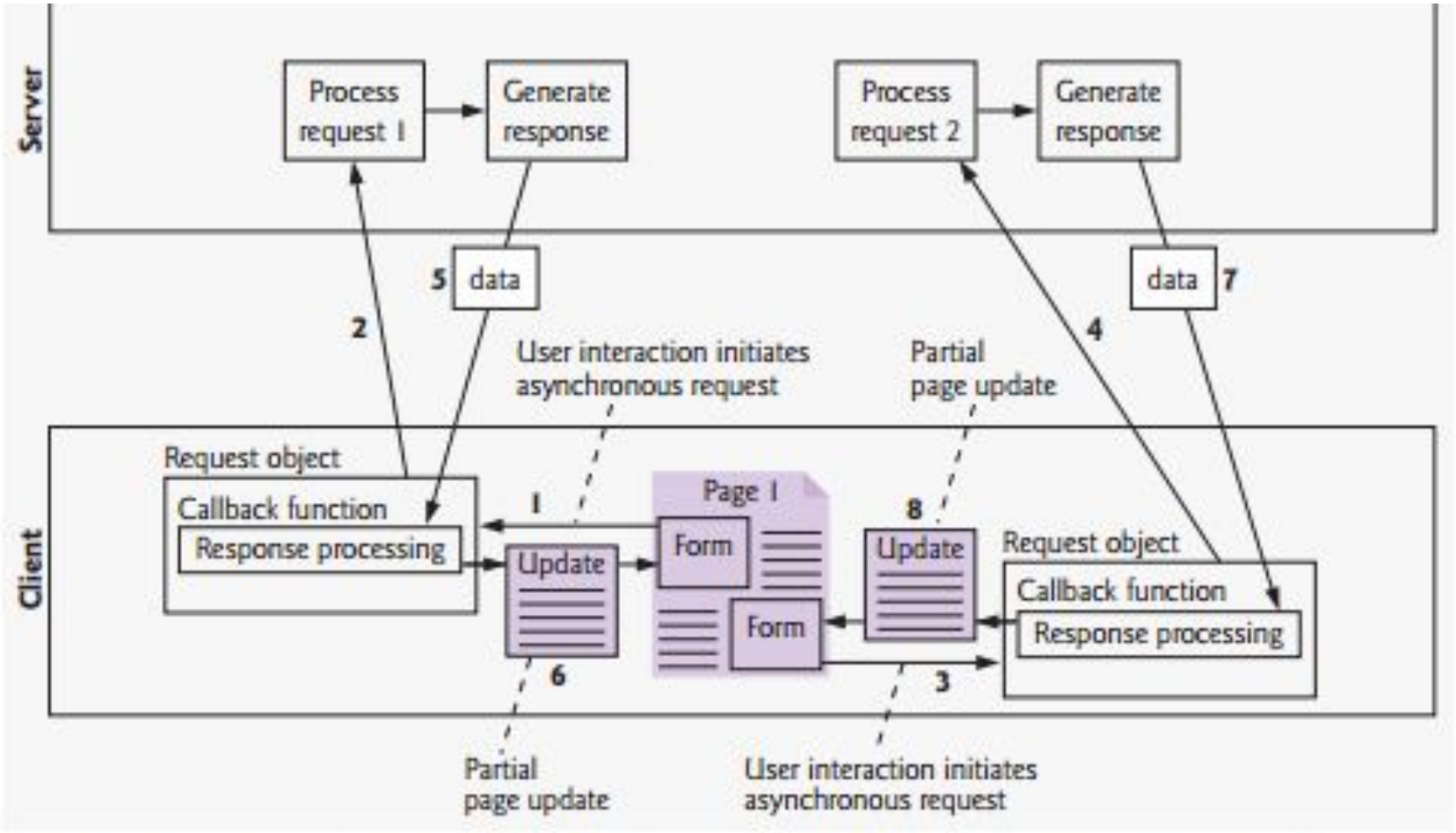


Fig. 15.2 | Ajax-enabled web application interacting with the server asynchronously.

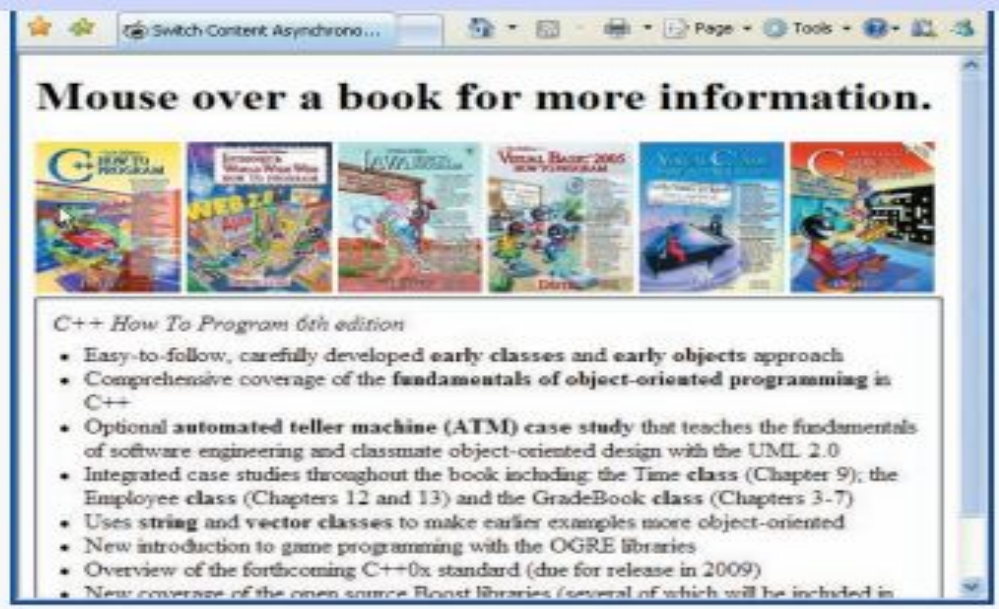
```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 15.5: SwitchContent.html -->
<!-- Asynchronously display content without reloading the page. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
   <style type="text/css">
      .box { border: 1px solid black;
             padding: 10px }
   </style>
   <title>Switch Content Asynchronously</title>
   <script type = "text/javascript" language = "JavaScript">
      <!--
      var asyncRequest; // variable to hold XMLHttpRequest object

      // set up and send the asynchronous request
      function getContent( url )
      {
         // attempt to create the XMLHttpRequest and make the request
         try
         {
            asyncRequest = new XMLHttpRequest(); // create request object

            // register event handler
            asyncRequest.onreadystatechange = stateChange;
            asyncRequest.open( 'GET', url, true ); // prepare the request
            asyncRequest.send( null ); // send the request
         } // end try
```

15.5 | Asynchronously display content without reloading the page. (Part 1 of 3.)

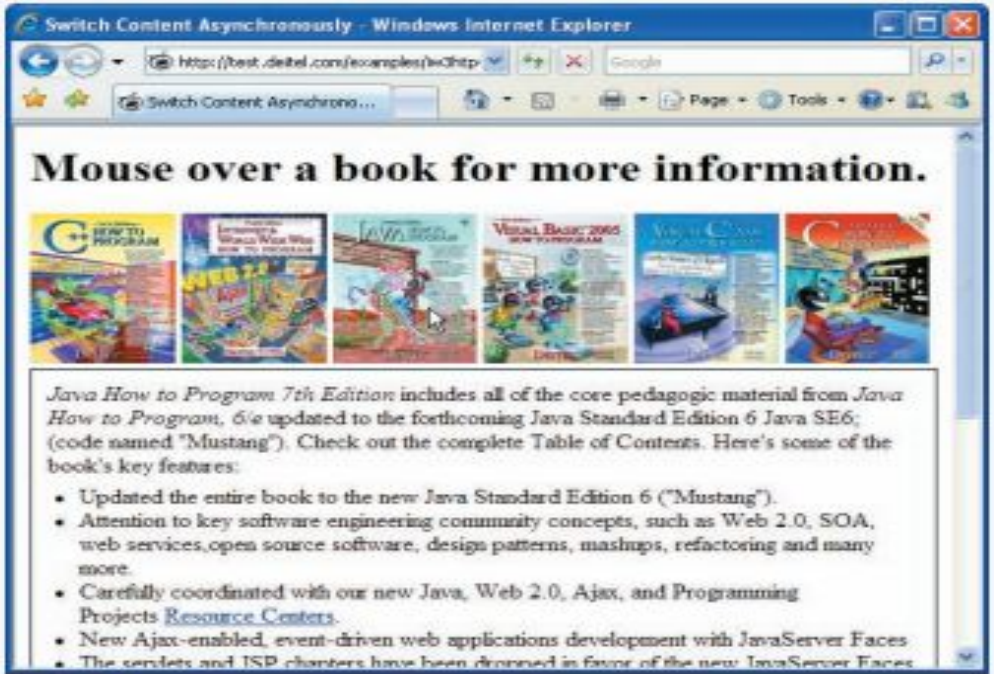
```
         catch ( exception )
         {
            alert( 'Request failed.' );
         } // end catch
      } // end function getContent

      // displays the response data on the page
      function stateChange()
      {
         if ( asyncRequest.readyState == 4 && asyncRequest.status == 200 )
         {
            document.getElementById( 'contentArea' ).innerHTML =
               asyncRequest.responseText; // places text in contentArea
         } // end if
      } // end function stateChange

      // clear the content of the box
      function clearContent()
      {
         document.getElementById( 'contentArea' ).innerHTML = '';
      } // end function clearContent
      // -->
   </script>
</head>
<body>
   <h1>Mouse over a book for more information.</h1>
   <img src =
      "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/cpphtp6.jpg"
      onmouseover = 'getContent( "cpphtp6.html" )'
      onmouseout = 'clearContent()'/>
   <img src =
      "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/iw3htp4.jpg"
      onmouseover = 'getContent( "iw3htp4.html" )'
      onmouseout = 'clearContent()'/>
   <img src =
      "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/jhtp7.jpg"
      onmouseover = 'getContent( "jhtp7.html" )'
      onmouseout = 'clearContent()'/>
   <img src =
      "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/vbhtp3.jpg"
      onmouseover = 'getContent( "vbhtp3.html" )'
```

```
      onmouseout = 'clearContent()'/>
   <img src =
      "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/vcsharphtp2.jpg"
      onmouseover = 'getContent( "vcsharphtp2.html" )'
      onmouseout = 'clearContent()'/>
   <img src =
      "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/chtp5.jpg"
      onmouseover = 'getContent( "chtp5.html" )'
      onmouseout = 'clearContent()'/>
   <div class = "box" id = "contentArea"> </div>
</body>
</html>
```

15.5 | Asynchronously display content without reloading the page. (Part 2 of 3.)



b) User hovers over *Java How to Program* book cover image, causing the process to repeat.



XMLHttpRequest Object Properties and Methods

Property	Description
`onreadystatechange`	Stores the callback function—the event handler that gets called when the server responds.
`readyState`	Keeps track of the request's progress. It is usually used in the callback function to determine when the code that processes the response should be launched. The `readyState` value `0` signifies that the request is uninitialized; `1` signifies that the request is loading; `2` signifies that the request has been loaded; `3` signifies that data is actively being sent from the server; and `4` signifies that the request has been completed.
`responseText`	Text that is returned to the client by the server.
`responseXML`	If the server's response is in XML format, this property contains the XML document; otherwise, it is empty. It can be used like a `document` object in JavaScript, which makes it useful for receiving complex data (e.g. populating a table).
`status`	HTTP status code of the request. A `status` of `200` means that request was successful. A `status` of `404` means that the requested resource was not found. A status of `500` denotes that there was an error while the server was proccessing the request.
`statusText`	Additional information on the request's status. It is often used to display the error to the user when the request fails.

Fig. 15.6 | `XMLHttpRequest` object properties.

Methods

Method	Description
`open`	Initializes the request and has two mandatory parameters—method and URL. The method parameter specifies the purpose of the request—typically `GET` if the request is to take data from the server or `POST` if the request will contain a body in addition to the headers. The URL parameter specifies the address of the file on the server that will generate the response. A third optional boolean parameter specifies whether the request is asynchronous—it's set to `true` by default.
`send`	Sends the request to the sever. It has one optional parameter, `data`, which specifies the data to be POSTed to the server—it's set to `null` by default.
`setRequestHeader`	Alters the header of the request. The two parameters specify the header and its new value. It is often used to set the `content-type` field.

Fig. 15.7 | `XMLHttpRequest` object methods. (Part 1 of 2.)

Method	Description
`getResponseHeader`	Returns the header data that precedes the response body. It takes one parameter, the name of the header to retrieve. This call is often used to determine the response's type, to parse the response correctly.
`getAllResponseHeaders`	Returns an array that contains all the headers that precede the response body.
`abort`	Cancels the current request.

Fig. 15.7 | `XMLHttpRequest` object methods. (Part 2 of 2.)

```
<!DOCTYPE html>
<html>
<head>
<script>
function loadXMLDoc(url)
{
var xmlhttp;
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {

document.getElementById('A1').innerHTML=xmlhttp.status;

document.getElementById('A2').innerHTML=xmlhttp.statusText
;

document.getElementById('A3').innerHTML=xmlhttp.responseTe
xt;
    }
  }
xmlhttp.open("GET",url,true);
xmlhttp.send();
}
```

Loading XML file(program continuation)

```
</script>
</head>
<body>

<h2>Retrieve data from XML file</h2>
<p><b>Status:</b><span id="A1"></span></p>
<p><b>Status text:</b><span id="A2"></span></p>
<p><b>Response:</b><span id="A3"></span></p>
<button onclick="loadXMLDoc('note.xml')">Get XML data</button>

</body>
</html>
```

Output


Retrieve data from XML file
Status:200
Status text:OK
Response: Tove Jani Reminder Don't forget me this weekend!
Get XML data

References

- “Internet and world wide web ,how to program” by Deitel and Deitel chapter 15

SOAP

SIMPLE OBJECT ACCESS PROTOCOL

- SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML.
- SOAP does not itself define any application semantics such as a programming model or implementation specific semantics; rather it defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules.

SOAP consists of three parts:

a. ***The SOAP envelope*** construct defines an overall framework for expressing what is in a message; who should deal with it, and whether it is optional or mandatory.
b. ***The SOAP encoding rules*** defines a serialization mechanism that can be used to exchange instances of application-defined datatypes.
c. ***The SOAP RPC*** representation defines a convention that can be used to represent remote procedure calls and responses.

Goals

- A major design goal for SOAP is simplicity and extensibility.

The SOAP Message Exchange Model

- SOAP messages are fundamentally one-way transmissions from a sender to a receiver, but SOAP messages are often combined to implement patterns such as request/response.
- SOAP regardless of the protocol to which SOAP is bound, messages are routed along a so-called "message path",
- which allows for processing at one or more intermediate nodes in addition to the ultimate destination.

A SOAP application receiving a SOAP message MUST process that message by performing the following actions in the order listed below:

- 1. Identify all parts of the SOAP message intended for that application
- 2. Verify that all mandatory parts identified in step 1 are supported by the application for this message and process them accordingly. If this is not the case then discard the message without affecting the outcome of the processing.

Relation to XML

- All SOAP messages are encoded using XML.
- A SOAP application MUST be able to process SOAP namespaces in messages that it receives. It MUST discard messages that have incorrect namespaces and it MAY process SOAP messages without SOAP namespaces as though they had the correct SOAP namespaces.
- SOAP defines two namespaces :
- The SOAP envelope has the namespace identifier "http://schemas.xmlsoap.org/soap/envelope/"
- The SOAP serialization has the namespace identifier "http://schemas.xmlsoap.org/soap/encoding/"
- A SOAP message MUST NOT contain a Document Type Declaration.
- SOAP uses the local, unqualified "id" attribute of type "ID" to specify the unique identifier of an encoded element. SOAP uses the local, unqualified attribute "href" of type "uri-reference" to specify a reference to that value, in a manner conforming to the XML Specification ,XML Schema Specification ,and XML Linking Language .
- With the exception of the SOAP must Understand attribute and the SOAP actor attribute

- A SOAP message is an XML document that consists of a mandatory SOAP envelope, an optional SOAP header, and a mandatory SOAP body.

- The SOAP envelope indicates the start and the end of the message so that the receiver knows when an entire message has been received.

SOAP Envelope element can be explained as:

- Every SOAP message has a root Envelope element.
- Envelope element is mandatory part of SOAP Message.
- Every Envelope element must contain exactly one Body element.
- If an Envelope contains a Header element, it must appear as the first child of the Envelope, before the Body.
- The SOAP envelope is specified using the *ENV* namespace prefix and the *Envelope* element.
- The optional SOAP encoding is also specified using a namespace name and the optional *encodingStyle* element, which could also point to an encoding style other than the SOAP one.

<?xml version="1.0"?>

<SOAP-ENV:Envelope

xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"

SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

- ...
- Message information goes here
- ...
- </SOAP-ENV:Envelope>

- The optional Header element offers a flexible framework for specifying additional application-level requirements.

  For example, the Header element can be used to specify a digital signature for password-protected services; likewise, it can be used to specify an account number for pay-per-use SOAP services.

SOAP Header element can be explained as:

- Header elements are optional part of SOAP messages.
- Header elements can occur multiple times.
- The SOAP header contains header entries defined in a namespace.
- The header is encoded as the first immediate child element of the SOAP envelope.
- When more than one header is defined, all immediate child elements of the SOAP header are interpreted as SOAP header blocks.

```
<?xml version="1.0"\\?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelop
    e"
SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-
    encoding">
<SOAP-ENV:Header>
<t:Transaction
xmlns:t="http://www.tutorialspoint.com/transaction/"
SOAP-ENV:mustUnderstand=“1"></t:Transaction>
</SOAP-ENV:Header>
...
...
```
- </SOAP-ENV:Envelope>

- The SOAP body is a mandatory element which contains the application-defined XML data being exchanged in the SOAP message.

```
<?xml version="1.0"?>

<SOAP-ENV:Envelope
.....
<SOAP-ENV:Body>
  <m:GetQuotation
   xmlns:m="http://www.tp.com/Quotation">
    <m:Item>Computers</m:Item>
  </m:GetQuotation>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- A corresponding SOAP response will look like :
- HTTP/1.0 200 OK
- Content-Type: text/xml; charset=utf-8
- Content-Length: nnn
-
- <?xml version="1.0"?>
- <SOAP-ENV:Envelope
- xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
- SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
- <SOAP-ENV:Body xmlns:m="http://www.xyz.org/quotation">
- <m:GetQuotationResponse>
- <m:Quotation>Here is the quotation</m:Quotation>
- </m:GetQuotationResponse>
- </SOAP-ENV:Body>
- </SOAP-ENV:Envelope>

- ***SOAP FAULT:***
- When an error occurs during processing, the response to a SOAP message is a SOAP fault and the fault is returned to the sender of the SOAP message.
- The SOAP fault mechanism returns specific information about the error, including a predefined code
- A SOAP Message can carry only one fault block
- Fault element is an optional part of SOAP Message
- A successful response is linked to the 200 to 299 range of status codes;
- SOAP fault is linked to the 500 to 599 range of status codes.

- **Error Description**
- SOAP-ENV:VersionMismatch
- Found an invalid namespace for the SOAP Envelope element
- SOAP-ENV:MustUnderstand
- An immediate child element of the Header element, with the mustUnderstand attribute set to "1", was not understood
- SOAP-ENV:Client
- The message was contained incorrect information
- SOAP-ENV:Server
- There was a problem with the server so the message could not proceed

```
<?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/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
    <faultcode xsi:type="xsd:string">SOAP-ENV:Client</faultcode>
    <faultstring xsi:type="xsd:string">
        Failed to locate method in class
</faultstring>

     </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- SOAP includes a built-in set of rules for encoding data types.
- This enables the SOAP message to indicate specific data types, such as integers, floats, doubles, or arrays.
- SOAP data types are divided into two broad categories: scalar types and compound types.
- Scalar types contain exactly one value, such as a last name, price, or product description.
- Compound types contain multiple values, such as a purchase order or a list of stock quotes.
- Compound types are further subdivided into arrays and structs.

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelop
    e"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
 <ns1:getPriceResponse    xmlns:ns1="urn:examples:priceservice"
    SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-
    encoding">
 <return xsi:type="xsd:double">54.99</return>
    </ns1:getPriceResponse></SOAP-ENV:Body></SOAP-ENV:Envel
    ope>
```

- **SOAP via HTTP**

The SOAP specification mandates that the client must provide a SOAPAction header

- For example, to access the labelFish Translation service, hosted by XMethods, you must specify the following as a SOAPAction header.
- urn:xmethodslabelFish#labelFish

- Here is a sample request sent via HTTP to the Xmethods labelfish Translation service:

- POST /perl/soaplite.cgi HTTP/1.0
- Host: services.xmethods.com
- Content-Type: text/xml; charset=utf-8
- Content-Length: 538
- SOAPAction: "urn:xmethodslabelFish#labelFish“
- <?xml version='1.0' encoding='UTF-8'?>
- <SOAP-ENV:Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/>
- <SOAP-ENV:Body>
- <ns1:labelFish xmlns:ns1="urn:xmethodslabelFish“
- SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
- <translationmode xsi:type="xsd:string">en_fr</translationmode>
- <sourcedata xsi:type="xsd:string">Hello, World!</sourcedata>
- </ns1:labelFish>
- </SOAP-ENV:Body></SOAP-ENV:Envelope>

- Here is the response from XMethods:
- HTTP/1.1 200 OK
- Date: Sat, 09 Jun 2001 15:01:55
- GMTServer: Apache/1.3.14 (Unix) tomcat/1.0 PHP/4.0.1pl2
- SOAPServer: SOAP::Lite/Perl/0.50
- Cache-Control: s-maxage=60, proxy-revalidateContent-Length: 539
- Content-Type: text/xml
- <?xml version="1.0" encoding="UTF-8"?>
- <SOAP-ENV:Envelopexmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"SOAP-ENV:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
- xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/>
- <SOAP-ENV:Body>
- <namesp1:labelFishResponse xmlns:namesp1="urn:xmethodslabelFish">
- <return xsi:type="xsd:string">Bonjour, monde!
- </return>
- </namesp1:labelFishResponse
- ></SOAP-ENV:Body>
- </SOAP-ENV:Envelope>

SOAP Message Embedded in HTTP Request

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
SOAPAction: "Some-URI"
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:GetLastTradePrice xmlns:m="Some-URI">
<symbol>DIS</symbol>
</m:GetLastTradePrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Message Embedded in HTTP Response

HTTP/1.1 200 OK
10/21/13 Simple Object Access Protocol (SOAP) 1.1
www.w3.org/TR/2000/NOTE-SOAP-20000508/ 5/34
Content-Type: text/xml; charset="utf-8"
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
<SOAP-ENV:Body>
<m:GetLastTradePriceResponse xmlns:m="Some-URI">
<Price>34.5</Price>
</m:GetLastTradePriceResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Thank You