JST: TOWARDS A USABLE WEB SITE DEVELOPMENT METHOD

R. Izquierdo[1], F.J. García[2], M. Andrés[2], A. Juan[1], P. Manrubia[3]
[1]*Laboratorio de Tecnologías Orientadas a Objetos (Universidad de Oviedo)*
Calle Calvo Sotelo s/n, 33005, Oviedo (Asturias, Spain)

[2] *Departamento Matemáticas y Computación. Universidad de La Rioja.*
Edificio Vives. Calle Luis de Ulloa, s/n. 26004 Logroño (La Rioja, Spain)

[3]*Comercio Electrónico B2B 2000*
Calle Larrauri 1-C, 48160, Derio (Vizcaya, Spain)

ABSTRACT

Based on the separation between content and presentation in web site development, the authors analyze several problems evidenced during years of work with XML/XSLT technologies. The paper proposes JST, an usable development method focused on reducing the interaction between the HTML designers and the programmers. The paper concludes with a performance evaluation of the technique.

KEYWORDS

JST, JSP, XML, XSLT, usability.

1. INTRODUCTION

When talking of usability [Nielsen2000] in web environments, the interaction of the user with the site always comes to mind. The target is the user experience: try to make it as easy and pleasant as possible, while taking certainly care of the graphical design. But, what about the usability of the web site development process itself? Ought it not be usable in a similar way? If web site usability is essentially focused on finding the actual needs of the user and providing a more efficient way to meet them, we do not understand why these ideas are not applied to the software development of those so usable web pages.

When developing a web site with dynamic content there are two important tasks to achieve. On the one hand it is necessary to develop the logic that will generate on demand the dynamic content, probably accessing to a data base or another data repository and processing it properly according to the business rules that dictate the site. But this is only part of the work. On the other hand, a well designed graphic presentation is necessary for the site to success. These two task clearly need different development skills and, consequently, they should be accomplished by different kinds of developers. However, the mutual interdependence between both teams are a source for new problems that we analyze in this paper.

Today there are a number of technologies that facilitate the generation of dynamic web content: Microsoft's ASP (Active Server Pages), PHP (PHP Hypertext Preprocessor) [PHP] or JSP (JavaServer Pages) [Sun2001] are only examples. Their basic aim is to provide a clean separation between content and presentation. JSP, for example, while making great strides in doing so, falls somewhat short. While JSP has simplified many server-side Java programming details, it still cannot quite make the clean break it was intended to. If you have delivered a large-scale application using JSP you will no doubt agree. JSP mingles content (pure data) with presentation in the same way static HTML does. When working with ASP, PHP or even JSP, usually there is only one document in which both content and graphical design are mixed. Consequently, those abovementioned technologies require too much interaction between the programmer

(who gets the data, knows how to do it) and the graphic designer (who creates the presentation style) [Marchal]. Therefore, these technologies are very powerful and useful but are not ideal.

An alternative that enhances the separation between content and presentation is the use of XML (eXtensible Markup Language) and XSLT (eXtensible Stylesheet Language Transformations) [Kay2001]. The main concept behind a web site using XML is, precisely, the separation of content –and the logic that generates it– and presentation. XSLT allows complete separation of content and style, something everyone writing web applications has been striving for but has been unsuccessful at achieving. This separation is much harder to obtain with HTML, even if CSS2 (Cascading Style Sheets, level 2) or other styling technologies are being used. Additionally, websites using this technique are able to reuse the XML encoded data through other channels (e.g., WAP terminals) or even in the context of Business to Business (B2B) interactions, only changing the applied XSLT. The main advantage of the pair XML/XSLT is, apparently, that the separation of content and presentation allows to create sub-teams in the project. XML solves the interaction problem between programmers and designers, as they do not have to work on the same files.

There is only a number of good programmers world around, and the group of good graphic HTML designers is limited. It is very difficult to find good programmers with graphic design skills. The authors of this paper are programmers who have been working in several non-trivial web site projects using XML/XSLT. The graphic skin of the site is very important for us, and, therefore we usually hire graphic designers. Interdependence has to be as low as possible since the development team is split into different groups working at different places and on different documents.

And here is where the usability matter comes up again. Thanks to our experience working in those projects using XML/XSLT, we can say that the so-called separation between content and presentation resulting from the use of XML leads to couplings in other aspects of the development method. These couplings are even more harmful for the project than those that are expected to be avoided. We devote the paper to argue this statement and, finally, we propose an alternative and usable development method.

The paper is organized as follows. Section 2, summarizes and shows some of the problems associated to currently used web development technologies, such as the pair XML/XSLT or JSP. Section 3 outlines the requirements that, in our view, should be fulfilled by a usable web development method. In section 4 we propose a usable development method that favors not only the separation between content and presentation, but also between the programming and HTML design teams. Despite this paper is focused on usability arguments, a performance evaluation is included in section 5, to support the adoption of the JST method.

2. EVIDENCED PROBLEMS IN WEB DEVELOPMENT PROJECTS

In the last three years, we have developed several web sites in which XML/XSLT was the selected technology to generate dynamic content. Below, we present the problems encountered in these projects.

Effort duplication

To develop a XSLT sheet, the HTML designers must work twice. It is very difficult for them to create the graphic view of the page and, at the same time, take care of the XSLT intricacies. Thus, they prefer to develop the HTML prototype using their favorite WYSIWYG editing tool, with the look & feel they have designed, and later, turn it into the final XSLT version. It is harder, indeed, when the page must be changed. The designer must change both versions (HTML and XSLT) or just modify only the XSLT, desynchronizing, consequently, the HTML page (which could be necessary for a look & feel refactoring).

Make HTML designers' work complicated

It is a fact that HTML designers are used to visual composition tools useful to focus on the aesthetics details of each page (this is the value they provide). By dragging and dropping they have an immediate feedback of the resulting visual effect. This is the most effective and productive way for them to work. Any deviation from this scenario should be comprehensively justified. Nevertheless, the use of XML/XSLT forces them to resort to additional and more complicated tools (XML and XSL parsers, test batteries, etc). They have a new environment in which the pages must be processed (the equivalent to "compile" for programmers). Although

programmers do not perceive this as a problem (since they are used to doing it), designers cannot understand the need for a change that does not improve its work. Moreover, the page (XSLT) cannot be tested without a XML document to be transformed. Thus, they have to wait for programmers to provide the XML content, or they must edit by themselves toy-XML documents, managing their versions and correspondences with their XSLT documents. Similar remarks can be made on the use of JSP. On the one hand, designers need new tools to insert Java scriptlets, and on the other hand, they need knowledge about the classes on which the JSP is going to run. New tools and more work to get the same look & feel.

Training problems for the designers

It is difficult to find professionals with artistic skills and XML/XSLT knowledge. They can be trained in the technologies, but problems persist. Designers never become experts on the new technology (they are not used to building algorithms, so they only grasp trivial transformations). XSLT forces to incorporate programming (loops, etc), which exceeds the simplicity promised to the designer. They must assume even the difficulty of debugging XSLT. The conclusion is that XSLT is easy but only from the point of view of programmers. It is a great obstacle for HTML designers. They do not understand the need for a more sophisticated technology when they have already created the look & feel with HTML (and perhaps they are right).

Make programmers' work complicated

Obviously, a source XML document is needed to apply a XSLT style sheet. This is very inefficient and makes more difficult the programmers' work. He has to finish each service he develops with an extra step to generate the XML document representing the page to be displayed. Imagine a user buying in our site. During the session the server must keep in memory an object model of the ongoing order. In every page transition the object model has to be transformed to XML so that the XSLT can be applied. This is very inefficient.

Another option is to manipulate DOM (Document Object Model) objects [DOM] instead of user classes directly in the memory. This implies using much more memory and code to manipulate, following the example, the order model. The model classes for this project (order, customer, etc) should be replaced with DOM classes (Text, Node, etc) losing the advantages of an object-oriented approach.

Separation between content and presentation?

The so-called separation between content and presentation favored by XML/XSLT, despite being higher than with other technologies, is far from being perfect. In several cases, in order to make possible the presentation of the web page, it is necessary to populate the content with data related to the navigation or the operations that can be performed through the page (e.g., data to make up combos or lists). Content is no longer mere data, but content-data plus presentation-data.

Higher dependency between programmers and HTML designers

And last, but not less important, designers need to know the XML's DTD in order to develop the XSLT. Programmers must explain it to designers, who cannot begin work on XSLT until the former have delivered the DTD. We are not only "underusing" a design resource, but also demanding an extra definition effort to the programmer in early stages of the project. This leads to inaccurate descriptions and immature decisions, in short, work that will be redone. As a result both teams will have wasted their time. And, what about the DTD changes? The programmer must inform the designer of the changes. This is the real problem, since changes in data structures are inevitable and extremely frequent during development [Beck2000].

It is said that XML/XSLT brings about the independence between presentation and data as they are located in different files. This is not enough to achieve actual independence. Only in a scenario where designers are not affected by XML structure changes, actual independence can be achieved. Designers have a strong dependency of how programmers model the data, something that they even should not know at all.
Programmers are not independent from the presentation either. Frequently, HTML designers asks programmers to include new elements in the XML document to simplify the XSLT transformation, which are

otherwise too complicated or even impossible. And this is a problem, because each porting to another device could require its own XML tag addition.

3. REQUIREMENTS FOR A USABLE WEB DEVELOPMENT METHOD

Web usability is focused on finding the actual needs of the user and providing him with the more efficient way to meet them. It would be necessary to do the same analysis for the development process, but from the point of view of its users: programmers and designers. Our aim is to study the tasks they must perform and offer them a simple process that enables them to tackle these tasks with the minimum steps and tools.

In short, we have looked for a usable process to develop usable web sites. In this line, we have established the indispensable requirements that, in our view, a development process must have to be considered as usable. We will describe the four more main requirements among a total of ten.

Requirement 1. Reversing the Information Flow

So far, programmers explain to designers either the DTD (in XSLT) or the class model (as in JSP or other technologies) where the latter must obtain the dynamic data. The communication flow should be just in the opposite direction: designers should impose the data they need in the page, and programmers should provide it. Designers do not care about how the data is modeled and therefore data changes should not affect to their work. Programmers, who know the model they have created, will know immediately where to fetch the data that the page needs.

Requirement 2. Minimise communications between designer and programmer

Indeed, the reversal of the information flow reduces substantially the communication between designers and programmers. However the independence of programmers must also be considered. Their work should not be affected by any change in the presentation that does not require new data from the application model. If, despite this level of isolation, communication is still necessary, it must be as simple as possible.

Requirement 3. Only the HTML designer can choose his working tools

Designers must work on the look & feel of the site. The process will be usable if it allows them to concentrate on their job, without disturbing them with collateral tasks. They should not be compelled to use specific tools imposed by the last technology chosen by programmers. Designers should view the page in edition without using additional tools or external preprocessors. Only an HTML edition tool is required. This is important: a preprocessor should not be necessary to view the page at design time; it could be necessary to view the page with real data. On the other hand, the process should provide designers with a testing platform where they can introduce and remove test data with the minimum effort.

Requirement 4. Avoid new proprietary script languages

Designers know HTML, Javascript and, perhaps, Flash. They should not, nor need, learn any new language. Otherwise it would be necessary to devote time to their training and they will need to include new tools in their development environment.

4. JST: USABILITY ADDED TO THE WEB DEVELOPMENT METHOD

Currently, we are using a simpler process for web development called JST based on the use of JST templates. The following example shows a JST template and how it is displayed in a browser. Note that no extra tool has been used to display the page. You can see the resulting web page using only simple HTML browser.

```
<html>
   <head>   <title>JST example</title>
   <script>
     function Item(name, price) {
       this.name = name;
       this.price = price;
     }
     function p(arg) { document.write(arg) }
   </script>

   <!------------------------------------------------------------------------>
   <Data>
   <script>
     name = "Peter"

     items = new Array()
     items[0] = new Item("Umbrella",1)
     items[1] = new Item("Suitcase", 18)
   </script>
   </Data>
   <!------------------------------------------------------------------------>

   </head>
   <body>
      <p>My name is <b> <script>p(name)</script> </b></p>
      Layout and data aren´t merged. Below are the static markup tags.
      Table contents are on Data section and are written dinamically.<br/>
      <table border="1">
        <tr>
          <td><script>p(items[0].name)</script></td>
          <td><script>p(items[0].price)</script></td>
        </tr>
        <tr>
          <td><script>p(items[1].name)</script></td>
          <td><script>p(items[1].price)</script></td>
        </tr>
      </table>
   </body>
</html>
```

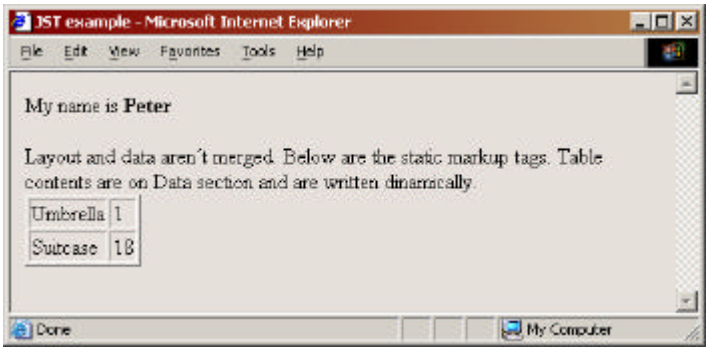


Figure 1. Sample JST document displayed on an internet browser

If the size of the table is dynamic the table can be generated using a Javascript for loop, which iterates through the data array. Within the for loop the `p()` function is called for each element of the array.

From the point of view of designers, working with JST involves creating the HTML page by focusing only on the navigation and look & feel. Designers must identify the dynamic part of the page (in the example a name and a table of items). Then, they must define the variables within the `<data>` tag, giving them a name and assigning a test value to them. Wherever they want to insert a data value, they write a call to the `p()` function with the name of the required data, or a more complex expression, as an argument.

Designer can perform this task using their favorite HTML editor since it is simply HTML with a minimum subset of Javascript. The `<data>` tag does not interfere (in HTML, unknown tags are just ignored,

though their content is processed). They do not need any extra tool, and they are familiar with the working environment. At this moment, designers have already finished their work. The data structure created in the `<data>` section is known as *designers' model*. There, designers simply indicate the data the page needs, independently of where the real data are and how they must be recovered.

Now the work of programmers begins. The data section in the JST template shows the dynamic data that the page needs to be displayed, reducing the communication flow between designers and programmers. Only semantics are needed (which can be added in the form of comments). Programmer must implement the code executed when a request for this page is received. This code will fetch the final data from an object model, XML, or DBMS (the *programmers model*) and will generate a string with Javascript code that will replace the `<data>` tag. Lest assume that the actual name is "Mary" and there are three items: a leather belt, a handbag and an umbrella. Using the JST processor[1], the page sent to the user will be just as before, except for the `<data>` section:

```
<script>
  name = "Mary"
  items = new Array()
  items[0] = new Item("Leather belt ",10)
  items[1] = new Item("handbag ", 79)
  items[2] = new Item("Umbrella",10)
</script>
```

Thus, if the JST template is loaded directly in a browser from the file system the test data are presented. But if the page is loaded by means of a web server (e.g., with a Servlet using the JST processor) the page is viewed with real data. Data substitution takes place on the fly during the request.

Double Model Pattern

The JST architecture is based on a variation of the MVC pattern [Buschmann96] denominated *double model pattern*. In the architecture of MVC, the view knows about the model (it must know what method must be invoked to get the model data). Therefore changes in the model cause changes in the views. This is just the problem with XML+XSLT. Whenever the model changes (DTD), it is necessary to modify the views (XSLT).

The double model pattern is necessary in situations where the cost of modifying views is too high. This pattern has two models: one for programmer and another for designers. There is also a new component, the *translator*, which copies data from the first model to the second (only in this direction).

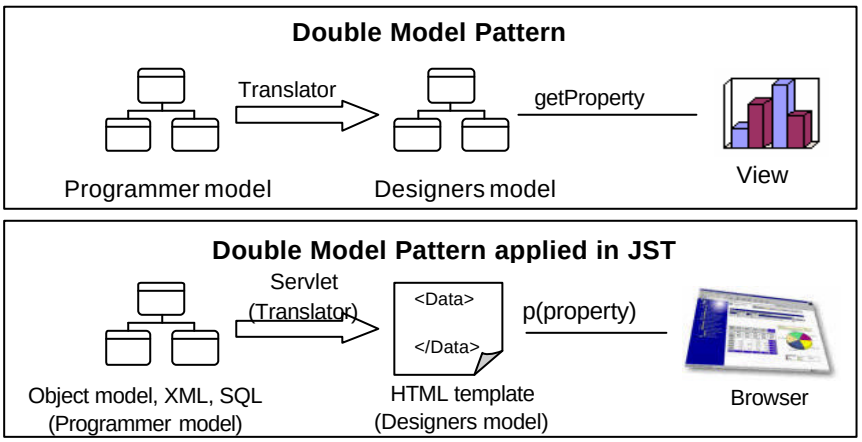


Figure 2. Double model pattern structure

[1] The JST preprocessor is extremely lightweight; just string substitution. Parsing XML, XSLT and applying the transformation is not comparable in terms of speed and used memory.

The views only know about designer's model. They become independent of programmer's model (which is subject to continuous refactoring during development [Fowler1999]). When programmers change the model, they only have to modify the translator in order to import the data from the new model.

One moment, … "changes that cause changes" … What is the difference between changing the translator and changing the views as before? Note that, with the new pattern, it is the same role who writes all the changes, therefore eliminating communication. Programmers who have changed the model will know how to change the translators. Designer will not be aware of the change. For sure, the amount of work to be done is the same as before but, at least, the propagation of this work between different roles has been avoided.

Other Frameworks Evaluated

We have evaluated other web technologies (JSP, Velocity [Velocity], WebMacro [WebMacro] and FreeMaker [FreeMaker]). Nevertheless, none of them fulfilled our requirements:

- All of them impose a proprietary script language or a set of tags that, in fact, becomes another way of putting logic and code into the HTML.
- Some of them are based on reflection. Therefore, designers have to know the classes of the model and their methods.
- They require a configured web server running. This configuration is not trivial in most cases.

In short, these technologies are oriented to facilitate the HTML construction phase to programmers. They are not intended for helping graphic designers.

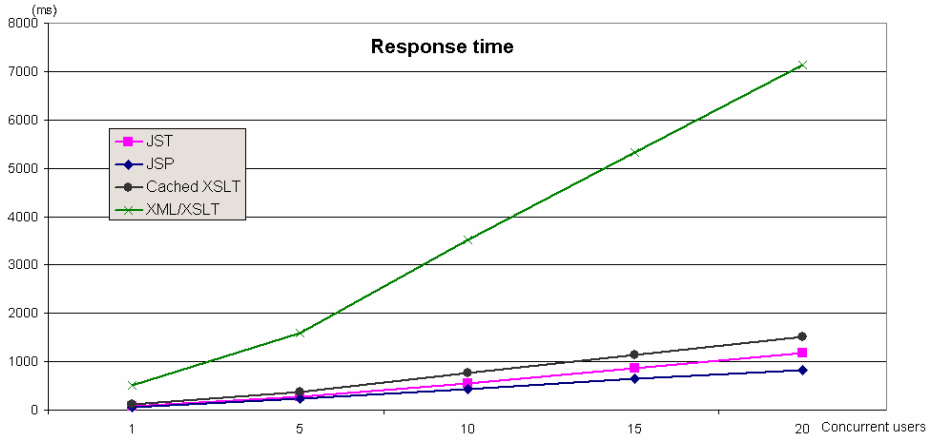
5. PERFORMANCE EVALUATION

Although the aim of the paper is to discuss problems related to the use of XML/XSLT, we include a section on performance so that the cost of adoption of JST can be evaluated.

We have analyzed the response time, bandwidth and memory needed to serve the result page of a search in one of our developments. We have run four experiments, each based on a different technology: a JSP version, a XML/XSLT version, an improvement over the latter consisting of caching the XSLT transformer, and, finally the JST version. The number of concurrent users has been varied from 1 to 20. The concrete features of the experiment's platform are intentionally omitted, since we are not interested in the specific figures, but in the comparative behavior of these technologies.

Response time

The JST response time is better compared to any other XSLT based version. Only JSP beats JST in this performance factor, but considering that the difference is very small and that the simplicity of use of JST is higher, we think that this factor cannot decide us in favor of JSP.



Occupied bandwidth

The occupied bandwidth is directly related to the size of the served pages. In this case, JST beats any other technology. Consider that, in order to display a page, a JST page contains only the table data and a Javascript loop to expand the table in the browser of the client. The rest of technologies send the expanded HTML table. In the example, the JST page is 35% lighter than the others.

Used memory

The size of the transformer and the DOM tree representing the XML content used in a XSLT transformation is very far from the amount of memory necessary to process a JST template or a JSP.

6. CONCLUSION

The use of JST has the following advantages:

- JST proposes a usable development process focused on solving tasks in a simple and effective way.
- Efficiency in the delivery of web pages since the processing of the template in the server is trivial.
- JST can be used to generate portal prototypes without writing code. The pages do not require a processor to be tested (test data is used in this case). Designers can navigate and test the site locally.
- JST is totally independent from the programming language used in the server. Any programming language can be used to replace the section `<data>` in the template.
- The learning curve for JST is insignificant. Any HTML designer can create JST templates and any programmer can insert the actual data in the template.
- JST really separates the view from the application model. Isolation is so high that, in fact, the view can be displayed without a model (which results in a prototype).
- Each member of the development team does what he can do: programmers, program, and HTML designers, design. The interaction between them is dramatically reduced.

ACKNOWLEDGEMENT

This work has been partially supported by the Ministerio de Ciencia y Tecnología of Spain (project TIC2002-01626) and by the Comunidad Autónoma de La Rioja (project ACPI-2002/06).

REFERENCES

[Beck2000] Beck, K., 2000. Extreme Programming. Addison-Wesley. Boston, USA.

[Buschmann96] Buschmann, F. et al, 1996. Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. John Wiley & Sons. Indianapolis, USA.

[DOM] Document Object Model. http://www.w3.org/DOM/.

[Fowler1999] Fowler, M., 1999. Refactoring. Improving the Design of Existing Code. Addison-Wesley. Boston, USA.

[FreeMaker] FreeMaker. Home Page: http://freemarker.sourceforge.net/

[Kay2001] Kay, M., 2001. XSLT Programmer's Reference 2nd Edition. Wrox Press Inc.

[Marchal] Marchal, B. Servlet for Programming Teams. http://developer.netscape.com/viewsource/marchal_xml.htm

[Nielsen2000] Nielsen, J., 2000. Designing Web Usability. New Riders Publishing. Indianapolis, USA.

[PHP] PHP. Home Page: http://www.php.net

[Sun2001] Sun Microsystems, 2001. Java Server Pages™ Specification, Version 1.2. Sun Microsystems, Palo Alto, USA

[Velocity] Velocity. Home Page: http://jakarta.apache.org/velocity

[WebMacro] WebMacro. Home Page: http://www.webmacro.org/

[XML] Bray, T. et al (eds), 2000. Extensible Markup Language (XML) 1.0 (Second edition). http://www.w3.org/TR/REC-xml/.