International Journal of Networked and Distributed Computing, Vol. 4, No. 2 (April 2016), 106-115

An Extended Web Displaying System based on Multiple Tablet Devices

Shota Imai,[1] Shun Shiramatsu,[1] Tadachika Ozono,[1] Toramatsu Shintani[1]

[1] *Department of Computer Science and Engineering, Graduate School of Engineering*
Nagoya Institute of Technology
Gokiso-cho, Showa-ku, Nagoya-shi, Aichi, 466-8555 Japan
E-mail: {imasho, siramatu, ozono, tora}@toralab.org

Abstract

We propose an extended Web displaying system based on multiple tablet devices to help users to conduct collaborative works anywhere without a large display. Users can use multiple tablet devices as a pseudo mobile large display on demand by using our system. For example, users can make discussions by using Web applications anywhere. We present management methods of the arranged displays and two types of synchronous messages. One uses positional information and the other uses operational information for Web page synchronization. We show two applications based on the extended Web displaying system, a piano score system and a map viewer system.

Keywords: Distributed Display System, Tablet Devices, Web, Synchronization

1. Introduction

We have been implementing an extended Web displaying system for tablet devices, which makes multiple tablet devices to operate as one large display unit for Web viewing and aims to support collaborative works[1]. Tablet devices have high mobility because of their battery performance, lightweight properties, larger screens than smartphones, and so on. Thus, the system enables users to use large displays and makes collaborative works. For example, users can make discussions by using Web applications with a pseudo large display based on multiple tablet devices anywhere.

There are a variety of extended display systems. Distributed Multihead X combines multiple displays on multiple PCs that are distributed on a network, and the displays are presented to the user as a single unified screen[2]. MaxiVista extends PC's display area by using another PC as a secondary display[3]. Ueda proposed Tenmads that is a software multi-display implementation that runs on Microsoft Windows and focused on cost effectiveness and simplicity in configuration so that most PC users can set up multi-display environments without any help from computer experts[4]. Seifert et al. proposed an approach for extending mobile user interfaces by using external screens[5]. Users can utilize more space and can thus overview a larger information context. Our focus is to realize Web applications on an extended display system without any modification to Web applications.

The usability of large displays is discussed in the paper [6]. One of large-display usability issues is losing the mouse cursor. As screen size increases, users can lose the mouse cursor. A system that can be used as a pseudo large display by using multiple tablet devices with touch-screen has a high usability be-

cause it has no mouse cursor. Lyons et al. studied about collocated groups of individuals using multi-display composition on two different types of mobile computers[7].

Our aim is to develop a new approach to displaying a Web application on extended displays without any modification to Web applications. Existing researches do not focus on issues particular to execute a dynamic Web application on extended displays of multiple tablet devices, which requires coordination techniques on a JavaScript event detection mechanism among separated Web browsers.

Fig. 1. Example of an extended Web displaying system

Figure 1 shows an execution example of our extended Web displaying system. In the example, two tablet devices are used for viewing a single Web page. Users operations for one tablet device influence the other in real time. When a user zooms in, zooms out, scrolls, and follows links on the left device, the system automatically redraws the displayed content on the right one in synchronization with the left one. When a user interacts with the right one, the displayed content on the left one is automatically redrawn by the system.

The rest of the paper is organized as follows. In Section 2, we explain the extended Web displaying system based on multiple tablet devices. In Section 3 and Section refOperational Synchronous Message, we show two types of synchronous messages: a positional synchronous message and an operational synchronous message, and explain an implementation of a synchronization mechanism for Google Maps. In Section 5, we show applications of the system. In Section 6, we discuss the advantage of the system, and review related works and systems in the area of display systems for collaborative works. Finally, we conclude the paper in Section 7.

2. An Extended Web Displaying System

An extended Web displaying system is defined as a pseudo display that consists of multiple displays. Our system is also designed to support collaborative working on users' tablet devices. Although the systems proposed in the paper [6,8] require a large display, our system requires no large display because users' tablet devices become a pseudo large display by using the system.

2.1. Management Methods of Device Placement Information

Device placement information is extremely important to use many devices as one large display system. All tablet devices need to have some coordination method to adjust their coordinate system of displays to realize a pseudo large display because the tablet devices do not share one absolute coordinate. For example, how many devices are used in all, how many rows and columns are in the device matrix, where is the device in the device matrix, etc. In this paper, a position of the device is written in (row, col) format.

We need support methods for making a device matrix because it is quite time-consuming. We adopted three ways to make a device matrix for detecting placement information (1) using an acceleration sensor, (2) using a Pair Swipe gesture, and (3) using a management device.

(1) Using an acceleration sensor is based on collision detection by using acceleration sensors of two tablet devices. Hinckley have described a pairing method by using built-in acceleration sensors of tablet devices. Signals available from acceleration sensors when two devices are in collision are used for pairing in Hinckley's method[9].

(2) Using Pair Swipe is based on a gesture recognition for pairing two tablet devices. Paring means that two adjacent displays share their own coordinate system. Suzuki et al. proposed a method to

generate a device matrix by a swipe gesture between the adjoining devices with touch screens. The operation is called "Pair Swipe"[10].

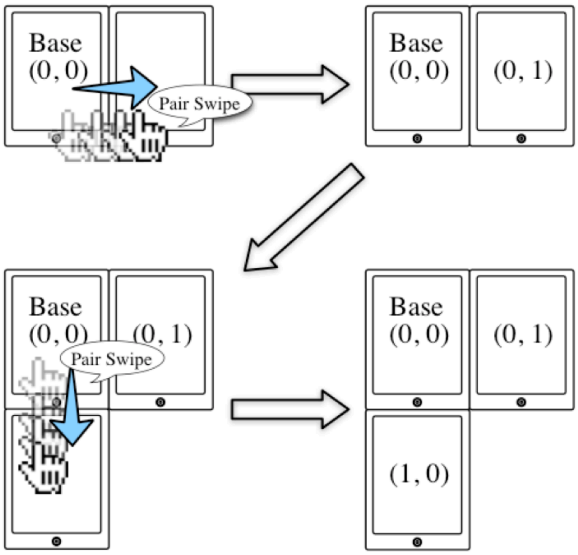

Fig. 2. Making a device matrix by using Pair Swipe

An example of Pair Swipe is shown in Figure 2. This example shows the process of making a device matrix of three tablet devices by using swipe gestures. The three tablet devices in Figure 2 do not share any device matrix initially. In the first step shown in the top-left of Figure 2, a user do a left-to-right swipe gesture from the left device to the right device, then the two devices share the device matrix including (0, 0) and (0, 1) as the top-right of Figure 2. In the second step shown in the bottom-left of Figure 2, a user do a top-to-bottom swipe gesture from the top-left device to the bottom device, all the devices share the device matrix including all of them as the bottom-right of Figure 2.

(3) Using a management device is based on a helper device to determine a device matrix. An example of a management device is shown in Figure 3. In this example, we can make a device matrix for detecting placement information by using a smartphone on the left side in Figure 3. There are one smartphone and four tablet devices in Figure 3. Firstly, the user input the number of tablet devices into the smartphone, then the smartphone displays placement information of the four tablet devices and the tablet devices display different numbers (1-4). Next, a user arranges the position of the four tablet devices on a grid pattern (2 by 2 in in Figure 3) as the placement information displayed on the smartphone. Now the user can assign positions to tablet devices by using the smartphone. Consequently, the smartphone and the four tablet devices can share the device matrix including all the tablet devices.

Fig. 3. Using a management device to make a device matrix

2.2. *System Architecture*

We show the system architecture of the extended Web displaying system in Figure 4. Our system is composed of seven parts: a user interface, an acceleration sensor, a Web browser, a Web content synchronizing module, a placement information (PI) management module, and a PI determining Module. The PI determining module contains a PI mode module, a device orientation module, a pair swipe PIDM, an acceleration PIDM, and a management device PIDM.

We can interact with the system by the user interface. The Web browser gives information such as URLs, scroll offsets, and zoom levels. Hereafter we define these types of information as Web content information. The Web browser displays a Web

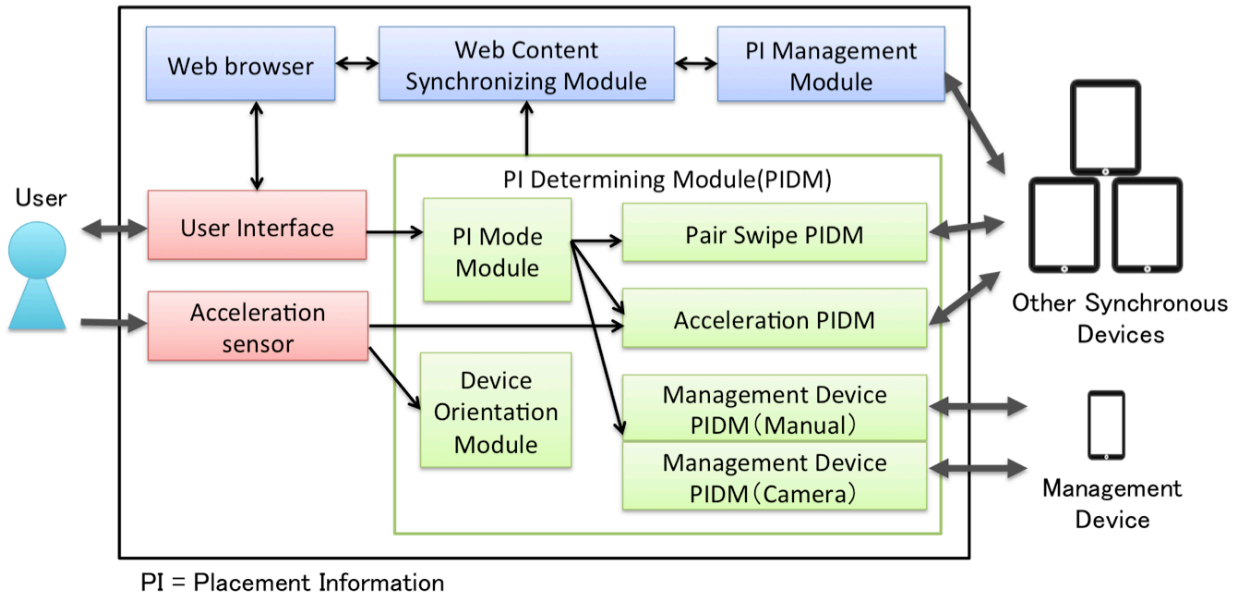

Fig. 4. System architecture

page, and exchanges Web content information with the Web content synchronizing module. The Web content synchronizing module reflects the Web content information that is received by the PI determining in displaying Web pages or scroll offsets. The PI management module exchanges Web content information between other devices. The module has information of devices to synchronize. So the module controls a group of the resulting large display.

The PI determining module (hereafter referred to as the PIDM) connects a display to a management device or other devices, and deals with the placement information that is generated by the process. Depending on the way in which to determine placement information, the PIDM decide a module to determine placement information. The pair swipe module decides a connection by swipe operation, and the acceleration PIDM deals a connection by the acceleration sensor, and the management device PIDMs deals a connection by management device. Furthermore, two management device PIDMs are prepared: Manual module that determines placement information by users with the management device, and camera module that determines placement information by using cameras. The pair swipe PIDM and the acceleration PIDM receive placement information from communication with other devices, and the management device PIDMs receive placement information from communication with management devices. We can select which PIDMs use by PI mode module. The device orientation module receives the orientation of the device from the acceleration sensor and other PIDMs use the information to determine placement information.

2.3. *Sending Synchronous Message*

The Web content synchronizing module checks users' interaction. A device touched by a user, which is called as an "operator" in this paper, sends a synchronous message to other devices.

Synchronous messages must be sent to all devices in the device matrix. A method to manage IP addresses of all devices in an integrated fashion is a cumbersome procedure because the IP addresses table must be updated every time when devices increase or decrease in number.

UDP broadcast is adopted in the system to send synchronous messages. The broadcast is sending packets to all nodes in a local area network. An operator sends a synchronous message to a broadcast address and all devices receive the message and re-

fresh their screen based on the message.

In this system, we implemented two types of synchronous messages: a positional synchronous message that synchronizes information of scroll orientations of Web pages, and an operational synchronous message that synchronizes information of user operations. The procedures of two types of messages are shown in the later sections.

3. Positional Synchronous Message

The positional synchronous message realizes synchronization among displays by using display location and position of Web pages. It contains information of a url, a scroll offset, and a zoom level. A method that exploits this synchronous message can use widely for generic static Web pages without specific settings. But the method cannot be adapted to dynamic Web pages that cannot synchronize Web pages by only location and position information.

3.1. *Structure of the Positional Synchronous Message*

The positional synchronous message contains five attributes listed in Table 1. We will describe these attributes in detail.

Table 1. Structure of the Positional Synchronous Message

attribute	description
ip_addr	an IP address of the sender, which is a machine that sent the message
date	a UNIX time when a message was sent
url	a URL of a displayed content
scale	a zoom level of a displayed content
(*scrl_x, scrl_y*)	a scroll offset, x and y coordinate

The attribute *ip_addr* is the operator's IP address, which is used to identify to reject received own synchronous messages. UDP broadcast messages reach to the device to which the message was transmitted. It is not needed to synchronize based on own synchronous messages. Devices that received the message determine if the message was sent from other devices or not by using the *ip_addr* attribute.

The attribute *date* is the time when the message was sent by the operator, which is in UNIX time. The *date* is used to synchronize based on the newest message. A UDP header does not have a sequence number. So, when two messages are sent by UDP, the second message will reach before the first one's arrival. UDP does not guarantee an order of a sequence of packets. It means that UDP does not promise $rt(mt) < rt(mt+1)$, and often $rt(mt) > rt(mt+1)$. Here, mt indicates the synchronous message that is sent on time t and $rt(mt)$ indicates the time when another device received mt. Redrawing based on mt after redrawing based on $mt+1$ will take place mis-synchronous between multiple devices. The synchronous messages must contain the message sent time and the system redraws their contents based on newest messages by using the time attribute.

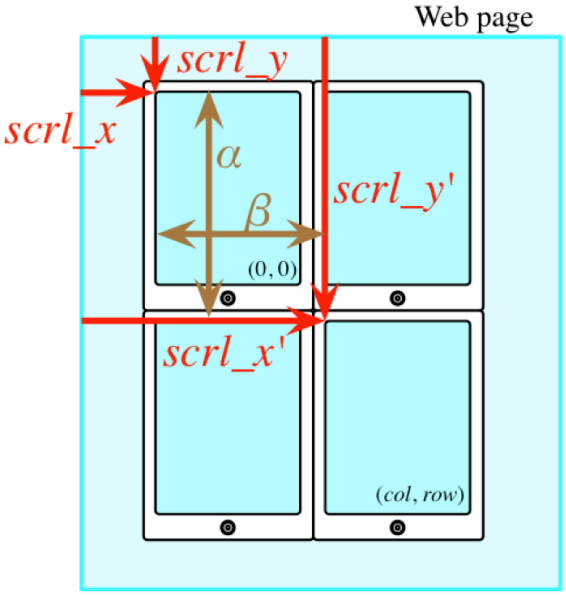

Fig. 5. Transform to coordinates of the device (0, 0)

The attribute *url*, *scale*, and (*scrl_x, scrl_y*) describe the state of displayed contents on the operator. The *url* is a content URL that is displayed on the op-

erator. The *scale* is zoom level of the displayed content on the operator. The (*scrl_x, scrl_y*) are x and y coordinates of a scroll offset on the operator. The offsets must be transformed to coordinates based on the device on at the position (0, 0) as shown in Figure 5.

When an operator is the device on position (col, row) and the content offset on the device are (*scrl_x', scrl_y'*), the (*scrl_x, scrl_y*) parameters are calculated by the formula below. In this formula, α and β are the height and width pixel count on the device screen respectively.

The system repeats sending messages to avoid packet loss. We conduct a preliminary experiment to determine the number of times messages to send, and the results show that messages should be sent in five repetitions.

3.2. Synchronization Algorithm

We explain a synchronization algorithm using positional synchronous messages. Figure 6 shows the procedure *receivePositionalSyncMessage* that is called when a positional synchronous message has reached a device. The left numbers on the code are line numbers. Those that follow are the same.

procedure *receivePositionalSyncMessage(mes)*
mes : positional synchronous message
$\langle ip\_addr, date, url, scale, scrl\_x, scrl\_y \rangle$

This device is disposed on (col, row).

```
1:  if (ip_addr ≠ own ip addr) ∧ (date > last_sync_date) then
2:    last_sync_date ← date
3:    if url ≠ current url then
4:      moveTo(url)
5:    end if
6:    if scale ≠ current scale then
7:      setZoomScale(scale)
8:    end if
9:    scrl_x' = scrl_x + row · α
10:   scrl_y' = scrl_y + col · β
11:   if (scrl_x' ≠ current scrl_x) ∨ (scrl_y' ≠ current scrl_y) then
12:     scrollTo(scrl_x', scrl_y')
13:   end if
14: end if
```

Fig. 6. The procedure that is called when the system receives a synchronous message

The line 1 in Figure 6 determines whether the arrived message was sent from self-device or not, and the message is the newest message or not. The last_sync_date must be allocated as a global variable while the system running. When the message was sent from other devices and it was the newest message, the process for synchronization (after the line 3 in Figure 6) begins.

The lines 3-5 in Figure 6 are the process to show the same contents on all devices. If a URL of content displayed on the device is different from displayed on the operator, content will change to the one that is displayed on the operator.

The lines 6-8 in Figure 6 are the process for zoom scale synchronization. If the zoom scale of a device is different from the operator, the zoom scale of the operator will be set to the device.

The lines 9-13 in Figure 6 are the process for scroll offsets synchronization. The parameters in a message are based on the device (0, 0), so they are transformed to coordinates based on the device that the message received. The formulas for the transformation are shown in the lines 9-10 in Figure 6.

4. Operational Synchronous Message

The operational synchronous message realizes synchronization among displays by using information on users' touch, keyboard, and censor events. In this method, the system detects operational events at the operator and generates the same events by sending them. The method that uses this synchronous message can use for dynamic Web pages whose content vary based on parameters provided by a user or a computer program.

4.1. Sending an Operational Synchronous Message

The system collects the events of users operation by monitoring touch operation, or keyboard input operation occurring on Web pages. The system executes JavaScript in a Web page on the browser, and event occurrences can be informed of the system. The DOM events for which the system listens are shown in Table 2.

Table 2. The types and trigger timing of DOM events

type	trigger timing
touchstart	a finger touched a DOM element
touchend	a finger moved on a DOM element
touchmove	a finger released from a DOM element
keypress	a key pressed on a DOM element

The type *touchstart*, *touchend*, and *touchmove* are events occurred by touch events. Moreover, *keypress* is an event triggered by a key press.

Table 3. The structure of the operational synchronous message

attribute	description
ip_addr	IP address of the sender, which is the machine that sent the message
date	UNIX time when the message was sent
type	operation type of an event
property	unique property of an event type

Detected events by the system are sent as operational synchronous messages to other devices. The structure of the synchronous message is listed in Table 3. The attribute *ip_addr* and *date* are the same as shown in Table 1. The attribute *type* is the identifier of the operation event by users. The attribute *property* is a set of attributes of the event type.

Examples of event properties are listed in Table 4. When the *touchstart*, *touchend*, and *touchmove* event are triggered, the events contain x and y coordinate of the finger. As the *keypress* event is triggered, the event has its key code.

Table 4. An example of event properties

type	property
touchstart	x and y coordinate of a finger
touchend	
touchmove	
keypress	the key code of a key pressed

4.2. Synchronization Algorithm

Figure 7 shows the procedure *receiveOperationalSyncMessage* that is called when an operational synchronous message has reached a device.

```
procedure receiveOperationalSyncMessage(mes)
mes : operational synchronous message
      ⟨ip_addr, date, type, property⟩
 1: if (ip_addr ≠ own ip addr) ∧ (date > last_sync_date) then
 2:   last_sync_date ← date
 3:   if type is "touchstart" then
 4:     touchdown ← true
 5:     ⟨x_s, y_s⟩ ← getLocalPoint(property)
 6:   else if type is "touchmove" then
 7:     touchdown ← true
 8:     if touchdown then
 9:       ⟨x, y⟩ ← getLocalPoint(property)
10:       swipeTo(x, y)
11:     end if
12:   else if type is "touchend" then
13:     ⟨x_e, y_e⟩ ← getLocalPoint(property)
14:     if touchdown ∧ ⟨x_s, y_s⟩ = ⟨x_e, y_e⟩ then
15:       tapPage(x_e, y_e)
16:     end if
17:     touchdown ← false
18:   else if type is "keypress" then
19:     keycode ← getKeyCode(property)
20:     doKeypress(keycode)
21:   else
22:     do the default actions
23:   end if
24: end if
```

Fig. 7. The procedure that is called when the system receives an operational synchronous message

At first, as is the case with positional synchronous messages, the line 1 in Figure 7 determines whether the arrived message was sent from self-device or not, and the message is the newest message or not. When the message was sent from other devices and it was the newest message, the procedure executes different action, depending on each event type.

The lines 3-17 in Figure 7 are the process to check the touch event. When the *touchstart* event is detected, the variable *touchdown* that indicates

whether users are touching a display or not is assigned true, and the *getLocalPoint* function assigned the local coordinates where the user touched at the time to the pair of variables $\langle x_s, y_s \rangle$. Similarly, when the *touchend* event is detected, the pair of variables $\langle x_e, y_e \rangle$ is assigned the local coordinates where the user touched. At this time, if $\langle x_s, y_s \rangle$ and $\langle x_e, y_e \rangle$ are the same coordinates, the *tapPage* function is called and the Web page is tapped at the coordinates by JavaScript codes. When the *touchmove* event is detected, the pair of variables $\langle x, y \rangle$ is assigned the local coordinates where the user touched at the time. The *swipeTo* function generates swipe event from the current coordinates to the coordinates $\langle x, y \rangle$ in the Web page.

The lines 18-20 in Figure 7 are the process to check the keyboard event. When the *keypressed* event is detected, the *getKeyCode* function assigned the pressed *keycode* to variable *keycode* from property attribute. The *doKeypress* function generates *keypressed* event that presses the key of *keycode* in the Web page.

4.3. Implementation for Google Maps

We explain our implementation of operational synchronization for Google Maps on iOS. A method to extract and reproduce operations on Google Maps consists the following three steps.

1. Making a special HTML file to handle a *native scheme* for Google Maps
2. Extracting and broadcasting *scroll events* on Google Maps
3. Reproducing the map scroll on Google Maps on receivers

1. Making a special HTML file to handle a *native scheme* for Google Maps is needed to communicate between the synchronization mechanism and UIWebView*. The page contains the following JavaScript code, which is a program to extract and notify scroll operations on Google Maps.

```
01: function initialize() {
02:   var lat = 35.156232;
03:   var lng = 136.924574;
04:   var mtid = google.maps.MapTypeId.ROADMAP;
05:   var latlng =
06:     new google.maps.LatLng(lat,lng);
07:   var opts = {
08:     zoom: 15,
09:     center: latlng,
10:     mapTypeId: mtid
11:   };
12:   var elemid = "map_canvas";
13:   var canvas =
14:     document.getElementById(elemid);
15:   map = new google.maps.Map(canvas, opts);
16:   google.maps.event.addListener(
17:     map,"drag", function(){
18:       var y = map.getCenter().lng();
19:       var x = map.getCenter().lat();
20:       var url =
21:         'native://dragPoint/' + x + ':' +y;
22:       var iframe =
23:         document.createElement('IFRAME');
24:       iframe.setAttribute('src', url);
25:       document.
26:         documentElement.appendChild(iframe);
27:       iframe.parentNode.removeChild(iframe);
28:       iframe = null;
29:     }
30:   );
31: }
```

The `initialize` function is called on loading the special Web page. The Web page contains a DIV element with id `map_canvas`. The page displays maps on the DIV element `map_canvas` by using the Google Maps API.

The `initialize` function consists of (1) setting an initial position (the lines 2-15) and scale (the lines 16-30) and (2) making a hook to communicate between the synchronization mechanism and a Google Maps object in UIWebView. The second step is the key point of the implementation. The `addListener` function in the line 16 is a function to add the hook function (the lines 18-30) to a Google Maps object. The hook function is called when a user drags the map.

2. Extracting and broadcasting *scroll events* on Google Maps is implemented as the hook function. The hook function consists of (2-1) extracting the

* The UIWebView is a Web page rendering engine on iOS.

current location of the map by using the Google Maps functions (the lines 18-19), and (2-2) sending the location to the synchronization mechanism (the lines 20-28). The key point is url in the lines 20-21, which is a special url to send the current location to the synchronization mechanism. The syntax of the url is the following.

`native://dragPoint/`*lat*`:`*lng*

The key point is using `native` scheme in this url, which can be used to identify the event as scroll events. The `http` or `https` schemes are detected on page transitions. The *lat* and *lng* are the current latitude and longitude respectively.

The system sends the current position to the synchronization mechanism by using an `IFRAME` element (the lines 22-28). The program invokes a `GET` request of HTTP on the UIWebView. Then the synchronization mechanism receives the `GET` request and checks if the request is a `native` request or not. If the request is a `native` request then the mechanism broadcasts a scroll synchronization message, which contains the scroll offset on the map.

3. Reproducing the map scroll on Google Maps on receivers is described below. When the synchronization mechanism receives the broadcast of the scroll synchronization messages, it reproduces the map scroll by creating and executing the following JavaScript on the UIWebView to reflect the scroll on its map object.

`map.panTo(new google.maps.LatLng(`*lat*`,`*lng*`))`

The *lat* and *lng* in the script are the current latitude and longitude respectively. This code simulates user's scroll operation to move to (*lat, lng*) on the map.

5. Applications

We show two applications based on the proposed extended Web displaying system, a piano score system and a map viewer system.

The first application shown in Figure 8 is a piano score system. This application can show many scores to pianists by placing some tablet devices horizontally on their piano. The system can display a large image of scores on a Web page. In general, pianists put many scores on the piano horizontally because they have to watch them at a time, but a large display is too heavy to be placed on the piano.

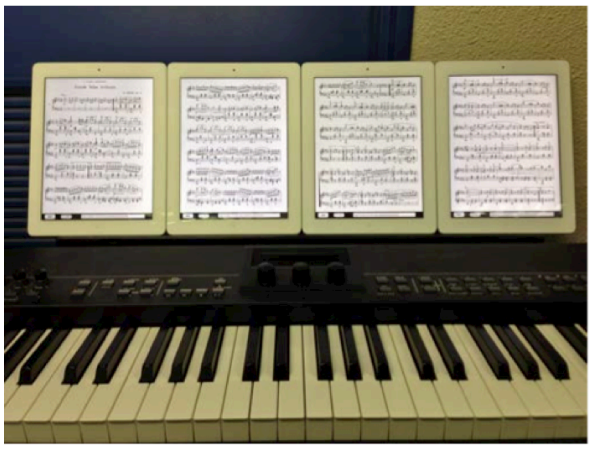

Fig. 8. Piano score system

The second application shown in Figure 9 is a map viewer. The system can display Google Maps on one integrated display with multiple iPads by using the method described in Section 4.3. Maps must be shown on a large display with high-resolution because maps have large amount of information[11].

Users can interact any devices. If a user scrolls the map on one of the four devices in Figure 9, the maps on the rest three devices also scroll in synchronization with the interacted device. When multiple users scroll different devices at the same time, the system exclusively processes the operations on a first-come-first served basis.

Fig. 9. Map viewer system

6. Discussion

There are several different display systems for collaborative works by using tablets or handheld devices. Cheng et al.[8] proposed a system that supports the use of tablet devices for interaction and collaboration with large displays. Users can interact with a subset of the large workspace on their tablet, while the same area is visualized on the large display as a rectangular frame. Liu et al.[12] have proposed a shared display groupware and explore whether the use of shared displays in classrooms can augment the handheld devices and enhance the effectiveness of handheld devices in promoting communication among participants. Baur et al.[13] proposed virtual projection which enables users to transfer data from their phone to a large screen and display it thereon. The ConnecTable[14] is an example of a hardware and an application that use BEACH[15] that is a CSCW system. When users connect two ConnecTable systems, the two display areas become a shared space automatically and users can move windows between two displays by stylus pens.

7. Conclusion

We proposed the extended Web displaying system based on multiple tablet devices to help users to conduct collaborative works anywhere without a large display. Users can use multiple tablet devices as a pseudo mobile large display by using our system.

We showed how to cooperate multiple tablet devices to display one Web content without any inconsistency. Furthermore, we explained three ways to determine the positions of the devices: an acceleration sensor, a pair swipe, and a management device. We proposed the two types of synchronous messages, the positional and operational synchronous messages, and their structures and algorithms. We described two applications, a piano score system and a map viewer system.

In the future, we would like to establish the evaluation method based on user manipulation types from the basic window manipulation, such as scaling, moving, and scrolling, to more complex manipulation and evaluate the proposed method.

References

1. S. Imai, S. Shiramatsu, T. Ozono, T. Shintani, "On a Synchronizing Module for Extended Web Displaying System Based on Multiple Tablet Devices," *In Proc. of SNPD2013*, 669–674 (2013).
2. Distributed Multihead X Project, "Distributed Multihead X," http://dmx.sourceforge.net/
3. Bartels Media, "MaxiVista," http://www.maxivista.com/
4. M. Ueda, I. Takeuchi, "Tenmads: A Software Distributed Multi-display Implementation for Practical and Low-cost Applications," *Software Technologies for Future Dependable Distributed Systems*, 195–199 (2009).
5. J. Seifert, D. Schneider, E. Rukzio "Extending Mobile Interfaces with External Screens," *In Proc. of INTERACT 2013*, LNCS 8118, 722–729 (2013).
6. G. Robertson et al., "The Large-Display User Experience," *IEEE Computer Graphics and Applications archive*, Vol.25 No.4, 44–51 (2005).
7. K. Lyons, T. Pering, B. Rosario, S. Sud, R. Want "Multi-display Composition: Supporting Display Sharing for Collocated Mobile Devices," *In Proc. of INTERACT 2009*, LNCS 5726, 758-771 (2009).
8. K. Cheng, J. Li and C. Müller-Tomfelde, "Supporting Interaction and Collaboration on Large Displays using Tablet Devices," *In Proc. of AVI'12)*, 774–775 (2012).
9. K. Hinckley, "Synchronous Gestures for multiple Persons and Computers," *In Proc. of UIST 2013*, 149–158 (2003).
10. R. Suzuki, S. Shiramatsu, T. Ozono and T. Shintani, "On an Implementation of a Smart Signage System based on Tablet Devices," *Journal of the Japan Society for Software Science and Technology on Computer Software*, Vol.30 No.2, 2_176–2_190 (2013).
11. R. Ball , M. Varghese , B. Carstensen , E. Dana Cox, C. Fierer, M. Peterson, C. North, "Evaluating the Benefits of Tiled Displays for Navigating Maps," *In Proc. IASTED-HCI '05*, 66–71 (2005).
12. C. Liu and L. Kao, "Handheld Devices with Large Shared Display Groupware: Tools to Facilitate Group Communication in One-to-One Collaborative Learning Activities," *In Proc. of WMTE2005*, 128–135 (2005).
13. D. Baur, S. Boring, S. Feiner, "Virtual projection: exploring optical projection as a metaphor for multi-device interaction," *In Proc. of CHI 2012*, 1693-1702 (2012).
14. P. Tandler C. Müler-Tomfelde, N. Streitz, and R. Steinmetz,,"ConnecTables: Dynamic Coupling of Displays for the Flexible Creation of Shared Workspaces," *In Proc. of UIST01*, 11–20 (2001).
15. P. Tandler. "Architecture of BEACH: The software infrastructure for roomware environments," *In CSCW 2000 Workshop on Shared Environments to Support Face-to-Face Collaboration*, 2–6 (2000).