Generate Dynamic Content on Cache Server

by

Aparna Yeddula

A project submitted to the Faculty of Graduate School of the

University of Colorado at Colorado Springs

in partial fulfillment of the

requirements for the degree of

Master of Science

Department of Computer Science

2002

This project for the Master of Science degree by

Aparna Yeddula

Has been approved for the

Department of Computer Science

By

_____

Advisor: C. Edward Chow

_____

Jugal K. Kalita

_____

Sudhanshu K. Semwal

Date _____

Generate Dynamic Content on Cache Server

By

Aparna Yeddula

Masters project directed by Professor C. Edward Chow

Department of Computer Science

Abstract

This project paper describes the implementation of a proxy cache using .NET web services, Java servlets, JSP custom tags and ESI resources to create and retrieve dynamic web pages on a cache server. Project paper includes the description of Edge Side Include (ESI) specification, installation of the ESI Test Server (ETS), examination of ETS process requests from the User and determination of the specific parts of the web page, which are needed for retrieval from the original server and finally performance testing with comparison of results using ESI edge suite and JSP custom tags. ESI allows dynamic content to be assembled at the very edges of the network**.** The usage of ESI 'include' and 'choose' tags is used to assemble a set of fragments of a web page. In order to create a dynamic cache server for generating ESI web pages based on JSP custom tags, JSP web pages with ESI tags will be created and the related tag library files and servlets will be developed for generating those web pages.

CONTENTS

FIGURES

Tables

CHAPTER 1

Introduction

With the World Wide Web (WWW) the user is able to retrieve all kind of information from the network without having any knowledge of the network. From the user point of view, it doesn't matter if the information he/she is looking for, e.g. a video clip, is on a computer in the next room, or on the other side of the world. With the use of Web growing so fast, it is to be expected that the WWW traffic on the national and international networks will also grow. Due to this enormous growth of traffic, congestion can occur on the local, national and international network backbone and affects the quality of service and the response times.

The quality of service and the response times can be improved by reducing the unnecessary network traffic. One answer to this problem is local **caching**, which is built into a Web browser. Web browsers, such as Internet Explorer and Netscape Navigator support this function. Files, graphics, Web pages are stored temporarily and can be retrieved to display on the screen as the end-user moves back and forth over a constrained set of Web pages. The Web browser also provides us with a way to by-pass the cache by holding shift/ctrl key and hit reload. Another answer to this problem is the **proxy cache [8]**. Web browsers have been given the ability to direct their resource requests to a local **web proxy server**, a device that is capable of altering the request before passing it on to the ultimate destination. Content delivery network (CDN) consists of client, proxy server, original web sites. In Figure1.1 CDN browser can be configured to request pages from a local server cache. Web proxy server acts as a conduit between Web server and browser by fetching documents if needed and passing them to the browser. Additionally, it can save copies of the documents to form a collection of the documents that are available when they are requested. Subsequent requests from other users of the cache get

the saved copy, which is much faster and does not consume Internet bandwidth over the often-congested network links.

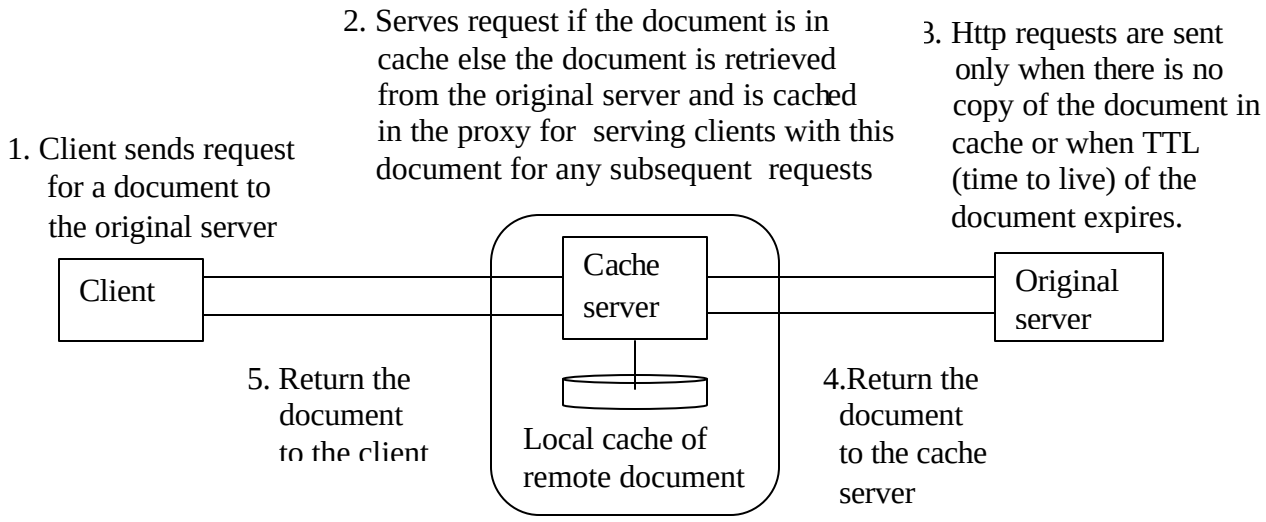

Figure 1.1. Content delivery with cache server

Traditionally the proxy server in CDN only serves the static web pages. It passes the dynamic web page request such as these .jsp, .asp, cgi script to the original web server. For web sites that serve dynamic content, the content on the web server can change for each individual user request or it can be updated frequently according to some schedule. For example stock quotes, auction-bidding pages, advertising banners, answer queries, news information, local time are such dynamic content. Generating dynamic web page imposes heavy burden on the original web server. To alleviate that, the generation of dynamic web pages can be done at the cache servers. One of the content delivery network providers Akamai [1] had proposed Edge Side Include (ESI) language for specifying how a web page can be dynamically generated. The rest of the paper is organized as follows:

Chapter 2: Discuss about the Akamai ESI language specifications

Chapter 3: Discuss about web server settings using Microsoft DOTNET and database access using Microsoft access and Active Server Page (ASP).

Chapter 4: Discuss how JSP custom tags to implement ESI and implementing the proxy on the web server.

Chapter 5: Testing the performance of my project with ESI.

Chapter 6: Conclusion and Future Work

CHAPTER 2

ESI specifications

In CDN serving dynamic pages is computationally intensive than serving static pages, because for static content the CDN needs to know what data its handling and what time to refresh the data, but for dynamic pages the CDN must also distinguish dynamic portions of the page from static, and know where to find dynamic data. ESI [2] language has this capability, ESI breaks pages into templates with common static elements like, logo, background, and navigational structure, and (HyperText Markup Language) HTML [3] fragments containing the dynamic portions of the page. Each fragment contains instructions about whether to cache the retrieved data and for how long should the cache copy be kept. Multiple users can share the template and the HTML fragment data. This allows edge servers to create dynamic pages locally, using locally cached content and referring back to the origin server only for missing data.

2.1. Akamai EdgeSuite2

The ESI language is conceptually similar in many ways to the Server Side Includes (SSI) function found in many server side script languages. It is an in-markup scripting language that is interpreted before the page is served to the client. The ESI assembly model is comprised of a template containing fragments. Figure 2.1 below shows a web page with 4 fragments, each fragment has its own time-to-live (TTL) attribute, which specifies how long the cache server maintains the copies.

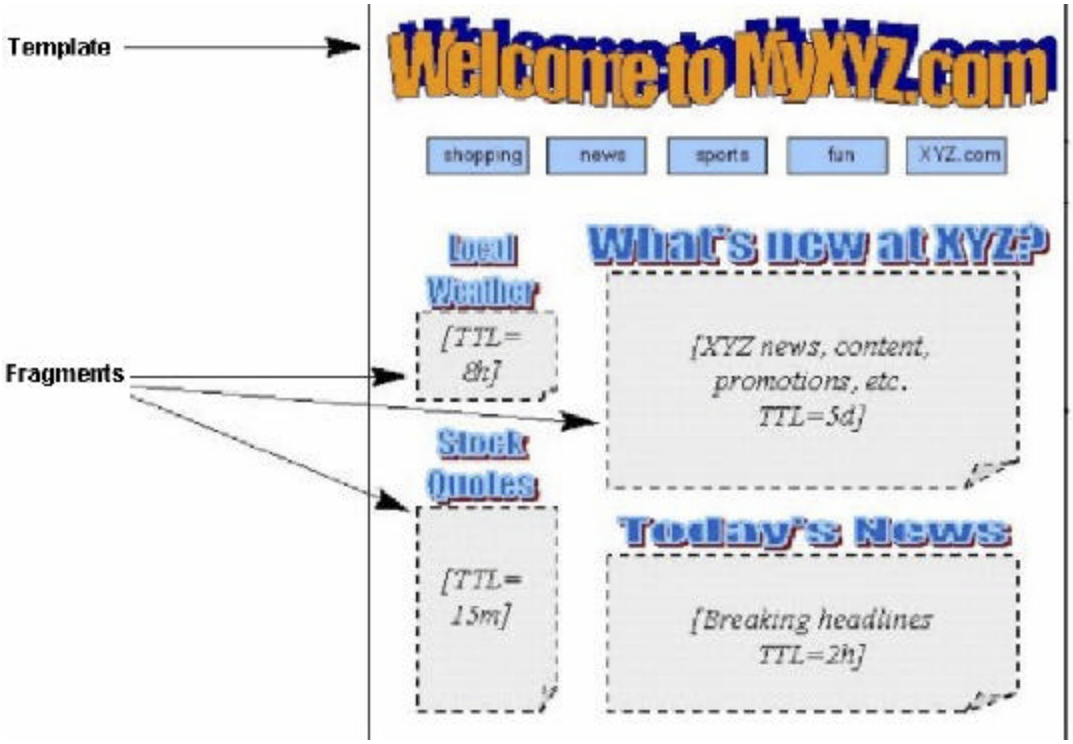

Figure 2.1. ESI template page containing ESI fragments and their expiration policies

The TTL value can be 5d (days) to 15m (minutes). The template is the container for assembly, with instructions for the retrieval of fragments, and is the resource associated with the (Universal Resource Locater) URL the end user requests. It includes ESI elements that instruct ESI processors (clients that understand ESI) to fetch and include a fragment's URI. The

fragments themselves can be any textual web resource, typically HTML markup. Because fragments are separate resources, they can be assigned their own cacheability and handling information. For example, a cache TTL of several days could be appropriate for the template, but a fragment containing a frequently changing story or advertisement may require a much lower TTL. Some fragments may require being marked uncacheable. ESI elements are specified in Extensible Markup Language (XML) with in an ESI-specific XML namespace. This allows them to be embedded in many common web document formats; including HTML and XML based server-side processing languages. EdgeSuite2 service delivers not only static content and streaming media, but also dynamic content from the network's edge.

How ESI delivers Dynamic Pages is shown in Figure 2.2 and explained in step by step below:

1. The user requests the content page, EdgeSuite running on the original web site directs the request to the closest cache server.

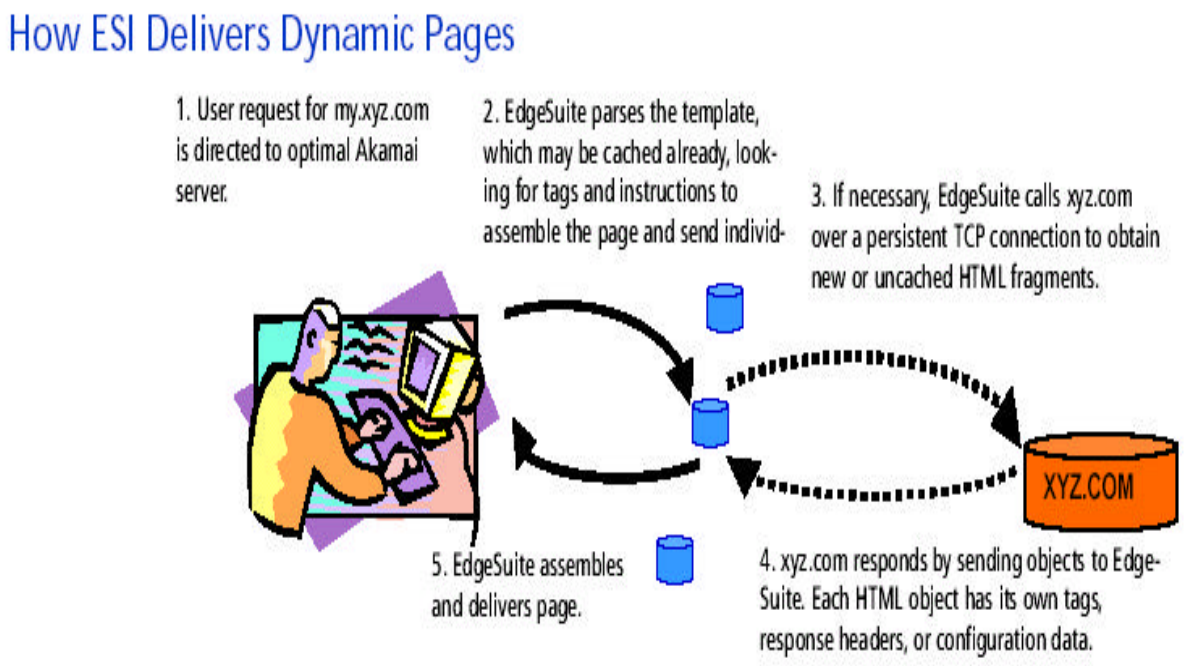

Figure 2.2. Edge Side Includes: How it works [3]

2. The template page associated with the request may already be cached, frequently used material. If the template isn't cached, EdgeSuite running on the cache server fetches it from xyz.com.
3. EdgeSuite sees the ESI language markup in the template; it reads the tags and instructions, conditions, and variables.
4. EdgeSuite calls xyz.com to request or validate any fragments.
5. The origin server here it is xyz.com, sends new objects back to EdgeSuite. Each object is an HTML fragment with its own associated configuration and header data.
6. EdgeSuite assembles and delivers the custom page to the user, and also caches appropriate objects for further use.

2.2. About ESI syntax

ESI can be embedded in documents such as HTML or XML. EdgeSuite ignores everything except elements that begin with <esi: or <! - -esi and ESI attributes can be arranged in any order within an ESI statement. ESI statements are case sensitive; ESI elements are lower case. ESI supported CGI environment variables require upper case.

2.3. ESI language elements

Total list of ESI language elements are listed in the www.esi.org web site. Some examples of the ESI language are shown in Table 2.1.

Table 2.1. ESI language elements

Type of task	Description	Type
Object inclusion	Create an include statement	Include
Conditional inclusion	Add conditional processing	Choose| when| otherwise
Alternative processing	Set alternative HTML to be used if ESI is not processed. Hide ESI statements if ESI is not processed	Remove <! - - esi - - >
Exception Handling	Set exception handling statement	Try | attempt | except
Comments	Add comments to code	Comment
Variables	Uses CGI variables	HTTP request and response headers

2.3.1. Object inclusion

The 'include' statement makes the essential ESI function, and it provides several optional attributes for alternative objects, error handling, caching, and dynamic processing.

Listing 2.1. Include statement

```
<esi:include src="http://www.akamai.com/frag1.html"
alt="http://www.akamai.com/frag2.html" onerror="continue" maxwait="500" ttl="4h"/>
                                        Or
<esi:include src="http://search.akamai.com search?query=$(QUERY_STRING{'query'})"/>
```

Of all the attributes shown in Table 2.2, only 'src' is mandatory rest of the attributes is optional (only some of the attributes from the www.esi.org site of the include statement are described in Table 2.2). The object specified by the 'src' or 'alt' is URL. A query string can also be added to the 'src' or 'alt' object as shown in the Listing 2.1 in it a query string is question

mark followed by 'key = value' pairs and value 'QUERY_STRING' is a CGI environment variable.

Table 2.2. 'include tag' statement attributes

Attribute	Type	Description
Src	Mandatory	The 'src' object must be fetch from the origin server
alt	Optional	The 'alt' object to be fetched if the 'src' object is not found
Onerror	Optional	The only argument 'continue' specifies ignoring failed fetches and continues serving the page without the results of the tag.
maxwait	Optional	A time-out period, in milliseconds, for EdgeSuite to wait for the src or alt to complete the fetch successfully
Ttl	Optional	A time interval for the fetched object to reside in cache before EdgeSuite revalidates that the object has not changed.

Another important attribute is the 'ttl' specifies the time-to-live. The TTL for the object is stored in EdgeSuite's cache. The max amount of time the content will be served before EdgeSuite issues an If Modified Since (IMS) request [3] to the origin server to check whether the object content has changed. EdgeSuite issues an IMS only if the object is requested. Value is an integer 0 or greater, examples ttl ttl=0s means that the object is cached but EdgeSuite will revalidate it every time it is requested. The unit specifier can be s (seconds), m (minutes), h (hours) or d (days). The specifers cannot be combined like 120m is ok, but 1d4h20m is not a valid entry.

2.3.2. Conditional Inclusion

Example for the conditional inclusion is choose | when | otherwise, comparable to the if-then-else mechanism in most of the languages.

Listing 2.2. Conditional Inclusion Statement

```
<esi:choose>
        <esi:when test = "$(REMOTE_ADDR) = = xyz.com">
                <esi:include src = "http://www.uccs.edu" />
        </esi:when>
        <esi:otherwise>
                <esi:include src = "http://www.web.uccs.edu" />
        </esi:otherwise>
</esi:choose>
```

Each 'choose' block must have at least one 'when' or more, the 'otherwise' element is optional. The 'when' tag is synonymous with 'if' and 'else-if', it evaluates the boolean expression using the test attribute. <esi:when test = "$(haha)"> where the 'haha' is the variable being tested. If the variable is empty or the test fails then it returns false.

2.3.3. Alternative Processing

Example for the alternative processing is remove statement, the use of remove statement is to include alternative HTML markup that browsers can display in the event ESI processing cannot be performed. Also can be used as <! - - esi and - - > tags to hide ESI statements in the event the content of the page is passed unprocessed to the browser.

Listing 2.3. Remove Statement

```
<esi:include src= "http://www.uccs.edu"/>
        <esi:remove>
                <a href = "http://www.uccs.edu/semester.html">www.semester.com</a>
        </esi:remove>
```

The remove statement provides for including valid HTML as output if the ESI markup is unprocessed, but removes the content if the markup is processed normally.

2.3.4. Exception Handling

Example for the exception handling is try | attempt | except. The use of it is shown in listing 2.4.

Listing 2.4. Exception Handling Statement

```
<esi:try>
        <esi:attempt>
            <esi:include src="http://www.akamai.com/ad1.html"/>
        </esi:attempt>
        <esi:except>
            <a href=www.akamai.com>www.akamai.com</a>
        </esi:except>
</esi:try>
```

EdgeSuite first processes the 'attempt' sub-block of the try block. A failed ESI include statement triggers an error; the processor then attempts to execute the contents of the 'except' sub-block. Statements other than 'include' do not trigger this error. The 'except' sub-block is not triggered if the 'src' object, or a default object is used, or if an onerror="continue" attribute is applied. If you use the onerror="continue" attribute in the include statement inside the try block, you run the risk of defeating the purpose in using 'except' block. If the 'attempt' fails and the onerror attribute tells EdgeSuite to skip the 'include', the 'except' block will not be processed and used for that 'include'.

2.3.5. Comment

We can add comments to a document using the 'comment tag'. It is formulated as follows:

```
        <esi:comment text="Just write some HTML instead"/>
```

These comments not processed, and are simply deleted by EdgeSuite when the file is processed. They are not included in the final output.

2.3.6. ESI Variable Support

Environmental variables supported in ESI are:

1. HTTP response and request headers
2. Akamai-specific environment variables

The environment variables are passed from templates to fragments, but not the other direction. The passing is automatic, but can override the value by setting a new value with the include statement for the fragment.

1. HTTP request and response headers

In EdgeSuite the HTTP headers can be used as variables, with the following conversion structure:

- The term "HTTP_" is prefixed to the name.

  For example, 'Accept-encoding' becomes 'HTTP_ACCEPT_ENCODING', 'Cookie' becomes 'HTTP_COOKIE', and 'Host' becomes 'HTTP_HOST'. The format of the variable reference is $(VARIABLE_NAME).

  ```
  <esi:when test="$(HTTP_COOKIE{'group'})=='Advanced'">
  ```

- The variable must be in upper case.
- Dashes (-) are replaced with underscores (_).

2. Akamai-specific environment variables

Variable names must be in UPPER CASE. The format of variable reference is $(VARIABLE_NAME). For example: $(QUERY_STRING)

An example of use in an include statement:

```
<esi:include src="http://uccs.edu/
        search?query=$(QUERY_STRING{'query'})"/>
```

Table 2.3. Akamai-specific variable supported in ESI

Variable Name	Type	Examples
GEO	Strings returned from several keys	Country code: US Region code: CA Network type: dsl
QUERY_STRING	A collection of name=value pairs, pairs separated by a "&"	'first'= 'Robin' & 'last'='Roberts'
REMOTE_ADDR	End user's IP address	123.234.243.132
REQUEST_METHOD	The HTTP request Method	GET, POST

2.4. Study of ESI

Example to show how to use 'include tag'

A document is viewed as consisting of a static document template and individual objects exhibiting different characteristics, which fill out this template. Figure 2.3 shows the snapshot of a personalized myyahoo.com page and the attribute TTL captures the length of time this object remains valid. Each individual object like the weather, stock quote and company news exhibit different sharing, cacheability and freshness time and this are referenced as different fragments in the template my.yahoo.com. Each fragment has different ESI include tag statements like

```
<esi:include src="http://www.yahoo.com/weather.html " ttl="5s"/>
```

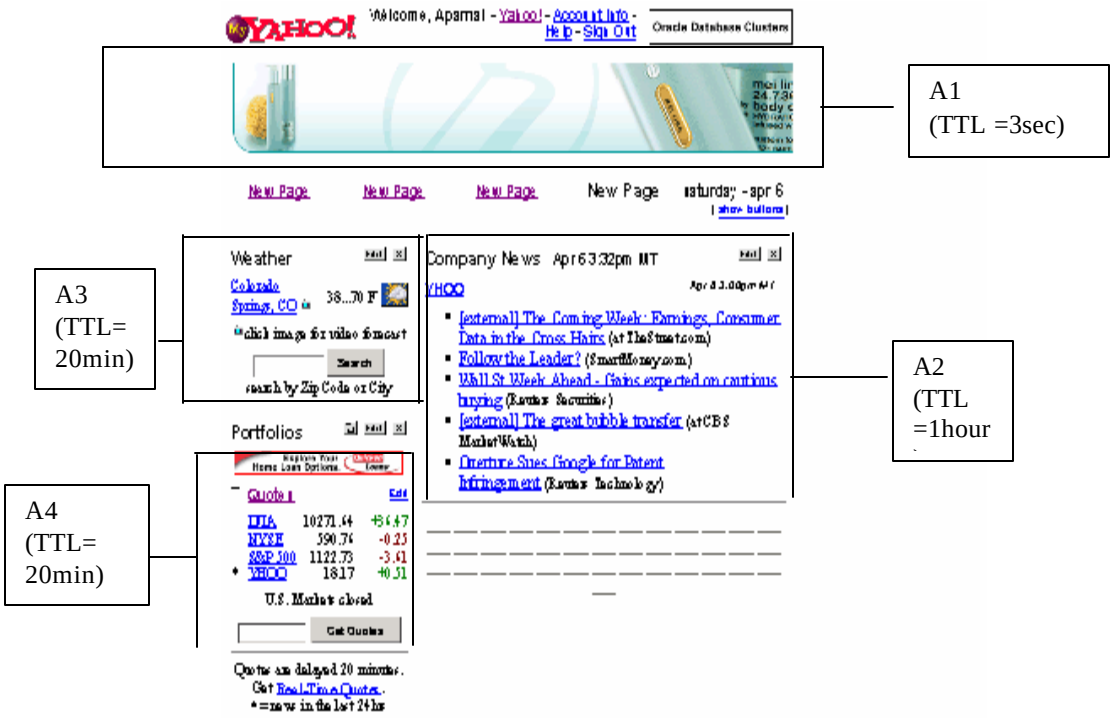

Figure 2.3. my.yahoo.com page exhibit different TTL

For the weather object and the attribute 'src' is URL to be fetched by the EdgeSuite. EdgeSuite validates these fragments and assembles and delivers the custom page to the browser. The attribute 'ttl' is set on a fragment; this value is used by the ESI to calculate the TTL value as defined in listing 2.5.

Example URL: http://gallop.uccs.edu/ayeddula/esi/esitag.html

Listing 2.5. ESI include tag example

```
<html>
<head><title>Sample web page for the ESI</title></head>
<body bgcolor = ccccee>
  <h4 align="center">Welcome to ESI page using Include Tag</h4>
  <p>
  <esi:include src="http://gallop.uccs.edu:8888/examples/staticpage.html" ttl="5s"/>
  <esi:include src="http://gallop.uccs.edu/ayeddula/tasks.html" ttl="5s"/>
```

```
</body>
</html>
```

Example to show how to use the REMOTE_ADDR variable

'REMOTE_ADDR' is an environment variable so must have to be in uppercase. The format of the variable reference is $(REMOTE_ADDR). An example to use in a conditional statement is shown in listing 2.6. The 'when' tag evaluates the expression using the 'test' attribute. The 'REMOTE_ADDR' gets the clients IP address and checks with the value, returns true if there is a match and processes the <esi:include - - /> tags else processes the <esi:include - - /> of the ESI 'otherwise' tag.

Example URL: http://gallop.uccs.edu/ayeddula/esi/conditionesi.html

Listing 2.6. ESI conditional statement example

```
<html>
<head><title>If Tag Example</title></head>
<body bgcolor=ccccee>
<h1>If Tag Example</h1>
<p> <esi:comment text="To test the codition statement."/>

<esi:choose>
        <esi:when test="$(REMOTE_ADDR) = ='128.198.60.23'">
                <esi:include src="http://gallop.uccs.edu/ayeddula/esi/staticpage.html"/>
        </esi:when>
        <esi:otherwise>
                <esi:include src="http://gallop.uccs.edu/ayeddula/tasks.html"/>
        </esi:otherwise>
</esi:choose>

</body>
</html>
```

CHAPTER 3

Web services specifications

The web services from the DOTNET provide us a simple, flexible, standards-based model for integrating network applications together over the Internet that takes advantage of existing infrastructure and applications. Web services provide us with application integration, i.e., taking a group of applications and turning them into user-friendly web applications. Those applications can run on different operating systems, can be created with different programming languages, and built with different object models. Developers can reuse without worrying about how to implement the service. Web services provide well-defined interfaces that describe the services they represent. Developers can assemble applications by using a combination of remote services, local services, and custom code.

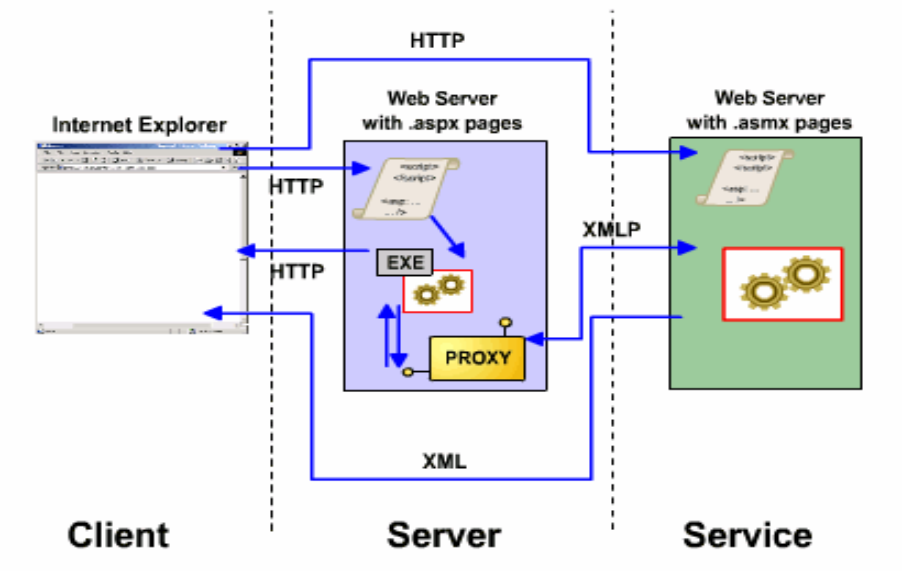

Figure 3.1. Illustrate how web services are used between client and web server [4]

Web services communicate by using standard web protocols and data formats, such as HTTP, XML, Simple Object Access Protocol (SOAP) and Extensible Markup Language Protocol (XMLP). Web service can be used internally by a single application, or it can be used externally by many applications that access it through the Internet. Because it is accessible through a standard interface, a web service allows disparate systems to work together. The web services model is independent of languages, platforms, and object models. Each time a service request is received, a new object is created. The request for the method call, and the object is destroyed after the method call is returned. Figure 3.1 shows web service model.

In the web service model, the web service developer:

- Creates the .asmx file that includes the namespace, classes, properties, and methods.
- Declares methods as Web methods that can be accessed over the Internet.

The following is an example of a simple .asmx file:

Listing 3.1. MathService.asmx

```
<%@ WebService Language="VB" Class="MathService" %>
Imports System.Web.Services
Imports System
Class MathService
   <WebMethod()> Public Function  Add(int1 As Integer, int2 As Integer) As Integer
      return(int1 + int2)
   End Function
End Class
```

Direct client model

1. The client browser issues a GET HTTP request .asmx page directly to the web service. The web service sends the service description to the client about which methods are available.

2. When we call a web service from a browser, we access the description page, which lists the methods that are included in the Web service. The protocol that is used in this case is HTTP, and the data is returned as XML.

Web server or the proxy client model

1. The client browser issues a GET HTTP request .aspx page to the web server. The server parses and compiles the page. The code invokes the proxy to call the web service. Here the request and response between the servers is using XMLP (shown in Figure 3.1). And the web server sends the response to the client in HTTP.

3.1. Calling a Web Service from a Browser

Example below shows how to call the web service from a browser; here we access the description page. Figure 3.2 shows how the description page looks; this page lists the methods that are included in the web service shown in listing 3.2. The protocol that is used in this case is HTTP. This HTML description page provides information about what a web service does, the methods it contains and their parameters, and its response type. In addition, you can use the description page to test the functionality of the Web service. For example, suppose that you access a web service called 'Service1.asmx' that is used to retrieve details from a Service1. The base URL for this service is 'http://gallop.uccs.edu/ayeddula/example1/Service1.asmx'. Entering the base URL with no extensions or parameters produces a page that displays information about the service and the methods it contains.

Example URL: http://gallop.uccs.edu/ayeddula/example1/Service1.asmx

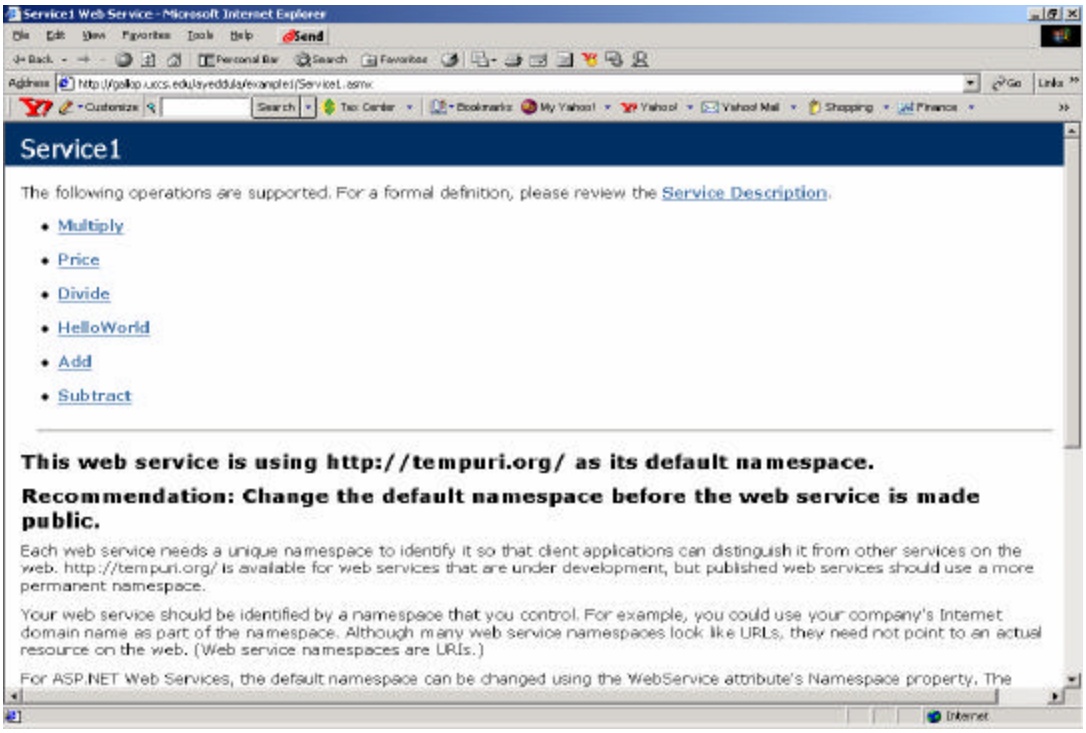

Figure 3.2. Description page

Listing 3.2

```
<%@ WebService Language="VB" Class="Service1" %>
Imports System.Web.Services
Imports System

Class Service1
        <WebMethod()> Public Function HelloWorld() As String
          HelloWorld = "Hello World"
    End Function
    <WebMethod()> Public Function Price() As String
                Price = "The item costs $10"
        End function
    <WebMethod()> public Function Add(int1 As Integer, int2 As
                    Integer) As Integer
                return(int1 + int2)
        End Function
   <WebMethod()> public Function Subtract(int1 As Integer,
              int2 As Integer) As Integer
                return(int1 - int2)
    End Function
    <WebMethod()> public Function Multiply(int1 As Integer,
                int2 As Integer) As Integer
            return(int1 * int2)
```

```
        End Function
    <WebMethod()> public Function Divide(int1 As Integer,
                int2 As Integer) As Integer
        return(int1 / int2)
    End Function
End Class
```

We can call the methods of a web service by clicking on that method and enter the parameters or we can also call the methods of a web service directly from a browser by passing the name of the method, the required parameters, and the values of the parameters to the URL to the Web service. The web service returns data in XML format when it is called from a Web browser by using HTTP. For example, if a web service named 'service1' has a method named 'Multiply' that takes two parameters named 'int1' & 'int2', you can call it directly from the browser by viewing the following URL:

http://gallop.uccs.edu/ayeddula/example1/Service1.asmx/Multiply?int1=3&int2=2

The following XML is data that was returned from this URL:

```
<?xml version="1.0" encoding="utf-8" ?>
        <int xmlns="http://tempuri.org/">6</int>
```

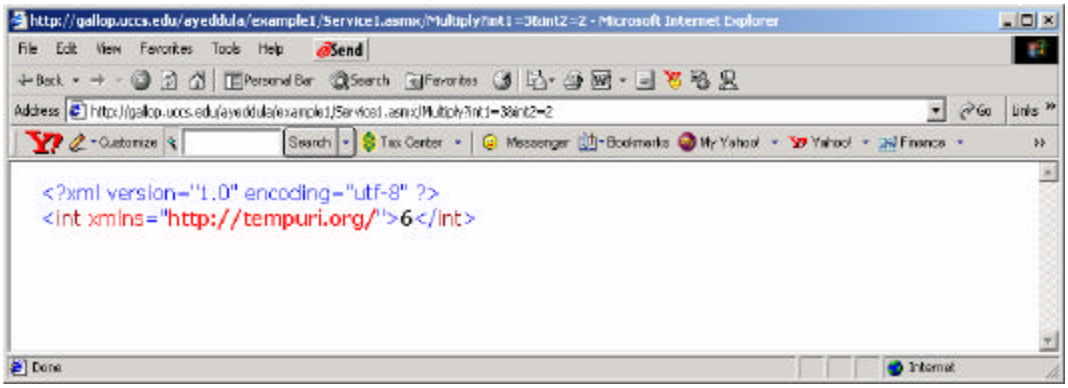

Figure 3.3. Return document in XML format

3.2. Creating Active Server Page with DB access

In order to create a '.asp' page for database access I have created a database using Microsoft Access and saved in the directory C:\ayeddula\ testaspfiles\ cs401signup.mdb.

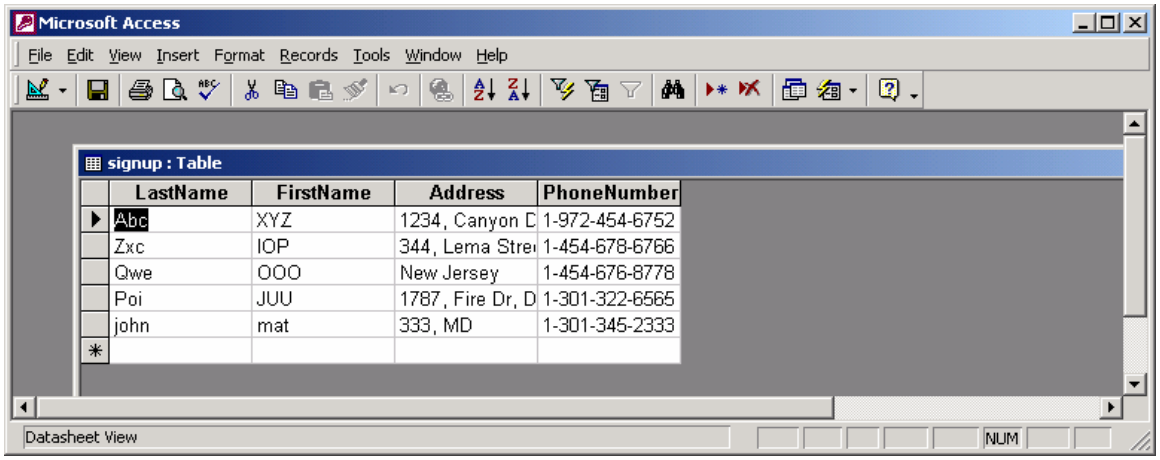

Figure 3.4. Database created using Microsoft Access

Then go to 'administrative tools' menu and double click on 'Data Sources (ODBC)' select 'System DSN' click on 'add' button to add new System Data Source then the Create New Data Source window will open as shown in the Figure 3.5.

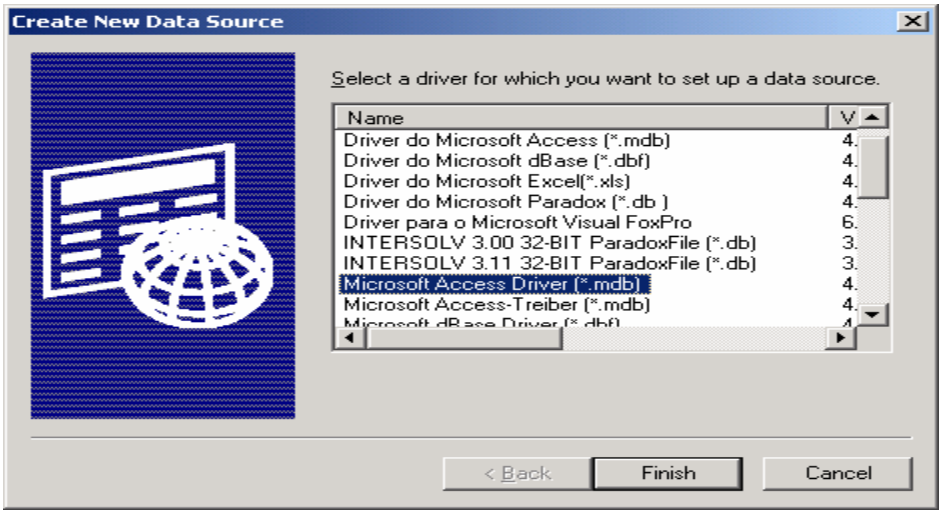

Figure 3.5. Create New Data Source Window

Click on the ‘Finish’ button to configure the ODBC setup. ‘testasp’ is the data source name which I can access in my ‘.asp’ for creating new connection to the database using the ODBC driver.

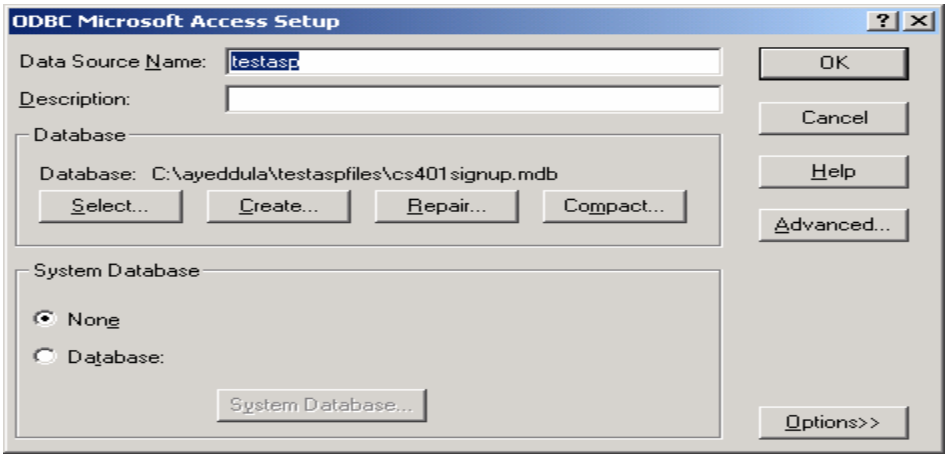

Figure 3.6. ODBC configuration

Listing 3.3 is my ‘.asp’ file ‘Server.CreateObject("ADODB.Connection")’ is used to create a connection to the ODBC. ‘Con.Open "testasp"’ is used to open a connection to the database ‘testasp’.

Listing 3.3. Active Server Page with DB access example

```
<%
        ' Dim myopt as String
        ' Dim mystr as String
         Set Con = Server.CreateObject("ADODB.Connection")
                Set RS = Server.CreateObject("ADODB.RecordSet")
                Con.Open "testasp"
                 myopt = Request.Form("address")
                 if myopt = "opt_lname" then
                        mylname = Request.Form("txt_lname")
                        mystr = "select * from signup where LastName='"& mylname &"'"
                elseif myopt = "opt_all" then
                        mystr = "select * from signup"
                end if
                 Rs.Open mystr,Con,adOpenDynamic
                 while not rs.EOF
%>
  <tr>    <td width="20%"><%=RS("LastName")%></td>  </tr>
```

```
<%
        RS.MoveNext
        end
        RS.Close
        set RS= nothing
        set Con=nothing
%>
```

Listing 3.4 Search.htm for the asp page

```
<html>
<head><title> Data Base Access using Microsoft Access </title></head>
<body>
        <form method="POST" action="response.asp">
        <h1 align="center">Address</h1>
        <table border="0" width="60%">
           <tr>
              <td width="38%"><input type="radio" value="opt_lname" checked
               name="address"> Last Name </td>
              <td width="62%"><input NAME="txt_lname" SIZE="30"> </td>
           </tr>
           <tr>
              <td width="38%"><input type="radio" name="address"
               value="opt_all">  Show All</td>
              <td width="62%"> </td>
           </tr>
           <tr>
                <td width="38%" align="left">
                <input type="submit" value="Submit"></td>
                 <td width="62%"><input type="reset" value="Reset" name="B1">
                </td>
           </tr>
        </table>
        <p>
        </form>
</body>
</html>
```

3.3. Study of .NET web services

3.3.1. Example 1

Example 1 show the change in date and time using DOTNET on the server side with '.asmx' page.

Example URL: http://gallop.uccs.edu/ayeddula/asmxtime/Service1.asmx

Listing 3.4.

```
<%@ WebService Language="VB" Class="timeasmx" %>
Imports System.Web.Services
Imports System
Class timeasmx
        <WebMethod()> Public Function GetTime() As String
            Dim dtNow As DateTime
            dtNow = DateTime.Now
            GetTime = dtNow.ToString
        End Function
End Class
```

3.3.2. Example 2

Example 2 shows changes in date and time using DOTNET on the server side with '.aspx' page.

Example URL: http://gallop.uccs.edu/ayeddula/aspxtime/WebForm1.aspx

Listing 3.5. WebForm1.aspx file

```
Sub Button1_Click(ByVal s As Object, ByVal e As EventArgs)
   Dim service As New aspxtime.localhost.Service1()
   Label1.Text = service.GetTime()
End Sub
```

Listing 3.6. HTML file

```
<form id="Form1" method="post" runat="server">
    <asp:button id="Button1" onclick="Button1_Click"
    runat="server"  Text="Click "></asp:button>
        <asp:label id="Label1" runat="server" ></asp:label>
        Click on the button to get the current time.
</form>
```

CHAPTER 4

JSP custom tag specifications

JSP custom tag can be used to implement XML languages including ESI. To use the JSP custom tags, we need to define three separate components: the tag handler class that defines the tag's behavior, the tag library descriptor (TLD) file that maps the XML elements names to the tag implementations and the JSP file that uses the tag library [5].

4.1. The JSP file

In the Hello.jsp file we have taglib directive, this directive has the form '<%@ taglib uri="…" prefix="…" %>'. The required uri attribute refers to a tag library descriptor file like shown in Listing 4.1. The 'prefix' attribute specifies that it will be used in front of whatever tag name the tag library descriptor defined. For example, if the TLD file defines a tag named 'hello' and the prefix attribute has a value of 'jspx', the actual tag name would be 'jspx:hello'. The tag could be used in either of the following two ways, depending on whether it is defined to be a container that makes use of the tag body:

<jspx:hello>body of the tag</jspx:hello>

Or just

<jspx:hello/>

Listing 4.1. Hello.jsp

```
<! DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
        <head>
                <%@ taglib uri="simple-taglib.tld" prefix="jspx" %>
```

```
                <Title><jspx:hello /></title>
        </head>
        <body>
                <H1 align="center"> JSP Custom Tag Test Page </H1>
                <H2>First JSP Custom Tag
                        <jspx:hellotest />
                </H2>
        </body>
</html>
```

4.2. Tag library descriptor file

After we define tag handler, next we need to identify the class to the server and to associate it with a particular XML tag name. In the TLD file basically we describe the mapping between the tags and the related servlets that processes the tags. For example the jspx:hello tag will be processed by cwp.tags.HelloWorldTag servlet code.

Listing 4.2. simple-taglib.tld

```
   <!DOCTYPE taglib
     PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
        "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_1.dtd">
  <taglib>
        <tlibversion>1.0</tlibversion>
        <jspversion>1.1</jspversion>
        <shortname>simple</shortname>
        <uri> simple-taglib.tld</uri>

         <tag>
           <name>hello</name>
           <tagclass>cwp.tags.HelloWorldTag</tagclass>
         </tag>
  </taglib>
```

4.3. Tag handler class

The Java class tells the system what to do when it sees the tag. This class must implement that javax.servlet.jsp.tagext.TagSupport interface. Listing 4.3 is an example of a simple tag that just inserts "Custom tag example (cwp.tags.HelloWorldTag)" into the JSP page wherever the corresponding tag is used.

Listing 4.3. HelloWorldTag.java

```
package cwp.tags;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.TagSupport;  // tells system what to do when it sees the tag

 // This is a simple tag example to show how content is added to the
// output stream when a tag is encountered in a JSP page.

public class HelloWorldTag extends TagSupport
{
        // doStartTag is called by the JSP container when the tag is encountered
        public int doStartTag()
        {
                try
                {
                        JspWriter out = pageContext.getOut();
                        out.println("<table border=\"1\">");
                        out.println("<tr><td> Hello World </td></tr>");
                 } catch (Exception ex) {
                  throw new Error("All is not well in the world.");
                 }
          // Must return SKIP_BODY because we are not supporting a body for this tag
          return SKIP_BODY;
        }
}
```

4.4. Implementing Proxy caching

The figure 4.1 shows the implementation of the proxy cache server.

1. At first the user tries to access a URL i.e. the client sends an HTTP request to the proxy server.

2. The proxy server checks its local cache for the requested object.

   - If it is a 'miss' (not available), the proxy server requests the object from the origin web server, stores the object in the cache and forwards to the client.
   - If it is a 'hit', i.e. object is found with an TTL expiration in the future, then the proxy server compares timestamps, i.e. the time of the request to the last requested object is less then the TTL expiration date then it is fresh, the cached object is sent to client directly; if stale, the object is requested from the origin web server, stores the object in the cache and forwarded to the client [9].

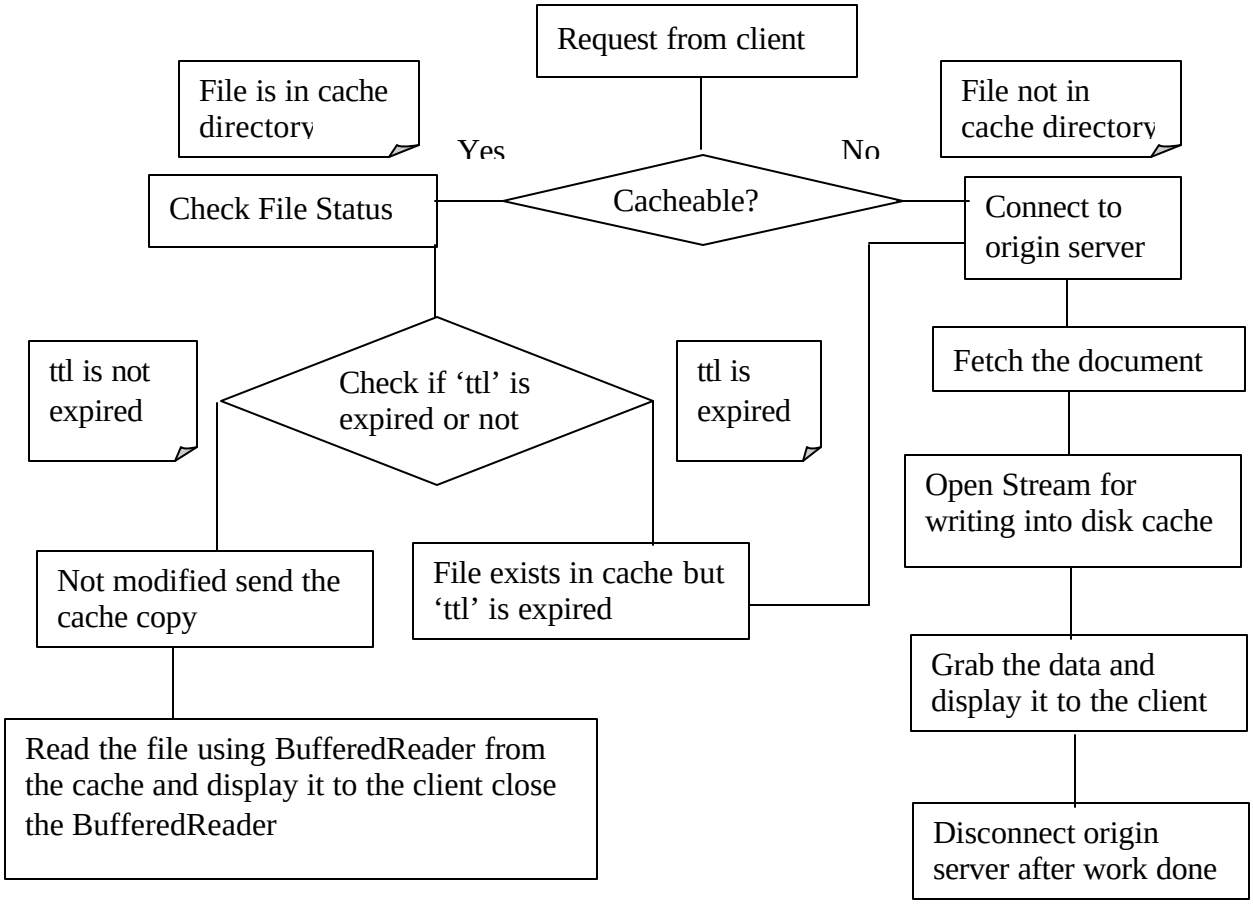

Figure 4.1. Implementing the proxy cache server

Include Tag jsp file

In the IncludeTag.jsp file in Listing 4.4 we have taglib directive tag as '<%@ taglib uri="include-taglib.tld" prefix="jspx" %>'. The required 'uri' attribute refers to a tag library descriptor file. The prefix attribute 'jspx' specifies that it will be used in front of whatever tag name the tag library descriptor defined. The tag is specified in the following way

```
<jspx:include uri="http://gallop.uccs.edu:8888/examples/staticpage.html" ttl="5"/>
```

Listing 4.4. IncludeTag.jsp

```
<%@ page import="java.util.*" %>
<html>
<head> <title> JSP Include Custom Tag </title> </head>
<body bgcolor=ccccee>
      <%@ taglib uri="include-taglib.tld" prefix="jspx" %>
         The current time is: <%= new Date() %>
         <h4 align="center">JSP Custom Tag Page Using Include Tag</h4>
         <a href="http://gallop.uccs.edu/ayeddula/includetag.txt"> view source </a>
         <p> TTL is set for 5 seconds <br>
        <jspx:include uri="http://gallop.uccs.edu:8888/examples/staticpage.html" ttl="5"/>
        <jspx:include uri="http://gallop.uccs.edu/ayeddula/tasks.html" ttl="5"/>
</body>
</html>
```

Tag library descriptor file

We defined tag handler, next we need to identify the class to the server and to associate it with a particular XML tag name. In the TLD file we describe the mapping between the tags and the related servlet that process the tags. The 'jspx:include' tag will be processed by 'cwp.tags.IncludeTag' servlet code. The 'uri' and 'ttl' are the two attributes which are set to true the uri = "http://gallop.uccs.edu:8888/examples/staticpage.html" ttl = "5"

Listing 4.5. include-taglib.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

```
<!DOCTYPE taglib  PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
"http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_1.dtd">
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <tag>
      <name>include</name>
      <tagclass>cwp.tags.IncludeTag</tagclass>
      <attribute>
         <name>uri</name>
         <required>true</required>
      </attribute>
      <attribute>
            <name>ttl</name>
            <required>true</required>
      </attribute>
  </tag>
</taglib>
```

Tag handler class

Listing 4.6 is the tag handler class, tag that just inserts "Custom tag example (cwp.tags.IncludeTag)" into the JSP page wherever the corresponding tag is used. The servlet starts to do some processing when it sees the tag. 'setUri (String name)' and setTtl (String name) are called to set the Tag attribute.

Listing 4.6. IncludeTag.java file

```
import java.io.*;
import java.net.*;
import java.util.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.TagSupport;// tells system what to do when it sees the tag.
import javax.servlet.http.*;
import javax.servlet.*;
public class IncludeTag extends TagSupport   {
          String uriName;                    // URI name
          String ttlName;                   // ttl name
          int ttlValue;                    // time to live value
          protected Writer writer;              // OutputStreamWriter
          protected BufferedReader reader;        // InputStreamReader
          protected String fileHost, fileName;     // FileHostName and FileName
```

```
protected int filePort;                 // FilePort
static int count =0;

// Cache Directory
String cacheDirector = "C:\\ayeddula\\tomcat\\jakarta-tomcat-
            4.0.1\\webapps\\examples\\WEB-INF\\classes\\cwp\\tags\\proxy\\";

public int doStartTag() throws JspException {
try {
        //content type to print on the browser
        JspWriter out = pageContext.getOut();       // JSP writer to the browser
        Process p= null;
        if(count == 0)          {
           Runtime runtime = java.lang.Runtime.getRuntime();
           p = runtime.exec("perl c:\\ayeddula\\alternate.pl");
          count++;
        }
        URL url = new URL (uriName);                    //split the URL
        fileHost = url.getHost ();
        filePort = url.getPort ();

        if (filePort == -1)   filePort = 80;            // default PORT is 80

        fileName = url.getFile ();
        ttlValue = getTtl(ttlName);                     // convert string to integer

         // To check if the page is in cache or not
        if (checkFile(fileName) == false)               {
                // File is * not * in cache directory get from origin server
                // Establish connection and get the file from the Origin server
                // also Save a copy in cache
                grab();
        }
        else            {
                // File is in cache directory
                // Check the files last modified date in the origin server
if (checkFileStatus(cacheDirector + getFileName(fileName),ttlValue) == 1) {
                // not modified send the cache copy to the browser
                BufferedReader in = null;
                File inFile;
                String line;
        in = new BufferedReader(new InputStreamReader(new
        FileInputStream(new File(cacheDirector+ getFileName(fileName)))));
        while ((line= in.readLine()) != null)
                 out.println(line);
         in.close();
```

```
            }
            else            {
                    // modified on the origin server save a copy in cache
                    // send the cache copy to the browser
                    grab();
            }
    }
    }
    catch(Exception ioe)   {
            throw new JspTagException("Error : ");
    }
            return SKIP_BODY;
    } // end do start tag

    // Check if will expired or not
    private int checkFileStatus(String filepath, int ttl) throws Exception          {
            ……….
    }

    // Set URI called when set the Tag attribute
    public void setUri (String name)       {
            uriName = name;
    }

    // Set Ttl called when set the Tag attribute
    public void setTtl (String name)       {
            ttlName = name;
    }

    // Gets File Name from the uri
    private String getFileName(String uri)         {
            return uri.substring(uri.lastIndexOf('/')+1);
    }


    // Check file is in cache or not
    private boolean checkFile(String s) throws Exception         {
                    ……..
    }

    //  get Time to live value in integer
    private int getTtl (String ttlName)     {
            int tmp;
            tmp = Integer.valueOf (ttlName).intValue();
            return (tmp);
    }
```

```
        // does the socket connection and prints the file on the browser
        public void grab () throws IOException        {
           connect ();
           try             {
                 fetch ();
           } finally      {
                 disconnect ();
           }
        }

        // connect () to the origin server
        protected void connect () throws IOException          {
        ……
        }

        // fetch () the web page and also store in cache
        protected void fetch () throws IOException   {
          PrintWriter outgoing;          // Stream for sending a command to the server.
          PrintWriter cacheOut = null;   // open file for writing into the disk cache
          String tmp;
          int len=0;

         // Open Stream for writing into disk cache
         ……..

        // Remove the headers
        ……

        // grab the data and display
           …….
        // closing the stream
        }

        // disconnect () after work is done
        protected void disconnect () throws IOException    {
           reader.close ();
        }
}//End IncludeTag class
```

CHAPTER 5

Performance Test

Listing 5.1 is the Perl program I used for testing performance of my code with the ESI. 'LWP::Simple' stands for the libwww-Perl library. LWP is a collection of Perl modules that provide a consistent and simple application-programming interface to the World Wide Web. It allows the developer to store the head or body of a web page (given its URL) in a scalar variable or file. Example,

```
#!/usr/bin/perl
use LWP::Simple;
my $content=get("yahoo.com"); #Store output of the web page (html and all) in content
```

The 'Time::HiRes' module implements a Perl interface to the 'usleep', 'ualarm', and 'gettimeofday' system calls. The 'gettimeofday ()' returns a 2-element array with the seconds and microseconds, the '($s2 - $s1) * 1000000' will convert the seconds into microseconds and the results is shown in microseconds. While testing the performance for the ESI, we can set proxy on the browser in 'Internet Options' but for testing the request-serving-time in perl program, we have to make sure the request goes through the ESI proxy, so the URL 'http://gallop.uccs.edu/ayeddula/esi/esitag.html' will change to 'http://wait.uccs.edu:8080/ayeddula/esi/esitag.html', as the ESI proxy is running on wait.uccs.edu at port 8080.

For installation and configuration information for the Linux version ESI Test Server see Appendix A.

Listing 5.1 Perl program to get request serving time

```
#!/usr/bin/perl
# rt_get.pl by Edward Chow, 2/3/2001
use LWP::Simple;
```

```
use Time::HiRes qw(usleep gettimeofday tv_interval);
$count = 0;
open(OUT, ">filert_get"); # to write to a file
print OUT get "http://gallop.uccs.edu:8888/examples/IncludeTag.jsp";
close(OUT);
for ($i=1; $i<4; $i++) {
        timeGetUrl ('http://gallop.uccs.edu:8888/examples/IncludeTag.jsp');
        sleep(1);$count ++;
        # timeGetUrl ('http://wait.uccs.edu:8080/ayeddula/esi/testesi.html');
        # sleep(1);$count ++;
}
sub timeGetUrl() {
      my ($url) = @_;
      my $s1;
      my $usec1;
      my $s2;
      my $usec2;
      my $time_diff;
      ($s1, $usec1) = gettimeofday;
       $doc = get $url;
      ($s2, $usec2) = gettimeofday;
      $length= length  $doc;
      $time_diff = ($s2 - $s1) * 1000000 + ($usec2 - $usec1);
      print "get $url size $length in $time_diff microseconds count $count \n";
}
```

5.1. Performance Test One

Following are the server configurations:

ESI Server

Server name: wait.uccs.edu
Port: 8080
Configuration: Redhat Linux 7.2, DELL dual Pentium, 1GHz, 1GB RAM.

Tomcat Server

Server name: gallop.uccs.edu
Port: 8888
Configuration: Windows 2000 advanced server, DELL dimension-4100, 933MHZ, 512MB RAM.

Original Server

Server Name: gallop.uccs.edu
Port: 80
Configuration: Windows 2000 advanced server, DELL dimension-4100, 933MHZ, 512MB RAM.

DataBase Server

Server Name: blanca.uccs.edu
Port: 3306
Database Server: MySQL
Configuration: Redhat Linux 7.2, DELL dual Pentium, 1GHz, 1GB RAM.

The ESI template page and the JSP Custom page, have same set of fragments for testing purposes.

5.1.1. Result - 1: ESI template page containing html fragments

Template page: http://gallop.uccs.edu/ayeddula/esi/esitag.html

Fragments are:

1. http://gallop.uccs.edu:8888/examples/staticpage.html ttl="5 Seconds"
2. http://gallop.uccs.edu/ayeddula/tasks.html ttl="5 Seconds"

5.1.2. Result - 2: JSP custom tag page containing html fragments

Template page: http://gallop.uccs.edu:8888/examples/IncludeTag.jsp

Fragments are:

1. http://gallop.uccs.edu:8888/examples/staticpage.html ttl="5 Seconds"
2. http://gallop.uccs.edu/ayeddula/tasks.html ttl="5 Seconds"

Table 5.1 shows the results, request-serving time in microseconds and request is sent every one second. TTL is set for 5 seconds.

Table 5.1. Performance Test One Results

ESI page performance test result one			JSP Custom Tag page performance test result two		
Size of packet	**Time taken to get the packets in microseconds**	**Relative time in seconds from start**	**Size of packet**	**Time taken to get the packets in microseconds**	**Relative time in seconds from start**
1501	75136	0	1503	22859	0
1501	12101	1	1503	9446	1
1501	12234	2	1503	9378	2
1501	12238	3	1503	11579	3
1501	12337	4	1503	9271	4

1501	**13756**	**5**	1503	**30208**	**5**
1501	12101	6	1503	11916	6
1501	12072	7	1503	9613	7
1501	12283	8	1503	9575	8
1384	12928	9	1503	11536	9
1384	**21825**	**10**	1384	**36810**	**10**
1384	12841	11	1384	12313	11
1384	12049	12	1384	9277	12
1384	12249	13	1384	10057	13
1384	12026	14	1384	14141	14
1384	**13584**	**15**	1384	**31444**	**15**
1384	12099	16	1384	9760	16
1384	12292	17	1384	11461	17
1384	12506	18	1384	14273	18
1501	13289	19	1384	9203	19
1501	**18139**	**20**	1503	**38308**	**20**
1501	12065	21	1503	9458	21
1501	12475	22	1503	11808	22
1501	12241	23	1503	12077	23
1501	12997	24	1503	9480	24
1501	**15106**	**25**	1503	**32519**	**25**
1501	12187	26	1503	10100	26

5.1.3. Performance test comparison between Result - 1 and Result - 2

In the figure 5.1 below, the graph shows an initial peak. It is probably due to the disk and file management. The operating system is responsible for these activities and for subsequent requests there on, the recall is available. TTL attribute is set to 5 seconds. Subsequent request interval is for every 5 seconds. The request serving time changes for the jsp file because the TTL expires after every 5 seconds the object is requested from the origin web server which is then stored in the cache and forwarded to the client. Hence, the processing time is greater after every 5 seconds of request interval.

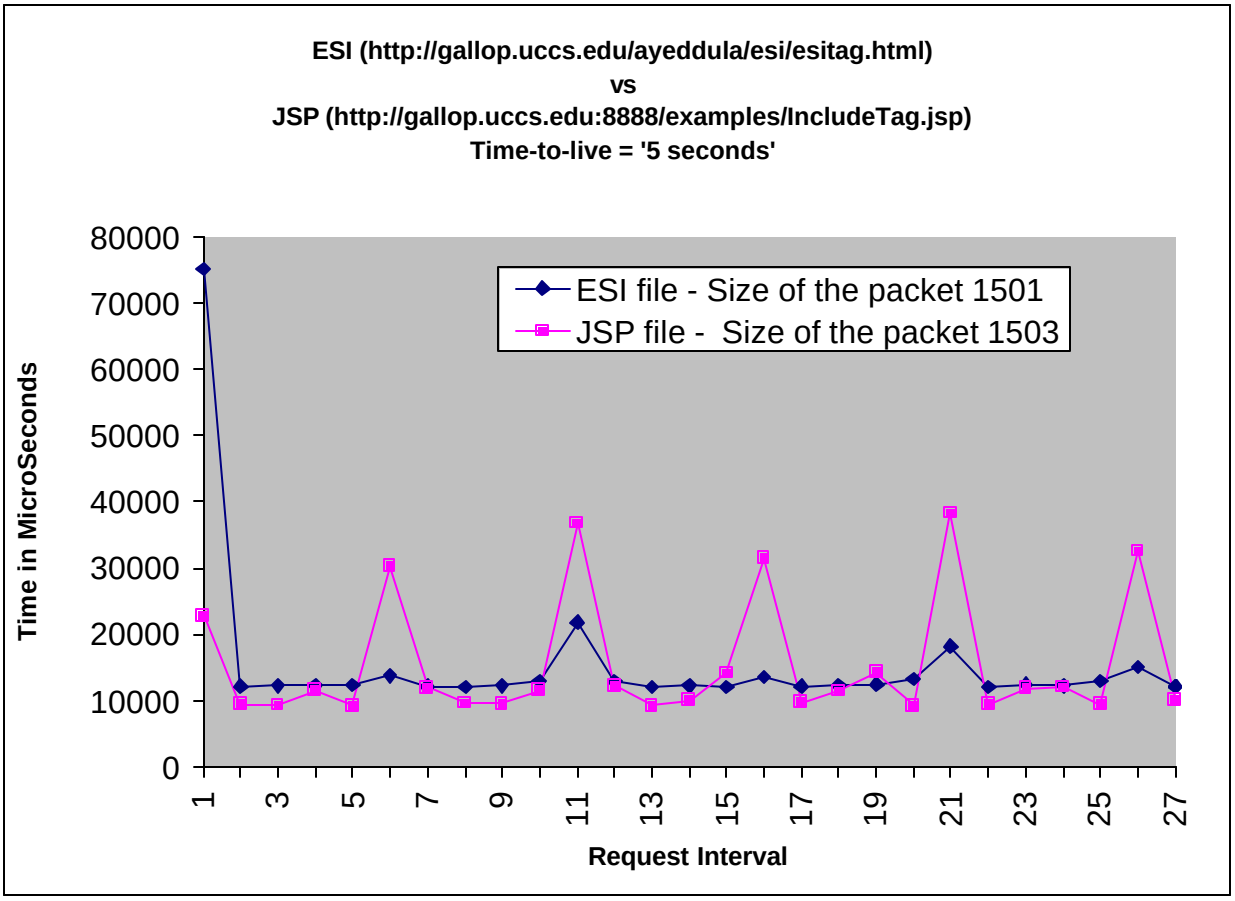

Figure 5.1. Performance test one results

5.2. Performance Test Two

Compared to the first test here the size of the file is increased.

5.2.1. Result - 1: ESI template page containing html fragments

Template page: http://gallop.uccs.edu/ayeddula/esi/testesi.html

Fragments are:

1. http://gallop.uccs.edu:8888/examp les/staticpage.html ttl="5 Seconds"
2. http://gallop.uccs.edu/ayeddula/tasks.html ttl="5 Seconds"

5.2.2. Result - 2: JSP custom tag page containing html fragments

Template page: http://gallop.uccs.edu:8888/examples/IncludeTag1.jsp

Fragments are:

1. http://gallop.uccs.edu:8888/examples/staticpage.html ttl="5 Seconds"
2. http://gallop.uccs.edu/ayeddula/tasks.html ttl="5 Seconds"

Table 5.2. Performance test two results

ESI page performance test result one			JSP Custom Tag page performance test result two		
Size of packet	**Time taken to get the packets in microseconds**	**Relative time in seconds from start**	**Size of packet**	**Time taken to get the packets in microseconds**	**Relative time in seconds from start**
2856	70243	0	2893	19788	0
2856	12917	1	2893	10161	1
2856	12997	2	2893	10159	2
2856	12888	3	2893	12656	3
2856	12952	4	2893	10253	4
2856	**16000**	**5**	2893	**37553**	**5**
2856	13368	6	2893	12533	6
2856	13621	7	2893	9976	7
2856	14300	8	2893	10081	8
2856	12827	9	2781	12255	9
2717	**19055**	**10**	2781	**37931**	**10**
2717	13147	11	2781	9892	11
2717	12679	12	2781	12551	12
2717	12820	13	2781	10034	13
2717	12664	14	2781	10037	14
2717	**14030**	**15**	2781	**39053**	**15**
2717	13135	16	2781	12415	16
2717	13944	17	2781	12787	17
2717	12813	18	2781	14708	18
2717	12909	19	2893	10204	19
2856	**19125**	**20**	2893	**39491**	**20**
2856	12913	21	2893	9842	21
2856	12735	22	2893	11104	22
2856	12794	23	2893	12460	23
2856	13200	24	2893	10006	24

2856	**14969**	**25**	2893	**39353**	**25**
2856	13582	26	2893	10117	26

5.2.3. Performance test comparison between Result - 1 and Result - 2

To compare to the first test above, I have increased the size of the file to find any dissimilarities. The results are request serving time increase which depends on the size of the file.

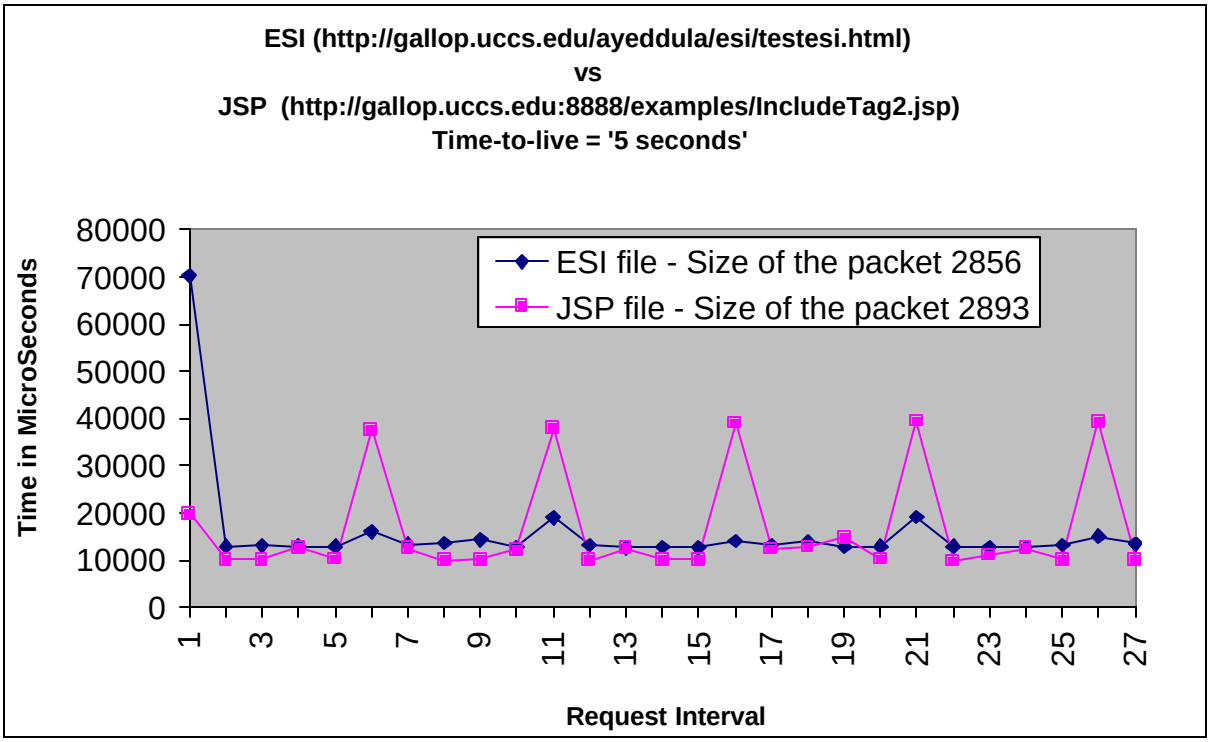

Figure 5.2. Performance test two results

5.3. Performance Test Three

5.3.1. JSP custom tag page containing html fragments

Template page: http://gallop.uccs.edu:8888/examples/database.jsp

Fragments are:

1. Including ‘database tag’

Table 5.3. JSP Custom Tag page results for performance test three

Size of packet	Time taken to get the packets in microseconds	Relative time in seconds from start
886	110886	0
886	29135	1
886	43820	2
886	58671	3
886	40206	4
886	33880	5
886	57600	6
886	22182	7
886	43527	8
886	30589	9
886	60000	10
886	33504	11
886	23431	12
886	40111	13
886	31016	14
886	33431	15
886	33373	16
886	29801	17
886	33639	18
886	38760	19
886	26190	20
886	36815	21
886	39981	22
886	26911	23
886	39847	24
886	39867	25
886	63902	26

5.3.2. Performance test – Request serving time

In Figure 5.3 a request is made to database; the Tag handler class makes a database connection using JDBC and ODBC. An ODBC-JDBC bridge is implemented as the Java package 'sun.jdbc.odbc' contains a native library used to access the ODBC. The tag handler class makes the connection to the database server that is running on blanca.uccs.edu.

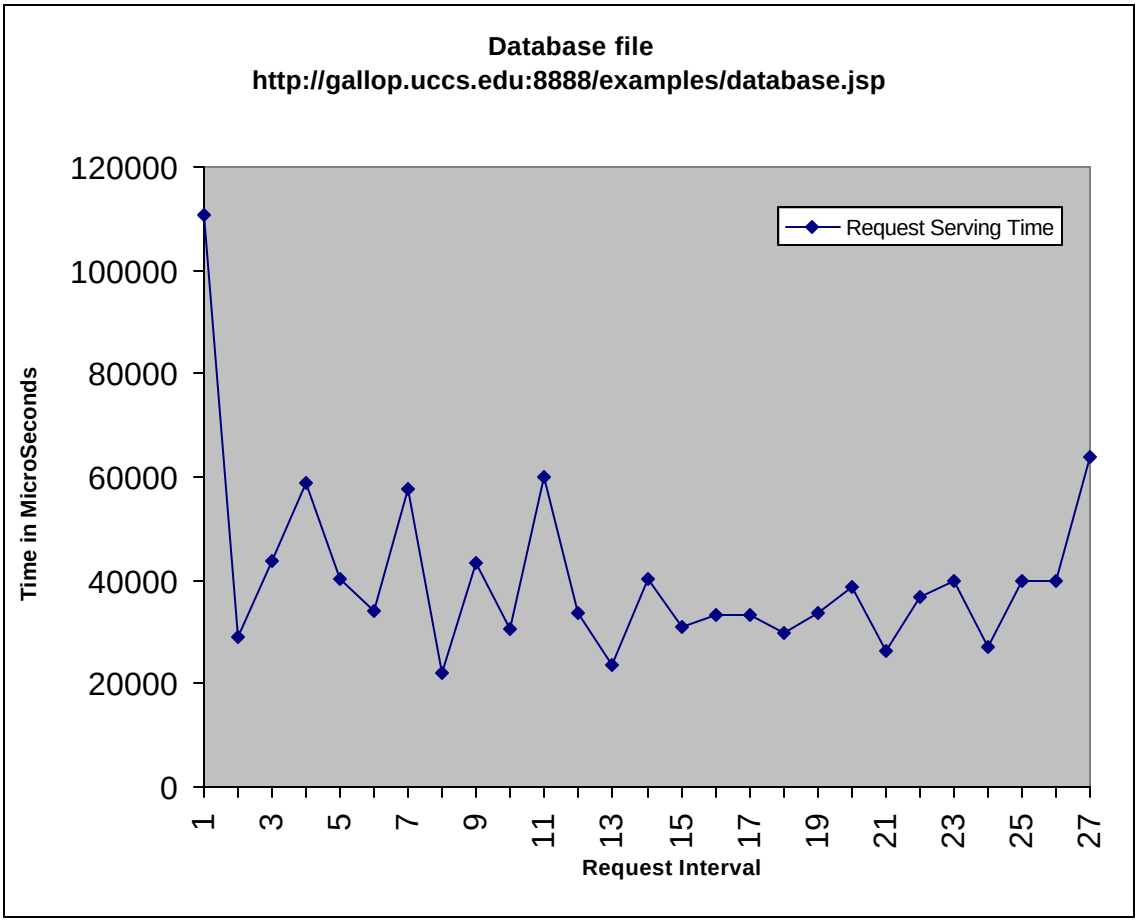

Figure 5.3. Performance test three results

5.4. Performance Test Four

Test four has one fragment in the template which has ttl set for TTL = '60 seconds' and the other fragment of the template page whose ttl is set for TTL = '120 seconds'.

5.4.1. JSP custom tag page containing html fragments

Template page: http://gallop.uccs.edu:8888/examples/IncludeTag2.jsp

The test four Fragments are:

1. http://gallop.uccs.edu:8888/examples/staticpage.html ttl="60 Seconds"
2. http://gallop.uccs.edu/ayeddula/tasks.html ttl="120 Seconds"

Table 5.4. JSP Custom Tag page results for performance test four

Size of packet	Time taken to get the packets in microseconds	Relative time in seconds from start
1930	197170	0
1930	24009	1
1930	28991	2
1930	23233	5
1930	23997	10
1930	21434	20
1930	20339	30
1930	22434	40
1930	24544	50
1930	24207	58
1930	29752	59
1930	**37076**	**60**
1930	28270	61
1930	23664	62
1930	21564	70
1930	20787	80
1930	22879	90
1930	21766	100
1930	23767	110
1930	24034	118
1930	21348	119
1930	**54112**	**120**
1930	22261	121
1930	28106	122
1930	24009	130
1930	25767	140
1930	22898	150
1930	24001	160
1930	25655	170
1930	21981	178
1930	22189	179
1930	**36202**	**180**
1930	33363	181
1930	25411	182
1930	28493	190
1930	27767	200
1930	27888	210
1930	20909	220
1930	28788	230
1930	26120	238

1930	23589	239
1930	**49791**	**240**
1930	23476	241

5.4.2. Performance test – Request serving time

The graph shows a small peak at 60 seconds request interval because only one page is beginning to be refreshed and at a 120 second request interval it showed double the original size of the peak as here both the fragments of the template are being refreshed.

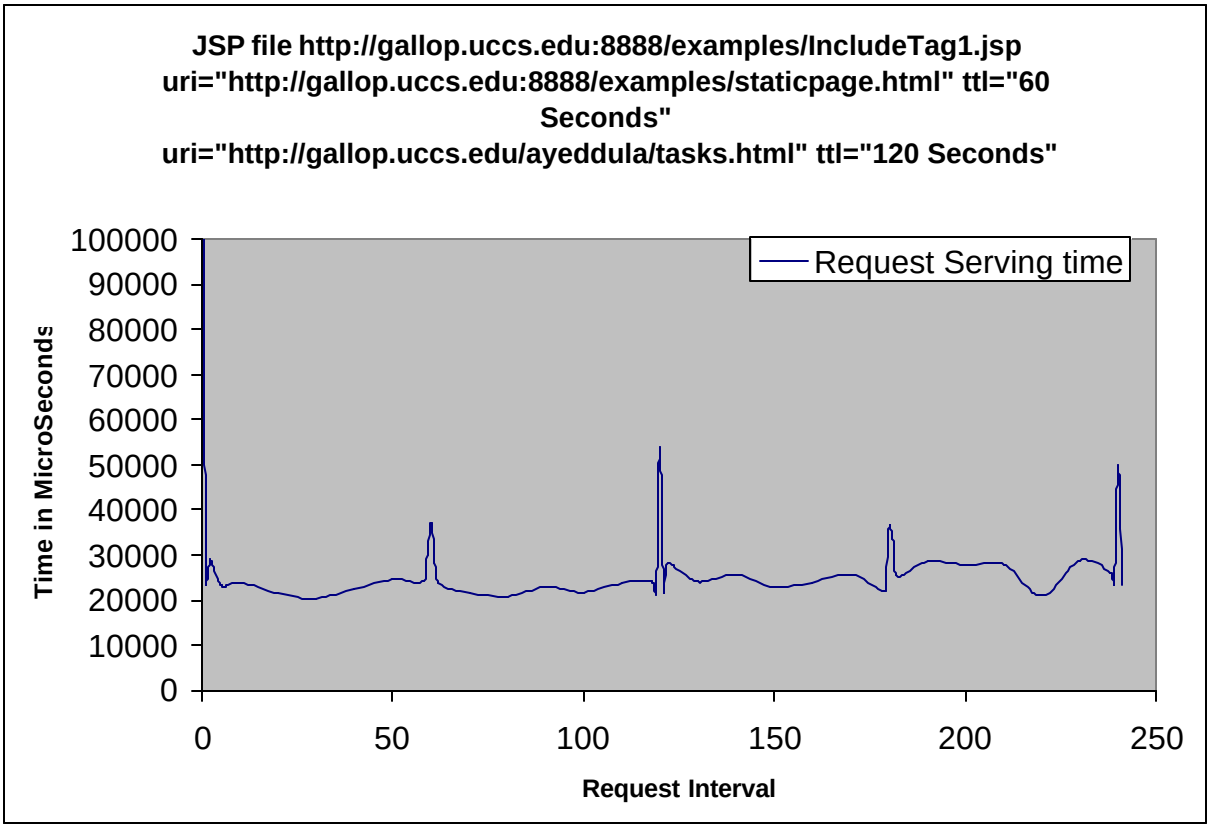

Figure 5.4. Performance test four results

CHAPTER 6

Conclusion and Future Work

The performance results show that JSP code takes lesser request processing time than the ESI Edge Suite. It is probably because of the development and testing environment, which has very less and complex network simulation. But in the real world, it could mean different as proxy servers are distributed all over the world and dynamic transfer of the web content could be very much faster than using JSP code or vice versa. And also, it entirely depends upon the design of the network and how the best solution can be achieved through an algorithmic design. In this project I have implemented the 'include' and 'conditional' selection using JSP. In the near future, I plan to work on other ESI functions such as filtering, merging, transforming of fragments and not just displaying them at different layout locations. I am also working on different extended ESI tags and would like to research on newer technologies to facilitate generic content processing not just content caching and also on what different kind of attributes can be included. For example, multiple src tags or operation tags. I am also interested in taking a look at from language specification point of view. The processing can also be done using JSP with servlets but that may not be my entire focus. The other possible extension for my future work is to allow the page to become an "active" web page, once it gets accessed for the first time and then allow it to actively or periodically send requests or retrieve information.

APPENDIX A

A.1. Setting up ESI test server and how to run code

Steps to installation ESI Test Server (ETS)

1. Go to the http://developer.akamai.com/esi_resources.html web site and download the Linux Version of the ETS server. The Linux Version downloaded file is 'ETS_1.0Beta3_Linux.tar.gz' and to extract the installation files use 'tar xzvf ETS_1.0Beta3_Linux.tar.gz'
2. Start installation by using the command './install'. While configuring, make sure the port we select for ETS is not taken or used by any other purposes. And set the origin server and its port number. My installation was not successful initially as it was running the web server on port 80. To change the port number go to the directory './usr/local/ETS/bin' at the prompt. Enter './ets_config' and make sure you also make changes in the './etc/httpd/conf/httpd.conf' file. To start the ETS server go to directory './usr/local/ETS/bin' and enter './apachectl start' and to stop enter './apachectl stop'.

A.2. Setting up Apache Tomcat and how to run the code

Steps to install Jakarta Tomcat and details on how to run hello.java file in tomcat.

1. Go to the http://arch.uccs.edu/~cs301/install/ web site and get the zip file 'jakarta-tomcat-4.0.1.zip' unzip the file.
2. Go to the directory file 'jakarta-tomcat-4.0.1.zip/common/lib' copy the file 'servlet.jar' into the directory 'j2sdk1.4.0-beta3/jre/lib/ext'.

3. To change the default port number (8080) go to the directory 'C:\ayeddula\tomcat\jakarta-tomcat-4.0.1\conf\server.xml' change the port 8080 to 8888.
4. To run the server go to the directory 'C:\ayeddula\tomcat\jakarta-tomcat-4.0.1\bin' and click on 'startup.bat'.
5. Go to 'start + control panel + system + advanced + environment variables' menu bar and add new system variables

   Variable name = JAVA_HOME

   Variable value = c:\j2sdk1.4.0-beta3
6. The java class files are saved into directory 'C:\ayeddula\tomcat\jakarta-tomcat-4.0.1\webapps\examples\WEB-INF\classes'.
7. 'javac -d C:\ayeddula\tomcat\jakarta-tomcat-4.0.1\webapps\examples\WEB-INF\classes HelloWorld.java' this command is used to save class file into the directory 'C:\ayeddula\tomcat\jakarta-tomcat-4.0.1\webapps\examples\WEB-INF\classes' automatically. To check the java file on browser just use 'http://localhost:8888/examples/servlets/HelloWorld'.
8. http://localhost:8888/examples/servlets/index.html for all examples

A.3. Setting up MySQL database server and ODBC driver

Steps to installation MySQL database server UNIX version

1. The mySQL server UNIX version is free. Download DBI-1_13.tar.gz and Msql-Mysql-modules-1_2209.tar.gz from www.mysql.com
2. For installation 'tar xvzf *.gz' will untar the tar files.

Steps to installation MySQL ODBC driver

1. Go to the www.mysql.com web site and download 'MyODBC-3.51.02.exe' Windows version.
2. For installation double click on the *.exe file.
3. Go to Start menu + Programs + Administrative Tools + Data Soruces (ODBC) and open the data source administrator click the 'add' button select the 'MySQL ODBC 3.51 Driver' click the 'Finish' button:

   Configure the following:

   Data Source Name: testdb (ODBC Data Source Administrator to connect to database)
   Host/Server Name(or IP): example.edu
   Database Name: testdb (to which database to be connected)
   User: username
   Password: password

Bibliography

[1] HTTP Hypertext Transfer Protocol http://www.w3.org/Protocols/

[2] Akami EdgeSutie http://www.akamai.com/index_flash.html

[3] ESI Resources http://www.esi.org/language_spec_1-0.html

[4] Microsoft .NET sample training modules

http://www.microsoft.com/traincert/training/developer/dotnet.asp

[5] "Core Servlets and Java Server Pages" by Marty Hall

[6] "Core Web Programming" by Marty Hall and Larry Brown

[7] Akamai Developer page: http://developer.akamai.com/pdf/ESI_Dev_Guide_4.3.2.pdf

[8] "Web Caching - An Introduction" http://www.cs.ubc.ca/spider/mjmccut/webcache.html

[9] "Oracle9iAS Caching Solution "

http://otn.oracle.com/products/ias/web_cache/pdf/9ias_caching_twp.pdf