Client-Side Rendering Mechanism: A Double-Edged Sword for Browser-Based Web Applications

Hao Han∗, Yinxing Xue† and Keizo Oyama∗‡
∗National Institute of Informatics, Japan
{han,oyama}@nii.ac.jp
†National University of Singapore, Singapore
yinxing@comp.nus.edu.sg
‡The Graduate University for Advanced Studies (SOKENDAI), Japan

Abstract—Rendering mechanism plays an indispensable role in browser-based Web application. It generates active webpages dynamically and provides human-readable layout of information. Client-side rendering system brings various flexibilities but also new problems. In this paper, we give a comprehensive analysis of possible advantages and disadvantages brought by client-side rendering mechanism in viewpoints of both developers and users based on practice and experience.

Index Terms—Rendering Mechanism, Template Engine, Web Application, Web UI, Practice and Experience

I. INTRODUCTION

Browser-based Web application is one of often used computer software applications designed for information interactivity on the Web. It is reliant on a common Web browser to render the application executable, and usually coded in browser-rendered/supported languages such as HTML (HyperText Markup Language) and JavaScript. At server side, rendering mechanism is indispensable for human-readable presentation and generates webpages based on template engine. Figure 1 describes the basic architecture of a typical server-side rendering system. In general, rendering system processes templates and content to generate the output Web documents. 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 webpage presentation from the underlying business logic and data computations in the context 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. Server responses may be determined by requests sent from client side such as data in a posted HTML form, parameters in the URL, or the type of Web browser being used. Server-side scripting (program running on the server) is used to change the web content on various webpages. Such webpages are often created with the help of server-side languages such as ASP, Perl, and PHP.

Web applications become more and more complicated with the advancement of diverse Web technologies in order to realize the demands of new techniques for the different kinds of users. The users prefer the Web applications containing the user-friendly interfaces and easy operations. For example, Ajax (Asynchronous JavaScript and XML) dynamically interchanges content which sends a request to the server for data. The server returns the requested data subsequently formatted by a client side script, which reduces server load time because the client does not request the entire webpage regenerated by the server-side rendering. This trend accelerates the development of interactive and animated websites. Many websites use the DHTML (Dynamic HTML) or DOM scripting technology to create the interactive Web pages. It contains HTML, client-side scripts (such as JavaScript), DOM (Document Object Model), etc. The scripts change variables programmatically in an HTML document, and affect the look and function of static content.

Client-side rendering mechanism is one of proposed approaches responding to this trend. Instead of a fully rendered HTML page, client-side rendering system produces skeletal segments, which are combined and transformed into an HTML page at client-side Web browser. XML (Extensible Markup Language), XSLT (XML Stylesheet Language Transformations [26]), JSON (JavaScript Object Notation) and JavaScript are usually used to generate or update the dynamic content (interactive and animated parts) at client side. Each segment
presents an individual topic or functional region of webpage and only the segment that subject to changes is transmitted. These interactive webpages bring the users various visual effects and consume fewer resources on the server.

As a developing technology always has its faults, client-side rendering mechanism also has both merits and drawbacks. In this paper, we propose a developing framework as a simple implementation of client-side rendering mechanism, and give a comprehensive analysis of possible advantages and disadvantages in different viewpoints such as accessibility, caching, personalization and speed. Our analysis emphasizes on issues like performance, practice and experience at client side rather than the business/processing logic at server side, and the specific Web browser plug-ins are not discussed. Some experimental evaluations about rendering cost and development time are also discussed.

The organization of the rest of this paper is as follows. In Section II, we present a developing client-side rendering framework and introduce some related approaches/systems. From Section III we give analysis of performance/functionality at client-side and application development based on various viewpoints. Finally, the conclusion and the future work are given in Section IX.

II. AN EXAMPLE OF CLIENT-SIDE RENDERING SYSTEM

There is no uniform and mature structure of client-side rendering system widely accepted by general Web applications. In order to provide an intuitive and visible image of client-side rendering for the further analysis and explanation, we present an example based on XML and XSLT technology, which is generally employed as a stylesheet processor used to transform XML data into HTML or XHTML documents.

We are developing an XML-based framework for generating flexible and extensible Web applications based on client-side rendering mechanism as shown in Figure 2. Here, each webpage is divided as a set of static segments and dynamic segments. Static segment always comprises and displays the same information in response to all requests from all users and in all contexts. (e.g. navigation bar and site information placed in page footer). Dynamic segment presents dynamically generated 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 requests from client side. At the client, client-side scripting is used to change interface behaviors within a specific webpage in response to mouse or keyboard actions or specified timing events. In this case the dynamic behavior occurs within the page presentation. Figure 2 describes the workflow and data flow of framework as follows.

1) The server responds to the request of client with the segment container file. A container is a layout page that surrounds or references static segments, XML data, XSLT files, or external files like JavaScript and CSS (Cascading Style Sheets) files in its page body. The XML data is encapsulated into a single XML document in order to reduce the connection time before it is sent to client side. The container file also contains the mapping information, which reflects the one-to-one mapping between the XML data and XSLT file.

2) The dynamic segment is generated at client-side browser. The XSLT processor transforms the corresponding XML data into HTML or XHTML document based on the one-to-one mapping. The static segments and generated dynamic segments comprise the main source of an entire client-side webpage.

3) When the end-users trigger the mouse/keyboard events defined in XSLT processors of dynamic segments, the variables in corresponding XML data are changed by JavaScript functions, and the XSLT processors retransform the updated XML data into the new HTML segments.

A more detailed description of this system can be found in [9]. As a widely used technology, the open source community has created a huge number of client-side templating solutions. Logic-less templating technologies follow the strict model-view separation rule that there should be little or no logic in templates. They usually provide a clean separation between presentation and logic without sacrificing ease of use, and are particularly well-suited for asynchronous and streaming applications. Twitter uses Mustache [16] and JSON to move significant pieces of functionality to the browser, and Google plus employs Java-friendly Closure [1]. LinkedIn moves from server-side templates to client-side JavaScript templates powered by dust.js [3].

Besides these logic-less templates, there are embedded templating options, which allow developers to embed regular

---

1 https://twitter.com/
2 https://plus.google.com/
3 http://www.linkedin.com/

Many related studies have given the discussion about "rendering mechanism and rich application" with "component-based" (Fiz [6]), "data-centric" (Hilda [27]), "strict model-view separation" ([19]), "flexible model-view separation" ([7]), "skeletal script" (FlyingTemplate [22]), "interactive behaviors learning" ([18]), "accessibility evaluation" ([5]) and others described in following sections.

III. ACCESSIBILITY

RIAs (Rich Internet Applications or we call them rich client applications) and supporting technology reflect an implementation of rendering mechanism in the user agent or in the browser at the client side. They provide more dynamic Web content and more attractive and interactive websites. However, not all end-users benefit from this interactivity. For example, users with weak or disabled eyesight access the Web using assistive technologies, such as a screen reader (e.g. IBM Home Page Reader ⁴) that delivers audio content. If a client-side rendering/programming technique is used by a webpage, the screen reader cannot read the page-based operation (page modification), which is particularly problematic for the keyboard navigation essential to accessibility. With the meta data or HTML5 [10], interface developers should be able to adapt interfaces to meet specific needs partially. However, meta data is recommended but not necessary in HTML page, and until now most of developers do not use HTML5 tags widely.

Rich defined XML is better for screen reader, not only in reading but also page-based operation [15]. Figure 3 shows an example of XML document employed in Figure 2. It adds semantics to webpage components and content so that assistive technologies can interpret their operation. Regional landmark roles provide a navigable structure within a webpage, and node names and attributes present a guarantee of accessibility of controls. For example, "writable" represents the dynamic content-update notifications: update, remove, and add (0: cannot, 1: can).

IV. DATA CACHING

A Web data caching is a mechanism for the temporary storage of Web documents, such as HTML pages, to reduce bandwidth usage, server load, and perceived lag. For Web applications distributed by HTTP (Hypertext Transfer Protocol), freshness, validation, and invalidation mechanisms are used for controlling caches. Dynamic webpages are often cached when there are few or no changes expected and the page is anticipated to receive considerable amount of Web traffic that would create slow load time for the server if it had to generate the pages on the fly for each request. Despite the popularization of broadband networks, page or page segment is still a most basic and often used data transfer unit for Web applications. So, it would lease the load of server if we reduce the data transfer of page.

Page segment and client-side rendering mechanism bring more efficient caching functionality at server side and client-side Web browser. Compared with the traditional page-based caching system, static segments would be widely cached and independent of changeable dynamic segments. Moreover, the XSLT templates of dynamic segments could be reused or cached at client Web browsers and the layout at client side does not affect the cache. This reuse/caching could reduce the data transfer between the server and client. As an experimental example, we reconstructed the response Web page (search result page) of Google Search into XML and XSLT data. The experimental results show that the transferred data size is much reduced as shown in Table I if page is rendered at client side.

### Table I

<table>
<thead>
<tr>
<th>Structure</th>
<th>Data Format</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server-side rendering</td>
<td>HTML</td>
<td>20,916</td>
</tr>
<tr>
<td>Client XSLT caching</td>
<td>XML</td>
<td>6,794</td>
</tr>
</tbody>
</table>

For the Web browsing at mobile devices, the page segments can be simplified for better presentation at mobile Web browsers. Android developers also recommend upgrading the Web UI to an XML layout [25]. Moreover, mobile devices could partially share the XML data in server side caching.

---

⁴http://www-03.ibm.com/able/
V. Layout Customization

A configurable page is a webpage designed with built-in layout personalization and content selection capabilities. Each piece of content, also referred to as a resource displayed within a cell of a layout region, can be rearranged, hidden or shown. Client-side XSLT customization is more flexible and powerful than CSS personalization, which is often used in blog pages. For example, it is easy to rearrange the layout or control the visibility by changing the attribute values of nodes.

- Layout arrangement (Figure 4): End-users can move segments by dragging and dropping operations to adjust the locations (e.g. object.style.left = event.clientX - event.offsetX + document.body.scrollLeft; object.style.top = event.clientY - event.offsetY + document.body.scrollTop;), which is more compact or suitable for user browsing habits than the default layout arrangement of segments.

- Visibility control (Figure 5): End-users can hide/Undisplay the undesired segments by setting the property “display” of attribute “style” to “none” (object.style.display = “none”);. If the hidden segments are deleted, the original execution environment of JavaScript would be broken and the JavaScript could not run normally if the hidden segments (XML data) are used/updated in JavaScript programatically.

VI. Speed and Delay

If server-side load is reduced and rendering tasks are moved to client-side, tasks of client-side are unavoidably increased. Client-side rendering and rich client applications usually bring users a “image” that they are executed heavily and high CPU/memory-consuming, which leads to delay of page loading or refreshing at client Web browser. Actually, this delay is caused by large data streaming and plug-in such as JSON text stream in map applications, Flash player and Silverlight, or JavaScript loading. Client-side rendering itself is not a high CPU-consuming process. We (OS: Windows 7, Browser: Internet Explorer 9, CPU: Intel Core i7 2.93GHz, RAM: 4.00 GB) captured performance data of examples given in Section IV by Windows Performance Analyzer Tools. As shown in Figure 6 and 7, there are not many differences in CPU usage percentage (e.g. maximum and average value) between presenting different types of webpages. A similar statistical result of main memory usage is also learned.

As clients become increasingly sophisticated, there is more for a browser to do. For websites that rely heavily on client-side rendering, it is essential to include this delay. Too much JavaScript on the client side makes the browser slow unless the user has a powerful computer. Many large websites use one common set of JavaScript files and one common set of CSS files in every template and webpage. Those files change over time and often contain elements that are no longer used on the webpage or anywhere on the website. We need to keep track of what elements are being loaded on each webpage. Client side uses JavaScript libraries that often contain massive amounts of functions, but only a handful of those functions are used actually. If we want to speed up presenting webpage, we should remove any JavaScript or CSS that are not being used on the webpage.

Moving JavaScript to the bottom of webpage also enables all other requests to be processed quickly, which would make webpage appear to load more quickly. The browser can begin rendering faster and do not wait for all of JavaScript to load at the top of webpage. However, not all JavaScript can be moved to the bottom of a webpage. If the JavaScript is a

library required for other components on the page to render, it must be loaded early in the page lifecycle.

More discussions about bridging the gap between the browser view of a UI and its JavaScript implementation are given in [14].

VII. DEVELOPMENT AND REUSE

We have to consider development efficiency and programming skill requirements if we employ client-side rendering mechanism. It is based on the separation of topic and functionality region of webpage. We implement not only the model-view separation in template but also the static-dynamic separation in segment. In the model-view separation, XSLT serves 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. The webpage 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 and high proficiency with JavaScript frameworks/libraries (e.g. jQuery [13] and Prototype.js [20]). Therefore, this XSLT-XML separation methodology and client-side rendering mechanism are not suitable for small personal webpages. It would waste the time by analyzing and has no big effect.

Here, we give two experiments to prove the abovementioned opinions and compare development time cost.

1) Experiment 1: Developers are not proficient in XSLT and write original JavaScript functions for rendering. We developed a CIM (Customer Information Management) system, which provides the add, search, show, update, and delete functions mainly. By the traditional method, it needs five basic functionality pages (and other pages such as listing page): add a new customer (input the customer information), search for a customer (input the search keyword), show the customer information (detailed information such as name, address, telephone, and statistic of purchase history), update a customer information (e.g. change the address, add a new contact), and delete a customer. By using dynamic page, the show page and update page are merged into a show+update page. The users can update the information in show page by clicking the corresponding value area (trigger Onclick event) without the page jumping/transition. The updated information includes changed user information and calculated new purchase statistic (e.g. sum, average price, and graphic statistic). Compared with the two standalone pages show and update, the show+update page needed much more time in programming and test as shown in Table II (same developing engineers and quality/test engineers). The developing engineers had to write large quantities of JavaScript to deal with the DOM and hidden values programmatically, and face up to the fact that XML syntax is far more restrictive than HTML. Moreover, browsers’ debugging support is still very poor compared with server-side debugging support. Therefore, although the sum of pages is reduced, the cost of programming and test becomes higher.

<table>
<thead>
<tr>
<th>Page (Function)</th>
<th>Programming (hours)</th>
<th>Testing (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>show</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>update</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>show+update</td>
<td>40</td>
<td>64</td>
</tr>
</tbody>
</table>

2) Experiment 2: Developers are proficient in XSLT and
JavaScript library. We upgrade a Web UI (User Interface) system, which is an online information processing system and has two versions. The UI generator of old one is developed by JSP (JavaServer Pages) and the new one is developed by XSLT. In a version upgrade, a new functionality needs to be added for personalizing webpage layout based on user access authorization. A developing engineer did the code update for two versions. Table III shows that the XML-XSLT architecture is more efficient. 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 webpage, and the webpage development can be subdivided into functionality-oriented development. For the dynamic segment, the JavaScript library is used to access or update client XML data. Compared with the traditional dynamically updating client HTML source, the XML data access/upgrade/ transformation process is independent of the layout of webpage, and avoids using the HTML tags and HTML-oriented functions such as innerHTML or innerText, which are lack of the possibility of code reuse. XML-oriented JavaScript functions are customized more easily than HTML parsing functions.

<table>
<thead>
<tr>
<th>Version</th>
<th>Language</th>
<th>Time (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old</td>
<td>JSP</td>
<td>184</td>
</tr>
<tr>
<td>New</td>
<td>XSLT</td>
<td>80</td>
</tr>
</tbody>
</table>

### VIII. DISCUSSION

The design philosophy behind our approach/opinion aims at achieving the following goals.

1) To extricate the server side from the overloaded rendering work requested simultaneously by multiple client ends.
2) To further separate the concerns of the Web application and distinguish the handling of various types of data.
3) To facilitate the personalization and customization of the displayed output webpages for the different users and types of client ends.

For the goal 1, in the client-server model, there is one common controversy about the choice of the thin-client architecture or the thin-server (full-client [12]) architecture. Yang et al. [28] found that thin-client systems can perform well on Web and multimedia applications in LAN (Local Area Network) environments. This shows that the rendering mechanism of the traditional Web application worked fine when it was usually at the server side. However, nowadays with the better and better graphical quality displayed by Web applications and more various requirements from users, the rendering tasks become more and more computationally costly. Thus, keeping the rendering mechanism still at server side becomes an obsolete design. Now there is a trend that TSA (Thin Server Architecture) is advocated recently [4][23]. Note that in our approach, we are not at the extreme to have most functions of Web applications exclusively limited at the server side or at the client side. The idea of our approach is not to use server-side templates to create and transmit the webpage, but to separate concerns using protocol between client and server and to get a much more efficient and less costly development model. Using TSA leads to following three potential advantages.

- The server side can only focus on the business logics.
- The client side can focus on the presentations.
- The communications between server and clients just exchange the raw data such as XML.

As shown in Table I, the exchanged data between server and client is reduced in our approach, which improves the usage of data caching. Besides, moving the rendering from server to client also facilitates the concurrency of presentation. The XSLT used in our approach, as a functional programming language, can easily enable concurrency, since the order of execution of side-effect free functions in functional programming is not important. Thus, the client can run the multiple same XSLT threads simultaneously to speed up the data presentation.

For the goal 2, after separation of presentation and content, for the ease of reusability, maintainability and extensibility, it is desired to further separate the concerns of the Web application on multiple dimensions [21]. The accessibility is one of the dimensions we try to separate concerns on. By adopting the RIAs or rich defined XML, the client side can provide navigable structure and guarantee of accessibility of controls (see Section III). Another dimension of concern important to the performance is the different types of content. For example, the static and dynamic content in the different segments in the output webpages should be differentiated. In the original server-side rendering mechanism, the data exchanged between server and client is usually webpages. In our approach, with the skeletal segments generated by XML, XSLT, JSON and JavaScript, the exchanged data between server and client can be narrowed down to the segment level. And distinguishing the static and dynamic segments also improves the efficiency of data caching (see Section IV). Except the different handling for static and dynamic content, the similar idea can be applied to yet some other types of content.

For the goal 3, as the user-adaptive and context-aware characteristic is the trend for the future Web applications [2], Web application should have the different rendering strategies for the different types of client ends and the various user requirements. Layout arrangement and visibility control (see Section V) is just one type of user customization and personalization. In some more sophisticated client ends like smart-phone, the gesture control in response to the users’ operations on the screen is better to be handled by the client end, otherwise waiting for the rendering results from server-side will cause the extra delay.

Certainly, the above merits of client-side rendering do not come without any compromise. As shown in Table II and Table III, using our approach together with the involved techniques
may introduce the extra learning curve and initial investment for the early separations than the commonly used server-side rendering mechanism (taking almost more than double time, see Table II). However, in the long term, the developers will benefit from this in the evolution and extension of the Web applications.

IX. CONCLUSION

We gave a comprehensive analysis of advantages and disadvantages of client-side rendering mechanism in different viewpoints of practice and experience. The experimental evaluations proved a proposed example framework supports the demands of users well and could bring the diverse new opportunities about the extensible reuse of Web applications.

There is still a lot of work to be done before it becomes a more mature technology since it currently cannot 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 for some developers or development scenarios. Therefore, as shown in Figure 8, the developers need to make a proper balance between the opinions and options of users and themselves usually.

As future work, we will explore further the problems of system security, scalability, and server-side database update of client-side rendering mechanism. Additionally, besides the currently developing JavaScript libraries and frameworks, we will develop more various supporting technologies for efficient client rendering in future.

X. ACKNOWLEDGEMENT

We gratefully acknowledge the advice and experiment support from Bo Liu (Fuji Xerox, Japan). This work was supported by a Grant-in-Aid for Scientific Research A (No.22240007) from the Japan Society for the Promotion of Science (JSPS).

REFERENCES