A Double-Model Approach to Achieve Effective Model-View Separation in Template Based Web Applications

Francisco J. García[1], Raúl Izquierdo Castanedo[2], and Aquilino A. Juan Fuente[2]

[1] Departamento de Matemáticas y Computación (Universidad de La Rioja)
C/ Luis de Ulloa s/n
26004, Logroño (La Rioja), Spain
francisco.garcia@unirioja.es
[2] OOTLab: Laboratory of Object Oriented Technology (Universidad de Oviedo)
C/ Calvo Sotelo s/n
33005, Oviedo (Asturias), Spain
{raul, aajuan}@uniovi.es

Abstract. Several works [20,22] have tried to enforce strict isolation between the model and the view in template based web applications by restricting the computing possibilities of the used templates. From the point of view of graphic designers this is a limitation that may make their work difficult. Besides, in this paper we state that this claimed strict isolation is impossible to achieve in practice for HTML template systems. We propose another approach to study and to attain an effective separation between model and view that does not necessarily restrict the expressive power of the template: the double-model approach. Finally we present an implementation of this approach in a renewed template system called JST2.

Keywords: template systems, Web applications, software architectures, MVC.

1 Introduction

When developing a web site with dynamic content, instead of writing the HTML code directly to the client's browser, developers have tended to make the most of some kind of *template system* (language and engine). Doing so, long and tedious code writing tasks can be alleviated and, a priori, among other undeniable advantages, it promotes the separation between presentation and business logic and it increases the clarity and maintainability of the application. The problem is to decide which is the more suitable template system, and, what are the more helpful features expected to be present in the template system. The answer, as usual, depends on the situation. But simplifying we can find two non exclusive tendencies: to provide the template system with a complex set of aids to reduce the programming effort, or to provide a reduced set of functional features, but in turn, contribute to force the separation between presentation and business logic.

The web developer's community has released plenty of template systems (at the time of writing, a simple search at SourceForge.net of the term "template engine" produces 214 results). We could define a template, in the context of web applications,

L. Baresi, P. Fraternali, and G.-J. Houben (Eds.): ICWE 2007, LNCS 4607, pp. 442–456, 2007.

as an HTML document in which several placeholders have been inserted. At execution time, the template engine replaces these placeholders by the actual dynamic content taken from the application. The eternal promise of template systems is that the template meets the rest of the application only at the end of the development. That is, in theory, graphic designers work on the templates completely independent from application programmers. But in practice, this is not true for most template systems.

The way the template placeholders are replaced by actual content has been evolving as the problems related to code inclusion in the template have been detected. In fact, the first template systems we can refer to (JSP [26], ASP or PHP [23]) allow the inclusion of code in the template. This code is programmed in the same language as the application, and it can gain access to the application in order to obtain the actual data necessary to generate the dynamic content. The possibility of placing code in the templates is a temptation that developers can hardly overcome. It is a back door for the inclusion of business logic in the template, a practice that involves evident coupling problems between business logic and presentation. These problems have been already evidenced, e.g. in relation to JSP, in the classic reference of Hunter [15].

The coupling problems can also be studied from the point of view of the relationship maintained between both parts of a typical web application development team: the graphic designers and the programmers. The use of this kind of template systems makes this relationship very close, and it is characterized by the constant flow of information from programmers to designers, who need to know what the application is like in order to program the access to it in the template. This need of communication have been already treated by us [16,17] in the context of XML-XSLT [30], but can be perfectly extrapolated to other environments.

A second step in the template systems evolution leads us to systems in which the inclusion of native code in the template is not possible. This fact eliminates the chance of accessing the application business logic, but does create other subtle coupling problems. Often, these template systems force the designers to learn a completely new, but equivalent, pseudo programming language in order to develop the template documents: e.g.: XSLT [30], FreeMarker [11], WebMacro or Velocity [31,3], Zope Page Templates [34] or Smarty [25]. Other systems, in order to be used, make necessary the development of special classes or adapters (Tapestry [2] or Tea). Designers are also compelled to use development tools that they are not familiar with. In our opinion one of the main objectives of any template system must be to allow designers to use the tools they are used to using. That is, it should not be necessary to use any tool apart from HTML (plus JavaScript, and perhaps Flash). This is the reason why we find interesting the *attribute template languages* like Zope Page Templates (ZPT) [34], Tapestry [2] or the one we present in this paper, JST2. In them the template directives are inserted in the host HTML document in the form of tag attributes. This feature carries with it one of the more useful aids that a designer may expect: the *previewability*, that is, the ability to see how the page will look like without using neither the template engine, nor tools different from the HTML editor or browser.

The fact is that, despite the efforts, the coupling between presentation and business logic, or *view* and *model* using MVC terminology [7], still remains. Very few works have been devoted to analyzing the sense and causes of this coupling in order to attack them at their root. A good introductory reference can be [10], in which several

template systems are analyzed. There is only one paper that formalizes the model and view separation [20]. In our opinion, the author forgets to consider one factor, JavaScript, that makes his work hardly applicable. This leads us to provide another way of facing the model-view separation problem by means of the use of a double-model, one for the view and another for the application. This can be considered the main contribution of this paper.

Finally, in the last two years a new breaking idea has emerged in the web development world, leveraging the development of Rich Internet Applications: AJAX [12]. Using already existing technologies in conjunction, AJAX has persuaded web designers to begin thinking of enriching their designs with features more typical of desktop application, even though this decision implies facing the difficulties derived from the use of the cryptic JavaScript DOM API or the need of learning how to use AJAX toolkits such as DoJo [8] or GWT [13], and so on. AJAX drove us to think of the possibility of processing templates in the user's browser. As we will show in this paper this strategy is the key to implement the double-model approach we propose in our renewed template system: JST2. The impossibility of strictly isolating the view and the model, the pernicious demands for the graphic designer required by the existing template systems, and the irruption of AJAX, with its corresponding processing power in the client side, are all forces that concur on JST2 as double-model based template system, decoupled, processed in the client, previewable and familiar to the designer.

Once has been presented the motivation and background, the remainder of the article is organized as follows. Section 2 deals with the related work. Section 3 presents the double-model approach. Section 4 is devoted to JST2. And to conclude, section 5 collects several conclusions and future work lines.

2 Related Work

There is plenty of documentation about template systems in the Internet. As for more academic research papers, the first reference we find is [19], where *TML* is presented, a language based on XML tags added to an HTML document. It relies on *TRiX*, a framework for TML templates processing, being its extensibility its main advantage. *Mixer* is presented in [32], a simple template system for servlet based applications that superficially analyzes the separation problem. Another reference is [1], where *PageGen*, an ASP based template system, is described. In this case the tight coupling of PageGen to its Data/Control database may prove restrictive. In [14] the convenience of using "standard templates", i.e. templates in which no non-HTML tags are allowed, is discussed. *STRUDEL* [9], is a framework for intensive data based Web application that includes its own template language, which can be criticized because it allows the direct inclusion of java code in the templates. Other interesting papers are [6,5], in which the <*bigwig*> language for Web applications is presented. It includes the *DynDoc* sublanguage to specify HTML template documents, and the *DynDocDag* data structure [5], that allows the representation of the template document in a way that allows the reconstruction of the document in the client

browser using a JavaScript processor. *DynDocDag* was designed to improve the caching of DynDoc documents, but it does not pay attention to the separation concern.

Related work about model-view separation. Nevertheless, when we look for bibliography related to the analysis of the coupling between the model and the view of a web application, only one work stands out. Parr [20] provides a definition for *strict separation* between model and view in template engines, as well as a definition of template and a classification of the different types a template can belong to. From a more practical point of view, he formulates a set of five rules that a template engine must follow in order to enforce *strict separation* and gives an *entanglement index* that can be used when evaluating template engines.

We can briefly summarize the Parr contribution enumerating his rules: (1) "the view can not modify the model", (2) "the view can not perform computations upon dependent data values", (3) "the view cannot compare dependent data values", (4) "the view cannot make data type assumptions", and (5) "data from the model must not contain display or layout information". The entanglement index "is the number of separation rules that a template may violate", being the minimum 1, since there is no way to prevent data of the model from containing display or layout information.

Parr warns of template engines that encourage separation and lays the responsibility of preserving it on the hands of well-intentioned developers. The essence of Parr's work is to give the rules to enforce, rather than encourage, the separation, forbidding, e.g., that views can perform computations of any type upon model data values. E.g., a view must not either compute book sale prices as "`$price*0.9`" (rule 2), or compare a price with a certain limit like in "`$price<25`" (rule 3), or index an array of names with the value of another variable, that is supposedly an integer, like in "`$names[$id]`" (rule 4).

Despite Parr's rules being well stated, we think that several of them can not be imposed upon most template system that produce HTML as output. Parr does not take into account the possibility of using JavaScript in the HTML documents. It's impossible to prevent template expressions to be inserted in the middle of a JavaScript code, like in:

```
<html><body>The final book price is
<script>document.write($precio$*0.9)</script> &euro;
</body></html>
```

The previous code is a StringTemplate [21] sample that, once processed and taking 30 as the price variable value, results in the following HTML output:

```
<html><body>The final book price is
<script>document.write(30*0.9)</script> &euro;
</body></html>
```

This fact may cause the violation of rules 2 and 3, and it makes the minimum value for Parr's entanglement index be 3. This leads us to state that it is impossible to enforce strict model-view separation while using HTML template systems. In the next section we show another approach to the analysis of the separation problem which, if applied, makes the entanglement index be at most three

3 The Double-Model Approach

The essence of model-view separation is that it pursues the protection of both sides of the application from changes in the other. If separation is not provided, changes in one part draw the other to be modified. If separation is provided each part can live isolated, even be developed independently.

Once we have stated that strict separation as Parr defined is impossible to achieve in HTML templates, we analyze another solution to obtain a weaker, but effective, degree of isolation in HTML templates.

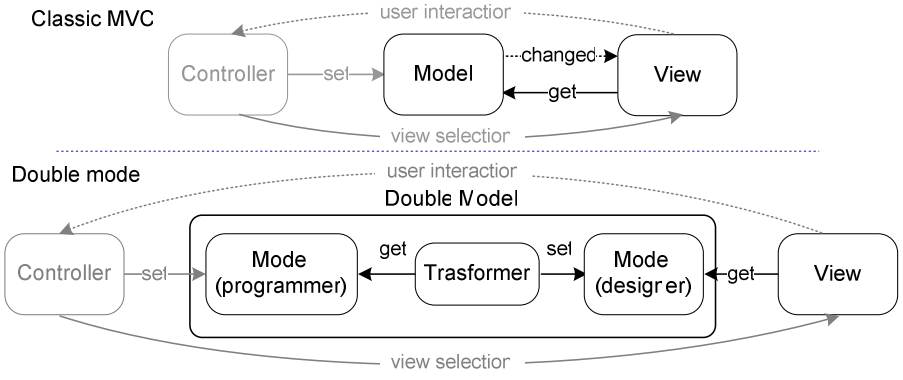

Fig. 1. MVC versus double-model approach

The foundation of the double-model is that each part of the application has a different model. On one hand, the view uses its private model that holds the necessary data to support the functionality of the page. The view is therefore completely matched up to its model. In fact the view can only use its model. No direct reference to the application logic is allowed, not even read-only accesses. HTML designers make up this model as a set of attributes (name-value pairs) that can be simple or compound, and populate it with *test values*. These values can be used during development in order to see the final appearance of the page. This helps achieving the previously mentioned previewability. Compound values may be structured according to data structures that clarify the design. We call this model, *designer's model*. It is developed during the graphical design process based on the functional requirements of the view. Once all these requirements are fulfilled, the model can be considered stable and it not need to change. E.g.: if the page shows a set of books of a certain shopping basket, the model will define data structures to hold the data of the books of the basket, as well as the user's identification and, perhaps, the current date.

It is important to say that in order to change the appearance of the view the designer's model does not need to change, unless a new functional requirement appears that demands new attributes in the model. It does not need to, but it can be changed. The subtle difference is that the decision whether to change it or not is only

taken by the one who created it: the graphic designer.. It is not a change caused by another change in the application, as it happens in other template systems.

Views are developed apart from the other model, which we call *programmer's model*, because it is developed by programmers. This is the classical application model, i.e., the M in the MVC acronym, usually implemented as a set of classes persisted on a data base, or even without classes, i.e. only a data base schema. The programmer's model can freely evolve as needed, surely subject to continuous refactoring during development. But this evolution does not affect the designer's model, which remains isolated in the view.

Obviously both models must be integrated so that the views can show actual information in their pages at execution time. This task is performed by the programmer, who is the one who knows the actual application model and how to extract actual data from it. This integration consists of supplying the view with the actual data from the programmer's model *re-structured* as the designer's model mandates. It is important to say that the view must be passive with respect to this data load, that is, the actual data must be pushed into the template, instead of the template pulling the data from the model. This requirement in favour of the *push strategy* was already argued as necessary by Parr in [20], and we adopt it as necessary also.

At this point of the presentation, the versed reader could think that almost any template system that follows the push strategy, implements the double-model approach. Consider for example Fremmarker [11] (with its DataModel), WebMacro or Velocity [31,3], Smarty [25], StringTemplates [21] or HTML::Template [27]. All of them have templates with attributes that are processed pushing values taken from the application model.

Push strategy is not enough. So, what is the key aspect that makes us say that we are in front of a double-model or not? This key is in an above paragraph: "… supplying the view with the actual data from the programmer's model *re-structured* as the designer's model mandates". This *transformation* between models is mandatory.

All the previously referenced template systems do not follow the double-model approach because they allow the designer's model to adopt data structures from the programmer's model. That is, they allow the programmer to suggest to the designer what the most appropriate designer's model is so that the models integration does not require transformation. This allows the programmer to push objects or data structures directly from its model into the templates, which are irremediably entangled with the application. Obviously these template systems encourage developers to use completely different models, but, the fact is that the back door is open, and that under deadline pressure, developers tend to save work using these kinds of resorts. Of course, data structures in both models may coincide (a book has a title and an author anywhere), but this coincidence cannot be used to simplify the transformation, which is always compulsory. And this must be taken not as a drawback, but as a model-view coupling vaccine.

Note that if the programmer changes its programmer's model, he is also responsible for changing the code that makes the models adaptation, but in no case the programmer's model modification is propagated to the view.

As a consequence of double-model approach, templates developed following this can be used in different applications, with the sole requirement that they belong to the same domain. This is a good test that allows us to state that double-model favours the separation between model and view. In fact, isolation is so high that the view can be displayed without a model (which results in a prototype).

3.1 Application Architecture for the Double-Model Approach (MVC+mT)

The transformation of data taken from the programmer's model into the format specified by the designer's model is performed in a new architectural component of the application, which is very close to the controller (the C in MVC). We call this component *transformer*.

The presence of the transformer modifies the typical control flow in MVC (or its adaptation to web applications, Model-2 [24]). We will refer to the architecture with the MVC+mT acronym. The control flow is represented in Fig. 2. The Transformer (T) is activated (3 in Fig.2) once the Controller has updated the Model (1, 2). Then the transformer loads (4) the suitable template (View), takes (5) the necessary data from the programmer's Model (M), and performs (6) the models transformation (the designer's model is the 'm'). Then the transformer pushes (7) the resulting data into the template, which is sent to the client (8).

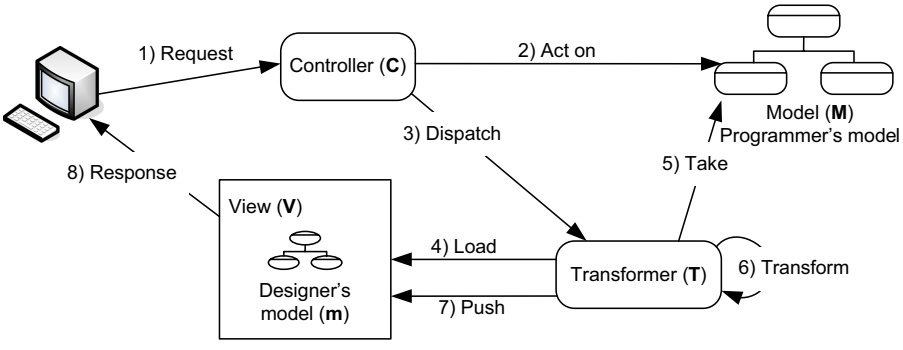

Fig. 2. Application flow in MVC+mT architecture

3.2 Double-Model Requirements

As a summary, we enumerate the conditions we require for a certain template system to be considered double-model compliant:

1. Both parts of the application, model and view, have their own model supporting their requirements, probably expressed using different languages, even paradigms.
2. The view is not allowed to access the programmer's model in any way.
3. The designer's model is initialized with test values.
4. The application uses the push strategy [20] to feed data into the template.
5. Data taken from the programmer's model can not be pushed directly into the template. They must be transformed in the way the designer's model mandates.

3.3 Double-Model and Entanglement Index

It is possible to relate Parr's work with the double-model approach. In fact we can say that *a template system that follows the double-model approach has a maximum entanglement index of 3*. This property is equivalent to saying that a template system that follows the double-model approach observes Parr's rules 1 and 4. Compliance of rule 1 is directly deduced from requirement 2 of section 3.2. Compliance of rule 4 is supported by requirements 1 and 5, because they guarantee that both parts of the application (classical model and view) use different data types that must be adapted.

4 JST2

JST2 (JST version 2) is an attribute based, script-driven, double-model based HTML template system that is processed in the user's browser. JST2 is an evolution of JST, previously presented at [16,17].

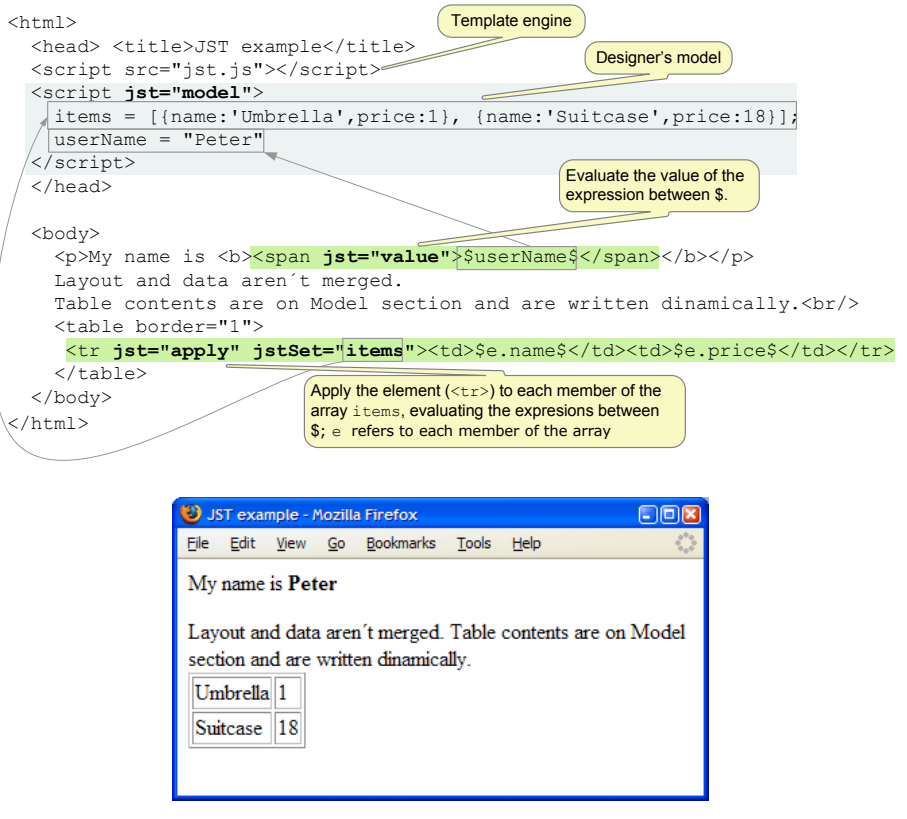

Fig. 3. JST2 example and its visualization on an internet browser

An important feature of JST2 is the use of JavaScript, which is at the foundation of JST (in such a way that JST stands from JavaScript Templates). JavaScript is used to define the designer's model, to populate it with test values, and to fill in the "holes" that the designer places in the HTML page as placeholders for dynamic data. Actually JST2 has no associated language, like other HTML templates have. JST2 defines a set of non-standard HTML-attributes that can be inserted in the HTML elements of the template. These HTML-attributes are processed by the JavaScript engine to perform the page rendering[1]. This is relevant for designers, who do not need to learn any other template language. They get by with their familiar tools: HTML and JavaScript, as well as the fact that they can work disconnected from the applications server in order to see their designs working. In fact, JST2 is valid for the development of standalone HTML documents. Fig. 3 shows a JST2 example, and its visualization in the Firefox browser (we stress the fact that the page does not need a server in order to be viewed, just double-clicking on the template document; this is full-previewability).

4.1 Working with JST2

From the point of view of designers, working with JST2 involves creating the HTML page by focusing only on the navigation and look and feel. Designers must identify the dynamic part of the page (in the example in Fig. 3 a user's name and a table of items). Then, they must define the variables within an `script` section that must be marked with the value `model` for the HTML-attribute `jst`. Variables are given a name and assigned a test value to them. Variables can be simple (of any JavaScript type), like `userName` in the example, or compound, like `items` in the example (an array of objects). These variables make up the so-called *JST model section*, which can be easily identified as the designer's model in the double-model approach.

A key point is that designer is free to structure his model as he wants, or knows. This is important for us because, up to certain extent, we do not want to force the designer to learn any particular syntax or to apply a unique way of doing things. E.g.: in the example, object literal notation is used to populate the `items` array, but there exist other possibilities, such us to use constructors of a JavaScript `Item` class, or to use two arrays of simple data (one for names and another for prices).

JST2 expressions. Wherever designers want to insert a data value, they place the variable name between a couple of `$` symbols (e.g., `$userName$`). A variable reference can appear alone or as part of a JavaScript expression, like in `$price*0.9$`. Therefore, designers have all the JavaScript power at their disposal,

[1] A JST2 template is processed by a script loaded with the page. For the processing of the template, the DOM JavaScript API is used. As was expected, many problems have arisen during the template engine development due to different browser peculiarities. A lot of tests have been performed, including form controls, structural elements (tables, divs …), styles, event handlers, and so on. A total of 112 different versions of browsers have been tested, resulting that JST2 can be used in Internet Explorer (from version 5.5), Firefox, Mozilla, SeaMonkey, Netscape (from version 6), Opera (from version 7), DeepNet, Safari (for Mac platforms) and Konqueror (for Linux). This covers approximately 98% of the available browsers (depending on the consulted browser statistics).

without the cost of learning a new programming language. JST2 expressions may appear as part of the value of HTML elements or attributes.

JST2 non-standard HTML-attributes. The way JST2 expressions are processed is determined by JST2 HTML-attributes. These attributes contain instructions for the template engine. Using attributes we get most editing tools not to complain about they do not understand these attributes, and they will not remove them. Besides, they will not change the structure or appearance of the template when loaded into a WYSIWYG editor or a web browser.

The most important JST2 HTML-attribute is `jst` indeed. Its value specifies the type of processing that the element that carries it must undergo. We call HTML elements marked with this `jst` attribute *JST2 elements*. There are six values for `jst`, which cover evaluations, conditionals, iterations and sub-template inclusions:

- `value`: the JST2 expressions inserted in the JST2 element, or any of its children, must be evaluated and their values must replace the expressions.
- `if`: the JST2 element will be processed only if the value of the conditional expression contained in an auxiliary HTML-attribute named `jstTest` is evaluated to `true`. If the condition is evaluated to `false`, the JST2 element is removed from the final document. All JST2 expressions are also evaluated as in `value`.
- `apply`: the JST2 element must be replicated for each one of the values contained in the array specified in an auxiliary HTML-attribute named `jstSet`. All JST2 expressions are also evaluated as in `value`. There are three implicit variables associated to the underlying iteration process that can be used in the expressions: `i`, the iteration index, `e` the iteration value, and `values`, the iterated set.
- `compApply`: like `apply`, but for a group of JST2 related elements. It also uses the auxiliary HTML-attributes `jstSet` and `jstTest`. For each set value, all the JST2 elements in the `compApply` group are evaluated. Only those whose `jstTest` expression evaluates to `true` are included in the final result.

Several simple examples[2]. Our objective is not to write here a detailed JST2 tutorial. Nevertheless, we want to illustrate the previously presented JST2 syntax with a few simple examples. Let's consider as the designer's model, the following script block:

```
<script jst="model">
 items = [{name:'Umbrella',price:1},{name:
          'Suitcase',price:48},{name:'Belt',price:18}];
 userName = "Peter"
 temperature = -2;
</script>
```

And below the samples:

```
<button jst="value" onClick="alert('Hello,
$userName$')">Say hello to $userName$</button>
<table border="1">
 <tr jst="apply" jstSet="items">
  <td>$e.name$</td><td>$e.price$ \$</td>
 </tr>
```

[2] JST2 tutorial and samples can be found at http://www.unirioja.es/cu/fgarcia/jst/

```
</table>
<p jst="if" jstTest="temperature<10">
  $userName$, you should wear your coat.
</p>
<table border="1">
 <tr jst="compApply" jstSet="items" jstTest="i%2==0"
     bgcolor="#CCCCCC">
  <td>$e.name$</td><td>$e.price$ \$</td>
 </tr>
 <tr jst="compApply" jstSet="items" jstTest="i%2!=0">
  <td>$e.name$</td><td>$e.price$ \$</td>
 </tr>
</table>
```

Once processed the resulting document will contain the following HTML:

```
<button onClick="alert('Hello, Peter')">Say hello to
Peter</button>
<table border="1">
 <tr><td>Umbrella</td><td>1 $</td></tr>
 <tr><td>Suitcase</td><td>48 $</td></tr>
 <tr><td>Belt</td><td>18 $</td></tr>
</table>
<p>Peter, you should wear your coat.</p>
<table border="1">
 <tr bgcolor="#CCCCCC"><td>Umbrella</td><td>1
$</td></tr>
 <tr><td>Suitcase</td><td>48 $</td></tr>
 <tr bgcolor="#CCCCCC"><td>Belt</td><td>18 $</td></tr>
</table>
```

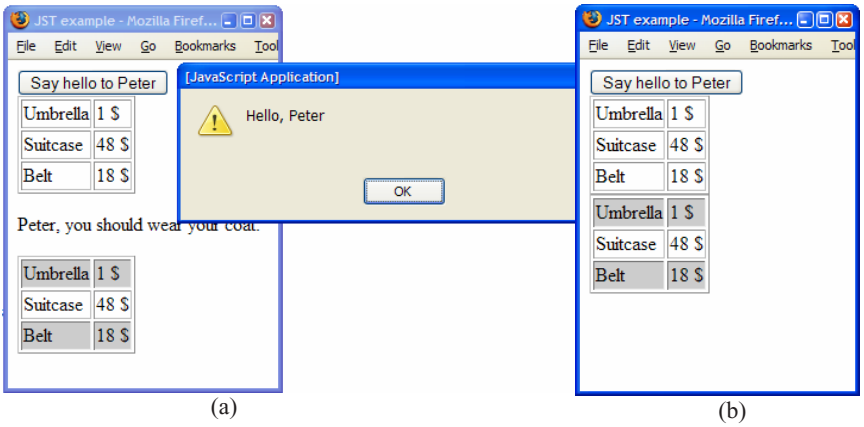

Fig. 4. JST2 examples, a) with temperature value -2; b) with temperature value 25

JST2 in the server-side. Once the work of designers is finished, programmers can proceed. The model section in the JST2 template shows the dynamic data that the page needs to be displayed, thus reducing the communication need between designers and programmers. The programmer must implement the code executed when a request for a

page is received. This code will fetch the final data from an object model, XML, or DBMS (the programmer's model) and will generate a string with JavaScript code that will replace the template model section. The JST2 preprocessor is extremely lightweight; just string substitution. Other template engines processing are not comparable in terms of speed and used memory. E.g.: let's assume that the actual name is "Mary" and there are three items: a leather belt, a handbag and an umbrella; then the page sent to the user will be just as before in Fig. 3, except for the model section:

```
<script>
 name = "Mary"
 items=[{name:'Leather belt',price:1}, {name:
       'Handbag',price:79},{name:'Umbrella',price:18}];
</script>
```

4.2 JST2 Implements the Double-Model

JST2 follows the double-model approach because it meets all of its requirements: (1) The designer's model is constituted by the model section, and the programmer's model can be supported by a DBMS, and XML data source, or any type of data structure in any programming language. (2) The view can not access the programmer's model. The view can only access its model section including the variable names inside JST2 expressions. This is stressed by the fact that the template can even be processed disconnected from the server where the application resides. (3) The model section is initialized with test values. (4) The template is passive from the point of view of its processing. At execution time, the previous test values are replaced with new data values. In the server side of the application, the part that acts as the transformer in MVC+mT architecture, collects all the necessary data and composes a string defining the new JavaScript model section (push strategy). And (5) during the previous processing, data taken from the programmer's model can not be pushed directly into the template, because they belong to different realms, they are even modelled in a different language. They must be transformed in the way the model section specifies.

4.3 JST2 Analysis

The use of JST2 effectively separates the model and view of a web application, and therefore further enhances the extensibility, maintainability, and reusability of the page templates. Besides of these positive properties, profusely treated in the related bibliography, we can mention another set of advantages.

Simplicity. JST2 relies on JavaScript, but it is difficult to appreciate it (apart from the model section and, perhaps, some expressions). This is a great help for designers, who do not need a lot of knowledge of JavaScript in order to use JST2. Besides, the syntax of JST2 is very simple and the learning curve is very soft.

Previewability. Thanks to the use of HTML-attributes in the JST2 elements, HTML editors allow designer to manipulate JST2 elements visually. Designers can work with JST2 elements as if they were normal HTML. Besides, due to the template engine being included as a script in the template page, the final appearance of the template can be observed without needing a server, with the only burden being the

visualization of test data instead of actual data. But this is not relevant for the aesthetics of the template.

Rapid prototyping. As a consequence of the previous feature, JST2 can be used for rapid prototyping of web applications, reducing deployment time. If fact, JST2 can even be used to rapidly develop stand-alone HTML.

Independence of the language used in the server-side of the application. JST2 processing consists only in string substitution, and this task can be performed with any programming language.

Exploitation of the client processing power. JST2 templates are processed in the client browser, thus relieving the server of this task.

JST2 helps to avoid cross-site-scripting (XSS). The usual technique to avoid XSS consists of escaping the potentially dangerous characters (such as <,>,&,…) by replacing them with their corresponding entity (`<`, `>`, `&`, …) . Using JST2, text is inserted in the JST2 elements as DOM TextNodes, which automatically escapes those types of characters.

JST2 simplifies AJAX interactions. AJAX processing is very simple with JST2. In fact, for an AJAX interaction to occur, we only have to record the original JST2 element that is going to be asynchronously changed, ask the server for the new data values required for the change, and then re-process the JST2 element with the new values. AJAX data interchanges in JST2 do not require the use of XML, not even JSON [18]; new data is received in the same format as was specified in the model section. That is, in the server side, AJAX requests, can be processed by the same modules that service normal requests.

JST2 problems. But we would not be honest if we did not recognize JST2 problems. One of the main JST2 drawbacks is precisely one of its foundations. JST2 relies on JavaScript and therefore it can not be used in browsers that have it disabled (at the time of writing, 90% of browsers has the use of JavaScript enabled, according to the last browser statistics). This is a limitation for the use of JST2 in devices such as mobile telephones or PDAs, which have available simpler browser versions without JavaScript support. However, it is a matter of time and technological maturity for this not to be a problem any more. Another potential problem is related to web accessibility. We have not investigated yet how accessibility tools process a JST2 template, but we can guess that problems may arise. And finally, a JST2 template is not an XHTML [28] document. Therefore JST2 pages can not show off the W3C stamp that marks it as XHTML compliant. This can be solved using the XHTML modularization mechanism described in [29].

5 Conclusions

When developing a web application, it is not enough to use a template system in order to achieve separation between model and view. The template system must have a set

of features that prevent developers from using programming shortcuts and backdoors that reduce their work load in first term, but entangle the view and the model.

Taking as a starting point the work in [20], we have stated that strict separation of model and view is desirable but impossible to achieve in HTML template systems due to JavaScript. Existing template systems can not observe all the rules stated in [20], but this does not mean we be resigned to release entangled web applications.

We have provided another less strict but effective approach for the analysis of the model-view separation problem, consisting of using two models, one for the view and another for the application. We have also proposed a modification of the MVC architecture that supports the double-model approach (MVC+mT). The double-model approach has been applied in JST2, a template system based on JavaScript. JST2 allows developing templates that are processed in the client browser.

As for future research lines we propose to relate our work to others such us [33], where a similar adaptation process occurs between the fornt-end and the back-end of a two-tier MVC based architecture that supports the MODFM methodology for the development of web applications; or [4] that presents the *dual-mvc* approach, in which the controller of a MVC application is split between the client and the server. The approach reduces client/server interactions in web applications by returning to the client more data than strictly necessary for page visualization, but that can be used in response to certain user actions. The idea is that the page in the browser has its own controller that fetches local model data instead of having to access the server. This approach has nothing to do with template systems, but shares the idea of having a double model for the application. The existence of these works leads us to consider that perhaps we are in front of an architectural pattern, and that it may appear in more fields apart from web applications. This will be our main line of future work, as well as to keep working in the study of the accessibility in relation to JST2 and the extension to make JST2 XHTML compliant.

Acknowledgments. Partially supported by Comunidad Autónoma de La Rioja, project ANGI-2005/19.

References

1. Al-Darwish, N.: PageGen: An Effective Scheme for Dynamic Generation of Web Pages. Information and Software Technology 45(10), 651–662 (2003)
2. Apache: Tapestry, http://tapestry.apache.org/
3. Apache: Velocity, http://velocity.apache.org/
4. Betz, K., Leff, A., Rayfield, J.T.: Developing Highly-Responsive User Interfaces with DHTML and Servlets. In: Proceedings of the 19th IEEE International Performance, Computing, and Communications Conference – IPCCC-2000, pp. 437–443 (2000)
5. Brabrand, C., Møller, A., Olesen, S., Schwartzbach, M.I.: Language-Based Caching of Dynamically Generated HTML. World Wide Web: Internet and Web Information Systems 5, 305–323 (2002)
6. Brabrand, C., Møller, A., Schwartzbach, M.I.: The bigwig Project. ACM Transactions on Internet Technology 2(2), 79–114 (2002)
7. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture. In: A System of Patterns, vol. 1, John Wiley & Son, West Sussex, England (1996)

8. Dojo, http://dojotoolkit.org/
9. Fernández, M., Florescu, D., Levy, A., Suciu, D.: Declarative Specification of Web Sites with STRUDEL. VLDB Journal 9(1), 38–55 (2000)
10. Ford, N.: Art of Java Web Development. Manning Publications Co. (2004)
11. FreeMarker, http://freemarker.sourceforge.net/
12. Garrett, J.J.: Ajax: A New Approach to Web Applications (February 18, 2005) http://adaptivepath.com/ publications/essays/archives/000385.php
13. Google: Google Web Toolkit,http://code.google.com/webtoolkit/
14. Halasz, S.J.: An Improved Method for Creating Dynamic Web Forms Using APL. In: Proceedings of the international conference on APL-Berlin-2000 conference, Berlin, Germany, pp. 104–111 (2000)
15. Hunter, J.: The Problems with JSP (2000), http://www.servlets.com/soapbox/problems-jsp.html
16. Izquierdo, R., García, F.J., Andrés, M., Juan, A., Manrubia, P.: JST: Towards a Usable Web Site Development Method. In: Proceedings of the IADIS International Conference WWW/Internet 2003, vol. 1, pp. 515–522. IADIS Press (2003)
17. Izquierdo, R., Juan, A., López, B., Devis, R., Cueva, J.M., Acebal, C.F.: Experiences in Web Site Development with Multidisciplinary Teams. In: Lovelle, J.M.C., Rodríguez, B.M.G., Gayo, J.E.L., Ruiz, M.d.P.P., Aguilar, L.J. (eds.) ICWE 2003. LNCS, vol. 2722, pp. 459–462. Springer, Heidelberg (2003)
18. JSON: http://www.json.org/
19. Kristensen, A.: Template Resolution in XML/HTML. Computer Networks and ISDN Systems 30, 139–249 (1998)
20. Parr, T.J.: Enforcing Strict Model-View Separation in Template Engines. In: Proceedings of the 13th International Conference on World Wide Web, WWW 2004, May 17-20, pp. 224–233. ACM Press, New York (2004)
21. Parr, T.J.: StringTemplates.http://www.stringtemplate.org/
22. Parr, T.J.: Web Application Internationalization and Localization in Action. In: Proceedings of the 6th International Conference on Web Engineering, ICWE 2006, ACM Press, New York (2006)
23. PHP: http://www.php.net
24. Seshadri, G.: Understanding JavaServer Pages Model 2 architecture. Exploring the MVC design pattern. JavaWorld.com (December 1999) http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html
25. Smarty.http://smarty.php.net/
26. Sun Microsystems: Java Server Pages™ Specification, Version 2.1. Sun Microsystems, Palo Alto, USA (2006), http://java.sun.com/products/jsp/
27. Tregar, S.: HTML:Template, http://html-template.sourceforge.net
28. W3C: XHTML™ 1.0 The Extensible HyperText Markup Language (2nd Edition), http://www.w3.org/TR/xhtml1/
29. W3C: XHTML™ Modularization 1.1. http://www.w3.org/TR/xhtml-modularization/
30. W3C: XSL Transformations (XSLT) Version 2.0 (2007), http://www.w3.org/TR/xslt20/
31. WebMacro, http://www.webmacro.org/
32. Wijkman, P., Dissanaike, S., Wijkman, M.: Mixer, Supporting the Model-View-Controller Design Pattern in Servlets. In: Proceedingds of the IASTED International Conference on Software Engineering SE 2004, Innsbruck, Austria, February 16-18, pp. 658–661 (2004)
33. Zhang, J., Chung, J-Y.: Mockup-Driven Fast-Prototyping Methodology for Web Application Development. Software-Practice and Experience 33(13), 1251–1272 (2003)
34. Zope: Zope Page Templates, ZPT, http://zpt.sourceforge.net/