CASE-BASED REASONING (CBR) FOR CHILDREN STORY SELECTION

IN ASP.NET AND C#

Demin Hu, M.S

Problem in Lieu of Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

December 2003

APPROVED:

Karl Steiner, Major Professor
Kavi Krishna, Chair of Department of Computer Sciences
Sandra L. Terrell, Interim Dean of the Robert B. Toulouse School of Graduate Studies
Hu, Demin, *Case-Based Reasoning (CBR) for Children Story Selection in ASP.NET and C#.*

Master of Science (Computer Science), December 2003, 71 pages, 34 figures, references, 52 titles.

This paper describes the general architecture and function of a Case-Based Reasoning (CBR) system implemented with ASP.NET and C#. Microsoft Visual Studio .NET and XML Web Services provide a flexible, standards-based model that allows clients to access data. Web Form Pages offer a powerful programming model for Web-enabled user interface.

The system provides a variety of mechanisms and services related to story retrieval and adaptation. Users may browse and search a library of text stories. More advanced CBR capabilities were also implemented, including a multi-factor distance-calculation for matching user interests with stories in the library, recommendations on optimizing search, and adaptation of stories to match user interests.
ACKNOWLEDGMENTS

I would like to thank my instructor Dr. Karl Steiner for his help, kindness and patience. I am grateful for all things he has done for me.
# TABLE OF CONTENTS

LIST OF FIGURES........................................................................................................iv

Chapter

1. INTRODUCTION .................................................................................................1
   1.1 CBR ..............................................................................................................2
   1.2 Advantage of CBR System .........................................................................3
   1.3 ASP.NET and C# .......................................................................................4

2. SYSTEM ARCHITECTURE ...............................................................................7
   2.1 XML Database ..........................................................................................8
   2.2 Story Web Service .....................................................................................12
   2.3 Story Web Client .......................................................................................22
   2.4 Deployment of Web Service and Web Client ...........................................26

3. BASIC SELECTING METHODS ..................................................................28

4. CASE-BASED REASONING (CBR) ..............................................................35
   4.1 Simple CBR Method ................................................................................37
      4.1.1 Retrieve and Reuse ........................................................................29
      4.1.2 Revise ...............................................................................................44
      4.1.3 Retain ...............................................................................................48
   4.2 Advanced CBR Method ..........................................................................50
      4.2.1 Ranking Using Distance ..................................................................52
      4.2.2 Edit Story .........................................................................................59

5. FUTURE WORK & CONCLUSIONS .............................................................66

REFERENCE LIST ..............................................................................................58
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Architecture of the distributed application of the project</td>
<td>8</td>
</tr>
<tr>
<td>2.</td>
<td>Story.xsd (dataset) in Visual Studio .NET</td>
<td>12</td>
</tr>
<tr>
<td>3.</td>
<td>The Solution Explorer of Story Web Service in Visual Studio .NET</td>
<td>14</td>
</tr>
<tr>
<td>4.</td>
<td>Browsing the StoryService.asmx file</td>
<td>17</td>
</tr>
<tr>
<td>5.</td>
<td>Browsing the GetAllTopic of StoryService Web method</td>
<td>18</td>
</tr>
<tr>
<td>6.</td>
<td>Net provide example of calling method</td>
<td>19</td>
</tr>
<tr>
<td>7.</td>
<td>Calling the GetAllTopics method manually via a HTTP Get method</td>
<td>19</td>
</tr>
<tr>
<td>8.</td>
<td>WSDL for StoryWebService</td>
<td>20</td>
</tr>
<tr>
<td>9.</td>
<td>Local Web Services</td>
<td>21</td>
</tr>
<tr>
<td>10.</td>
<td>Web Form Pages structure and the Page base class</td>
<td>22</td>
</tr>
<tr>
<td>12.</td>
<td>The Solution Explorer of Story Service and StoryWebClient Setups</td>
<td>27</td>
</tr>
<tr>
<td>13.</td>
<td>Selecting story by Topic (topicForm.aspx) in action</td>
<td>28</td>
</tr>
<tr>
<td>14.</td>
<td>Selected story is displayed in the browser</td>
<td>31</td>
</tr>
<tr>
<td>15.</td>
<td>The user inputs character and location</td>
<td>32</td>
</tr>
<tr>
<td>16.</td>
<td>The matching score and recommended story are displayed in the browser</td>
<td>34</td>
</tr>
<tr>
<td>17.</td>
<td>A CBR cycle</td>
<td>35</td>
</tr>
<tr>
<td>18.</td>
<td>A CBR cycle for the Story selecting</td>
<td>36</td>
</tr>
<tr>
<td>19.</td>
<td>A user interface for attribute selecting</td>
<td>39</td>
</tr>
<tr>
<td>20.</td>
<td>A user is selecting attributes</td>
<td>43</td>
</tr>
<tr>
<td>21.</td>
<td>Match score and recommended story displayed in the browser</td>
<td>45</td>
</tr>
<tr>
<td>22.</td>
<td>The preferred title and recommended story title are not identical</td>
<td>46</td>
</tr>
<tr>
<td>23.</td>
<td>An unsuccessful matching</td>
<td>47</td>
</tr>
<tr>
<td>24.</td>
<td>The system advice how to improve input fields</td>
<td>48</td>
</tr>
<tr>
<td>25.</td>
<td>The newly recommend story after the input changed</td>
<td>49</td>
</tr>
<tr>
<td>26.</td>
<td>The changed user interface for advanced CBR method</td>
<td>51</td>
</tr>
<tr>
<td>27.</td>
<td>User selecting the story attributes in advanced CBR method</td>
<td>56</td>
</tr>
<tr>
<td>28.</td>
<td>User input weight percentages</td>
<td>56</td>
</tr>
<tr>
<td>29.</td>
<td>The Result with same weight percentages of each attribute</td>
<td>57</td>
</tr>
<tr>
<td>30.</td>
<td>The input weight percentages are changed</td>
<td>58</td>
</tr>
</tbody>
</table>
31. Showing the result after weight percentage changed.........................................................48
32. The browser shows the modified story ..............................................................................59
33. The DataList of story database before editing .................................................................63
34. Editing is in action .............................................................................................................64
35. The edited story is saved and displayed.............................................................................64
CHAPTER 1
INTRODUCTION

While keyword searches can be effective in many circumstances, some search and matching problems may benefit from other techniques. For example, some problems may involve trying to determine the closest match based on not a single keyword or phrase, but on a collection of attributes. Each attribute may contribute independently to the suitability of the match, and the value of each attribute may be weighted individually. When the best match is found, it still may not suit the users request, so ideally, additional adaptation of the result will take place. These capabilities are commonly associated with Case-Based Reasoning (CBR) systems, and we apply CBR techniques to the problem of managing, searching, and presenting a library of children’s stories. In order to facilitate web deployment of these capabilities, we have employed Microsoft’s .NET Framework to develop our system.

Selecting an ideal children’s story from a large database can be a complex problem. In particular, users may have preferences for material targeted at certain age ranges as well as for different story styles, characters and settings. For instance, a story about “Teddy Bear Tales” may be suitable for a 4 or 5-year-old, but not for a 12-year-old. Moreover, a story about a little mermaid may suitable for a little girl, but may not good for a boy whose favorite topic is space adventure. CBR provides a mechanism for matching stories based on a combination of user preferences, and then modifying the story to better fit the elements that were not closely matched.

From the beginning of the Internet’s existence, the available technologies for web-based application developers have been a mixed bag of various tools and language. For this reason web programming has been a tedious and error-prone task. Most web applications have been
developed with hard-to-read code that is not object-oriented or event-driven. Web programmers have longed for a way to harness the mature, object-oriented and event-driven techniques that have been available to traditional client/server programmers. Microsoft has created an entire suite of tools that provides many of these services – the .Net Framework [1].

1.1 CBR

Case-based reasoning (CBR) is a recent approach to problem solving and learning that has received a lot of attentions over the last few years. It has grown from a research novelty, to a powerful technique used in a variety of commercial applications.

In CBR systems, expertise is embodied in a library of past cases, rather than being encoded in classical rules. Each case typically contains a description of the problem, plus a solution and/or the outcome [5]. CBR systems utilize this knowledge of previously experienced, concrete problem situations (cases). CBR can solve a new problem by remembering a previous similar situation and by reusing information and knowledge of that situation.

The root of CBR in AI is found in the works of Roger Schank on dynamic memory and the central role that a reminding of earlier situations and situation patterns has in problem solving and learning [6].

The creation of the first actual system based on CBR is handed to Janet Kolodner who created a system called CYRUS. The system was built on Schank’s theories and was basically a question-answering system that contained knowledge of various travels and meetings. The memory model developed for this system was also used in other systems like MEDIATOR, CHEF and CASEY.
Bruce Porter and his group at the University of Texas developed another set of models for CBR that combine general domain knowledge with specific case knowledge. This same approach was used in the GREEBE system, which is a law application.

Another scientist who also has contributed to the field of CBR and law is Edwina Rissland with her group at the University of Massachusetts. They used cases not to produce a single answer but to interpret a situation in court and to produce arguments for both parties. This work has led to the HYPO system and later on the CABARET system that combines case-based and ruled-based systems.

Currently CBR is spreading over the world. CBR systems are being deployed not only in the USA and Europe, but also in India, Japan and other Asian countries. CBR systems are being used in both academic and commercial settings. There are a variety of specialized CBR tools and software packages, as well as numerous conferences dedicated to CBR.

1.2 Advantage of CBR System

Kolodner describes some clear advantages of CBR systems [6]. A CBR system does not require as much information as a rule-based system normally would at the initial launch since a CBR system can learn over time. Debugging is also made easier because there are often far fewer interactions between cases than there can be between rules.

Because CBR systems store previously solved problems they can re-use the solution to solve an equivalent problem. This can improve efficiency since this system may require less time to come up with a solution. A CBR system can also recognize previous mistakes, which eliminates the
possibility of doing them again. CBR systems can also explain why a particular case was chosen. This provides a benefit over some rule-based systems[14].

CBR system can be very scalable if implemented correctly. Different forms of indexing and parallel algorithms can be used to speed up the processing and it would appear that CBR systems might be able to approach, or even achieve, real-time performance. By real-time we assume the author means that a human would not feel as if they were waiting for an answer from the system [14].

There is also some ongoing research into collaborative case-based reasoning where one system can ask another system if it has the solution to a problem. An example of how this could work is a GPS navigation system for cars. If the driver of a car wants directions to an unknown location, his CBR based navigation system can ask other cars if any of them know the route. It could also select between routes given by many other cars and select one that suits the kind of route the driver prefers (fastest, most scenic, shortest distance, etc.).

1.3 ASP.NET and C#

ASP.NET is a set of technologies in the Microsoft .Net Framework for building web application and XML Web Service. ASP.NET pages execute on the server and generate markup such as HTML, WML and XML that is sent to a desktop or mobile browser. ASP.NET pages use a compiled, event-driven programming model that improves performance and enables the separation of application logic and user interface. ASP.NET pages and ASP.NET XML Web Service files contain server-side logic written in Visual Basic .NET, C#. NET, or any .NET compatible language. Web applications and XML Web Service take advantage of the features of
the common language runtime, such as type safety, inheritance, language interoperability, versioning, and integrated security [21].

ASP.NET picks up where ASP leaves off and provides a vast improvement over the former technology. Visual C# is a new object-oriented programming language that is an evolution of the C and C++ languages, available as part of the .NET framework. C# attempts to capture the power and robustness of C++ while eliminating its shortcomings and pitfalls. It provides a simple and type-safe language for developing application [3].

An XML Web service is a programmable unit of software, exposed on the Web through SOAP, described with a WSDL file and registered in UDDI. Microsoft Visual Studio .NET and XML Web Service provide a simple, flexible, standards-based model that allows developers to assemble applications regardless of the platform, programming language, or object model [1].

Web Forms pages serve as the user interface for web application developed with .NET. A Web Forms page presents information to the user in any browser or client device and implements application logic using server-side code [2]. Web application programming presents challenges that do not typically arise when programming traditional client-based application, among the challenges are [22]:

- Implementing a rich web user interface. A user interface with a complex layout, a large amount of dynamic content, and full-featured user-interactive objects can be difficult and tedious to design and implement using basic HTML facilities. It is particularly difficult to create a rich user interface for applications likely to run on many different browser and client device platform.
• Separation of Client and Server. In a web application, the client (browser) and server are different programs often running on different computers. Consequently, the two halves of the application share very little information; they can communicate, but typically exchange only small chunks of simple information.

• Stateless execution. When a web server receives a request for a page, it finds the page, process it, send it to the browser, and then, effectively, discard all page information. Put another way, the server has no memory of pages that it has processed.

• Unknown client capabilities. In may case, web application are accessible to many user using different browser. Browser has different capabilities, making it difficult to create an application that will run equally well on all of them.

• Complication with data access. Reading from and writing to a data source in a traditional web applications can be complicated and resource-intensive.

• Complication with scalability. In many case web applications designed with existing methods fail to meet scalability goals due to the lack compatibility between various components of the application.
CHAPTER 2
SYSTEM ARCHITECTURE

Distributed applications are typically created one tier at a time, beginning with the data layer, and then moving to the middle-tier, business-rule object, and finally creating the user interface. The business object is created on a Web server to provide the performance and scalability necessary for a distributed application [44]. In addition, business objects are typically implemented as XML Web services so that clients can use a standard Internet protocol to communicate with the business object from any platform.

In this project, I created a multi-tiered, distributed application to implement the CBR system. The application consists of three logic tiers: data, business objects, and user interface. The data tier is a database in XML format; the business-object tier handles accessing the data and distributing data to the clients. The user interface tier is a Web-based application. Figure 1 illustrates the architecture of the application.

![Diagram of distributed application architecture]

Figure 1. Architecture of the distributed application of the project.
For the Web portion, the web Forms Designer is used to create a web page that is compatible with standard HTML. On the server, the Web Forms code calls the XML Web Service to retrieve data that contains story information from the XML database. Communication with the XML Web Service is handled using HTTP, XML and TCP/IP.

2.1 XML Database

In the .NET Framework, XML is very important. It serves as the foundation for many of the .NET technologies. There are some specific advantages of XML over other data formats [2]:

- XML documents are easily readable and self-describing –like HTML, and XML document contains tags that indicate what each type of data is.
- XML is interoperable –nothing about XML ties it to any particular operating system or underlying technology.
- XML documents are hierarchical –It’s easy to add related data to a node in an XML document without making the document unwieldy.
- You don’t have to write the parser – Several types of object-based parser components are available for XML. The .NET platform contains support for the Internet-standard XML.
- Changes to your document won’t break the parser –Assuming that the XML you write is syntactically correct, you can add elements to your data structures without breaking backward compatibility with earlier versions of your application.

At the same time, Visual Studio .NET hides many of the implementation details from you. Relational data expressed in XML is abstracted in the form of a DataSet object. In fact, Visual Studio .NET generates XML schema by automatically inspecting an existing database structure.
In this project, I put all the stories in an Xml database. Each story has different attributes and different features. For example, a story could have a title, a topic, a story character, a story location, and an age level. These attributes can be represented in XmlNodes.

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Children_Stories xmlns="http://localhost/StoryWebClient/">
  <Story>
    <StoryTitle>A Bear at Bedtime</StoryTitle>
    <StoryContent>
      One bear in a bed is cuddy, and two are better still. With three teddy bears, you are sure to be warm, and just one more is no problem at all.
      Five teddy bears in a bed can help you sleep, while six teddy bears are very good indeed. Seven is lucky number for bears. And eight teddy bears are best of all.
      But nine teddy bears in a bed? Be careful! There may not be room for you ...
    </StoryContent>
    <StoryCategory>
      <StoryTitle>A Bear at Bedtime</StoryTitle>
      <StoryForChildAge>5</StoryForChildAge>
      <StoryTopic>Teddy Bear Tales</StoryTopic>
      <Character>Teddy Bear</Character>
      <Location>bed</Location>
    </StoryCategory>
    <Description>
      <CharacterType>bears</CharacterType>
      <LocationType>indoor</LocationType>
    </Description>
  </Story>
  <Story>
    <StoryTitle>The Teddy Bear's Picnic</StoryTitle>
    <StoryContent>
      Bears often decide to have picnics, and just as often, things don't go exactly according to plan. Once ten little bears planned a picnic.
      One little bear tried to catch a butterfly and didn't come back for hours. Another attempted a somersault, and couldn't get up again.
      The third bear went swimming, when he should have been helping.
      The fourth got the hiccups and was no use at all.
      The sixth little bear was stung on the nose by a bee, because he wondered if its honey might be tasty. The seventh got so excited that she had to have a rest! A crab chased her sister. She forgot all about making sandwiches.
      It was not surprising that the ninth little teddy bear decided to go home. So the tenth bear tucked into the buns and cakes all on his own, and was soon a very full little bear indeed!
      At last, all the little bears did gather together for their party, but after all the excitement, they fell fast asleep ... every one of them!
    </StoryContent>
  </Story>
</Children_Stories>
```
One of the purposes of XML is to manage the storage of data. The XmlNode object represents a node in the XML document. It exposes an object hierarchy that contains object attributes and child nodes, as well as every other part of an XML document. If we described this document in English, we’d say that it contains a top level “Children Stories” and node “Story.” The “Story” is a child of the “Children Stories” element. The “Story” element itself contains four child nodes of its own : “Story Title,” “Story Content,” “StoryCategory,” and “Description.” The element “StoryCategory” contains five child nodes of its own: “storyTitle,” “StoryForChildAge,” “StoryTopic,” “Character,” and “Location.” The element node “Description” contains 2 child nodes of its own: “ CharacterType” and “LocationType.”

Visual Studio .NET has the capability to create XML schemas automatically from a data source. This means that you can set up a database and Visual Studio will reverse the structure of the database into XML schemas. The Visual Studio .NET created the story XML database schema as the following:

```xml
<?xml version="1.0" ?>
<xs:schema id="Children_Stories"
  targetNamespace="http://localhost/StoryWebClient/"
  xmlns:mstns="http://localhost/StoryWebClient/"
  xmlns="http://localhost/StoryWebClient/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-..."/>
```
Figure 2 shows the dataset of story database in Visual Studio .NET.
DataSets are represented as XML [2]. The structure of the dataset – the definition of what tables, columns, data types, and constraints – is defined using an XML Schema based on the XML Schema definition language (XSD). Just as data contained by a dataset can be loaded and serialized as XML, the structure of the dataset can be loaded from and serialized as XML Schema [24].

2.2 Story Web Service

The ASP.NET page framework serves as a framework for XML Web services in managed code. XML Web Service can access many features of the .NET Framework, such as authentication, caching, and state management [26]. Developers are then free to focus on creating or accessing
XML Web services without needing to write infrastructure code. In the ASP.NET application model, Web pages intended for the browser use the .aspx file extension. To differentiate XML Web Service from regular ASP.NET pages, XML Web services use the .asmx extension [21].

XML Web Service consists of an XML Web Service entry point and the code that implements the XML Web service functionality. In ASP.NET, the .asmx file serves as the addressable entry point for the XML Web Service. It references code in precompiled assemblies, a code-behind file, or code contained in the .asmx file itself [1]. Figure 3 shows the Solution Explorer for the story Web Service in Visual Studio .NET:

When we create an XML Web Service, we are creating an application that exposes functionality to XML Web Service clients. When we access an XML Web Service, our client application locates, reference, and uses the functionality contained within that XML Web Service. For story Web Service, it includes files: StoryyService.asmx, StoryService.asmx.cs, AssemblyInfo.cs, Global.asax, Global.asax.cs, StoryWebService.vsdisco, StoryWebService.csproj and Web.config. If you wish, you could put the some classes in the .asmx.cs file, or in another file under the same namespace as the .asmx file. Story .Xml database is in Web Service, but actually it is loaded from WINNET System32. The Story Web Service and Story Web Client are in the directory: D:/Inetpub/wwwroot.
Creating an XML Web Service in managed code, we indicate the method that is available through that XML web Service by placing the WebMethod attribute before the method declaration of a Public method. Private methods cannot serve as the entry point for an XML web Service, although they can be in the same class and the XML Web Service code can call them. The WebMethod attribute must be applied to each public method that is available as part of the XML Web Service [2]. All of the methods exposed by a particular Web Service (.asmx file) must have
a unique name. Exposing multiple overloads of a method of a class that has several overloads with the same name could cause significant problems. The following are examples of some WebMethods in StoryService.asmx.cs:

```csharp
[WebMethod (Description="get the story all information for advanced story select.")]
public StoryForSelect[] SelectStories()
{
    ArrayList arr = new ArrayList();
    XmlDocument doc = new XmlDocument();
    doc.Load("Story.xml");
    XmlNode docElement = doc.DocumentElement;

    foreach (XmlNode n in docElement.ChildNodes)
    {
        XmlNodeReader reader = new XmlNodeReader(n);
        reader.Read();

        foreach (XmlNode n1 in n.ChildNodes)
        {
            foreach (XmlNode n2 in n1.ChildNodes)
            {
                if (n2.Name == "StoryTitle")
                {
                    XmlNode n3 = n2.NextSibling;
                    XmlNode n4 = n3.NextSibling;
                    XmlNode n5 = n2.ParentNode;
                    XmlNode n6 = n5.NextSibling;
                    XmlNode n7 = n6.FirstChild;
                    XmlNode n8 = n6.LastChild;

                } // end if
            } // end inner foreach
        } // end first foreach
    } // end foreach
    return (StoryForSelect[])arr.ToArray(typeof(StoryForSelect));
} // end function
```

The WebMethod attribute above the SelectStories method tells the compiler that this is a Web Service method. We can have as many methods as we wish within our code-behind class, but
only those that have the WebMethod attribute applied to them will be callable from the web. We set the Description inside the attribute so that it would be displayed when the user views the service in a Web browser [1].

Because we can actually think of Web services as web pages that output their results in XML, we know that we could call them directly with a browser. When we browse to the Web Service location with our browser, however, we see a nicely formatted screen created for us by the ASP.NET Framework. A line in the machine.config file on the server maps.asmx requests to a specific class in the framework that displays this page to us. This page displays the names and descriptions of the available methods. If we drill into these methods by clicking on their links, an interface allows us to provide parameters and call the method from within our browser [1]. This is a valuable debugging tool. Figure 4 shows the WebMethods of StoryService in Solution Explorer.

In the following example, we replace the variable values in the query string with some real name parts. We can also see from the test that the SOAP call looks different than the HTTP Get call above.

```xml
POST /StoryWebService/StoryService.asmx/GetAllTopics HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: length

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<ArrayOfString xmlns="http://www.localhost/StoryWebService/">
  <string>string</string>
  <string>string</string>
</ArrayOfString>
```
Figure 4. Browsing the StoryService.asmx file.

Figure 5, Figure 6, and Figure 7, show the SOAP process for WebMethods in StoryService.
Figure 5. Browsing the GetAllTopic of StoryService Web method.

Figure 6. .Net provide example of calling method
The SOAP uses an HTTP Post to make the call to the WebMethod [1]. In the header there is a SOAPAction parameter, which tells the Web service the full name of the method being called. In the body of the request, you can see the XML that makes up the SOAP method call and the parameters sent. In the example generated by .NET, there are placeholders shown for the parameters, but if you had the proper tools, mainly an application with which you could submit HTTP request manually, you could send this SOAP header with real data in the parameter to invoke the Web method.

By performing these steps, we are actually browsing the server on which the web service lives. Clicking on the link to the .vsdisco file returns XML. This XML is called discovery information, and it tells the calling application - in this case the Visual Studio .NET IDE - which methods are in this service and the URL to call to learn more about the methods. This URL is a link to what is known as the Web Service Definition Language (WSDL) We can view the web service by

![Calling the GetAllTopics method manually via a HTTP Get method](http://localhost/StoryWebService/StoryService.asmx/GetAllTopics?/Accept..)
browsing the link: http://localhost/StoryWebService/StoryService.vsdisco?wsdl or:
http://localhost/StoryWebService/StoryService.asmx?wsdl. The link calls the
StoryService.asmx, the web service, with a query string parameter of wsdl. This is handed by all
Web Services, and returns the actual WSDL [28]. Figure 8 is WSDL for StoryService.

Figure 8. WSDL for StoryWebService

UDDI is important because it appears as an option for searching and web service in the Add Web
Reference tool of Visual Studio .NET. UDDI is the Universal Description, Discovery and
Integration (UDDI). This is a directory of Web Service on the Internet. Just as you would search
for any other type of web resource using a search engine on the Web, the UDDI is a searchable repository for services available from locations everywhere [25].

In the left hand pane of figure 9 there is a link to the live Microsoft UDDI directory, where we can find any web services that have been registered by various companies [2]. There is also a link to the Test Microsoft UDDI directory and, last, a link to any Web references on the local web server. If you click on the link for local Web services, the server will be found, which will enable the discovery of any Web services in that directory or that directory below it.

Figure 9. Local Web Services

In the browser’s address field, type the base URL for the XML Web Service and provide the WSDL query string by using this format:

http://localhost/StoryWebService/StoryService.asmx?WSDL
2.3 Story Web Client

Web Forms is an ASP.NET feature that can be used to create user interfaces for Web applications [19]. Figure 10 is Web Form Page Structure in the Story Web Client; the .aspx file is compiled dynamically when a user browses the page. Its relationship to the class file is established with a script directive at the top of the page.

The .aspx and .aspx.cs make up a single Web Form page. The .aspx file contains the visual elements of the Web Forms page such as, HTML elements and Web Forms control. The .aspx.cs is a hidden, dependent file of the Web Form. It contains a code-behind class for the Web Forms page, containing event-handler code. The .vsdisco file is an XML-based file that contains links to resources providing discovery information for an XML Web service. Web.config is an XML-based file that contains configuration data on each unique URL resource used in the project. AssemblyInfo.cs is a project information file that contains metadata about the assemblies in a project [22].
We access the service by instantiating an object whose type name is the name of the web service. In the following example, because we are calling the Service1, we instantiate an object of type service1. The constructor to the service takes no parameters, and as soon as it is instantiated, we can call any Web Method we want on it. The same goes for properties. In addition to the Web Methods in the service, if there are properties, we can use them as well. The calling code looks like this:

```csharp
private void Page_Load(object sender, System.EventArgs e)
{
    if(!IsPostBack)
    {
        try
        {
            // bind available topics
            Service1 s = new Service1();
            DropDownList1.DataSource = s.GetAllTopics();
            DropDownList1.DataBind();
            DropDownList1.Items.Insert(0, "Select");
        } // end try
        catch(Exception ex)
        {
            Response.Write("Exception: " + ex.ToString());
        } // end if
    }
    else
    {
        // dont need this item after first selection is made.
        DropDownList1.Items.Remove("Select");
    } // end else
}
```

Figure11 is the Solution Explorer of Story Web Client in Visual Studio .NET.
the ASP.NET page framework performs many Web application services. For example, the ASP.NET page framework captures information posted with the Web Forms page, extracts the relevant values, and makes the information accessible via object properties.
In this project, the browser presents the user with a form, and the user interacts with the form, causing the form to post back to the server. However, since all processing that interacts with server components must occur on the server, this means that for each action that requires processing, the form must be posted to the server, processed, and returned to the browser.

The following example describes the browser to server round-trip. In the Select Story by Topic method, you select the DropDownList, then you click the submit button. The selected item is submitted to the server, and the processing occurs in the server. After the processing is done, the stories under this topic are returned to the browser, and the server discards the information about your selection. If you want to repeat the selection, the whole process will repeat again. In Web forms, most user actions – such as clicking a button – result in a round trip [22].

The following is a typical example of PostBack Events in the aspx.cs code in the Story Web Client. The<asp:button> objects automatically postback when clicked. No additional code is required to handle that event unless you want to do something more than postback to the server.

Normally, when a page is redraw, each control is redrawn from scratch. The web is stateless, and if we want to manage the state of a control, we must do by ourselves. In the following example, we use the Session event to preserve state.

```csharp
protected void Button1_Click(object sender, System.EventArgs e)
{
    //Set the session var and go to next page.
    Session["character"] = TextBox1.Text.Trim();
    Session["location"] = TextBox2.Text.Trim();
    Response.Redirect("CBRSelFirstDisplay.aspx");
}

private void Page_Load(object sender, System.EventArgs e)
{

```
if(!IsPostBack)
{
try
{
    // bind available topics
    Service1 s = new Service1();
    DropDownList1.DataSource= s.GetAllTopics();
    DropDownList1.DataBind();
    DropDownList1.Items.Insert(0, "Select");
}
} // end try
catch(Exception ex)
{
    Response.Write("Exception: " + ex.ToString());
}
} // end if
else
{
    // dont need this item after first selection is made
    DropDownList1.Items.Remove("Select");
} // end else

2.4 Deployment of Web Service and Web Client

Deployment in Visual Studio .NET differs from traditional setup and deployment in many respects. Visual Studio .NET deployment uses Microsoft Windows Installer technology to create an installer that controls the entire installation process [44]. We use the Setup project method to deploy the Web Service and Web Client. Figure 12 shows the Solution Explorer of StoryService and StoryWebClient Setups in Visual Studio .NET.
Figure 12. The Solution Explorer of Story Service and StoryWebClient Setups in Visual Studio .NET
CHAPTER 3
Basic Select Methods

We have developed five basic ways to search a story: select a story according to story attributes, select a story by title, select a story by topic, select a story by child age, and select a story with more attributes. Another technique allows a user to search for a story by entering multiple attributes. We build these methods in a modular fashion in order to reuse them as part of the Advanced CBR Method. The following explains two search methods: select a story by topic, and select a story according to user-defined attributes.

Figure13, Selecting story by Topic (topicForm.aspx) in action
In order to select a story by topic, we first construct a Web Form named topicForm.aspx. The user will be presented with a DropDownList of Topics from the story database. When the user selects a topic, the user will be presented with all of the stories in our database for that topic. Choosing a title then takes them to the story. Selecting a story by topic (topicForm.aspx) is shown in Figure13.

In the DataGrid, ButtonColumn is used with the CommandName of “Select” and simple bound columns. Because the ButtonColumn is a Selected button, the SelectedIndexChanged event id fires when clicked.

```xml
<asp:DataGrid id="DataGrid1" Width="90%" DataKeyField="Story Title" AutoGenerateColumns="False" runat="server">
    <HeaderStyle Font-Names="Verdana" Font-Bold="True" BorderColor="White" BackColor="OliveDrab"></HeaderStyle>
    <Columns>
        <asp:ButtonColumn Text="Read Story" CommandName="Select"></asp:ButtonColumn>
        <asp:BoundColumn DataField="Story Title"HeaderText="Story Titles"></asp:BoundColumn>
        <asp:BoundColumn DataField="Children Age"HeaderText="Children Age for Story"></asp:BoundColumn>
    </Columns>
</asp:DataGrid>

private void Page_Load(object sender, System.EventArgs e)
{
    if(!IsPostBack)
    {
        try
        {
// bind available topics
            Service1 s = new Service1();
            DropDownList1.DataSource= s.GetAllTopics();
            DropDownList1.DataBind();
        }
        catch(Exception ex)
        {
```
```
Response.Write("Exception: " + ex.ToString());
}
}//end if
else
{
//don’t need this item after first selection is made.
DropDownList1.Items.Remove("Select");
}// end else

If it is not in PostBack situation, the string “Select” is inserted into the list. This serves two purposes: it shows the user a clue as to what to do first, and second, it forces the user to select a topic. This causes the SelectedIndexChanged event to be fired for the control, which retrieves the story. On the PostBack, we remove the “Select” from the list, so it won’t cause problems in the handler code.

   this.DropDownList1.SelectedIndexChanged += new System.EventHandler(this.DropDownList1_SelectedIndexChanged);
this.DataGrid1.SelectedIndexChanged += new System.EventHandler(this.DataGrid1_SelectedIndexChanged);
this.ID = "topicForm";
this.Load += new System.EventHandler(this.Page_Load);

The value string name = (string)Session["SelectedStory"] when the server has completed the request from the topicForm and sent the result to the new browser to display the story. This is because the server is stateless, and sees each request as a single, connectionless client request. Unlike a client/server-based application, the server does not distinguish multiple pages requested by a single client as related to one another in any way. Certainly, we know this because we have been dealing with the connectionless HTTP protocol [39]. The Session is part of the HttpApplication class, and thus they are implemented in the Gobal.asax file over the Web Application. Session is stored in memory on the server.
Search by topic is a very traditional technique. It has the benefit of always producing a result, but the results are limited by the user’s knowledge of the topic. Figure 14 shows the selected story in the browser.

I would like to mention another basic story selecting method similar to key words search. In this technique, the user enters a story character and story location, and the system retrieves stories based on match scores. An example is shown in Figure 15.

The C# language System.Text and System.Text.RegularExpressions help to simplify the code. Regular expressions are a powerful language for describing and manipulating text [3]. A regular expression is applied to a string – that is, to a set of characters. These strings could even be an
entire text document. The .Net Framework provides an object-oriented approach to regular expression matching and replacement. The base class Library namespace System.Text.RegularExpressions is the home to all the .NET Framework objects associated with regular expressions. The central class for regular expression support is Regex, which represents an immutable, compiled regular expression

The .NET RegularExpressions namespace allow you to search a string, and to return the results in a collection. The collection returned is of type MatchCollection, which consists of zero or more Match objects. In this example, after you input “bear” and “forest,” the system retrieves all story text matching these two words in an object match collection. We also use the whitespace to match the story text, and then the same way, we can get the total words in this story text. In order find the best matches, we score matches according to the following formula:

\[
\text{matchScore} = \frac{(\text{countc} + \text{countl}) \times 100}{\text{countw}}.
\]

Countc represents the total times the input character appeared in the story, countl represents the total times the input location appeared in the story, and countw means the total number of words in the story.

![Figure 15. The user inputs character and location.](image-url)
The following is part of the aspx.cs code for calculating this score:

```csharp
Session["character"] = TextBox1.Text.Trim();
Session["location"] = TextBox2.Text.Trim();

Service1 se = new Service1();
StoryForEdit[] arEdit = se.EditStories();

foreach (StoryForEdit sp in arEdit)
{
    string[] subs = new string[6];
    subs[0] = sp.Title;
    subs[5] = sp.Content;

    string sc = name1.ToLower();
    string sl = name2.ToLower();
    string stext = sp.Content.ToLower();
    // get counts for distance calculator
    // get the input character count in each story
    Regex theReg1 = new Regex(sc);
    MatchCollection theMatches1 = theReg1.Matches(stext);
    foreach (Match theMatch in theMatches1)
        countc++;
    // get the input location count in each story
    Regex theReg2 = new Regex(sl);
    MatchCollection theMatches2 = theReg2.Matches(stext);
    foreach (Match theMatch in theMatches2)
        countl++;
    // get the word count of each story
    Regex theReg3 = new Regex(" ");
    MatchCollection theMatches3 = theReg3.Matches(stext);
    foreach (Match theMatch in theMatches3)
        countw++;
    // compute the match score
    matchScore = ((countc + countl) * 100) / countw;
    // the highest matchScore will be selected
    //Response.Write(matchScore + " ");

    h.Add(sp.Title, matchScore);
    countc = 0;
    countl = 0;
    countw = 0;
    matchScore = 0;
}
Figure 16. The matching score and recommended story are displayed in the browser.

Figure 16 shows the matching score and the recommended story in the browser. While this selection method is easy to implement, the results may not always closely match the request. For example, if the user requested characters or locations that were not present in the story database, the most closely matching story would not reflect those options, and might be a disappointment for the user.
CHAPTER 4

Case-Based Reasoning (CBR)

Case-based reasoning (CBR) systems work by finding past problem situations (or cases) similar to a current situation, and using the past case to suggest a solution to the current problem. Additionally, when an exact match is not found, a CBR system will adapt the solution to better fit the current situation, and possibly store this new case and solution in the case-base [14]. Techniques for achieving this can vary considerably.

When we talk about CBR, we often talk about a CBR cycle. Figure 17 is a typical CBR cycle. At the highest level of generality, four processes may describe a general CBR cycle [14]:

- RETRIEVE the most similar case or cases.
- REUSE the information and knowledge in that case to solve the problem.
- REVISE the proposed solution.
• RETAIN the parts of this experience likely to be future problem solving.

Retrieving one or more previously experienced cases, or reusing the case in one way or another solves a new problem. We can also revise the solution based on a previous case, and retain the new experience by incorporating it into the existing knowledge-base (case–base). The four processes each involve a number of more specific steps, described below. Figure 18 is a CBR cycle for story selection.

An initial description of a problem defines a new case. This new case is used to RETRIEVE a case from the collection of previous cases. The retrieved case is combined with the new case – through REUSE – into a solved case, i.e. a proposed solution to the initial problem. Through the REVISE process this solution is tested for success, e.g. by being applied the real world environment or evaluated by a user, and repaired if it failed. During RETAIN, useful experience is retained for future reuse, and the case-base is updated with new learned cases, or by modification of some existing cases [15].

Children Stories

Select story
Topic="Fairy Tales"
Story for Child Age =8
Story Character="snow white"
Story Event
Location="forest"

Search for stories that satisfy the description

Output sent to case Base

Select the story closest to the user request

Figure 18. A CBR cycle for the Story selecting.
4.1 Simple CBR Method

Our CBR system for selecting children’s stories works by comparing requests for stories with the stories available in the case base. These requests may specify various attributes of the story, such as character, location, or topic. For example, let us assume the user want to select a story with the topic of “Fairy Tales.” He starts by entering some attributes about the selection, such as “Story Character” is “Snow White.” He also wants a story suitable for a child who is nine-years-old. The system searches the catalogue and retrieves those stories that satisfied those conditions. This search can be a traditional one or can be based on similarity (i.e., the match can be only partial). In the third step, the retrieved cases are ranked. The story with attributes most similar to those entered by the user is ranked first.

A case-based reasoner is heavily dependent on the structure and content of its collection of cases – often referred to as its case memory. Since a problem is solved by recalling a previous experience suitable for solving the new problem, the case search and matching processes need to be both effective and reasonably time efficient. Further, since the experience from a problem just solved has to be retained in some way, these requirements also apply to the method of integrating a new case into the memory. The representation problem in CBR is primarily the problem of deciding what to store in a case - finding a proper structure for describing case content, and deciding how the case memory should be organized for retrieval and reuse [51]. In this project, our cases are XML-based structures.
4.1.1 Retrieve and Reuse

Given a target problem, we want to retrieve cases from memory that are relevant to solving it. A case consists of a problem, its solution, and annotations about how the solution is derived. The Retrieve task starts with a problem description, and ends when the best-matching case has been found. Retrieval subtasks include Identify Features, Initially Match, Search and Select, and are executed in that order. The identification task basically comes up with a set of relevant problem descriptors. The goal of the matching task is to return a set of cases that are sufficiently similar to the new case –given a similarity threshold of some kind, and the selection task works on this set of cases and chooses the best match [14].

A question that should be asked when deciding on a retrieval strategy is the purpose of the retrieval task. If the purpose is to retrieve a case which is to be adapted for reuse, this can be accounted for in the retrieval method [5].

In this method, we used five attributes of the story. These attributes are story title, story topic, target age, story character and story event location. The user can input each attribute according to their preferences (selecting from a DropDownList). Then the system collects input data from the user. The search results are collected and then ranked using a similarity metric that measures the “distance” between item and the “ideal” solution. Figure 19 is a user interface for user selecting attributes.
We explored different methods for calculating the distance score. In one method, the distance was based on the number of matching attributes. If a selected attribute matches the story attribute, then the match score is incremented by 1. With this technique, each attribute has the same weight – that is, each attribute contributes to the overall match score to a similar degree. The .aspx.cs code for performing this calculation follows:

```csharp
Service1 ser = new Service1();
Stories[] aStories = ser.childrenStories();

foreach (Stories st in aStories) {
    if (name1 == st.Title) countTitle++;
    if (name2 == st.Age) countAge++;
}
```
if (name3 == st.Topic)
    countTopic++;
if (name4 == st.Character)
    countCharacter++;
if (name5 == st.Location)
    countLocation++;

    countTotal = countTitle + countAge + countTopic +
    countCharacter + countLocation;
    h.Add(st.Title, countTotal);

    countTitle = 0;
    countAge = 0;
    countTopic = 0;
    countCharacter = 0;
    countLocation = 0;
    countTotal = 0;

} // end foreach (Stories st in aStories)

This search was implemented in the following way. Whenever a user selects an attribute, scores
for each case are updated depending on whether they match this individual attribute. For example,
if the user requests stories with the title “Teddy Bear Tales,” any case where the title matches
“Teddy Bear Tales” will have their match score incremented, otherwise, the match score does not change.

After retrieving the entire story database and calculating a match score for each story, we put this
match in a hash table. Each story’s title is a key, and the match score is a value. We sort the
values, get the highest match score and the story with the highest match score is selected.

Sometimes the selection may not closely match what the user requested. We have developed an
ability for the system to suggest methods for improving the search results. We illustrate this cycle
in detail with the following example.
We use CBRSelectSecond Web Form to collect user input. When we select each attribute from the DropDownLists, we call the Web Methods from Web Service. We use the DropDownList SelectedIndexChanged method, and the form is automatically submitted whenever the selection changes because we set the AuoPostBack property to true. In the SelectedIndexChanged handler, we are able to access the item that is currently selected using the SelectedItem property of the DropDownList. The programming code for calling these web methods is the following:

```csharp
if(!IsPostBack)
{
    try
    {
        // Load all story titles
        Service1 s1 = new Service1();
        DropDownList1.DataSource = s1.GetAllTitles();
        DropDownList1.DataBind();
        DropDownList1.Items.Insert(0, "Select");

        // Load all children age scales
        Service1 s2 = new Service1();
        DropDownList2.DataSource = s2.GetChildrenAge();
        DropDownList2.DataBind();
        DropDownList2.Items.Insert(0, "Select");

        // Load all story topics
        Service1 s3 = new Service1();
        DropDownList3.DataSource = s3.GetAllTopics();
        DropDownList3.DataBind();
        DropDownList3.Items.Insert(0, "Select");

        // Load all story characters
        Service1 s4 = new Service1();
        DropDownList4.DataSource = s4.GetAllCharacters();
        DropDownList4.DataBind();
        DropDownList4.Items.Insert(0, "Select");

        // Load all story event locations
        Service1 s5 = new Service1();
        DropDownList5.DataSource = s5.GetAllLocations();
        DropDownList5.DataBind();
        DropDownList5.Items.Insert(0, "Select");
    }
    catch(Exception ex)
    {
    }
}
```
The DropDownList1 handle event is as follows:

```csharp
this.DropDownList1.SelectedIndexChanged += new System.EventHandler(this.DropDownList1_SelectedIndexChanged);
this.DropDownList2.SelectedIndexChanged += new System.EventHandler(this.DropDownList2_SelectedIndexChanged);
this.DropDownList3.SelectedIndexChanged += new System.EventHandler(this.DropDownList3_SelectedIndexChanged);
this.DropDownList4.SelectedIndexChanged += new System.EventHandler(this.DropDownList4_SelectedIndexChanged);
this.DropDownList5.SelectedIndexChanged += new System.EventHandler(this.DropDownList5_SelectedIndexChanged);
this.LinkButton1.Click += new System.EventHandler(this.LinkButton1_Click);
this.ID = "CBRSelectSecond";
this.Load += new System.EventHandler(this.Page_Load);
```

Because it is round trip, the web service will have no memory about this calling. The server sees each request as a single, connectionless client request. After the process is done, all information will be discarded.

In the Reusing phase, the CBR system is to make use of the case that has been retrieved. This involves determining how the retrieved case(s) differ from the current case and how they can be used to help solve it. Figure 20 illustrates a user selecting attributes from DropDownLists.
Figure 20. A user is selecting attributes

In this example, the user hoped to find a story with the following attributes:

- Story title: The Teddy Bear’s Picnic.
- Target age: 8.
- Story Topic: Teddy Bear Tales.
- The Story Character: Teddy Bear.
- The story Event Location: picnics
After the user clicks the Submit button, the system will search for a matching story, and the result will be sent to the browser.

4.1.2 REVISE

Once a CBR system has retrieved the best matching case, it may then attempt to revise the solution if necessary. The system may also record how frequently a case has been excluded from selection. This information can be used to cull infrequently accessed cases from the case-base, and thereby improve system performance [10].

Figure 21 shows the results of a search on the following attributes:

- You Selected Story Title: The Teddy Bear’s Picnic
- You Selected Children Age: 8
- You Selected Story Topic: Teddy Bear Tales
- You Selected Story Character: Teddy Bear
- You Selected Story Event Location: picnics

Each story in the case-base receives a match score based on these attributes. The highest score possible is 5. The highest match score in the case-base is for the story “The Teddy Bear’s Picnic” which has a match score of 4, so the system recommends the story “The Teddy Bear’s Picnic.”
Figure 21. Match score and recommended story displayed in the browser for simple CBR method.

For this example, it is successful and the recommended story title is the same as the input title selected by the user. However, story titles may not always be known or may not match. In the following example, while the input story title is “The Teddy Bear’s Picnic”, the recommended story title is “Snow White and Seven Dwarfs.” Figure 22 shows the result.
Figure 22. Another example show the preferred title and recommended story title are not identical.

When the match score is less than 3, the search is considered unsuccessful and the system will provide suggestions for improving the search. Figure 23 shows an unsuccessful result.
Figure 23. An unsuccessful matching.

The following code generates the suggestion:

```csharp
if (Convert.ToInt32(arr[i]) < 3)
{
    //if (ac.Count() > 3)
    foreach (Stories st in aStories)
    {
        if (ac[0].ToString() == st.Title)
        {
            if (name2 != st.Age)
                achange.Add("Children Age");
            if (name3 != st.Topic)
                achange.Add("Story Topic");
            if (name4 != st.Character)
                achange.Add("Story Character");
            if (name5 != st.Location)
                achange.Add("Event Location");
        } // end if
    } // end foreach e

    Labeltry.Text += " Your selection is not very good, please improve you selection in these fields :" + "<br>";
    DataGrid1.DataSource = achange;
    DataGrid1.DataBind();
} //end if
```
When a given set of attributes produces a low match score, the browser will display advice for modifying the attributes. In this example, the system tells us to improve the selection in these three fields: Children Age, Story Topic and Event Location. Figure 24 displays the result.

![Figure 24. The system advice how to improve input fields.](image)

4.1.3 Retain

During the RETAIN phase of a CBR system, useful experience is retained for future reuse, and the case base is updated with a new learned case, or by modification of some existing case [46]. For example we might modify our selection according to the advice of the system, updating these three fields: Children Age, Story Topic and Event Location.

For example, we could change the attribute selection in these three fields. In the Children age field, we select age 9. In the Story Topic field, we select the topic “Bible Story.” In the Event Location field, we select “unknown.” With these new attributes, we find a successful match as
shown in the following figure. The new recommended story here is “Noah’s Ark” which represents a new learned case. Figure25 shows the newly recommended story.

![Figure25](image.png)

Figure25. The newly recommend story after the input changed according to the system advice.

The following examples represent a simplistic implementation of CBR matching intended primarily to illustrate the RETRIEVE, REUSE, REVISE and RETAIN phases of a CBR process. More sophisticated methods for producing match scores would greatly improve the functioning of our demonstration system. Our simple scoring system provides scores of 0 for an attribute that does not match and 1 for matching attributes. A more sophisticated system would provide a range of values. For example, if the user wanted stories related to “Animal Tales,” then a story with the topic “Teddy Bear Tales” should be considered a close match and the topic “Bible Story” less close. Similarly, if we chose an age of 8, then a story with an age of 9 would be closer than a story with age of 12. Also, different attributes may be considered more significant and may be weighted differently. We demonstrate such capabilities with a more advanced CBR method.
4.2 Advanced CBR Method

Our previous example demonstrates how a CBR system includes four main processes: retrieval, reuse, revision and retain. These four CBR processes are an adequate way to describe a running system. However, our basic CBR system only recognizes attributes that match or do not match. It does not provide any insight into how closely attributes match, and oftentimes this distinction among attributes can be significant. For example, when you select the topic “Teddy Bear Tales” the topic “Animal Tales” should be recognized as a better match, but “Teddy Bear Tales” and ”Bible Story” should be recognized is a poor match. Our basic CBR system also does not support the ability to assign different weights to the match scores for individual attributes.

Our advanced CBR method is capable of generating a range of values when considering how closely one individual attribute matches another, and using the weighted sum of these attribute scores in calculating the final match score. Such a scoring system is more likely to produce better matches when an exact match is not found. Figure26 shows the interface for specifying the weights for our Advanced CBR Method.

The form here is similar to our other data entry screens. However, we now use TextBox controls to collect weight rather than DropDownLists. The .aspx code is made up of simple entry fields with validation controls attached. Validation takes place when the form is submitted to the sever. Here we used two types of validation controls. Users are required to enter data on each field on the page. If you navigate to this page and click the submit button without typing anything in the TextBox, you should see an error message displayed. Because we use DHTML, we can see the error message displayed without a round trip. We also use CompareValidation to compare the contents of one control to some static value. Since the following values represent percentages,
they should all be less than 100. If the user enters an invalid percentage, an error message such as “your input weight percentages is not correct” will appear in the browser.

Figure 25. The changed user interface for advanced CBR method.

<asp:RequiredFieldValidator id="RequiredFieldValidator6" runat="server" ErrorMessage="you should input a percentage" ControlToValidate="TextBoxTW"></asp:RequiredFieldValidator>

<asp:CompareValidator id="CompareValidator1" runat="server" ErrorMessage="percentage <100" ValueToCompare="100" Operator="LessThan" ControlToValidate="TextBoxTW" Type="Integer"></asp:CompareValidator>
We also know that the total weight percentages should equal 100. If the total is not equal to 100, the user must modify their input. The following code also controls the user input in the .aspx.cs file.

```csharp
string s1 = (string)Session["wTitle"];  
string s2 = (string)Session["wAge"];  
string s3 = (string)Session["wTopic"];  
string s4 = (string)Session["wCharacter"];  
string s5 = (string)Session["wLocation"];  

int i1 = Convert.ToInt32(s1);  
int i2 = Convert.ToInt32(s2);  
int i3 = Convert.ToInt32(s3);  
int i4 = Convert.ToInt32(s4);  
int i5 = Convert.ToInt32(s5);  

int i = i1 + i2 + i3 + i4 + i5;  

if (i != 100)
    Response.Write("Percentage total should equals 100.");
else
    //Response.Redirect("CBRSelThirdDisplay.aspx");
    Response.Redirect("CBRSelForthDisplay.aspx");
```

For our advanced CBR calculations, we have chosen to exclude the title from the weighting calculation because the title is unique and producing a range of scores would be difficult or impossible.

### 4.2.1 Ranking Using Distance

The following code and illustrations demonstrate the operation of this more advanced match score technique.
// count age distance
int iname = Convert.ToInt32(name2);
int iage = Convert.ToInt32(st.Age);
if (iname == iage)
    distAge = 0;
else if (iname > iage)
    distAge = iage - iname;
else
    distAge = iname - iage;

// count the topic select distance
if (name3 != st.Topic)
{
    if (name3 == "Teddy Bear Tales")
    {
        switch (st.Topic)
        {
            case "Animal Tales"
            {
                distTopic = 1;
                break;
            }
            case "Nursery Rhymes"
            {
                distTopic = 2;
                break;
            }
            case "Fairy Tales"
            {
                distTopic = 3;
                break;
            }
            case "Christmas Treasury"
            {
                distTopic = 4;
                break;
            }
            case "Bible Story"
            {
                distTopic = 5;
                break;
            }
        }
    } // end switch (st.Topic)
    } // end if (name3 == "Teddy Bear ...
else if (name3 == "Fairy Tales")
{
    switch (st.Topic)
    {
        // dist between teddy and fairy = 3
        // dist between teddy and animal = 1
        // dist between fairy and animal = 3-1 = 2
        case "Animal Tales"
        {
            distTopic = 2;
            break;
        }
        // dist between teddy and Christmas = 4
        // dist between fairy and Christmas = 4-3
        case "Christmas Treasury"
        {
            distTopic = 1;
            break;
        }
        // dist between teddy and Nursery = 2
        // dist between fairy and Nursery = 3-2 = 1
        case "Nursery Rhymes"
        {
            distTopic = 1;
        }
    }
break;
// dist between tedd and Bible=5
// dist bteween fairy and bible = 5-3=2
    case "Bible Story" :
        distTopic=2;
        break;
    }// end switch
} //end if (name3=="fairy...
else if(name3=="Animal Tales")
{
    switch(st.Topic)
    {
    // dist between teddy and Animal =1
    // dist between teddy and Christmas = 4
    // dist between Animal and Christmas = 4-1=3
    case "Christmas Treasury" :
        distTopic=3;
        break;
    // dist between teddy and Nursery=2
    // dist between animal and Nursery= 2-1=1
    case "Nursery Rhymes" :
        distTopic=1;
        break;
    // dist between teddy and bible=5
    // dist between animal and Bible=5-1=4
    case "Bible Story" :
        distTopic=4;
        break;
    } // end switch
} // end if (name3== "animal...
else if (name3=="Christmas Treasury")
{
    switch(st.Topic)
    {
    // dist between teddy and Christmas =4
    // dist between teddy and Nursey=2;
    // dist between Christmas and Nursery=4-2=2
    case "Nursery Rhymes":
        distTopic=2;
        break;
    } // end switch
} // end else if
else
    //if (name3=="Nursery Rhymes")
    // dist between teddy and bible=5
    // dist between teddy and Nursery=2
    // dist between Nursery and Bible = 5-2=3
    distTopic=3;
    } // end if (name3!=st.Topic
else
    distTopic=0;

In the following example, Figure 27, and figure 28 show the user inputs:

- Story Title: Mary had Little Lamb
- Story for Children Age: 10
- Story Topic: Nursery Rhymes
- Story Character: bear
- Event Location: forest
- The input weight percentages:
  - Story for Children Age: 25
  - Story Topic: 25
  - Story character: 25
  - Story Location: 25

Given our selected attributes and weights, the system matched 6 stories; “Goldilocks and the Three Bears,” “The Horrible Hat,” “The Ugly Duckling,” “The Hare and the Tortoise,” “The Exciting Life of Uncle Bobo” and “The Most Beautiful Bear.” We know from the database, all these stories are about teddy bear and animal tales. The topic distance between “Teddy Bear Tales” and “Animal Tales” is 1 since they are very close. Figure 29 shows the result.
We count the total distance using the formula:

$$\text{distTotal} = \frac{(\text{distAge} \times 100)}{it^2 \times 8} + \frac{(\text{distTopic} \times 100)}{it^3 \times 5} + \frac{(\text{distCharType} \times 100)}{it^4 \times 5} + \frac{(\text{distLocType} \times 100)}{it^5 \times 5};$$
Figure 29, The Result with same weight percentages of each attribute

If we conduct another search with different weights we can see how the results have changed:

- Story Tile: Mary had Little Lamb
- Story for Children Age: 10
- Story Topic: Nursery Rhymes
- Story Character: bear
- Event Location: forest

- Story for Children Age: 10
• Story Topic: 50
• Story character: 10
• Story Location: 30

Here, we put more weight percentages on the story topic and story location, less weight percentages on the Story for Children Age and Story Character.

Figure 30. The input weight percentages are changed.

This time, the recommended stories are “The Hare and The Tortoise” and “The Most Beautiful Bear.” The story “The Hare and The Tortoise” has the topic “Animal Tales” and event location “forest.” The location matches our selection, and the topics “Teddy Bear Tales” and “Animal Tales” are very close (distance of 1). According to the user attribute selection and weight percentages, the recommended story “The Hare and Tortoise” is the best decision.
Another important capability of a CBR system is adaptation of cases. When an exact match is not found, the system can attempt to modify the closest match to better fit the user-selected attributes. In our demonstration system, we illustrate this capability by adapting the character and locations to better match the user’s selection.

Consider the following example:

- Story Title: Mary had Little Lamb
- Story for Children Age: 10
• Story Topic: Nursery Rhymes
• Story Character: bear
• Event Location: forest

According to your Selection, Your ideal story may look like this:

Bear had a little lamb, Its fleece was white as snow. And everywhere that bear went, The lamb was sure to go. It followed her to forest one day. That was against the rule. It made the children laugh and play. To see a lamb at forest. "Why does the lamb love bear so?" The eager children cry. "Why, bear loves the lamb, you know," The teacher did reply.

Figure32. The browser shows the modified story.

If the retrieved story does not match a particular attribute (like location or character) we can modify the retrieved story to better match the users request. In this example, the user requested a story with the title “Mary had a little Lamb” but with the character “Bear” and set in the location “Forest.” The system collects this information and sends a request to the server.

We use the following code behind to handle this process:

```java
Service1  se= new Service1();
StoryForEdit[] arr = se>EditStories();
ArrayList newa = new ArrayList();
foreach(StoryForEdit aa in arr)
{
    if(aa.Title==title)
    {
```

System.Text.RegularExpressions is the home to all the .NET Framework objects associated with regular expressions. The central class for regular expression support is Regex, which represents an immutable, compiled regular expression. For example, Regex is used to adapt our story text. The original story contains the sentence “Mary had a little Lamb.” The user selected the character “bear”, not “Mary”, so in our adapted story, “Mary” is replaced by “bear.” Similarly, “school” is replaced by “forest.” This direct replacement technique is simplistic, but is serves to illustrate the adaptation capability of CBR systems.

Story editing is done with the form CBREditForm.aspx. We use the DataList to collect user input and save and display data. Actually, The DataList supports this much the way the DataGrid does, but it has automatic layout functionality. We define the button with special CommandNames: edit, cancel, and select. Here we just used the edit and cancel and save commands. If there were any type of button controls in an item template that has the CommandName set to edit, then the DataList EditCommand event will be fired automatically on its click. Cancel and select behave similarly, causing the CancelCommand and SelectCommand events to fire.
When the EditCommand event is fired because the user clicked a button-type with CommandName edit, we set the EditItemIndex of the DataList to the currently clicked item. This causes the DataList to be re-created using the EditItemTemplate.

```html
<asp:DataList id="DataList1" BorderColor="#ffffff" BorderWidth="1" GridLines="Both" CellPadding="6" CellSpacing="0" width="100%" DataKeyField="Story Title" RepeatColumns="2" runat="server">
  <ItemTemplate>
    ......<br/>
    <%# DataBinder.Eval(Container.DataItem,"Story Title") %>
    ...
    <%# DataBinder.Eval(Container.DataItem,"Children Age") %>
    ...
    <%# DataBinder.Eval(Container.DataItem,"Story Character") %>
    ...
    <asp:LinkButton id="showButton" CommandName="ShowStory" runat="server">Show Story</asp:LinkButton>
    <asp:Label Visible="False" ID="labelStory" Runat="server">
      <%# DataBinder.Eval(Container.DataItem,"Story Content") %>
    </asp:Label>
    ...
    </ItemTemplate>
  <EditItemTemplate>
    <font face="Arial" size="2"><b>Children Age :</b></font>
    <asp:TextBox id="TextBoxAge" runat="server" text='<%# DataBinder.Eval(Container.DataItem,"Children Age") %>' >
      ......<br/>
    </asp:TextBox>
    ...
    <asp:LinkButton id="saveButton" CommandName="save" runat="server">save</asp:LinkButton>
    <asp:LinkButton id="cancelButton" CommandName="cancel" runat="server">cancel</asp:LinkButton>
    ......<br/>
  </EditItemTemplate>
</asp:DataList>
```

Figure 32 is a DataList of story database, just before editing. Figure 33 is during the editing phase.

The story “Mary had a Little Lamb” is editing.

Figure 32. The DataList of story database before editing.
When we start to edit, we call the FindControl method of the Item to get the TextBox values the user modified [1]. We can’t access these controls directly via a class-level variable, because they
are created only when the EditItemTemplate is used and must be accessed dynamically through FindControl. When we find the controls, we can get their text, and save the record to the database. We also use the DataKeyfield to store the unique record with each row.

The following code shows how to handle the CommandNames and FindControls.

```csharp
protected void DataList1_ItemCommand(object source, System.Web.UI.WebControls.DataListCommandEventArgs e)
{
    if (e.CommandName == "ShowStory")
    {
        Label showLabel = (Label)e.Item.FindControl("labelStory");
        showLabel.Visible = true;
        LinkButton hideLink = (LinkButton)e.Item.FindControl("hideButton");
        hideLink.Visible = true;
    }
    if (e.CommandName == "HideStory")
    {
        Label showLabel = (Label)e.Item.FindControl("labelStory");
        showLabel.Visible = false;
        LinkButton hideLink = (LinkButton)e.Item.FindControl("hideButton");
        hideLink.Visible = false;
    }
    // handle the save command
    if (e.CommandName == "save")
    {
        string titleName = DataList1.DataKeys[e.Item.ItemIndex].ToString();
        TextBox textAge = (TextBox)e.Item.FindControl("TextBoxAge");
        TextBox textTopic = (TextBox)e.Item.FindControl("TextBoxTopic");
        TextBox textCharacter = (TextBox)e.Item.FindControl("TextBoxCharacter");
        TextBox textLocation = (TextBox)e.Item.FindControl("TextBoxLocation");

        // if find, updat datasouce
    }
}
```

Figure 34 the modified story is saved and displayed.
CHAPTER 5
FUTURE WORK & CONCLUSIONS

There are a number of enhancements we have considered for our system. The current system does not truly update the case-base, but rather stores changes in the cache. After the browser disconnects, all updates are lost. For the next step, we could build the user database in SQL server, use sqlDataAdaper to get work done easily and keep the user’s history record. The current scoring mechanisms are rather simple. We would like to investigate more sophisticated techniques for producing match scores. Finally, the adaptation of story text works by direct substitution of text and the results are not always grammatically correct. We would like to create a more sophisticated text adaptation capability as well. While this development represents a basic proof-of-concept, we found ASP.NET to be an effective platform for making CBR capabilities available in a web-compatible form.
REFERENCE LIST


[38] Quick UDDI (qUDDI) Registration Service, qUDDI.EraServer.NET, 2002.


[52] Lenz, M. *Experience from deploying CBR applications in electronic commerce*. In proceedings of the German Workshop on CBR.