A NEW APPROACH TO RECYCLE WEB CONTENTS
The DOM Tree as the Support for Building New Web Pages

Luis Álvarez Sabucedo and Luis Anido Rifón
Telematics Engineering Department, Universidade de Vigo, Spain

Keywords: Interoperability, web contents, DOM tree.

Abstract: After an initial period of populating the web with a large amount of resources; a new situation has to be faced in the scope of the WWW: the over production of web contents. Currently, there is a lot of resources available in the web and new approaches are scanned to improve some features. In particular, this paper tackles the reusability of contents. The aim of this paper is an alternative method to provide a simple and effective manner to reuse contents in order to create new web resources. This approach is based on the use of the DOM tree that defines the web resource to build up new web pages. The presented method involves minor changes on the server side and no change at all on the client side. Besides, this proposal can take advantage of resources already developed using on-the-fly technologies.

1 INTRODUCTION

Over an initial period of time, in the mid 90s, a lot of contents for the Internet were developed. In fact, we were witnesses of an exponential growing of web contents on a myriad of different web sites. Therefore, the Internet community soon foresees the problem of the lacking of contents moving to the excess of contents. In order to overcome that situation, during these last years, a research trend in Web environments is related to information retrieval, interoperability and content reuse. Briefly, we can outline some of the more outstanding: the use of metadata, the development of data standards formats (e.g. RSS standard), intelligent agents, crawlers, etc. All of them provide suitable solutions that overcome limitations on the excessive amount of information and improve the current situation where nearly no interoperability would be possible if not measures were taken in the short term.

As a consequence and to overcome the presented scenario, we propose the development of a platform that provides agent users with server side processing support. Using this platform we will be able to reuse contents in a standard way regardless of particular technologies. The final goal we are looking for is a suitable way to compose contents in a single web page from contents already available in other web sites and resources already developed.

In the insofar developed technologies/initiatives, we notice the existence of a gap of functionality not yet fulfilled as this is not possible just by using server side programming deliver this service. So, the main goal for this project is to provide a suitable and simple way to recover partially documents and therefore, be able to compose new web contents in a collage fashion. Our proposal will deal with those requirements: a server-side simple support for partial data retrieval in web environments, i.e., XML-like contents. In order to achieve this goal we propose the use of URL (Universal Resource Locator)(Network Working Group, 2007) schemas with some upgrade to be able to address a single certain node included in any document available as web content. As web browsers, in general, allow us to introduce any URL with nearly no pattern filtering, the only needed upgrade is concern the web server, which must be able to deal with this enhanced URL schema.

As all web pages may be defined in terms of its DOM (Document Object Model) tree(Mozilla, 2007), we can *re-make* new web contents just by mixing nodes from different web resources, see Figure 1. In order to do this, we just need to be able to address nodes in the DOM tree remotely from the web server responsible for content providing. With this idea in mind, when a user agent requests a web page from a server, this server must collect nodes from those servers where contents are actually stored and com-

Álvarez Sabucedo L. and Anido Rifón L. (2008).
A NEW APPROACH TO RECYCLE WEB CONTENTS - The DOM Tree as the Support for Building New Web Pages.
In *Proceedings of the Fourth International Conference on Web Information Systems and Technologies*, pages 284-289
DOI: 10.5220/0001519202840289

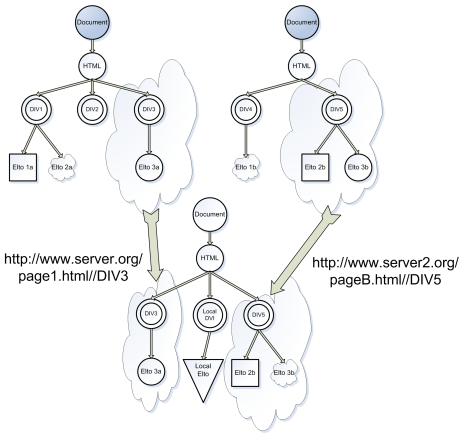

Figure 1: Merging DOM trees.

pose the final submission that must be provided in response to the original request by the user agent. This works properly as any web content can be rendered from and to a DOM tree.

Thus, we can compose *live* the new page from contents recollected all over the World Wide Web, i.e., there is not intermediate steps, contents are gathered in a full automatic fashion to make up new contents.

It is important to bear in mind that the page is made of several already existing nodes, no matter where they come from. It is possible to include contents created dynamically from any DBMS (Data Base Management System) by using any technology, static contents or what ever that can be accessed as a DOM tree. As a consequence, by using just HTML and no server side technologies to develop dynamic contents, we are providing contents that may change according its original sources.

The advantage from other technologies already available, those are briefly described later on, is related to provide support with no need to update software on the client side or contents themselves on the server: just by upgrading web server functionalities we can meet the requirements as we will show.

This paper, therefore, will present a simple solution to contribute in the general trend towards a more accessible Internet with little cost. The idea for this contribution lies on the selective recovery of information from already existing web contents with no modification on legacy information systems.

The rest of this paper is organized as follows. Firstly, we will introduce a key concept for the proposal in this paper: DOM trees. Then, we will present our proposal and all technical details will be addressed. Later on, we will discuss details related to the implementation of the proposal. Once the development is completed, we will show the testing phase by mean of some examples deployed for testing proposes. Finally, some conclusions are included.

2 DOM TREE

This contribution takes place in the current Web environment where a lot of on going solutions are being carrying out to solve the presented problem. To be able to manage ourselves in the context of the present proposal, we must deal with a key concept: DOM.

The DOM tree is just a simple way to lay out contents from a HTML page. This representation is conceptual, so there are several ways to present it; nevertheless, the most usual way to do it is by mean of a tree-like form diagram, as already shown in Figure 1. Valid DOM trees always have a root node for the document itself and pending nodes to render the full page. This structure may be as complex as desired and, obviously, it will get larger as contents get more and more complex.

Once an agent has already downloaded a web page, it can create automatically the corresponding DOM tree. Likewise, the agent can get/update any piece of the information by accessing the proper node. In order to implement that functionality, there are already plenty of software libraries that make it easy to parse it in order to look for information and change values. To accomplish this particular issue about accessing and updating information from DOM tree in the client side, we must take the DOM initiative from W3C(W3C, 2007a) into account. According to the W3C, this project is

> a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.

By using this project, it will be possible to filter data on the client side by using a standard API for all client agents. The main aim for this project is to support transformation of contents on the client side by using user agents, mainly, web browsers.

This solution will not fulfil our needs as far as the point for this project is to express an API to access information on the client side, but it does not allow to compose contents as requested.

3 THE PROPOSAL

As previously stated, our solution will provide a server side mechanism to recover contents from the

server and submit that information to the client. This development must allow us to reuse contents already developed and insert them into a new page without client side processing. The solution proposed to the present problem involves minor modifications on the server processing and no client changes at all. As a case of use we may refer, for instance, to any blog where it is needed to introduce information from any other web site with no modification on the latter as we will not be able to perform such modifications to fulfil our needs. This platform will allow users to pick up a certain node from any web resource by addressing the proper node in the DOM tree and inserting it on its own DOM tree (see Figure 1). The result, as we will show on the testing phase, is a new web resource, a web page, where information collected from several external web sites is available.

To achieve this goal we need to perform two major, and almost unique, upgrades in current systems:

- Upgrade the requested URL format to express nodes in a DOM tree.
- Insert a module on web server to collect just the requested data.

So far, when an agent user requests a web resource, no matter if this is an HTML page, a flash element or an image; it just sends an URL to ask for the desired resource as it is known as a file in its own filesystem. But we need to be able to express an element from the DOM tree in the requested resource, of course, a HTML/XHTML or XML-based resource. We may consider this new requested condition as a refinement of the previous situation.

At first sight, we could think of two already in use solutions such as Xpath(W3C, 2007b) or the use of notation similar to anchors in HTML, i.e., $file.html\#node1$. We dismiss the first option due to the high level on complexity that would introduce in further development phases. The latter option is more suitable due to its simplicity and meeting the required conditions for this solution. The applied criterion is to keep the system as simple as possible while it meets the requested capabilities. We chose the following format:

http://server.org/resource.xhtml//node1

By using this format, we mean that the desired resource is just *node*1, an element in the file *resource.xhtml* located in the server *server.org*. Main reasons to choose this format are:

- This schema for URLs fits with the current specifications provided in the RFC about URL.
- As far as we can determine the resource with no possible misunderstanding, we are able to know what is the real request from the user agent. We can assure this as far as we completely detached the file containing the node from the node itself by the double '/'. As a result, there is not overlap with any other possible request and no misunderstanding is possible.
- Most of already working web clients are able to manage this pattern to address contents with no changes.

There is a remaining issue not solved yet. The remaining problem is about the delivery of contents themselves. Current web servers process each request by performing the requested operation on the proper file o files by mean of a HTTP connection. At this point, we would like to outline that HTTP protocol has nothing to do with contents themselves. This protocol is just responsible for the submission of contents through data networks despite any other consideration.

To properly deliver this solution, we must modify the performance of web server but not the HTTP protocol itself that will remain providing exactly the same functionalities. Instead of finding the requested file and submitting it, our web server must deal with an additional problem: parsing the file and gathering the requested information. This server must get the file and process it until it just gets the desired piece of information, the desired node. The implementation of this behaviour does not require a large amount of resources. By using already developed tools we can find the proper node and submit it in a quite simple way, as described in the next section.

4 SOME NOTES ON DEVELOPMENT

Insofar, we have presented the ideas required to implement this system. The only software implementation required is an enhancement on the web server. In this way, the client software, mainly browsers, will need no upgrade to be able to use this additional feature.

In order to test the proposed schema, we decide to implement this mechanism by modifying an already working server. So the first decision we had to face was the election of the right web server to work with. The final selected web server was Tornado(Neil Conway, 2007). The main reasons for this selection are:

- This software is provided under GPL(Free Software Foundation, Inc., 2007) license, so we are allowed to modify source code to fulfil our needs.

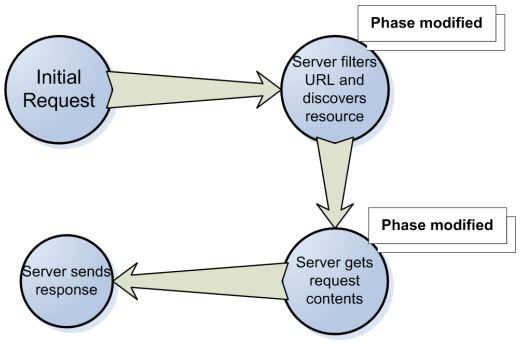

Figure 2: Server processing requests.

- The software platform where this project is being developed, actually is an ongoing project, is Java. So we can get advantage of an OOP (Object Oriented Programming) code where a lot of resources are available to work with little effort for integration in our project.
- The project has the right state of maturity, i.e., it is big enough to provide functional result but not huge enough to introduce too much work on modifying the source code for the new requested capabilities.

Independently of the selected web server, the needed changes in the work flow diagram of the web server (see Figure 2) are:

- Deal with the requested URL to discover the real requested information and to improve the management of error message with regard to the existence of the requested resources.
- Filter the response to the client in order to send just the proper piece of information.

To fulfil the functionality from the first point, we just locate the proper piece of code and filter the final part of the URL to properly search for the resource requested.

```
private File translateURI(String uri)
 throws HTTPException
{ { ...
 /* We need to parse the incoming
   request to get the desired node*/

String relURI =
 uri.substring(uri.indexOf('//', 7));
  // Getting the relative URI
String  requestedNode =
relURI.substring
   (relURI.lastIndexOf(':')+1);
   // Getting the requested node
... } ..
/* When the URL is properly located,
 we can check if the requested file
 actually exists */
 if (requestFile.exists() ==false)
  throw new
    HTTPException(HTTP.NOT_FOUND);
..
```

An interesting point in this situation is related to the error message in case of no resource available. We may make a distinction between two different cases. We can deal with the situation where no resource is available because there is no the requested file and submit, therefore, the 404 error message. But also, it is possible to find the file but we may not be able to send any piece of information due to the inexistence of the requested node or even due to the improper composition of the file as they may be not XHTML or HTML compliant. We decided to use the same error message, as far as we are dealing with a partial document as if it was a full document. Nevertheless, other options are also possible.

There is only an issue left to implement this solution: parse the file and recover the right node. In our case, a Java project, this may be solved by using any of the existing libraries for DOM processing. In our case, we decide to use de library jaxen(The Werken Company, 2007). Obviously, this improvement is the previous step to the final submission of the information to the client agent. In our case, the resultant piece of code is as follows:

```
public static Node GetContent
    (String requestedNode, Object doc)
{ Node node = null;
  try {
  /* We declare an object from class
DOMXPath to get the desired node */
   XPath xpath = new
     DOMXPath(requestedNode);

 /*We get a list with all nodes with the
  proper name */
List value = xpath.selectNodes(doc);

 /*In case of more than one node, we
 submit just the first one It would be
 possible to send all of them */

 Iterator resultIter = value.iterator();
  if (resultIter.hasNext()) {
   node = (Node) resultIter.next();}
  }
  /* Default exception handling */
   catch (XPathSyntaxException e) {
     throw new
       HTTPException(HTTP.NOT_FOUND);}
   catch (JaxenException e) {
     throw new
       HTTPException(HTTP.NOT_FOUND);}
   return node;
}
```

When the information is already located, we only need to serialize it and submit it through a TCP socket. With quite little additional upgrade on this code and the one responsible for data serialization, we could provide more options to recover different parts of the document. Nevertheless, no more features will be developed so far in order to get feedback from users to improve further developments. So far, just support for recovering a certain node in the requested document is provided. As this project gets support and on-line experience provides feedback, we will develop support for more advanced queries.

5 AN EXAMPLE

In order to test the already developed project, we deployed several experiences. As our server is so far the only one with these capabilities, the first step in this testing process was to clone some web pages from the Internet to our own server. The first obvious test is to request for some particular element from the DOM tree. This test was successfully accomplished with no special incidences.

The first serious attempt to test our system consists of a simple web page, as explained in the introduction, where contents were address using our new schema; we, therefore, developed a web page made, mainly, of frames.

By using this concept we can compose any web content as they can be accessed by mean of its DOM tree. It is important to notice that the DOM tree corresponding to this source code, at first glance, may seem to be quite simple, but we may be dealing with a huge structure.

One of the main constrains for web design using this technique is due to limitations to use of external addresses. To access external resources, the only suitable tag included in the HTML 4.01 specification is the tag *frame* so we are forced to use it. In this situation, it would be very suitable to be able to use other tags, such as *div* to reference external resources. As this tag provides a certain position to place certain contents and it is widely supported for laying out with CSS, it would quite useful to be able to use it as container for external resources.

But the most interesting test was about its function as a real interoperability booster. As previously stated, the aim in this project is to allow the content interchange in the World Wide Web. The expected way this tool is going to be use is to include contents from several web resources in just one single new web resource, mainly, a web page. To achieve this goal, we designed a web page with contents both, from its own and externally located. In the insofar web pages, it is a quite usual practice to include as contents for frames someone else's full web page; an application for this project is to allow web designer to include just a single piece of an existing web page in their own web sites.

Several concerns may arise on content presentation. DOM tree nodes do not just include information about data but also about presentation so problems may arise if no measures are taken. During the design of the test page some resources had to be devoted to fix presentation bugs. These problems may get worse when dealing with contents poorly formatted or even wrong. So we must bear in mind problems on placing external HTML code on new pages.

The key concept to achieve a higher level of content reusability involves merging different DOM trees from several sources into a new single tree where all the information is stored. Besides, these transformations are not in the scope of the final client, which just will ask for a simple web resource, neither in the scope of the HTML designer that just ask for contents in a simple way regardless of where they are allocated. The only needed tool to build up contents is the use of frames as previously presented.

It is important to notice that this goal could be accomplished just by using other already developed technologies, but by using this technique we can provide these contents with no need of dynamic contents or client side processing.

6 CONCLUSIONS

The main idea of this paper is to propose an alternative way to reuse contents in a cost-effective way. As the only needed changes are on to the software installed in web serves, not in the information itself, and no changes in client side, this technique can spread quickly with nearly no efforts in the short term and support legacy information systems. This proposal deals with is related to partial recover, server side processing of static contents. In this particular situation we could not find any suitable solution in terms of simplicity with short time-to-market.

The main goal in our proposal is to provide web developers with the chance to reuse contents from already existing web contents. This is possible with no changes on the contents themselves. To be able to perform this operation we mix different DOM tree in a single new tree. The undertaken steps to fulfill our proposal are related with information recovery by sending pieces of files on HTTP channels. It, therefore, is possible to build up contents just by using

HTML code depending on the external contents referenced in our source code. Web designers do not need to take care of availability of those resources or even if there is installed on that server an engine for dynamic contents. Thus, it is possible to reuse partially contents with no need to program anything neither on the server where contents are stored nor on the client side. The only requirement is use of the URL where the desired content is located. This idea makes possible to reuse contents from legacy information systems with no extra efforts.

Before hand, the main goal for this initiative was to upgrade web server capabilities and provide a new way to increase interoperability among web contents. This goal was completely achieved: web contents can be shared in a new way rendering a higher integration level. Nevertheless some concerns must be taken into account. A part from legal issues about contents under copyright, some problems about design may arise if contents are provided under poor HTML coding.

ACKNOWLEDGEMENTS

We want to thank “Ministerio de Educacción y Ciencia” and “Xunta de Galicia” for their partial support to this work under grants TIN2007-68125-CO02-02 and "Diseño y desarrollo de un marco semántico para el modelado de servicios en la administración pública. Aplicación a la provisión de servicios frente al ciudadano.” (PGIDIT06PXIB322285PR).

REFERENCES

Free Software Foundation, Inc. (2007). Gnu general public license. Web available. http://www.fsf.org/licenses/gpl.htm

Mozilla (2007). Gecko dom reference. Web available. http://www.mozilla.org/docs/dom/domref/.

Neil Conway (2007). Tornado http server. Web available. http://tornado.sourceforge.net/

Network Working Group (2007). Rfc 1738 - uniform resource locators (url). Web available. http://www.faqs.org/rfcs/rfc1738.html.

The Werken Company (2007). jaxen: universal java xpath engine. Web available. http://jaxen.codehaus.org/.

W3C (2007a). Document object model. Web available. http://www.w3.org/DOM/.

W3C (2007b). Xml path language. Web available. http://www.w3.org/TR/xpath.