JWebPresenter: A Universal Web Presenting Tool

Liang Zhao and Hideo Yamamoto
Faculty of Engineering, Utsunomiya University
Utsunomiya, Tochigi, 321–8585, Japan.
Email: {zhao, yamamoto}@is.utsunomiya-u.ac.jp

***Abstract*— The JWebPresenter project aims at creating a universal Web presenting tool. It uses bitmap image as the presenting format to avoid compatibility problem. To get the maximum efficiency, protocol-independent techniques are used to handle the cache/proxy issue and to avoid duplicate image fetching. Other features include platform independence, scalability, simplicity and open source.**

I. Introduction

We consider the way to perform a presentation, i.e., *presenting*. Recently, the primary way to give a presentation has being shifted from old-fashioned methods to PC-Projector (PP) based method. PP based presentation has advantages over the old-fashioned methods, but there also exist problems in real applications. In particular, we observe that existing tools cannot satisfy our requirement for *distance presenting* ([7]). Here let us summarize it as the next problem.

Suppose there are a number of audiences who may not locate locally and may have different kinds of information devices (e.g., computers, PDAs or even cell phones). How can a presenter perform the presentation? If there is a solution, how efficient is it? Here we can assume that a public network (e.g., the Internet) is available. See Fig. 1.

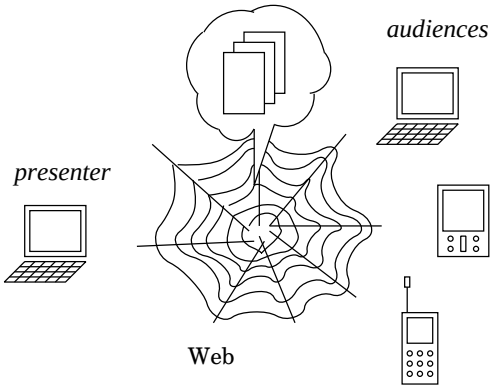

Fig. 1. Illustration of the Web presenting problem.

Speaking this kind of presenting, one may notice that it is much like a TV learning program. Actually our objective can be said to be an *efficient, networked, TV-program like distance presenting system*. This paper shows our solution. We note that it is a visual-only solution. An audio solution could be obtained by any existing audio streaming tool, see e.g., `http://linux-sound.org/netaudio.html`.

A. Relations to other works

Computer based distance learning has a long history. As far as we know, most previous studies (e.g., [1], [2], [5]) are based on HTML, XHTML or SMIL, which are shown to be efficient for *interactive* or *self-training* courses. On the other hand, however, these systems cannot fit for our requirement well. This is because that:

1) Basically HTML (and others) only defines the *structure* of contents, not the *appearance* of the presentation (which is left as a work of browsers); on the other hand, the most important factor in a presentation is that *all audiences can have the same appearance (screen), even if they are using different kinds of devices*.
2) In a presentation, a presenter usually wants to control what the audiences see (i.e., all audiences can see nothing but what the presenter wants to show). Thus a self-playing HTML page, a Flash clip or other things that require client interactivity is not preferred.
3) Text based presentations such as the HTML suffer the compatibility problem (font, language, browser-different or browser-dependent rendering etc).
4) Other limitations. For instance, the important full screen function is not available in existing browsers.

It is easy to see that these problems can be solved by *image based Web presenting*. Currently, this means to transmit the *bitmap-image* based screenshots of a local presentation to distance audiences. We note that it is easy to obtain image-based slides with existing PP presentation tools by *exporting into image* (available in PowerPoint) or by *capturing the screen* (works for all PP presentation tools).

Actually bitmap-image based distance presenting is not a new idea. MagicPoint ([3]) and VNC tools ([4], [6]) can do presenting in the same way. Comparing to such existing tools, JWebPresenter is a presenting-only tool. It is more simple, efficient, scalable and feature-rich (with respect to the functions of *presenting*), see Section III for a comparison. The contribution of this paper also includes a relatively detailed study on the data amount of such presenting.

The rest of this paper is organized as follows. In Section II, we explain basic ideas of JWebPresenter. In Section III, we show a simple solution and detailed data comparisons. Finally in Section IV, we conclude with remarks.

II. Basic ideas in JWebPresenter

Let us first give an overview of JWebPresenter.

JWebPresenter provides no utility for creating contents. It only cares *presenting*. One can use one's favorite tool to create the contents (slides), and use JWebPresenter to present them (after converting the contents into supported images). See an illustration in Fig. 2.

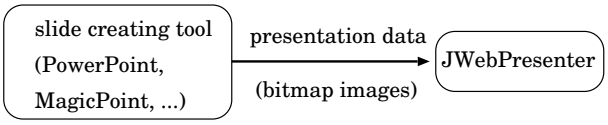

Fig. 2. Illustration of using JWebPresenter in a presentation.

JWebPresenter works in the server-client style, where presenter is the server, and audiences are clients. They are connected by a (TCP/IP) network. We need an address (a URL) for presenting, which points to the *currently displayed* slide. Fig. 3 illustrates the data transferring in JWebPresenter, where the address is denoted by "slide".

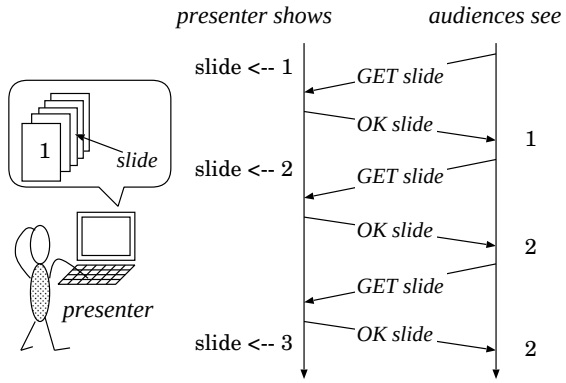

Fig. 3. Illustration of data transferring in JWebPresenter presenting.

As said before, we provide no interactive ability between server and client. This is because that, unlike a self-training course, in a presentation, audiences are preferred to be in "read-only" mode – it is the presenter who leads the presenting. As a consequence, the server only need to care the slides, whereas a client must fetch slides periodically.

The above idea was adopted in MagicPoint (by the mgpnet utility), which employs the *auto refresh* feature of HTML to tell client the time period to access the *unique* presentation URL (mgpnet has a built-in tiny HTTP server). In practice, unfortunately, this simple idea has poor performance if there is no new idea. Firstly, since image files are generally large, we want to avoid duplicate fetches (which is not available in MagicPoint). Secondly, we want to have considerations on the client cache issue (cache is not available for MagicPoint since mgpnet generates screenshots on-the-fly).

For that purpose, we introduce *control parameters* to provide clients an easy way to control the fetch. One parameter is the *sequence_number* of the slide, by which a client could cache fetched slides and thus skip duplicate image fetch. Another parameter is the *suggested_next_fetch_time*, which gives clients a hint of when to perform the next fetch (the auto refresh feature of HTML is not used since we want a protocol-independent implementation).

There is another kind of cache problem. In preliminary study, we observed that the internal cache function of proxy, browser or Java plugin (we use the Java language) can prevent an audience to correctly fetch the target if the URL is static (as in MagicPoint). Thus we add two more parameters *slide_url* and *next_ctl_url*, which specify the *dynamic* URL of the currently displayed slide and the *next* (also dynamic) URL containing control parameters (for the next slide). Besides, we also have static URLs linking to the current displayed image and control parameter file for a new client to keep up with the presenting.

Let us denote the slide URL by "slide", and denote the control parameter URL by "ctl". (Both "slide" and "ctl" vary during a presenting.) The pseudo-code of our basic implementation (which considers no cache function for simplicity) can be written as Fig. 4 and Fig. 5 for server and client, respectively.

```
while not finished {
  determine new "slide" and new "ctl"
  put control parameters into old "ctl"
}
```

Fig. 4. Pseudo-code of a JWebPresenter server.

```
while not finished {
  get control parameters from old "ctl"
  renew "slide" and "ctl"
  if sequence_number has changed {
     get the slide from new "slide"
     set timer for the next fetch
  }
}
```

Fig. 5. Pseudo-code of a JWebPresenter client.

We note that multiple sources can be used to adapt for the client heterogeneity such as the resolution and processing power (see [7]). Our implementation supports this.

III. A simple solution: JWebPresenter SS

In this section, we show a simple solution JWebPresenter SS, which uses unmodified images. An advanced solution will be discussed in the next section.

A. Server implementation

In JWebPresenter SS, we employ an HTTP or an FTP server to provide clients the control parameters and slides. This can greatly simplify our implementation, as well as

obtaining good scalability (since modern HTTP/FTP servers are extensively studied and well tuned). We only have to take care of the shown slides and the creation of control parameters. For that purpose, we provide the presenter a small Perl script (and a shell script for UNIX/Linux users).

Many HTTP/FTP servers have been tested, including

- Apache (http://www.apache.org/),
- Savant (http://savant.sourceforge.net/),
- thttpd, mini_httpd (http://www.acme.com/software/),
- tinyweb (http://www.ritlabs.com/en/products/tinyweb/),
- vsftpd (http://vsftpd.beasts.org/)

that run under various platforms, though even a simplest HTTP/FTP server should work. Fig. 6 shows a screenshot of the interface of our Perl script, which is running on a PDA (iPAQ h3600) powered by Linux (Familiar 0.7.2) and the thttpd HTTP server.

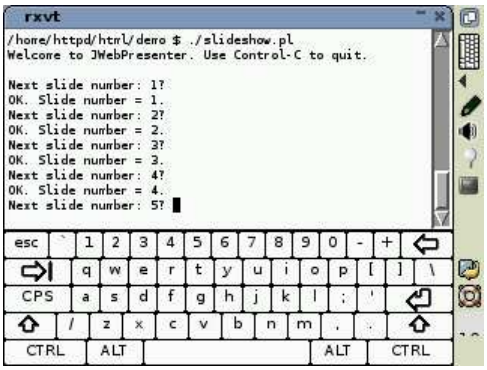

Fig. 6. Interface screenshot of the server Perl script.

B. Client implementation

It is more difficult to implement a client than a server. We have chosen Java as the programming language in order to use its rich and portable standard library.

The client can run either in applet mode (need no installation) or in application mode (requires an installation). Its capability depends on the hosting Java Virtual Machine (VM in short), i.e., it supports all protocols and all image formats supported by the Java VM. According to our tests, currently a desktop Java VM (version 1.4.2) can handle protocols HTTP and FTP, whereas the one shipped with Zaurus SL-C760 (PDA) can only handle HTTP. On the other hand, they all support the GIF, JPEG and PNG image formats.

Screenshots of the client working in applet mode are shown in Fig. 7 (Windows XP) and Fig. 8 (Zaurus SL-C760). A screenshot demonstrating the application mode is shown in Fig. 9 (Redhat Linux 9). We note that in application mode, the information bar ”Java Applet Window” in applet mode does not exist. In that case, an audience has no way to know the presenting tool (in full screen mode).

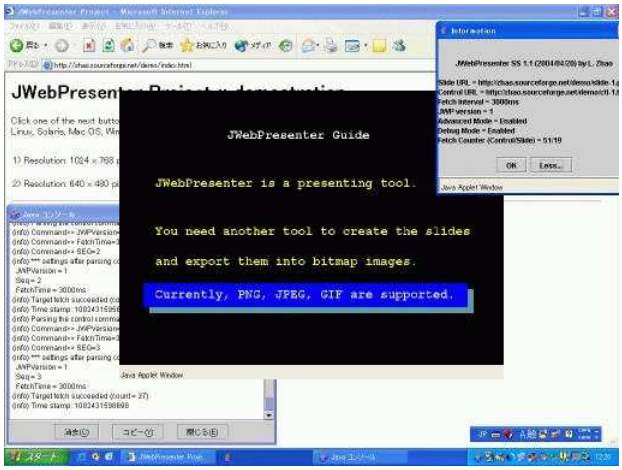

Fig. 7. Screenshot of the client in applet mode (Windows XP).

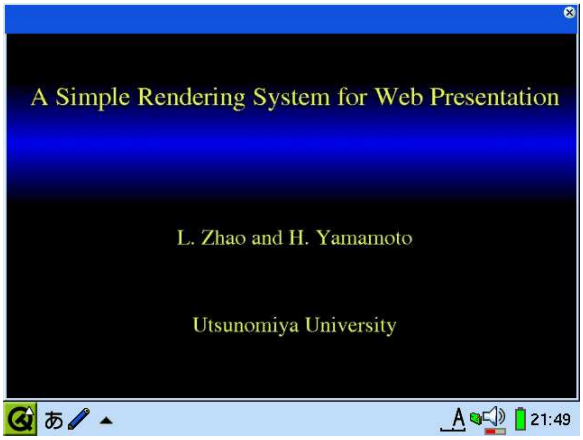

Fig. 8. Screenshot of the client in applet mode (Zaurus SL-C760).

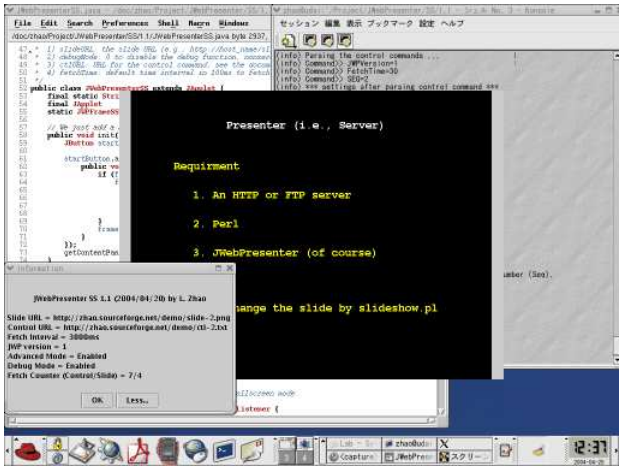

Fig. 9. Screenshot of the client in application mode (Redhat Linux 9).

Recall that we want all audiences to have the same screen. Therefore our implementation provides clients little functionality. The current implementation allows a client to change the window size (including full screen mode), enable or disable the auto resize option (i.e. to use the original image size or to fit for the window), and can get (debug) informations on the presenting.

Finally, we note that recently the number of Java embedded information devices (including computers, PDAs and cell phones) is increasing rapidly. We have also developed a client JWebPresenter MDIlet (using the MDIP 1.0 API) with J2ME Wireless Toolkit (see http://java.sun.com/j2me/). Fig. 10 shows a screenshot (running on the emulator).

Fig. 10. Screenshot of JWebPresenter MIDlet.

C. On the performance of JWebPresenter SS

For the purpose of estimating the efficiency of bitmap image based presenting, we have performed the next two tests. They may not be very extensive, but can give us a rough estimation on the data amount.

Firstly, we randomly picked up 20 PowerPoint files from the Internet. Each file is exported into three formats GIF, JPEG and PNG by PowerPoint 2002. In exporting, we use the default settings of PowerPoint except that the resolution is set to 1023×768 which matches the mainstream of today's PP presentation. The comparison of data amount is shown in Fig. 11, where the data items are as follows.

- #slide: number of slides,
- PPT (KB): size (in KB) of the original PowerPoint file,
- GIF (KB): size of the obtained GIF files,
- JPEG (KB): size of the obtained JPEG files,
- PNG (KB): size of the obtained PNG files,
- Mixed (KB): sum of the minimum sizes of slides by mixing the GIF, JPEG and PNG files.

The last line in Fig. 11 shows the sums for all items.

From Fig. 11, we can see that bitmap image based presenting does not require much data amount. (In fact, comparing to the original PowerPoint files, the total size even decreased.) On the other hand, the average data amount per slide is

$$20451/478 < 43 \text{ KB}.$$

No	#slide	PPT (KB)	GIF (KB)	JPEG (KB)	PNG (KB)	Mixed (KB)
1	30	2102	6837	2026	1997	1088
2	9	283	104	532	74	74
3	34	1478	2880	2805	3412	2487
4	50	1041	2499	5474	5375	2125
5	4	2689	339	302	1222	231
6	37	83	823	2741	510	510
7	13	706	874	1225	3040	728
8	49	2863	2031	3817	6122	1727
9	13	485	231	736	627	215
10	11	1483	942	1165	1342	708
11	29	108	551	3315	418	418
12	17	390	504	1727	502	385
13	18	89	1698	1105	955	938
14	20	872	1799	1282	4210	1170
15	19	94	1354	1390	504	504
16	18	678	2740	1408	2312	1400
17	52	2238	2441	4210	4112	1972
18	27	3057	1899	2504	5109	1607
19	18	1648	3850	2011	3109	1893
20	10	296	318	667	851	271
SUM	478	22683	34714	40442	45803	20451

Fig. 11. Raw data amount (in KB) comparison.

If it takes 30 seconds for a presenter (for us, we often take one or more minutes) to explain one slide, this shows that the required bandwidth is at most

$$20451 \times 1024 \times 8/(478 \times 30 \times 1000) < 12 \text{ (kbps)}.$$

In real applications, we must consider the overhead, which include control parameters and packet headers. For that purpose, we performed another test by comparing JWebPresenter SS with VNC tools running in the read-only mode, i.e., a VNC client can watch the server screen but has no user interactivity. (We note the mgpnet utility of MagicPoint did not work on our Linux machine, hence the data amount of MagicPoint is not available.) The VNC tools that we have tested are RealVNC v4.0 beta ([4]) and TightVNC v1.2.9 ([6]). Since in our preliminary tests, TightVNC shows its superior performance to RealVNC, we only compare TightVNC and JWebPresenter SS in the following.

For TightVNC, we do a PowerPoint presentation on a Windows PC running TightVNC server, and receive the presentation by a Linux PC running TightVNC viewer. For JWebPresenter SS, on the contrary, we do the presentation on the Linux PC, and receive it by the Windows PC. All packets are captured and analyzed by ethereal 0.10.0a (http://www.ethereal.com) running on the Linux PC. See Fig. 12 for the result comparison, where the same presentation files in Fig. 11 are used. The items are

- TightVNC (max_compression): data amount (in KB) of TightVNC using its maximum compression level,
- TightVNC (normal): data amount of TightVNC using it normal compression level,
- JWebPresenter SS (mixed): data amount of JWebPresenter SS using mixed formats (see also Fig. 11).

From Fig. 12, we can see that JWebPresenter SS performs

No	#slide	TightVNC (max_compression)	TightVNC (normal)	JWebPresenter SS (mixed)
1	30	2007	7528	1299
2	9	351	351	114
3	34	2118	6216	2768
4	50	2526	8882	2449
5	4	266	1460	260
6	37	1112	2780	688
7	13	939	3893	822
8	49	2840	8802	2026
9	13	425	1014	282
10	11	1202	4078	795
11	29	1604	5774	519
12	17	1136	2804	454
13	18	930	2714	1040
14	20	1816	5243	1288
15	19	1187	3609	584
16	18	2476	6737	1529
17	52	3886	13684	2223
18	27	1153	3310	1770
19	18	4857	9249	2011
20	10	349	911	312
SUM	***478***	***33180***	***99039***	***23233***

Fig. 12. Packeted data amount (in KB) comparison with TightVNC.

much better than TightVNC. Let us give two more remarks.

Firstly, TightVNC is slow, especially in high compression level. This is because that it computes the data on-the-fly (this is the feature of VNC tools and MagicPoint). On the other hand, JWebPresenter SS works on pre-created files (this is the feature of most presentations) and utilizes a Web server to transmit them to clients efficiently. We have no idea on the number of clients that a TightVNC server or MagicPoint can serve concurrently, but clearly JWebPresenter SS is more scalable (the number can be several tens, hundreds or even more, depending on the performance of the Web server).

Secondly, the presentation quality of TightVNC often goes bad (notable noise) in high compression level. We consider that this is due to its usage of JPEG compression, since JPEG is known to have problems for non-natural pictures which are common in presentations. On the other hand, JWebPresenter SS can choose the best one from GIF, JPEG and PNG (for each individual slide), thus can get better flexibility.

Therefore we can conclude that, for distance presenting, JWebPresenter is superior to VNC tools (and MagicPoint). Notice that Fig. 12 also gives us a rough estimation on the average data amount per slide (including overhead), which is

$$23233/478 < 49 \text{ KB}.$$

Suppose again that one slide takes 30 seconds, we can get an estimation on the required network bandwidth (for a real-time presenting), which is

$$23233 \times 1024 \times 8/(478 \times 30 \times 1000) < 14 \text{ (kbps)}.$$

Finally, for clarity, we give a detailed comparison of the three Web presenting tools MagicPoint (mgpnet), VNC tools and JWebPresenter SS in the following table.

TABLE I

FEATURE COMPARISON OF WEB PRESENTING TOOLS.

feature	*MagicPoint*	*VNC tools*	*JWebPresenter*
platform	UNIX/Linux (may fail)	Windows UNIX/Linux	independent (PC, PDA, cell phone...)
protocol	HTTP	VNC	HTTP, FTP (independent)
full screen	no	yes	yes
auto resize	no	no	yes
image creating	on-the-fly	on-the-fly	offline
data amount	n/a	large	small
cache/proxy consideration	none	none	yes
scalability	n/a	poor	good
require client installation	no	yes	no (applet) yes (application)
package size	big	big	small
non-presenting feature	slide creating	desktop management	none
main requirement	NetPBM, Perl Web browser	none	Java VM, Perl Web server

IV. CONCLUSION AND REMARK

In this paper, we have shown a Web presenting tool JWebPresenter SS, whose efficiency is shown by comparing with other solutions including the MagicPoint and VNC tools. JWebPresenter is an open source project, see `http://zhao.sourceforge.net/` for details.

We also have a developing version JWebPresenter AS, which tries to automatically reduce the data amount by introducing techniques (e.g., motion compensation) from video compression. Currently the simplest difference mode (i.e., transmit only the difference between two pictures) is implemented. Preliminary tests shows this method can further reduce the data amount by about 10%. Further details are omitted due to the incompleteness.

ACKNOWLEDGMENT

This research is partially supported by the International Communications Foundation (ICF), Japan. We would like to thank the Open Source Development Network, Inc. for hosting the JWebPresenter project.

REFERENCES

[1] H.Y. Chen, G.Y. Chen and J.S. Hong, "Design of a Web-based Synchronized Multimedia Lecture System for Distance Education", in *Proc. IEEE International Conference on Multimedia Computing and Systems Volume II* (1999), pp. 887–891.
[2] S. Sampath-Kumar, A. Banerjea and M. Moshfeghi, "WebPresent – A World Wide Web based tele-presentation tool for physicians", *SPIE Medical Imaging* 97, vol. 3031, pp. 832–843 (1997).
[3] MagicPoint, `http://www.mew.org/mgp/`.
[4] RealVNC, `http://www.realvnc.com/`.
[5] N. Sala, "Multimedia Technologies in University Courses: Some Examples", in *Proc. IEEE International Conference on Multimedia Computing and Systems Volume II* (1999), pp. 979–981.
[6] TightVNC, `http://www.tightvnc.com/`.
[7] L. Zhao and H. Yamamoto, "A Simple Rendering System for Web Presentation", in *Proc. ICACT 2004*, pp. 627–631 (2004).