Object Oriented Mapping for HTML Documents

Francesco Garelli <garelli@acm.org>
Carlo Ferrari <carlo@dei.unipd.it>
Department of Electronics and Informatics
Università di Padova
via Gradenigo 6a - 35131 Padova. Italy

Abstract

Emerging distributed technologies aim to provide simple and powerful tools for web services design and implementation. Main vendors provide modern frameworks so that a good coordination between web interfaces and object oriented paradigms can be supported. These frameworks aim to divide a web application in different logical tiers and they describe these tiers by a object oriented architecture. Unluckily the object oriented facilities are usually constrained to the application components which belong to the application, while a comfortable way to access external web services which run on different hosts, is missing.

In this article we describe a strategy which simplifies the creation of object oriented container for existing web services. First we introduce a method to collect relevant information from existing HTML services, like search engines or web based mailboxes. Then we define a tool which convert a HTML template into a Java class. This class acts as a wrapper for the services that a web site offers. A application model access to the web site by calling the class instances methods.

1 Introduction

Sometimes a question that appears easy and harmless reveals to be very difficult and hostile . Among those questions there is the following one: "*What is a world wide web document*?". If you believe that question is trivial try to ask it to different people; unlikely you will get a single and definitive answer.

The reason you will probably get different answers is because the HTTP protocol and the HTML language evolved during the last years towards complex features.

The HTML format was created by Tim Bernerns-Lee and Robert Caillau at CERN in 1991. The language only described the logical structure of documents because the original motivation was to keep track of experimental data. Quickly non physicist users wanted more control over the appearance of the pages. Starting from HTML 3.0 browsers got commercial value and software companies, namely Netscape and Microsoft, proposed some non-standard extensions. At the same time web server developers introduced specific protocols in order to support dynamic context.

Since 1991 the HTTP based architectures evolved both on client and server sides [2]:

- *a better look and feel.* Web pages became more interactive. New graphical items, like layers or styles, some scripting (e.g. javascript) and Java applets provide an improved appearance.

- *dynamic and user customized contents.* Modern web sites are based on dynamic environments. Usually a user needs to authenticate himself before he can accesses to the provided services. Thanks to this authentication, the web application can maintain user related information between different sessions and the showed pages can be customized according to the user's desires. For example all web-based email services require this authentication. Even when the authentication is not a requirement, the retrieved content often depends on the information that the user submitted to the web server in advance.

These complex features make the development of web applications more difficult. In order to face this increasing complexity, vendors are designing new object oriented paradigms on the server side. The most challenging emerging technologies are the .NET platform from Microsoft and the J2EE platform from Sun [3]. These architectures aim

to offer an homogeneous and practical environment to develop applications both in classic graphical user interfaces and web-based interfaces. The web application is built in accordance with object oriented design and the use of third-part components: this strategy reduces the development time and effort. These frameworks don't allow the reuse of existing online web services although.

In this article we propose a object oriented mapping for HTML documents in order to access to existing web services. Thanks to this mapping we can envelope a HTML document inside a object oriented container. In the next section we discuss the properties of web interfaces and some modern frameworks which support their development. The third section deals with a typical HTML language problem: the presence of logical and layout data mixed in the same page. Therefore it describes a method to extract relevant information from documents. The forth section introduces a process to envelope this relevant information in a object-oriented container. It describes also a Java tool which makes this process easier and faster. The last section resumes the article content and it hints at some open problems.

2 Object oriented paradigms and web interfaces

The need for an object oriented representation of web information depends on the . The browser is no more a tool for leafing through textual documents, but it is a graphical terminal for applications which run remotely. As graphical interface the web browser supplies many advantages which a classic client application built by the local operating system widgets doesn't offer:

1. *the web interface is auto-explaining,* it contains the interface for the remote application and at the same time the documentation for that interface. Hence even a novice or occasional user fells at one's easy.
2. *the web interface is universal,* it is well-known by most people and it is independent of the local operating system.
3. *the web interface is light*, since all the logic (and the code) of the application runs remotely on the server. Hence almost all the maintenance of the application is on the server side and the expensive maintenance on client side is minimum.

Thanks to those advantages web-based applications are evolving and becoming more complex. In particular the logic tier of web application are becoming larger and major vendors propose object oriented frameworks which support the design and the development of the application logic. Unluckily the HTML format is not an object oriented language and it is not suitable for an object oriented architecture. Actually some efforts toward object-oriented mapping of HTML are standing out; the most studied problem is how to hide the CGI-BIN protocol behind a object oriented layer. A real effective tool is a framework developed by CapeConnect [4]. This framework translates a HTML form request to a remote method invocation . The remote method is exported by a CORBA compliant server which implements the application logic. Moreover the framework converts the remote call result to a result web page according to a page template. Hence the web interface appears to the application logic as a CORBA client.

Although this product, as other frameworks, can't model the web interface as a remote server. As consequence the application logic can't use the web as a remote component and it can't use online services. If this abstraction were available, agents which browse an internet site, could be easily developed. They could be fast interfaces to legacy web applications (e.g. search engine or SMS sender sites) or they could collect and organize information from the Internet.

3 The document representation

In order to model a web service as a remote component, an object oriented representation of HTML documents is needed. At the moment we don't take into consideration XML documents because we need an access to existing web services which normally provide a HTML interface. Anyway XML support appears to be a simple extension.

A natural object based representation for HTML document is the DOM model. This model describes a web page as a tree whose nodes are the tags and the attributes of the documents. A DOM tree is a complete representation of the document because it contains both the content and its structure; in fact it is quite easy to retrieve the original document from the DOM tree. Unluckily all those information can be excessive for our aim.

The problem we are facing is how to envelope a web service inside a component. Actually we are not interested in all the information inside the page but only to some substrings which contain relevant data: images and static text are useless information. For example if our application needs to use a web service to retrieve a list of recipes, we could model the query as a method invocation and the search result as an object. The object content should be just the recipes and not the layout information.

Hence we need a technique to extract relevant information from a document. Modern web sites create dynamic pages by mixing static HTML code with scripting code

```
<html>
 <body bgcolor=”green”>
  This page has been seen 432 times
  <a href="hello.html">reload</a>
  <form action="/search.asp">
   <option type="text" name="keys">
   <option type="submit">
  </form>
 </body>
</html>
```

Figure 1. Simple HTML document

```
<html>
 <body bgcolor=<%=bgcolor type="String"%> >
  This page has been seen <%=counter type="Integer"%>
times
 </body>
</html>
```

Figure 2. Simple HTML template

(e.g. ASP or PHP pages). In a similar fashion we propose a HTML template. This template contains normal HTML code and some new special tags which we call *xtags*. The *xtags* resemble the ASP or PHP languages sintax:

<%=*attribute_name* type="*attribute_class*">

where the field *attribute_name* is a string identifier related to the relevant data and the *attribute_class* is the data type. If no type is present, the *String* type is assumed.

In the template the HTML code denotes useless data while special tags are placed where in normal documents dynamic information are placed. By a comparison between a HTML document and a related template, relevant substring can be extracted. Figure 1 shows a simple HTML document and Figure 2 shows a possible related template. From the comparison between the two documents the following data can be extracted:

identifier	type	value
bgcolor	String	green
counter	Integer	432

In order to make the comparison algorithm simple some constraints are required: a special tag can stands either inside the text between two tags (as for the counter identifier) or in place of a tag attribute value.

Sometimes a HTML document contains a collection of similar items whose number is unknown in advance. For example quite all search engines return a set of data contained in the same layout (often a table row). Since the number of items is unknown the template can’t contain appropriate *xtags*.

```
<html>
 <body>
  <h1>Foreign songs</h1>
  <table>
   <tr>
     <td><b>Author</b></td>
     <td><b>Title</b></td>
   </tr>
   <tr>
     <td>Beatles</td>
     <td>Let it be</td>
   </tr>
   <tr>
     <td>Cat Stevens</td>
     <td>Father and son</td>
   </tr>
   <tr>
     <td>Pink Floyd</td1>
     <td>Another brick in the wall</td>
   </tr>
  </table>
 </body>
</html>
```

Figure 3. Simple document with repetitions

Hence we need a different sintax to manage possible repetition in HTML documents. A valid solution could be the following notation:

<%loop%> html code <%/loop%>

The two special tags enclose the code which can repeat many times. Of course the enclosed code can contain many *xtags*. Figure 3 and 4 shows a simple example and its template; the following table represent the relevant information from comparison:

identifier	repetition	value
title	no	Foreign songs
author	1	Beatles
song	1	Let it be
author	2	Cat Stevens
song	2	Father and son
author	3	Pink Floyd
song	3	Another brick in the wall

```
<html>
 <body>
  <h1><%=title%></h1>
  <a href="main.html">Home</a>
  <form action="/search.asp">
   <option type="text" name="keys">
   <option type="submit">
  </form>
  <table>
   <tr>
     <td><b>Author</b></td>
     <td><b>Title</b></td>
   </tr>
   <%loop%>
   <tr>
     <td><%=author%></td>
     <td><%=song%></td>
   </tr>
   <%/loop%>
  </table>
 </body>
</html>
```

Figure 4. Simple template with loops

4 The object-based wrapper

Once a valid technique to extract information from a HTML document, is available, the task of store it in a component is quite simple. Each HTML document corresponds to a single object while each template corresponds to a class. The information that the comparison process creates, is stored inside the public attributes of the object. The attribute identifier and its type can be those that each *xtag* provides. Figure 5 shows a Java class for the "songs" template. The *title xtag* corresponds to a public attribute with a anonymous identifier and type equal to *String*. Instead the *author* and *song* attributes have a type equal to *Collection* because their *xtags* are inside a *loop* block and so they refer multiple values.

In order to define the class, the only requirement is the document template. Instead when an object is created, the document is mandatory. In fact only the comparison between HTML document and its template produces the information which attributes contain.

A web document doesn't contain only useful data but also it owns references to other pages. Two kinds of references are possible; HTML form and HTML link. The former, denoted by the *<form>* tag, is a parametric reference; in fact when a user submits a form, he specifies some values which affect the response. The latter, denoted by the

```
public class Simple extend HtmlObject {
 public String title;
 public Collection author;
 public Collection song;
 public Simple(String url);
 public HtmlObject mainHtml() {...}
 public HtmlObject searchAsp(String keys) {...}
}
```

Figure 5. Java classes

<a> tag, is a reference with no parameters.

In both cases when the browser follows a reference, it downloads a different HTML document. Since in our model a HTML document corresponds to an object, we need a construct that supports some parameters and that returns an object. This construct is just a method invocation: hence for each form or link that is present in a template, the class must own a different method. If this method maps a HTML link, it doesn't require any parameters. Otherwise, i.e. when it maps a form, its signature defines some parameters which corresponds to the form parameters. Figure 5 shows the two methods *mainHtml* and *searchAsp*. In the original document they correspond exactly to the HTML link whose reference is "main.html" and to the HTML form whose action is "search.asp".

The creation of a Java class beginning from a HTML document, requires some effort. In our work we developed some useful tools to make this task easier:

- *htmlObject class.* It is the base class for each component. It implements the algorithm which compares a downloaded page with a template.
- *htmlbin compiler*. It is a tool which generates a Java class from a document template. This class follows the rules that we described in the previous section: it owns the appropriate attributes and methods. Since the class extends HtmlObject, it is able to download a web page and to compare it with the correct template. The class implements a *load* method that saves the comparison result in the attributes. Finally the class provides a
- *html registry.* It is an external text file which join each URL to a template and to a class able to process it.

Thanks to those tools the human action is minimal. Figure 6 shows the four simple steps so that a web service can be enveloped in a object:

1. The developer downloads a document from the web site that he wants to envelope

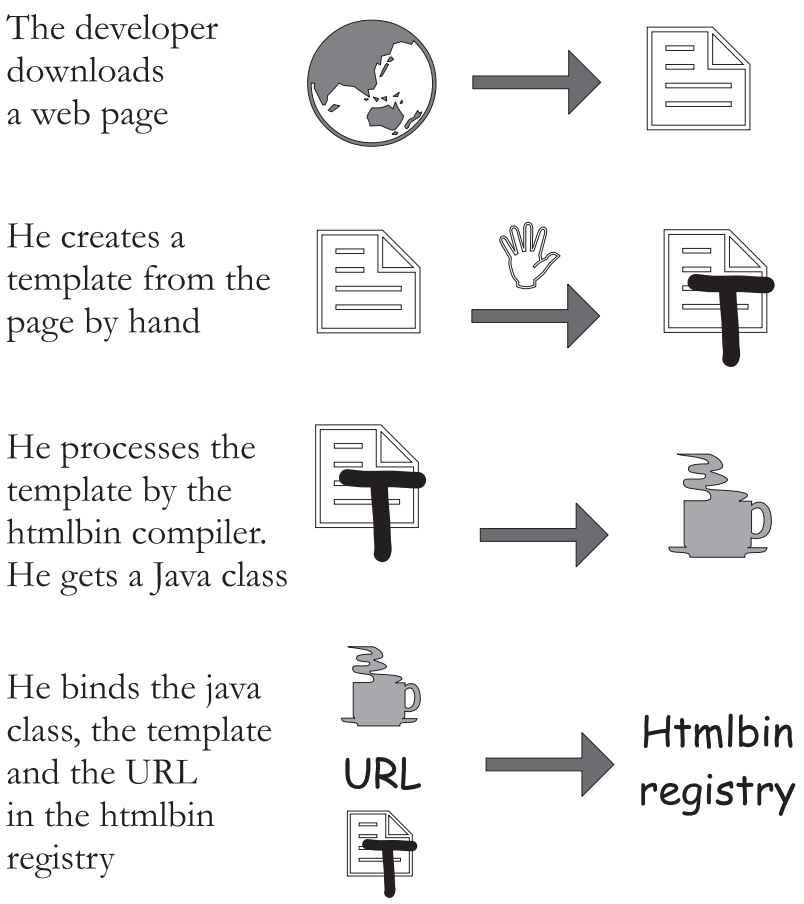

Figure 6. The developer envelopes a web document in a Java object

2. He identifies repeated blocks in the document structure and he replaces them with the *loop* special tag. He replaces also all dynamic information (i.e. information that depends on user actions) with *xtags*. This task must be achieved by hand using a text editor. A graphical tool could be convenient in future.
3. He processes the template by the *binhtml* compiler. He gets a Java class which envelopes a web service.
4. He inserts the site URL, the template location and the class name into the *html registry*.

Therefore the developer can access to the registered online web service in his application. In fact the *HtmlObject* class provides a static method *call* which create web objects beginning from a URL. Other modules can call this method passing as parameter the URL where the service is located. As figure 7 shows the method retrieves the Java class that can process that URL from the *html registry* and then it calls the class constructor. The class constructor downloads a web page according to the specified URL and it retrieves from the *html registry* the related template. Then it compare the two documents and it stores the comparison result in its attributes. Once the constructor terminates, the instantiated object contains the information from the web page and other modules in the application have access to this information. Then the static method returns the object to its caller. For convenience the object is returned with a type equal to *HtmlObject*, the parent of each class which the *binhtml* compiler generates. Hence the caller module must cast the object according to its real type before it can read its attributes.

5 Conclusion

This article describes a object oriented mapping and a Java tool which simplify the building of object container for web services. Thanks to this technique web developers can easily improve component reuse in their application.

The actual prototype proved to work correctly with sim-

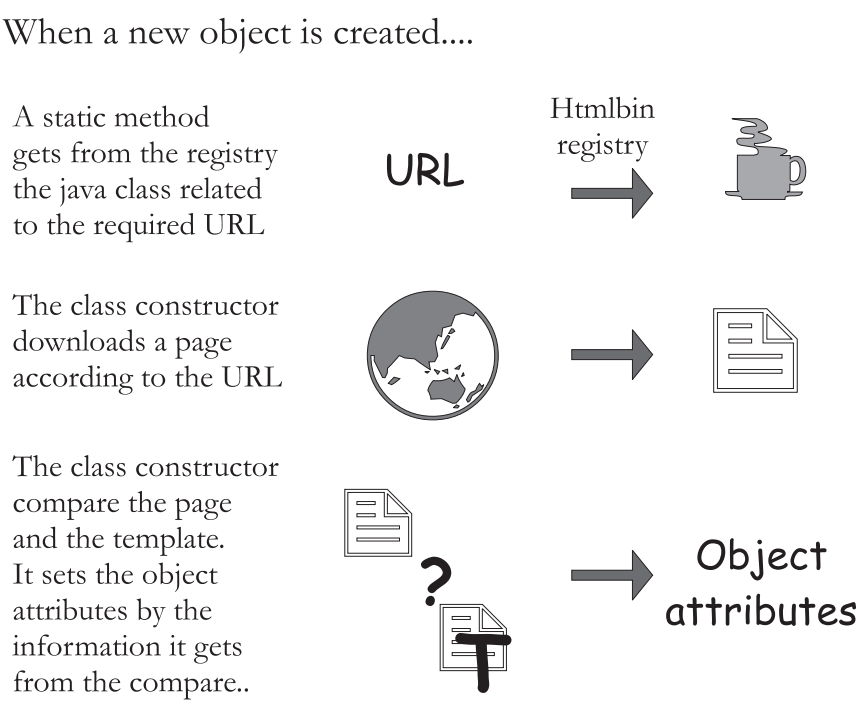

Figure 7. The creation of a wrapper object

ple web services but showed some problems with more complex ones. This happens both because current web sites uses enhanced HTML extensions (e.g Javascript) which our tool doesn't recognize, and because many HTML pages contain sintax errors. Moreover the comparison algorithm showed some instability when it must locate repeated blocks; actually it can be proved that the algorithm is not deterministic in case of particular documents and we are looking for some improvement. Anyway in our experience these harmful documents are quite rare.

Finally we are evaluating the opportunity to add support for XML. Nowadays XML is used principally as content language inside web server but it is not yet a diffused language for web sites because only few browsers manage it. Probably XML will be a better choice for embedded devices or services which don't require a browser. In this case an extension of our work can simplify the interaction between XML documents and the object oriented paradigm.

Acknowledgements

The authors would like to thank Nicola Zingirian for the fruitful discussions. This work has been done under the support of the MURST 60% project.

References

[1] J. Nielsen, *Hypertext and Hypermedia.* Academic Press , San Diego, CA, 1990

[2] P. Fraternali, *Tools and Approaches for Developing Data-Intensive Web Application: A Survey.* ACM Computing Surveys 31, 3 (Sep. 1999), 227-263

[3] P. Perrone, V. Chaganti. *Building Java Enterprise Systems with J2EE*. SAMS Publishing, 2000

[4] Cape Clear Software, *CapeConnect Technical Overview*. http://www.capeclear.com/products/whitepapers/capeconnect_techoverview.pdf

[5] Metadata, *Javacc Documentation*. http://www.webgain.com/products/code_analyzer/documentation.html