Problems, Solutions and New Opportunities: Using Pagelet-Based Templates in Development of Flexible and Extensible Web Applications

Hao Han
National Institute of Informatics, Japan
han@nii.ac.jp

Bo Liu
Fuji Xerox
liu.bo@fujixerox.co.jp

ABSTRACT
As a standard programming model for Web application development, HTML template engines (systems) are used to separate the business logic and data computations from the Web page presentation. In this paper, we present the current problems about template engines in Web application development and propose the solution: construction of an XML-based template engine for developing flexible and extensible Web applications. We explain that the approach supports the demands of users well and could reduce the cost of development. We also discuss the new opportunities brought to the effective browser cache usage, code reuse, data integration, etc.

General Terms
Algorithms, Design, Experimentation, Performance

Categories and Subject Descriptors
H.4 [Information Systems Applications]: Miscellaneous

Keywords
Template Engine, Application Framework, XML, XSLT, Script, DHTML, Mashup, Web UI

1. INTRODUCTION
As an important method of information distribution, Web applications become more and more complicated in order to realize the demands of new techniques for the different kinds of users. The users prefer the Web applications containing the friendly user interface and easy operations. This trend accelerates the development of interactive and animated Web sites. For example, many Web sites use the DHTML or DOM scripting technology to create the interactive Web pages, which contains HTML, client scripts, DOM, etc. The scripts change variables programmatically in an HTML document, and affect the look and function of static contents. Although the interactive Web pages bring the users various visual effects, the Web pages are still not flexible in the further usages such as customization, information extraction and integration, etc. Moreover, in the opinions of developers, it is difficult to develop, debug and upgrade the DHTML documents. The interactive and animated parts might be coarse-grained components since they depend on the DOM structure of Web page. The developers want to develop the extensible Web applications quickly and easily including the processes of coding, debug, maintenance, etc. Also, the developed applications are expected to be convenient in the case of version upgrade, code reuse and even extension to other platforms. However, the current Web application architectures can not arrive at a solution satisfactory to both users and developers. The dynamic visual effects are realized by client script functions, which bring the flexible operation but increase the development cost. Therefore, the developers need to make a proper balance between the opinions and options of users and themselves usually.

In this paper, we propose an XML-based framework for developing flexible and extensible Web applications. Each Web page is divided as a set of pagelets, which are combined into an HTML page at the client side by using XML, XSLT and script. Our research emphasis is laid on the presentation and functionalities at client side instead of the business logic (or other server-side program logic) such as the database retrieval at server side.

The organization of the rest of this paper is as follows. In Section 2, we present the current problems about template engines in Web application development, and give the research motivation and an overview of the related work. In Section 3 we explain the pagelet-based framework in detail. We discuss the new opportunities brought to the Web applications by the evaluation and comparison of the proposed approach with the traditional development approaches in Section 4. Finally, the conclusion and the future work are given in Section 5.

2. CURRENT PROBLEMS AND RESEARCH MOTIVATION
A template engine is designed to process Web templates and contents to generate the output Web documents such as HTML/XHTML files. In general, a template engine specifies a template and fills in the template with the assigned values to obtain the output page. Templates are frequently used to separate the Web page presentation from the underlying business logic and data computations in the con-
text of Web application development. In the case of Web application development, this means that no logic computation in HTML and no HTML in logic computation roughly. Several works have given the discussion about whether the enforcing strict model-view separation in template engines is possible [11] or impossible [5] to achieve in practice. Besides the widely used and general-purpose JSP/ASP/PHP technologies, several template generators/systems are proposed for different purposes. Some systems are based on the page transition or data navigation modeling. T-Web (Template-Web) is a software application generator for design and construction of Web applications and Web services based on the Web Transition Diagram [13], which reflects the Web page Transition. By only composing a diagram using T-Web editor without manual programming, T-Web generator can automatically generate a Web application or Web service from the diagram. Without the manual programming, the calculation ability of generated Web application is still weak. Some are the partitioning systems. Hilda [15] is a high-level language for data-driven web applications, which allows dynamic partitioning of the Web application between the client and the server in a manner that is completely transparent to the developer. FlyingTemplate [14] produces a skeletal script which includes only the dynamic values of the template parameters and the bootstrap code. It is an automatic partitioning system that transfers more of the task of generating HTML to the client browsers. There are some component-based framework such as Fiz [3], which provides a library of flexible components ranging various kinds of objects and value editors.

Although these proposed systems decrease the development cost, there are some unsolved problems. For example, by using DHTML, some Web pages could be merged into a single Web page and the sum of developed Web pages is reduced. However, the development of DHTML pages is a time-consuming job. The developing engineers had to write large quantities of scripts to deal with the DOM and hidden values programmatically. Therefore, although the sum of pages is reduced, the cost of programming and testing becomes higher. Moreover, the presentation of rendered Web pages is not the final phase at the client side. The further usage of Web application includes reuse, information extraction, data integration and others. For example, the popular mashup technology needs to reuse the components of Web applications. Many approaches are proposed to realize the integration of HTML-based Web applications by end-user programming [7], or clipping [4], or extracting [6, 8]. These approaches scratch the surface of problem, not the cause brought by server-side page rendering mechanism.

3. SOLUTIONS

We give the solution, an XML-based framework for developing flexible and extensible Web applications. Here, each Web page is divided as a set of pagelets. A pagelet presents an individual topic or functionality region of Web page as shown in Fig. 1. Instead of a full rendered HTML page, the proposed template engine produces skeletal segments, which are combined and transformed into an HTML page at client-side Web browser. XML, XSLT and script library (JavaScript class package) are used to generate or update the dynamic contents (interactive and animated parts) at client side. As shown in Fig. 2, the pagelets in client are classified into the static and dynamic types. A static pagelet is a static segment of a rendered Web page. It always comprises and displays the same information in response to all download requests from all users and all contexts (e.g. Service Tool Pagelet and Site Info Pagelet in Fig. 1). A dynamic pagelet is a dynamically generated segment of a rendered Web page. It presents content that can change in response to different contexts or conditions. There are two ways to create this kind of effect/change. Generally, server-side program is used to change the page source determined by such conditions as data in a posted form, parameters in the URL, the type of Web browser, or a database. At the client, client-side scripting is used to change interface behaviors within a specific Web page in response to mouse or keyboard actions or specified timing events. In this case the dynamic behavior occurs within the page presentation.

Figure 1: A possible example of pagelet layout

Figure 2: Outline of pagelet-based templates

Fig. 2 describes the workflow and data flow as follows.

1. The server responds to the request of client with the pagelet container file. A container is a pagelet layout page that surrounds or references static pagelets, XML data/segment (e.g. database query result data), XSLT files (template processor used to transform XML data into HTML or XHTML documents), or external files (e.g. JavaScript, CSS files) in its page body.
The XML data/segments are encapsulated into a single XML document in order to reduce the connection time before they are sent to client side. The container file also contains the mapping information, which reflects the one-to-one correspondence between the XML data/segment and XSLT file.

2. The dynamic pagelet is generated at client-side browser. The XSLT processor transforms the corresponding XML segment into HTML or XHTML segment based on the one-to-one mapping. The static pagelets and generated dynamic pagelets comprise the main source of an entire client-side Web page.

3. When the end-users trigger the mouse/keyboard events (defined in XSLT processors, see Fig. 3) of dynamic pagelets, the variables in corresponding XML segments are changed by JavaScript functions (See Table 1). Then the XSLT processors retransform the updated XML data into the new HTML pagelets.

4. After the end-users’ submitting, the updated XML data are used to realize the database update (e.g. writable or read-only, event type) of XML data.

![Figure 4: An example of container file](image)

<table>
<thead>
<tr>
<th>Name</th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadInfo</td>
<td>load the XML data from encapsulated XML</td>
</tr>
<tr>
<td>getIviValue</td>
<td>get integer value from node</td>
</tr>
<tr>
<td>getNext</td>
<td>get next sibling node</td>
</tr>
<tr>
<td>lock</td>
<td>lock the setting information</td>
</tr>
<tr>
<td>unlock</td>
<td>unlock the setting information</td>
</tr>
<tr>
<td>getLstrErr</td>
<td>get the code or last error message</td>
</tr>
<tr>
<td>getErrMsg</td>
<td>get the error message by index</td>
</tr>
<tr>
<td>add</td>
<td>append node to the end of sibling nodes</td>
</tr>
<tr>
<td>replace</td>
<td>replace/update the node value</td>
</tr>
<tr>
<td>del</td>
<td>delete the node</td>
</tr>
<tr>
<td>insert</td>
<td>insert the node at the index</td>
</tr>
<tr>
<td>exchange</td>
<td>exchange the index/position of nodes</td>
</tr>
<tr>
<td>remove</td>
<td>remove all of the child nodes from node</td>
</tr>
<tr>
<td>clear</td>
<td>clear node value of node</td>
</tr>
<tr>
<td>transform</td>
<td>transform the XML into HTML by XSLT</td>
</tr>
</tbody>
</table>

4. COMPARISON AND DISCUSSION

We compare the pagelet-based system with other developed systems (introduced in Section 2) in development efficiency, applicability, reuse, programming skill requirement, security, server/client caching, extensibility and other fields. The pagelet approach is based on the separation of topic and functionality region of Web page. We implement not only the model-view separation in template but also the static-dynamic separation in pagelet. In the model-view separation, XSLT works as client-side template and offers a great expressiveness, allowing a translator to produce complex HTML documents from XML data. This separation leads to the division of labor and high efficiency. The Web page designer can adjust the presentation or layout without having to change the program logic, which is always much riskier. However, XSLT files development requires mastering a quite different language compared with JSP/ASP/PHP. Therefore, this XSLT-XML separation methodology is not suitable for small personal pages. It would waste the time by analyzing and has no big effect.

Pagelet separation methodology brings the easy and efficient development for large-scale Web applications. The programming job could be analyzed and divided according to the abilities or proficiencies of developing engineers. One engineer does not need to write an entire template file of a Web page, and the Web page development can be subdivided into functionality-oriented development. For the dynamic pagelet, the provided JavaScript functions are used to access or update client XML data. Compared with the traditional dynamically updating client HTML source, the XML data access/update/translation process is independent of the layout of Web page, and avoid using the HTML tags and HTML-oriented functions such as innerHTML or innerText, which are lack of the possibility of code reuse. XML syntax is far more restrictive and HTML coding often has been relatively careless. Similarly, the XML-oriented JavaScript functions are constructed and upgraded more easily than HTML parsing functions.
Pagelet separation methodology also brings more efficient caching functionality at server side and client-side Web browser. Compared with the page-based caching system, static pagelets would be widely cached and independent of changeable dynamic pagelets. Moreover, the XSLT templates of dynamic pagelets could be reused or cached at client Web browsers. This reuse/caching could reduce the data transfer volume between the server and client. We reconstructed the search result page of Google Search using the proposed approach. The experimental results show that the transfer data size is much reduced from 20,916 bytes to 6,794 bytes.

For the Web browsing at mobile devices, the pagelet-based pages can be easily simplified for better presentation at mobile Web browsers. Moreover, mobile devices could partially share the XML data in server side caching system with the general desktop Web browsing. It is also a big difference between the pagelets and traditional portlets [9].

The possible new opportunities may be brought to the further usage of Web applications by the pagelet structure. Information extraction on the Web focuses on the acquisition of information/knowledge from Web applications mainly. Many methods are proposed to extract the target information from Web pages excluding the HTML tags by analysis of HTML sources. These methods evolved from machine learning or pattern recognition or XPath designation are maintenance costly over a long period of time. In the case that RDF-like description is given for functional-semantical semantic in the container files, the information extraction from Web applications becomes convenient and the XML data of dynamic pagelet is a clean extraction result. Also, the mashup of Web applications becomes much systematic. The reuse of Web application data, computation and UI becomes more efficient and thorough. It is easier to transform the pagelets into mashup components than general Web applications [1] and portlets [2], especially for typical users with no or little programming experience. Moreover, it may accelerate/realize the following technique development: 1) XML-based message exchangeability between the desktop computers and mobile devices, 2) Client-side templates customization by end-users, 3) XML-based data transfer (layer) such as Ajax+JSON architecture, 4) Active Web pages containing Microsoft Object Library or Flex/Flash, 5) Efficient retrieval and update of client-side XML data in XML/XML-enabled database (e.g. XPath, XQuery operation) or by object-relational mapping technique, and 6) Multilingual Web UI.

Client-side XML transformation also brings new problems to search engines because the dynamic pagelets are not seen as general HTML sources. The container files do not contain information present on Web pages itself. The crawlers need to parse the pagelets to acquire the information. Moreover, the client transformation costs time and leads to an unavoidable short delay in page presentation at client-side Web browsers. Also, the current architecture in our ongoing implementation is proposed without strong consideration of security, such as the problems that may occurs in client-side XML accessing or XSLT client-side caching.

Some JavaScript frameworks/libraries (e.g. jQuery [10] and Prototype.js [12]) are proposed to simplify the event handling, animating, and Ajax interactions in Web application development. Compared with these frameworks/libraries, pagelet concentrates on the light-weight XML-oriented operation and further reuse of Web applications at client side.

5. CONCLUSION

We proposed a pagelet-based template engine/system for Web application development, which supports the demands of users well and could reduce the cost of development. We also discussed the diverse new opportunities about the extensible reuse of Web applications. There is still a lot of work to be done on this pagelet system before it becomes a mature technology. As future work, we will modify the development environment (e.g. a friendly GUI editor) for the generation of Web applications more easily as described in this paper. Moreover, we would like to explore further the problems of security, system’s scalability, and server-side database update. Additionally, besides the currently developing JavaScript class library, we will provide more various functions for efficient client XML access in future.

6. ACKNOWLEDGEMENT

We gratefully acknowledge the advice from Takehiro Tokuda (Tokyo Tech) and Hirofumi Yokouchi (Gumma Univ.). This work was partially supported by a Grant-in-Aid for Scientific Research A (No.22240007) from the Japan Society for the Promotion of Science (JSPS).

7. REFERENCES


[9] J. A. Demers, J. Gehrke, J. M. Marzetta, and M. Tatsubori and T. Suzumura. HTML templates that fly: a friendly GUI development environment (e.g. a friendly GUI editor) for the generation of Web applications more easily as described in this paper. Moreover, we would like to explore further the problems of security, system’s scalability, and server-side database update. Additionally, besides the currently developing JavaScript class library, we will provide more various functions for efficient client XML access in future.


