Experiences in Web site Development with Multidisciplinary Teams. From XML to JST

Raúl Izquierdo[1], Aquilino Juan[1], Benjamín López[1], Ricardo Devis[2], Juan Manuel Cueva[1], César F. Acebal[1]

[1]Laboratory of Object Oriented Technology. University of Oviedo. Spain
{ric, aquilino, benja, acebal, cueva}@lsi.uniovi.es
[2]Ricardo Devis & Asociados. Madrid, Spain
devis@ieee.org

Abstract. XML and XSLT apparently offer many advantages in Web site development. However after using them in several real projects we have found many disadvantages that yield a development process neither so productive nor easy. We propose a simple development process centered on reducing unnecessary interaction between programmers and graphic designers so they can focus on their fundamental tasks.

Introduction

The most usual criticism to technologies like Servlets, PHP or JSP is the lack of separation between data and presentation[1]. Using XML/XSLT programmers generate XML and designers write XSLT style sheets solving the separation problem because they are centered on different tasks.
In *CE B2B2000* (http://www.ceb2b2000.com) we have been using XML in several real projects (from B2B integration to portals like http//www.saunierduval.es, http://www.caloryfrio.com and http://www.conaif.com). Due to the great importance that visual appearance has on the web CE B2B usually works with external graphic design companies. Therefore independence between programmers and graphic designers it's a needed requirement since the development team is composed by people from different companies working in different places.

Problems in XSLT based Web Development

Complicating Graphical Designer Work

The graphic design team works twice. It is easier for them to work with a WYSIWYG tool than to describe in XSLT text what they want to see. Therefore they usually create first a HTML page with the look & feel they want and they must repeat

In ICWE 2003, LNCS 2722.
http://springerlink.metapress.com/content/0wbn77tpdcxrecny/?p=c56e0f9c4d044911acb0c2e82b359fa3&pi=85

their work writing the XSLT document. Moreover the synchronization between both versions must be kept during development.
Other problem is that designers use to rely heavily on visual composition tools that allows them to focus in the aesthetics details. With XML/XSLT they will have to add more complicated tools (XML and XSLT parsers, test batteries, etc). Therefore now they have a new environment in which the pages must be "*compiled*". Although this doesn't may be perceived as a problem for programmers (because they use this kind of processing tools) designers cannot appreciate the need for a change that doesn't improve their work in the aesthetical area.

Training Problems for the Designers

When working with XSLT it is difficult to find professional art designers with XML and XSLT knowledge. Time can be dedicated to their training but problems persist:

- Designers never become experts in new technologies. They are not trained to build algorithms so they only grasp trivial transformations.
- XSLT finally must incorporate programming which exceeds the *simplicity* that was promised to designers. They must assume the difficulty of debugging XSLT.
- The project depends on the XSLT trained designers, so teams can't be replaced when the results are not those expected.

The conclusion is that XSLT is easy, but only from the point of view of programmers.

Complicating the Programmer Work

To apply a XSLT style sheet a source XML document is needed. Suppose the implementation of a shopping site. During a session the server will keep in memory an object model of the ongoing order. In every page transition the object model has to be transformed to XML to be presented in HTML. Obviously this is very inefficient.
Another option is to directly manipulate DOM objects in memory instead of user classes. But this option uses much more memory and requires more code to manipulate the order model. The right class model for this project (order, customer, etc) has to be replaced with DOM classes (Text, Node, etc) loosing the advantages of object oriented modeling.

Greater Dependency between programmer and designer

The designer needs to know the XML tags to transform. This implies that:

- The designer cannot begin to work until the programmer has finished the DTD.
- Whenever a DTD changes, the programmer and the designer must have a meeting. This was the *greatest problem* we found in our projects, since changes in data structures are something inevitable and extremely frequent during development [3].

It's said that XML/XSLT separate presentation and data because they are described in different files. But true independence would mean that designers were not affected by

XML structure variations. Therefore *designers have a strong dependency on how programmers model data* (which is something that they even should not know).

JST: A Simple Web Site Construction Method

Web usability is focused on finding the user needs and provide them the most direct solution[2]. In the development process programmers and designers are the *users*. It is necessary to analyze their tasks and provide a process *with minimum steps and tools*.
Now we are using a more simple process for web development denominated JST. A JST template is just an HTML file that even can be opened directly from the file system by a browser:

```
<html> <head>
<script>
function p(arg) { document.write(arg) }
</script> </head>
<Data> <script>
name = "Raul"
</script> </Data>
<body>
My name is: <script> p(name) </script>
</body> </html>
```

From the point of view of the designer working with JST just consists of:
1. To isolate the dynamic parts of the page (in this case a person’s name) and to assign them a default value in the data tag.
2. The function *p* is used to insert the dynamic data into the static markup.

The <*data*> tag content is known as the *designer’s model*. When editing the page the designer simply indicates what he needs independently of what the real data is and where it is located.
Now the work of the programmer begins:
1. The *designers model* in the HTML template shows the dynamic values needed by the page, minimizing communication between designers and programmers.
2. The programmer must implement the code that extracts the real data (from an object model, SGBD, etc) and generate a string with the right dynamic content values.

So if the JST template is opened in a browser directly from the file system the default test data is presented. But if the page is requested through a Web server a simple servlet can change the data tag content on the fly.

Double Model Pattern

The JST architecture is based on a MVC[5] variation named *double model pattern*. In the MVC architecture a *view* directly use the model methods. So changes in the model interface produces changes in the views. This is just the problem with XML+XSLT: whenever the model changes (XML structure) it is necessary to modify the views (XSLT).

The *double model* pattern is used when modifying views has a high cost. It is based on using two models: the programmer's model and the designer's model. A new component, the *translator*, copies data from the first to the second model. Views only know the designer's model so they become independent of the programmer's model (which is subject of a continuous refactoring during development). Whenever the programmer changes its model he has to modify the translator to feed the designer's model with the right data.
But when the programmer's model changes, what is the difference between changing the translator and changing the views? The difference is that by using the new pattern only programmers carry out the required changes eliminating communication. Programmers have changed the model and also know how to change the translator. Designers do not need to be aware of the change. The dangerous dissemination of work among different roles has been avoided.

Conclusions on JST

The main properties of JST are:

- The processing of the template in the server is trivial and efficient (just a string substitution: the data tag content).
- Language independence. Any programming language can replace the data tag.
- JST *really* separates the view and the model. Independence is so high that views can even work *without a model* (deriving in a prototype which uses the default data tag content).

References

1. Benoit Marchal. Servlet for Programming Teams. http://developer.netscape.com/viewsource/marchal_xml.htm
2. Jakob Nielsen. Designing Web Usability. New Raiders Publishing. 2000
3. Kent Beck. Extreme Programming. Addison-Wesley. 2000
4. Wei Meng Lee, Ngee Ann Polytechnic, Singapore. Tailoring Content Using XML and XSL. http://www.vbxml.com/wap/articles/wap_xml_xsl/default.asp
5. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. "Pattern-Oriented Software Architecture, Volume 1: A System of Patterns". John Wiley & Son. 1996