

Web contents recognition and seamless movement in multiscreen environment

Sangmin Park, Hyeontaek Oh, Sanghong An, Jinhong Yang, Hyojin Park, Junkyun Choi

Department of Electrical Engineering

KAIST

Daejeon, Republic of Korea

{deadpk, hyeontaek, ancom21c, sunupnet, gaiaphj1, jkchoi59}@kaist.ac.kr

Abstract—Nowadays, more and more users are surfing web pages using smart devices, such as smart phone, tablet PC, and smart TV. Those devices have some limitations, so that users want to utilize multiple devices to get improved user experience. Multi-screen environment involves web contents sharing, moving, controlling, and managing, and implementing multi-screen web page incurs several problems, which are distinct from conventional web page implementation. Implementing multi-screen environment is not a problem of a single browser, but of multiple browsers, so it requires some kind of browser to browser communication. Actually there are many ways to achieve the multi-screen communication, but we have tried to use only web standard based techniques. Besides implementation, we pick out and describe technical issues and tactics, and also multi-screen scenarios are investigated to see how multi-screen environment can be utilized usefully.

Index Terms—content recognition, session control, seamless movement, multiscreen

I. INTRODUCTION

More and more people are using not only single device, but multiple devices. The devices are smart phone, smart pad, or tablet PC, so the people can use them for various purposes. Web searching and surfing are one of the important purpose. As the devices get smaller, people are surfing web sites or searching web pages more often. We can get numerous services and information, let us to follow up information centric world. Web pages, in other words, are much closer to us than ever.

Those web page consists of many subareas based on their contents; for example, news web page can have text area, video area, related news area, and people's opinion area. In that case, users may want to select, copy, zoom in and out, move, and share a certain area of the web page for their demand and convenience. Selecting, copying, and zooming areas can be performed by using re-rendering of web browser, but the others are not. Those functions, sharing and moving areas, do not happen at only one web browser, but involves a couple of browser; so additional managements are required. This is what we call multiscreen environment.

Users use the web pages and services for various purposes. The user might want to do series of things; such as reading news article, navigating other articles, leaving his opinion, or checking related pages. It is hard to do those things with only one device; it have limited screen size and uncomfortable

interfaces. For those reasons, it is required to select, move, copy, or share a certain area of web pages among multiple devices, which is called multi-screen environment.

In this paper, we suggest web based framework to provide web content area recognition and seamless movement among web browsers, and actually implemented prototype of the technique. With content area recognition users can specify what content can be moved and then users can share or move it. This framework only use web standard so that it can be deployed to any web browsers as long as the browser sticks to the web standard. In that sense users do not need to suffer from above limitations or some kind of burdens.

To support moving and sharing web contents among browsers in multiple devices, several technical approaches can be applied. First approach is installing native application to the device. Second one is providing different access address for user, and the last one is using hardware solutions such as DLNA standard. Each approach has additional implementation issues and some limitations. For the first case, we need to implement native application for each device platform. For the second case users should manage content addresses in their own hand, and the last case has hardware dependency.

For apple cloud service [?], we can have multi-screen like service. We store media files on cloud, and they are can be accessed by several Apple devices. The media services, however, does not support screen to screen communications. It only provide media play from stored media on cloud storage. It implies the service is not seamless nor screen to screen service.

Another case, Digital Living Network Alliance (DLNA) [?], uses Universal Plug and Play (UPnP) for media management, discovery and control. UPnP defines the type of device that DLNA supports and the mechanisms for accessing media over a network. The DLNA guidelines then apply a layer of restrictions over the types of media file format, encodings and resolutions that a device must support. DLNA gives seamless media movement among devices, but it requires appliances to be DLNA certified.

II. BROWSER TO BROWSER COMMUNICATION

To implement seamless multi-screen environment, the contents' data should be delivered from one browser to another browser.

It is important that where the contents came from and where to go to, so the session information should be managed. It is session control.

Then the actual contents' data should be transmitted among multiple browsers, and it's called data transmission.

A. Direct communication

The other method is using direct browser to browser communication; WebRTC [?] and WebSocket [?].

1) *Web real-time communications (WebRTC)*: There are a number of proprietary implementations that provide direct interactive rich communication using audio, video, collaboration, games between two peers' web browsers. IETF and W3C collaborates to make standards about WebRTC (Web Real-Time Communication). They produce architecture and requirements for selection and profiling. They also identify state information and events and define APIs.

2) *WebSocket*: WebSocket is a web technology providing for bi-directional communications channels over a single TCP connection. The WebSocket API is being standardized by the W3C, and the WebSocket protocol has been standardized by the IETF as RFC 6455.

B. Indirect communication

In contrast to direct communication, browser does not communicate with the other browser directly. The browser need to connect to server in order to transmit data to the other browser. The server is responsible for transmitting data from one browser to the other browser, so it mediate browser communication. It is basically client-server model, which is currently available implementation method.

1) *Session control*: For discovering devices and identifying devices, browsers should keep the session information. Since the HTTP [?] is stateless protocol, the multi-screen data should be managed with a session concept.

Actually the session is used and stored for various web sites in several purposes; Log in information, control options, page information, device data, and so on. Those sessions are distinguished among browsers or client IDs, so that the server can serve client aware services. In that sense, we can use the sessions to characterize client information, client web page, web content use, or page data. Multi-screen environment should keep track of those page and client information continuously, and server need to manage those data for browser communication.

2) *Data transmission*: Once a session control is over, the actual contents' data should be delivered between browser. There are several ways to implement the server and client, but it need to be based on available server client communication.

Comet is a method for communication between server and client for some continuous communication. It can be used to implement HTTP streaming, or HTTP long polling. The client request last long so the server and client can interact to each other as long as the link open. For these reasons, comet is used for server push service, but have limitation; such as buffering or browser constraints.

Another method is using Socket.io. It is not a single transmission method, but a series of. It involves Flash socket, JSONP polling [?], XHR polling [?], and so on, so it checks that a certain browser support the transmission connection. It serves unified API so the developer do not need to worry about the connection types.

III. COMPOSING MULTI-SCREEN SUPPORTING WEB PAGE

Once a browser to browser communication is established, the web page should have multi-screen supporting web contents. The content type should be defined in advance, and the transmission data have to be compromised. We cannot serve arbitrary content to be moved from one browser to the other browser, since we cannot predict the internal data. There are several ways to implement those web pages.

A. Framework level composition approach

The framework level supervised web page composition can be one approach to implement multi-screen web page. The framework specify which web area can be moved, and which element should be involved. Its kind of template engine, which we can use HTML web page easily. Swig, RoR, or Play framework are the examples of template engines, and they enable developers can build their own web pages in a simple way.

Like the case, we can implement framework, and the framework already specify the movable contents, and data to be transmitted. This scheme gives simple approach for implementing multi-screen web page and may takes relatively low effort.

It has, however, limitations obviously. The effort for building framework is main key; the framework should have various element, catch exceptions, specify data transmission, and synchronize the browser to browser data. Developer might suffers from limited object, and the home page likely to be simple or constrained form.

B. CSS level composition approach

One of the fundamental W3C standards for developing web applications is cascading style sheets (CSS) [?]. CSS is a language for dening the presentation semantics of HTML elements, including their positioning, layout, colour, and fonts. The main driving force behind adopting CSS has been the separation of structure from presentation.

In CSS the most important, widely supported and heavily used simple selectors are the element selector, the class selector and the id selector. An element selector picks all HTML elements with a certain name, while the class and id selector pick HTML elements by their class or id attribute respectively. An element selector is specified by the name of the element only, while the prefix "." identifies a class selector and the prefix "#" an id selector.

For content area recognition combination of jQuery library and JavaScript are used. Using DOM parsing and jQuery selectors lets us to access content object easily, since they provide structured and hierarchical access method to DOM object.



Fig. 1. Screen UI autofitting

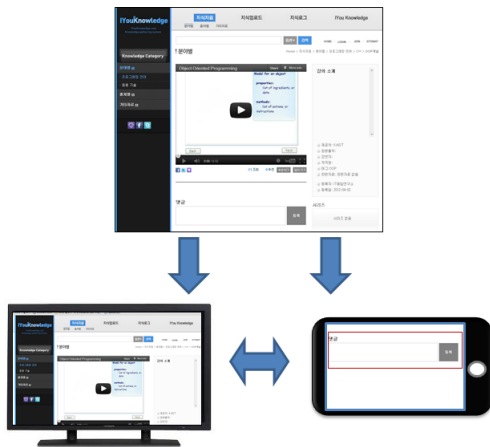


Fig. 2. Content synchronization

IV. REQUIREMENTS FOR MULTISCREEN ENVIRONMENT

We have covered some technical issues for implementing multi-screen environment. In practical case, it is not enough; there are several implementation issues and considerations to make multi-screen environment. For the first, we need to pick out what areas can be moved. That is considered as a content recognition, so that we can pre-define what data and information should be delivered. For the next, we need to provide some continuity for seamless content movement. They are described below for more detail.

A. Content recognition requirements

Devices have different size of display and different resolution, so web page can be rendered differently based on device screen size. Fig. ?? shows that how a web page is displayed on various type of screen.

Fig. ?? shows that one web page can be displayed in two (or more) displays at the same time. In this example, user can see content through bigger screen (i.e. TV, monitor), and at the same time, user can do interaction with content through smaller screen (i.e. smart phone). To support this functionality, devices should be synchronized in real-time.

B. Content movement requirements

When a content is moved through browser communication, the content should have same context in the other side. It is called web continuity. Surely, the web content technology should provide the web continuity.

1) *Web application/service continuity*: Users access to the web pages and get services from them. This is called web service, and is accessed through various environment. The web services may involves several technologies, such as SOAP, WSDL, UDDI, and so on, and the clients utilizes those things to communicate to the other computer on a different network. Those standards, which specifying web services and interfaces, can be used to implement component based web application, and the service also provide services to other applications, which uses HTTP, XML, or SOAP. This kind of scheme promise web service connection and integration, and we call it web application/service continuity.

2) *Web media continuity*: Web media continuity is a continuity that a multimedia file can be played over several devices seamlessly. It involves some session managing and media file transferring, so that a user can watch the media without discontinuity. In that sense, when if a user is watching movie with his smart phone, he can watch continuously with a smart TV. Related technologies are various, and DLNA and HTTP live streaming are general.

DLNA uses UPnP protocol for transferring media among devices, and is applied to several appliances. A device discovers and access to a router, then it can detect other DLNA support devices on the same network area. Media data transferring and playing is available among DLNA support devices, so many company are support DLNA. HTTP live streaming is another technique which utilize HTTP protocol for transferring media file. While conventional streaming service requires RTSP/RTP connection [?], HTTP live streaming uses only HTTP. It is crucial feature; RTSP/RTP may incur firewall problem, or start up cost.

V. SYSTEM DESIGN

We think of an education system, and the user are trying to see those contents in a multi-screen environment. In other words, a user is going to move subareas, or content, from one device to the other device in a seamless manner.

Object mover has contents to be moved. In object mover side, user decide which content or area should be moved. Object listener is another device and may get the moved content. These object mover and listener are not fixed, so that one device can be a object mover or object listener; it is a bi-directional content movement.

Once a object mover decide which content is going to be moved to object listener, then the content move message is sent to session manager server, which is crucial part of our system. The session manager server controls and synchronizes mover and listener so the listener can get content from mover seamlessly. Session manager server contains and maintains the session information of connected devices. The session information is about browser and device information, and

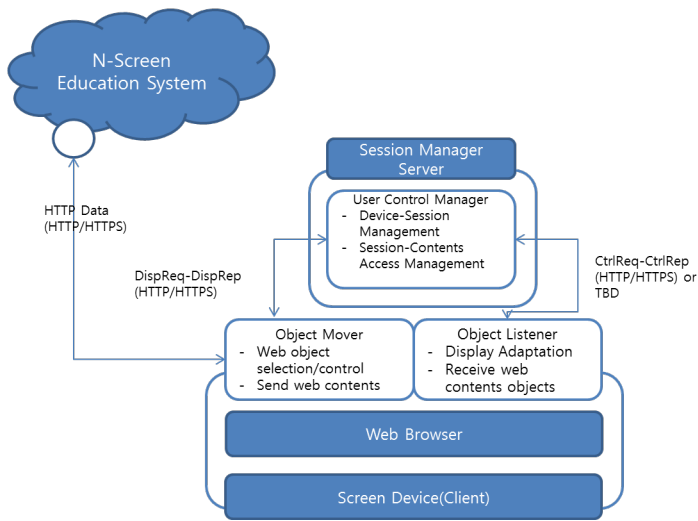


Fig. 3. System diagram

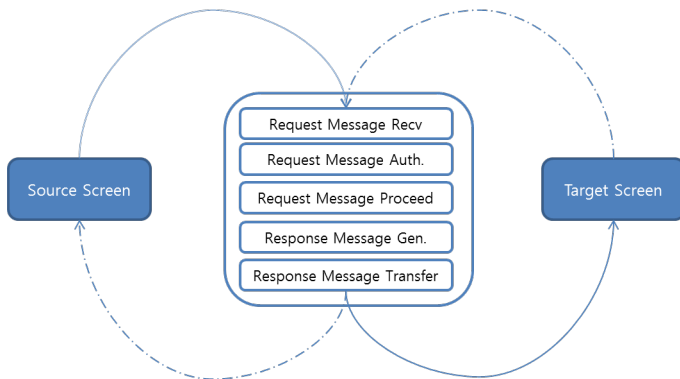


Fig. 4. Message request and response

content use data, so that the manager incorporate device recognition and seamless movement schemes.

At object mover side in Fig. ??, user selects and controls web content that moves to other web browser. If the User control manager is notified that a content is going to be moved, or shared to object listener, then it gets content information from object mover. Delivering data to object listener, session manager server Object listener receives events from session manager server.

This system uses client/server model for controlling services. The session manager server is connected to both mover and listener, so that it gets request and response from them. Each request might be a content moving, or sharing request, and the response might be content data, which is going to be shared or moved. The requests and responses occur continuously so the synchronization maintains.

The messages are mediated by session manager, which is located between object mover and object listener. As described in Fig. ??, a request for content movement is sent to session manager. If the request is identified, then it ready to send content data to target screen. If the content is delivered successfully, then session manager continuously send message

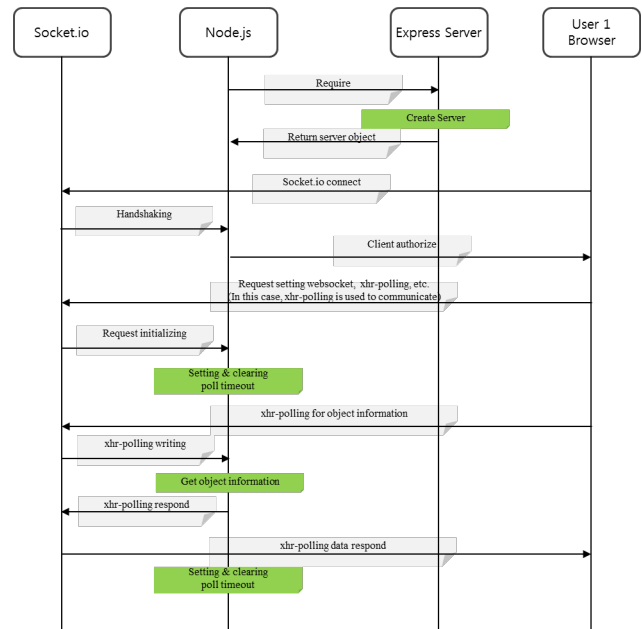


Fig. 5. Content recognition flow

among devices to synchronize the content data.

VI. IMPLEMENTATION

To implement server and client, we use Socket.io [?] and Node.js [?]. Node.js is for server side program coding and the Socket.io is for server to client communication. Content moving between web browsers need to manage content data and session management so that we set up session manager server. Session manager server is implemented using node.js and express server. Server can provide contents moving service by maintaining client session and relaying contents information from one client to the others. Even though each client uses different type of web browser, web standard ensures that browsers can communicate to each client.

For the first, Node.js establish web server via express server like in Fig. ?. Now the server is responsible for exchanging data among browsers. Then the clients are trying to communicate to the server with various ways. Socket.io provide the communication; Socket.io test some ways that a browser might support.

The content data and synchronization data flow is like the Fig. ?. Each browser get its own ID, which will be used to distinguish the browsers. The ID is important, since one browser can send data to a specific browser with the browser ID. In that sense, every single content data is tagged with the browser ID, and it enables us to identify from where the data comes. The server may broadcast, or just send received data to other browser and it can synchronize or share contents.

Our system basically focus on education scenario. It involves basic electric black board and shared screen. Students make some memos and can share and take over them to the other students. It does not require any other plug in, or native

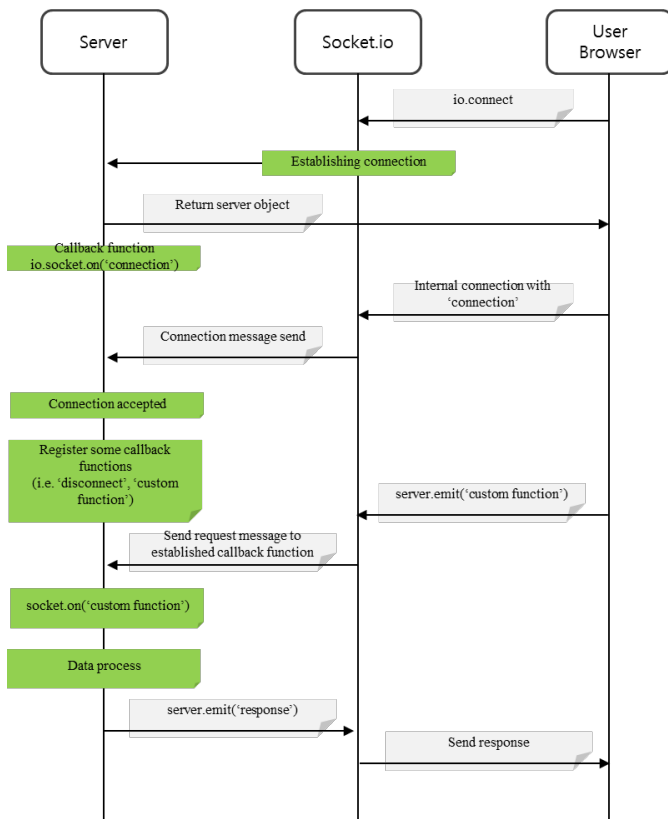


Fig. 6. Client/server flow

application support, since our implementation is based on pure web technologies.

VII. CONCLUSION

The number of web accessible device increases and the demand for web content and service varies. For those web contents provide several web services, so it is hard for the mobile devices, which have limited resolution and screen size, to show the whole services in just one screen size. Users are having troubles with scrolling and seeing small contents.

The size of mobile devices can not be enlarged easily, since the mobility matters. It encourages us to utilize more than one devices to fulfill user demand. Utilizing more than one devices is called multi-screen environment, and using multiple devices increases readability. Each device has small portion of the whole web contents and user can see the contents separately.

In this paper, some technical issues and implementation issues are covered. Some techniques about browser to browser communications involving direct/indirect communication are covered, and web site composition approaches are conveyed. For seamless data delivery concepts for the multi-screen environment, we list down several requirements.

For implementing multi-screen environment we uses Node.js for server, which is session manager. Session manager store session information of devices, and they are used to distinguish each device. With the session information the session manager can know a content come and go to what device.

For server client communication, we use Socket.io; it provide current several client server communication techniques. With the server client communication, two devices can share its content data and synchronize their status in real-time manner.

Using the content moving and sharing, some note sharing and exchange is possible, and it let students or conference participants share their ideas or opinions. For the future work, we may vary the type of movable contents, so we may can freely divide and separate web sites for more convenience.

ACKNOWLEDGMENT

This work was supported by the IT R&D program of MKE/KEIT(10039804) and the KCC (Korea Communications Commission), Korea, under the R&D program supervised by the KCA (Korea Communications Agency). (KCA-12-921-05-001).

REFERENCES

- [1] Apple iCloud, <http://www.apple.com/icloud/>
- [2] Digital Living Network Alliance(DLNA), <http://www.dlna.org/>
- [3] W3C Working Draft, Web real-time communication between browsers, <http://www.w3.org/TR/webrtc/>
- [4] IETF RFC6455, The WebSocket Protocol, <http://tools.ietf.org/html/rfc6455>
- [5] IETF RFC 2616, HTTP 1.1, <http://tools.ietf.org/html/rfc2616>
- [6] W3C, XMLHttpRequest Level 2, <http://www.w3.org/TR/XMLHttpRequest2/>
- [7] JSON-P, <http://www.json-p.org/>
- [8] W3C, Cascading style sheets, <http://www.w3.org/Style/CSS/>
- [9] Socket.io, <http://socket.io/>
- [10] Node.js, <http://nodejs.org/>
- [11] IETF RFC2326, Real Time Streaming Protocol, <http://tools.ietf.org/html/rfc2326>