# A Distributed Web Browser as a Platform for Running Collaborative Applications

Yasushi Shinjo, Fei Guo, Naoya Kaneko, Takejiro Matsuyama, Tatsuya Taniuchi, Akira Sato
Department of Computer Science
University of Tsukuba
1-1-1 Tennoudai, Tsukuba, Ibaraki 305-8573, Japan
Email: yas@cs.tsukuba.ac.jp, {kaku,nkaneko,tmatsuya,taniuchi}@softlab.cs.tsukuba.ac.jp, akira@cc.tsukuba.ac.jp

***Abstract*—Most existing collaborative applications on the Web require centralized servers for storing shared data and relaying communication messages among browsers. This means that users of these applications must fully trust centralized servers that hold and relay potentially sensitive and important data. Furthermore, users can lose access to their data if centralized servers go out of service.**

**This paper proposes building a distributed Web browser as a platform for Web-based collaborative applications to address these problems with centralized servers. A distributed browser consists of multiple browser nodes. Each node looks like a regular Web browser, is operated by a single user, but works together with other nodes. An application of the distributed browser runs across multiple nodes, and can make use of resources in both a local node and remote nodes. Multiple users can use a single application together. The distributed browser provides authenticated and secure inter-node communications for applications.**

**This paper describes an implementation of a distributed browser, called Subspace. Subspace uses an instant messaging system, Skype, to perform user authentication and secure communication among browser nodes. Reusing the overlay network and social features of Skype makes the implementation of Subspace extremely simple. Several applications on Subspace including simple collaborative browsing and comment sharing have been developed. These implementations demonstrate that Subspace provides useful facilities utilized as a platform for developing Web-based collaborative applications.**

***Index Terms*—Distributed systems, web browsers, distributed online social networks, social networking services, instant messaging systems, collaborative browsing, web annotation**

## I. Introduction

People on the Internet collaborate not only with messaging tools, such as email and instant messaging (IM), but also with Web-based collaborative applications. Examples of Web-based collaborative applications are wikis, blogs, bookmark sharing, images and video hosting, social networking services (SNSs) or online social networks (OSNs), and Web conferencing. Collaborative browsing and Web annotations are also interesting collaborative applications that overlay existing Web resources. Web browsers are essential front-end tools for these collaborative applications for users.

Central Web servers in most existing Web-based collaborative applications store shared persistent data and relay communication messages among users. This centralization causes technical and social problems [7]. One technical problem is the potential lack of scalability. Administrators of the micro-bogging system Twitter must work hard to avoid service disruptions and to avoid the famous Fail Whale [1]. Social problems include privacy and trust issues. It is not trivial for casual users to maintain access control lists (ACLs) of resources in central servers to maintain their privacy. Users must trust central servers when they upload their sensitive and important data. It is hard to completely block unplanned data disclosures and malicious break-ins [19]. In addition to privacy and trust issues, users have to think about losing access to their data in centralized servers if these servers shut down. For example, developers and users who had been attracted by Google Wave since May 2009 [23] began to worry about their code and data when Google announced Wave development would be suspended on August 2010 [22].

To address these problems with centralized servers, we propose building a distributed Web browser, or distributed browser for short, as a platform for Web-based collaborative applications. A distributed browser consists of multiple browser nodes as a distributed operating system that consists of multiple workstations. Each node looks like a regular Web browser and is operated by a single user, but it can work together with other nodes. An application of the distributed browser runs across multiple nodes, and can make use of resources in both a local node and remote nodes. Multiple users can use a single application together. The distributed browser provides authenticated and secure inter-node communications for applications.

This paper explains the implementation of a distributed browser called Subspace, which was implemented as an extension of the Web browser Google Chrome [24]. Subspace runs collaborative applications written in JavaScript and provides authenticated and secure inter-node communications for these applications by using an instant messaging system (IMS), Skype [2], [29]. IM is a popular social service on the Internet, and most users install some IM programs on their personal computers (PCs). Since we can reuse the overlay network and social features of Skype, Subspace becomes very simple to implement.

We have developed several applications on the distributed

[1] http://www.nytimes.com/2009/02/15/magazine/15wwln_consumed-t.html, http://royal.pingdom.com/2009/02/18/social-network-downtime-in-2008-2/

COLLABORATECOM 2011, October 15-18, Orlando, United States

DOI 10.4108/icst.collaboratecom.2011.247088

browser Subspace including simple collaborative browsing and comment sharing. These implementations demonstrate that Subspace provides useful facilities as a platform for developing Web-based collaborative applications.

The rest of the paper is organized as follows. Section II describes the system model that a distributed browser should provide. Section III explains how our distributed browser, Subspace, was implemented by using Google Chrome and Skype. Section IV describes applications running on Subspace. Section V and VI present future and related work while Section VII summarizes the key points and concludes the paper.

## II. System Model

This section explains the system model that a distributed browser should provide. We also discuss user grouping, communication patterns, application types, and storage in a distributed browser.

### A. Networked Workstation Model

The goal of a distributed browser is to provide an execution environment for collaborative applications that are used by a group of people who trust each other. Our distributed browser provides a system model as the networked workstations in Fig. 1 show.

Fig. 1a shows three workstations connected to a local area network (LAN). Each workstation has I/O devices, such as a bitmap display, a keyboard, and a mouse, and these I/O devices are dedicated to a single user. He/she mainly uses his/her workstation and runs commands locally. Alice in Fig. 1a controls the "more" process with her keyboard to view a long file. The output for the "more" process appears on her display. Users of these workstations can collaborate with network-capable programs. The three users Alice, Bob, and Carol in Fig. 1a are having a conversation with the "phone" processes. The "phone" is a text chat program like "talk" in Unix but it allows more than two users to have a conversation. Bob and Carol in Fig. 1a share the X11 application "xterm" with the collaboration tool "xtv" [1]. In addition to using their own local workstations, users can use remote workstations. For example, Alice in Fig. 1a runs the remote shell (rsh) process and the "write" process to send a message to Bob. The message appears on Bob's display.

Each workstation node in a networked workstation runs a network operating system (OS) that provides communication facilities across machine boundaries [30]. Workstations in a network OS often share files with a network file system, such as the Sun network file system (NFS), and share a password file with the Sun network information system (NIS). If an OS provides a higher degree of network transparency, this OS is called a distributed OS [30]. A distributed OS usually has a distributed kernel that provides inter-process communications across machine boundaries.

Fig. 1b shows the system model that the distributed browser should provide. A distributed browser consists of multiple *browser nodes*. Each node has I/O devices, such as browser windows, a keyboard, software buttons, menus and a mouse, and these I/O devices are dedicated to a single user. Each node can use two types of networks. The first is a regular LAN that is connected to the Internet. The second is a *protected overlay network* that is connected with other browser nodes.

A user mainly uses his/her node and executes programs in a local node. We call an instance of a program *a process*, which is the same term as in a distributed OS. Each process has an internal resource as a tree of the document object model (DOM). A process can use the browser node's I/O devices. Alice in Fig. 1b runs a "view" process; she can control the process with her mouse and keyboard, and the output of the process appears in a browser window. A process has the attribute *user identifier* (UID), which means the owner of the process. The UID of a process is used for access control the same as in a regular OS. For example, Alice owns a file in a shared storage and wishes to allow Bob to read the file, Alice adds the ACL entry "allow Bob read" to the file. When a process that has Bob's UID tries to read the file of Alice, this access will be granted.

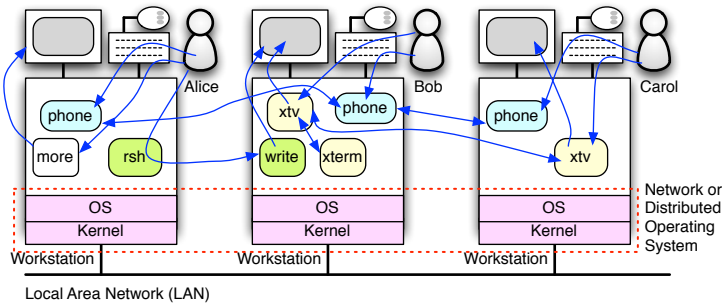
Users of the distributed browser can collaborate with network-capable applications. The three users Alice, Bob, and Carol in Fig. 1b are playing the same video with the "co-player" processes. Bob and Carol are sharing a browsing session with the application "co-browse". In addition to using their own browser node, users can use remote nodes. For example, Alice in Fig. 1b is running the "send" process to send a screen shot image of the browser window. The screen shot image appears in Bob's browser widow by the "receive" process.

A distributed browser must provide secure inter-process communications across node boundaries in the protected overlay network. A process in this overlay network can be addressed with a UID in a collaborative application. For example, the single collaborative application of "co-player" in Fig. 1b includes three processes, and each process is addressed with each user's UID.

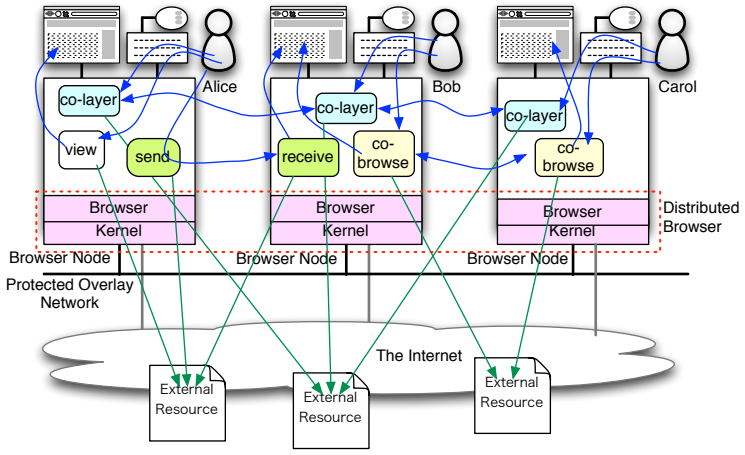
### B. User Grouping

It is useful to group users to describe access control configurations in a distributed browser. For example, consider that Alice has ten pictures of classmates, and Alice wishes to give the access rights of these files to classmates Bob, Carol, Dave, and Eve. With no grouping facility, Alice has to add these classmates to the ACLs of ten files. However, with a grouping facility, Alice only makes a group called "Classmates" once, and adds a single entry "Classmates" to the ACLs of the ten files. Multiuser operating systems usually have such grouping facilities.

While it is obvious that having a grouping facility is useful, it is not a simple problem as to who should maintain group member lists. In an early distributed OS for a LAN, since users of a system are *closed*. A distributed OS has a password file (or a list of users), /etc/passwd in Unix, and a group definition file, /etc/group in Unix. System administrators maintain these files.




(a) Workstations connected to local area network in network or distributed operating system.




(b) Browser nodes connected to overlay network in distributed browser.

Fig. 1. System models in distributed operating system and distributed browser.

This closed model is suitable for companies and university laboratories.

A distributed browser, on the other hand, should deal with a wide variety of ideas about grouping. An individual person belongs to several groups, such as friends in offices and schools, neighborhoods, circles, someone who likes the same stuff, and alumni associations. Furthermore, members of a group are ambiguous. For example, consider that Alice and Bob are classmates. Alice thinks Bob is her friend, but Bob thinks that Alice is not his friend.

It is preferable for a distributed browser to deal with two types of groups to express complicated social relations.

- Personal group. An individual user maintains this type of a member list. Only this user uses the member list in this type of group. When a member list is interpreted, the list is expanded to the members in it. The user must maintain the member list.
- External group. An external trusted user or organization maintains this type of a member list. A user does not have to maintain members in a list in this type of group.

### C. Communication Patterns

Conventional Web-based collaborative applications store shared data on centralized servers, and use them as a hub for inter-browser communication. In a distributed browser, processes of a collaborative application directly communicate with one another, and they need no mediation by central servers.

Interprocess communication in a distributed browser can be based on the client-server model. An application can consist of a single helper server process and other client processes. This helper server process should run on the PC of a collaborating member. Section IV presents applications based on the client-server model. A distributed browser does not prevent developing distributed applications based on a peer-to-peer model.

### D. Application Types

Applications of regular (non-distributed) browsers are classified into three types:

(1) Code in JavaScript and cascading style sheets (CSS) delivered from a server.
(2) Add-ons (or extensions) written in HTML, JavaScript, CSS, XML user interface language (XUL) for Firefox. Add-on programs are loaded from local files.
(3) Plug-ins that deal with media types other than standard texts and images. Representative examples of plug-ins are Java, Flash Player, QuickTime, RealPlayer, and Win-

dows Media Player. Plug-ins can be written in C and C++ typically based on the Netscape plug-in application programming interface (NPAPI). Plug-in programs are loaded from local files.

Applications of a distributed browser resemble Type (2) and (3) programs that are loaded from local files. Applications can be loaded from trusted shared storage. A distributed browser and its applications should not interfere with these applications for regular Web browsers.

### E. Storage

It is preferable for a distributed browser to provide storage service as a distributed file system in a distributed OS. The storage of a distributed browser can hold not only shared persistent data among users and applications but also application code.

A storage server should be executed on the PC of a collaborating member. If a storage server is running in a public space, we may face privacy and long-term availability issues as discussed in Section I. We need to encrypt content and make backup copies when we use such servers.

Shared storage is useful if it can provide high availability with replication. However, replication causes a revocation problem. For example, suppose that Alice revokes the access right for her file to Bob; this change should be delivered to all replications of the file. However, it is hard to deliver the control message to offline replicas. Replicating mutable items requires a classic cache coherent algorithm.

## III. Subspace: Distributed Browser based on Google Chrome by Using Skype Instant Messaging System

We are implementing a distributed browser, called Subspace, which is an extension of Google Chrome. We chose Google Chrome as the base browser. This is because, first, its source code is available as Chromium [2]. The second reason is that its process model [24] is suitable for the workstation model described in Section II.

As explained in Section II, a distributed browser must provide secure inter-process communication across browser nodes. We use an instant messaging system (IMS) in Subspace to accomplish inter-process communication. We chose Skype as an IMS because it provides an application-to-application (AP2AP) communication facility [29]. The AP2AP communication allows stand-alone external programs to use Skype's overlay network. In addition, since Skype has a decentralized architecture, this fits in well with the goal of a distributed browser. Subspace provides remote procedure call (RPC) to applications by using Skype's AP2AP facility.

### A. Overview of Distributed Browser Subspace

Fig. 2 overviews the distributed browser Subspace, which consists of two programs, the process manager and the RPC module, running on unmodified Google Chrome.

[2]http://src.chromium.org/

*The process manager* fulfills the concept of processes on a distributed browser and provides JavaScript API to these processes. A process is implemented as a browser tab on Google Chrome. When a browser tab is created on Google Chrome, a new OS process is also created for isolation. A process on Subspace inherits the features of a tab on Google Chrome. A Subspace process has internal memory resources as DOM tree nodes. Unlike the base Google Chrome, a process on Subspace has an attribute UID, as described in Section II. An UID is a string name in an IMS. The process manager is an extension (add-on) of Google Chrome and written in JavaScript.

*The RPC module* provides an inter-browser RPC facility for processes by using an external IMS program. The RPC module also provides IMS facilities, such as obtaining the identifier of the user who is running the IMS program. A user identifier is used by the process manager. The RPC module is a Google Chrome plug-in, and written in C and C++ based on NPAPI.

An application of Subspace consists of a group of processes that are running on multiple browser nodes. A process executes not only the regular built-in code for Web browsing but also special JavaScript code. A process can call the JavaScript API of the process manager. Non-collaborative browsing is also done as in executing a process. In Fig. 2, the process "view" executes the built-in code for Web browsing.

### B. API for JavaScript Programs Running in Distributed Browser

The process manager exports JavaScript API for collaborative applications. The API functions of the process manager are classified into two categories: interprocess communication and process management.

Table I lists the important API functions for interprocess communication. Applications use these functions to achieve RPC asynchronously. When a client calls send_request(), a request message for RPC is sent to the server immediately, but the client does not wait for a reply message to arrive. When a reply message is received, a callback function is called. When a server calls accept(), it registers a callback function. When a request message is received, the callback function is called. The server in the callback function obtains the arguments of RPC, performs a task, and sends a reply message to the client.

Table II lists the process management API. The function new_process creates a new child process that has the specified user ID. A process can terminate a process with kill(). In addition to these functions, application can use regular API functions of Google Chrome.

### C. Using Subspace Communication Facility outside Browser Nodes

JavaScript programs in Subspace can communicate with one another based on RPC. We made it possible to use this communication mechanism from C and Ruby programs running outside browser nodes (Fig. 2). These C and Ruby programs outside browser nodes can work together with JavaScript programs inside browser nodes. For example, we

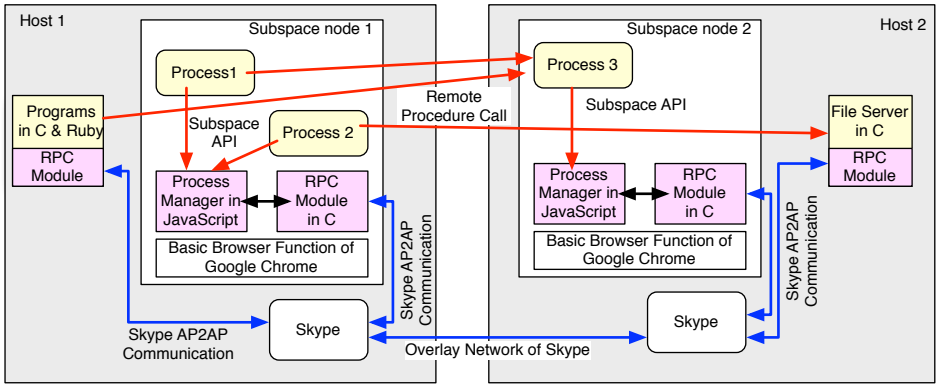



Fig. 2. Overview of distributed browser Subspace.

have implemented a file server in C and its client in JavaScript inside browser nodes. We have also implemented Ruby clients that send requests to JavaScript servers inside browser nodes.

### D. Using UIDs for Access Control and Services

All communication messages among processes across browser nodes in Subspace are associated with the UIDs of processes. These UIDs are user names in Skype IMS. Users and applications can use these UIDs for the following places and purposes.

(1) The reference monitor in the RPC module. In this place, users can describe access control policies with method names and UIDs. These policies are checked every time a RPC message is sent and received. The current implementation cannot check the parameters in request and reply messages. The reference monitor works not only on the server side but also on the client side. In addition to user names in Skype, a user can use his/her contact list to describe access control policies. A user can also define personal groups of users.

(2) Individual server functions of RPC. When a request arrives from a client, a call back function is called with the UID of the client process. The callback function can use the client UID for various purposes. For example, a server in collaborative browsing shows the client UID in a pop-up window to the user to ask if the user of the client can join the current session or not. When a client in annotation sharing tries to store a comment in storage, a server can attach the UID of the client to the comment. After that, when a client tries to retrieve comments from storage, the server can use the client UID as a trusted extra query parameter. The server can select comments that are allowed to be read by the client of the UID.

TABLE I
API FOR REMOTE PROCEDURE CALLS.

| Function Name | Description |
|---|---|
| $s$=bind($n$,$v$) | This function binds RPC with node name $n$ and service name $v$, and returns an identifier $s$ of the server in the program. The return value is passed to function send_request(). |
| send_request($s$,$m$,$a$,$c$) | This function sends an RPC request message to server $s$. Parameters $m$ and $a$ are the method and arguments of the RPC. When a reply message arrives, callback function $c$ is called with the reply message. |
| accept($v$,$c$) | This function registers service $v$ of RPC. When a request message arrives, callback function $c$ is called. The callback function takes an identifier of the request, a method name, arguments, and the name of the client node. The callback function performs an RPC task, and returns a reply message to the client. |

### E. Shell Program and "ps" Command

Subspace has a shell program that is a command interpreter as a shell in Unix. This shell accepts some built-in commands. The "ps" command of the shell shows the list of current running processes as the ps command in Unix.

## IV. Applications in Distributed Browser Subspace

We have implemented some applications in Subspace to test and verify that it provides enough facilities for developing collaborative applications. Since current Subspace uses Skype IMS, we do not need applications in which Skype has similar facilities, such as voice/video calling, a presence service, and instant messaging.

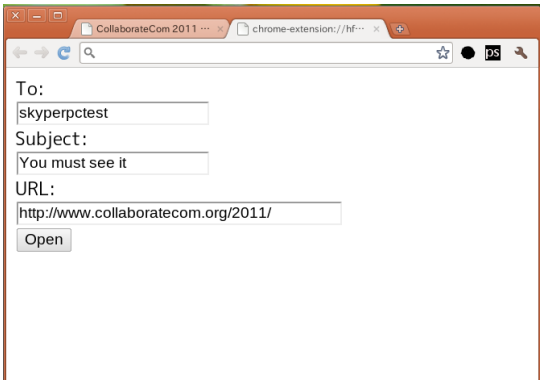
### A. Simple Collaborative Browsing

Collaborative browsing or *co-browsing* allows users in remote places to see the same Web page. We have implemented a simple co-browsing application that has two main features [12].

- Push Web pages: Pushing a Web page allows the sender to force a specific Web page to appear on the receiver's window.

TABLE II
API FOR PROCESS MANAGEMENT.

| Function Name | Description |
|---|---|
| $p$=new_process($u$) | This function creates a new process with UID $u$. The created process has a tab of Google Chrome and a DOM tree of the tab. |
| kill($p$) | This function kills process $p$ and closes the tab. |




(a) Client node that is sending URL.


(b) Server node that is receiving URL.


(c) Server node that opens tab of URL.

Fig. 3. Screenshots of simple collaborative browsing application.

- Hand-over control: This feature allows a sender who has logged in to a session to give control of the Web session to a receiver. A sender with this facility can pass a session to the receiver without telling him/her the user name or password for the site.

We are working on higher level co-browsing features, including sync-surfing, co-scrolling, and co-filling.

Fig. 3 shows screenshots of our simple co-browsing application. This was developed based on a client-server model. A client sends a URL to a server. Fig. 3a is a screenshot of the client node. This node has two tabs, i.e., two processes. One process is opening a site. The other process is executing our co-browsing application. The user is filling in the boxes with the receiver Skype name, subject, and URL.

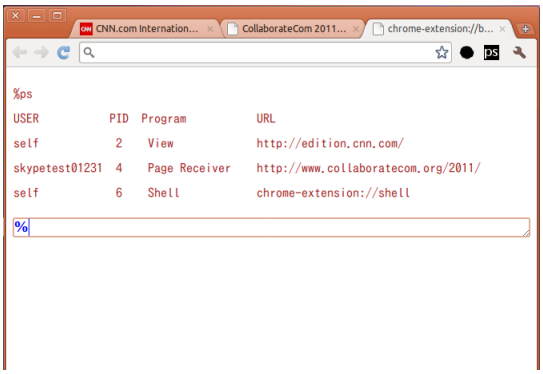



Fig. 4. Results for ps command in server node.

Fig. 3b is a screenshot of the server node, where an RPC server is waiting for a request message. The user is browsing a news site. When the server receives a request, it creates a window and shows the client user name, subject, and URL. If the user pushes the "OK" button, the server creates a new process with a tab, and opens the received URL in the process. The browser node obtains the HTML file of the received URL from the HTTP server, parses the HTML file, and obtains inline images from the HTTP server. Finally, the tab reveals the requested site as seen in Fig. 3c.

Fig. 4 shows a screenshot of the server node when the user is opening a shell window and typing the ps command. The result for the ps commands means that three processes are running in the browser node. The first and last processes are owned by the user him/herself, and the second process is owned by the client process. Therefore, the second process is running at a lower privilege, and cannot see the content of the other processes even if they have the same origin.

The client in Fig. 3a sends a simple URL to the server. In addition to sending a simple URL, we can send two items.

- A submission form: With this feature, a user can open a Web page that is created with a POST method. The client process sends a form to the server process, which receives the form, creates a process and its DOM tree nodes, and fills the DOM tree nodes with the received form.
- Cookies: With this feature, a user can sends session cookies to a remote friend. This makes it possible to accomplish the handing-over control mentioned above.

We can run a client from a command line in our simple co-browsing application. The following command line has the same effect as the client in Fig. 3a.

```
$ open-url -s "You must see it" skyperpctest \
http://www.collaboratecom.org/2011/
```

### B. Comment Sharing by Using Tuple Space

We have implemented a simple comment sharing application in Subspace, which uses a Linda-type tuple space as a backend store [6]. We have extended the Linda tuple space

model with access control capability. The following is an example of Alice adding a comment.

```
out(["comment-sharing",
     "http://www.tsukuba.ac.jp/",
     "I like it"],
     ["Bob", "Carol"])
```

This function `out()` takes two arguments: a tuple and an ACL. The tuple consists of the name of application `"comment-sharing"`, the URL of the page, and the comment text. Unlike Linda, the function takes an additional argument, i.e., an ACL, which is a list of users who are allowed to read the tuple.

To read comments, the client calls function `rd_all()` (read all) as follows.

```
rd_all(["comment-sharing",
        "http://www.tsukuba.ac.jp/",
        nil])
```

This function is called when the user visits the URL. The first and second elements are fixed, and the third element `nil` means a wild card.

When our tuple space receives this request, it first performs pattern matching with the argument, and collects all the tuples as a regular Linda tuple space. Next, our tuple space checks the ACLs of the matched tuples, and removes a tuple if the client does not have permission. Finally, our tuple space returns the readable tuples of the URL.

We have implemented a tuple space server by extending that of the Ruby language, called Rinda [27]. First, in this implementation, we extended the Linda server of Ruby to use the RPC facility of Skype. Next, we added the access control feature to the Linda server.

In the current implementation, the Linda server runs outside the browser node, and should be executed on the PC of a trusted user. The PC should run the Skype client and should always be online to maintain availability.

### C. Screen Capture

Google Chrome has an extension called “Screen Capture” [13]. A user can capture the browser screen as a portable network graphics (PNG) image with this extension and add annotations to the image. He/she can save the image to not only a local file but also remote picture sharing sites, such as Picasa, Facebook, and Sina Microblog.

We changed this saving function to send the image to a remote RPC server of Subspace. Fig. 5 shows screenshots of our modified screen capture extension. The user captures the screen image, adds text, lines, and highlighting to the image, and sends the image to the remote user “skyperpctest” of Skype. The client sends an image transfer request to the server. When the remote user accepts the request, the server creates a new process, creates a DOM tree, and adds an IMG node of HTML to the DOM tree. Next, the client begins sending the image, and the server receives it and stores it in the IMG node.

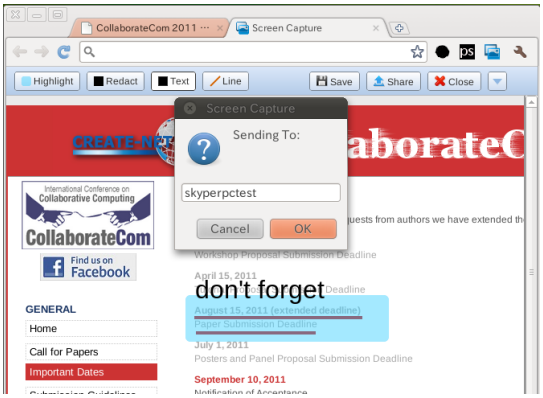



Fig. 5. Screenshot of modified screen capture extension.

## V. Next Steps

We are developing our distributed browser project, and we have many tasks to implement.

The current Subspace implementation depends on the central servers of Skype IMS. When these central servers are down by accident [3], [17], collaborative applications do not work. This dependence also means that users must trust Skype IMS. If Skype IMS is cracked, inter-browser communication becomes vulnerable to man-in-the-middle attacks. Users must use Skype IMS although they may prefer other IM systems.

We are thinking about using alternative IMSs or SNSs to solve this problem with dependence. Candidates for IMS are Yahoo!Messenger, Facebook Chat over eXtensible Messaging and Presence Protocol (XMPP), and Google Talk [26].

When we use multiple IMSs or SNSs, we have to identify a single person in them. For example, consider that Alice has an account Alice-skype-123 with Skype and Alice-yahoo-345 with Yahoo!Messenger. We have to combine Alice-skype-123 and Alice-yahoo-345 into a single person. To achieve this, we can use a public key as in some decentralized OSNs (DOSNs) [5], [7], [8].

We need more collaborative applications. First, we are working on higher levels of co-browsing features, including sync-surfing, co-scrolling, and co-filling. We wish to support dynamically changing Web pages as in [20] without centralized servers. Second, we are also interested in playing video collaboratively. These applications are sensitive to communication delays. We will evaluate our approach of IMS though implementing these applications.

## VI. Related Work

Co-browsing is one of our most important applications, and we have implemented the simple co-browsing application described in Section IV. There are many co-browsing systems, browser extensions, and services [12]. Web conferencing services usually include a co-browsing feature. Most of these systems require centralized servers to relay control messages among browser nodes. Our distributed browser does not require centralized servers. Brosix is an IMS, and has a

co-browsing feature [4]. However, Brosix requires centralized servers and provides no application development framework.

Web annotation is another key application of our distributed browser. Many Web annotation services have been proposed and developed [16]. Most existing Web annotation services depend on centralized servers. Our distributed browser stores comments on a server running on a trusted member's PC in a small group.

ShiftSpace is a rich Web annotation system, and it is trying to eliminate dependence on centralized servers [35]. The current implementation of ShiftSpace uses centralized servers to store rich Web annotations. Collaborative browser nodes in ShiftSpace communicate with one another through a backing store where keys are URLs. The collaborative browser node in our distributed browser, on the other hand, communicates through an overlay network in RPC style.

Skype is an IMS, and provides an application development framework [2], [29]. TalkAndWrite achieves collaborative editing, and IDroo achieves a whiteboard facility by using this framework [3]. Our prior report has described passing access rights over the Skype network in an anti-spam scheme based on capability-based access control [28]. Some desktop and application sharing services use Skype for binding PCs, but they do not use the Skype overlay network for exchanging control messages for sharing. If we share a Web browser screen in these services, we can implement co-browsing. However, they require centralized servers and a wide bandwidth to transfer screen images. Skype4Games provides a framework for developing online games on the Skype overlay network [33]. Our distributed browser provides a framework for developing Web-based collaborative applications on the Skype overlay network.

Many academic and commercial SNSs or OSNs have been proposed and developed. Some of these are called decentralized OSNs (DOSNs) [5], [7], [8]. Many peer-to-peer (P2P) and distributed hash table (DHT) systems and protocols have also been proposed and developed. These distributed systems have the goal of providing services without depending on centralized servers. The goal of our distributed browser was to provide a service for Web-based collaborative applications without depending on centralized servers. To achieve this, we reused an existing system, Skype IMS, and made our implementation much simpler than that in these systems. In DOSN, an IMS can be used to exchange public keys for identity as an out-of-channel mechanism in the bootstrap phase. Our distributed browser uses an IMS as a daily communication network for user authentication, message encryption and NAT traversal. Collaborative applications can use the overlay network of the IMS in RPC style. Users can use the user names of an IMS to describe access control policies. Our distributed browser also differs from these systems in that it provides the workstation model described in Section II. Furthermore, unlike DOSN, we are trying to eliminate the dependence on Skype and to avoid single points of failure by adding an alternative IMS, as discussed in Section V. The domain name system (DNS) also has alternative servers, which achieves very high availability.

Collaborative browsing and search (COBS) is a research project to achieve co-browsing and annotation-sharing in a distributed way [34]. This is implemented in COBS by a Web browser front-end and a DOSN backend. The COBS browser is designed to use a DHT as backend storage while the current implementation uses some central servers to demonstrate their ideas. The current implementation uses a centralized XMPP server for co-browsing. In our terminology, COBS is also implementing a distributed browser. The differences between COBS and ours have already been discussed as differences between DOSN and our distributed browser. In summary, the differences between COBS and ours are first, our distributed browser provides a workstation model, second, our implementation is much simpler because it reuses existing IMSs, and finally, we have tried to avoid single points of failure by having multiple IMSs.

Opera Unite is a technology that runs an HTTP server in a Web browser [32]. If a user runs a file server, he/she can allow the public or limited people to access files in his/her PC. Applications in the HTTP server of Opera Unite are written in JavaScript. Central servers in Opera Unite make HTTP servers accessible from the Internet. Central servers also perform user authentication every time an HTTP connection is established. On the other hand, our distributed browser does not require central servers after the initialization phase.

WebSocket is part of the HTML5 standard, and allows JavaScript code running on a Web browser to connect with a Web server to exchange various types of data [11], [14]. WebSocket does not allow running a server in a browser for security and direct communication among Web browsers. BrowserSocket makes it possible for JavaScript code running on a Web browser to act as a server for WebSocket [25]. BrowserSocket is implemented as an extension and a plug-in of Firefox but does not provide authenticated communication. Our distributed browser provides authenticated communication, and allows applications to address nodes with user names in an IMS.

Web Storage API and Indexed Database API are also parts of the HTML5 standard and they allow JavaScript code running on a browser to store persistent data in the browser [15], [21]. This standard does not deal with any access control mechanisms with user identifiers. Our distributed browser has access control mechanisms with user identifiers, and stored files and tuples can be shared among users and applications.

SPORC is a framework for collaborative applications by using untrusted centralized servers [10]. In SPORC, centralized servers are used to realize operational transformation (OT) [9]. The main application of SPORC is collaborative editing, and it is implemented by reusing the source code of Google Wave [18], which is also based on OT. The objective of our distributed browser is similar to that of SPORC. However, ours is different from SPORC in that it provides the communication facilities of IMS for collaborative Web-based applications.

[3]http://shop.skype.com/apps/ and https://extras.skype.com/

The Illinois browser operating system (IBOS) is a microkernel based operating system for Web browsers, which provides strong isolation among activities that have different origins [31]. IBOS does not support collaborative applications.

## VII. Conclusion

We proposed the idea of a distributed browser, and the workstation model as a system model for a distributed browser. A distributed browser consists of multiple browser nodes, where each node has I/O devices and is dedicated to a single user. Collaborative applications are a group of processes running in multiple nodes. A distributed browser achieves network transparency for processes as distributed operating systems.

We discussed the implementation of a distributed browser, called Subspace, which achieves inter-browser communications by using the existing IMS of Skype. This design simplifies the implementation of Subspace. Subspace also provides a remote procedure call facility for processes, where the user names of the IMS are used for addressing communication peers and describing access control policies.

We have implemented several applications in Subspace, including simple co-browsing, snapshot sharing, and simple comment sharing. These implementations have demonstrated that Subspace provides useful facilities for developing Web-based collaborative applications.

We plan to enhance co-browsing and Web annotation facilities in the future, and evaluate our approach in such applications that are more interactive and delay sensitive. We also intend to eliminate single points of failure by using multiple IMSs together.

## References


[1] H. Abdel-Wahab and M. Feit, "XTV: a framework for sharing X Window clients in remote synchronous collaboration," in *IEEE TRICOMM '91 Communications for Distributed Applications and Systems*, 1991, pp. 159–167.

[2] S. A. Baset and H. G. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol," in *25th IEEE International Conference on Computer Communications (INFOCOM)*, 2006, pp. 1–11.

[3] T. Brock. (2010, Dec.) Skype Outage Today. [Online]. Available: http://blogs.skype.com/enterprise/2010/12/skype_outage_today.html

[4] Brosix. Features - Brosix Instant Messenger. [Online]. Available: http://www.brosix.com/features/

[5] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta, "PeerSoN: P2P Social Networking: Early Experiences and Insights," in *the Second ACM EuroSys Workshop on Social Network Systems (SNS '09)*, Mar. 2009.

[6] N. Carriero and D. Gelernter, "How to Write Parallel Programs: a Guide to the Perplexed," *ACM Computing Surveys*, vol. 21, no. 3, pp. 323–357, September 1989.

[7] A. Datta, S. Buchegger, L. Vu, T. Strufe, and K. Rzadca, "Decentralized Online Social Networks," *Handbook of Social Network Technologies and Applications*, pp. 349–378, 2010.

[8] A. Datta, "SoJa: Collaborative Reference Management using a Decentralized Social Information System," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2010)*, 2010.

[9] C. A. Ellis and S. J. Gibbs, "Concurrency control in groupware systems," in *ACM SIGMOD international conference on Management of data*, ser. SIGMOD '89, 1989, pp. 399–407.

[10] A. J. Feldman, W. P. Zeller, M. J. Freedman, and E. W. Felten, "SPORC: Group Collaboration Using Untrusted Cloud Resources," in *the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*, Oct. 2010.

[11] I. Fette and A. Melnikov, "The WebSocket protocol (draft-ietf-hybi-thewebsocketprotocol-14)," *IETF Internet Draft*, Sep. 2011. [Online]. Available: http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-14

[12] R. Good. (2007, Mar.) Co-Browsing Tools And Technology: A Mini-Guide. [Online]. Available: http://www.kolabora.com/news/2007/03/22/cobrowsing_tools_and_technology_a.htm

[13] Google. chrome-screen-capture - Capture webpage screenshot in Chrome. [Online]. Available: http://code.google.com/p/chrome-screen-capture/wiki/EnglishFAQ

[14] I. Hickson (ed.), "The WebSocket API," *W3C Working Draft*, Apr. 2011. [Online]. Available: http://www.w3.org/TR/2011/WD-websockets-20110419/

[15] I. Hickson (ed.), "Web Storage," *W3C Working Draft*, Sep. 2011. [Online]. Available: http://www.w3.org/TR/2011/WD-webstorage-20110901/

[16] J. Hunter, "Collaborative Semantic Tagging and Annotation Systems," *Annual Review of Information Science and Technology*, vol. 43, pp. 187–239, 2009.

[17] J. Kirk. (2011, Jun.) Skype Sign-in Problems Knock Millions Offline. [Online]. Available: http://www.pcworld.com/article/229604/skype_signin_problems_knock_millions_offline.html

[18] S. Lassen and S. Thorogood, "Google Wave Federation Architecture," *Google Wave Protocol*, May 2009. [Online]. Available: http://www.waveprotocol.org/whitepapers/google-wave-architecture

[19] J. Li, M. Krohn, D. Mazières, and D. Shasha, "Secure untrusted data repository (SUNDR)," in *the 6th conference on Symposium on Opearting Systems Design & Implementation (OSDI'04)*, 2004.

[20] D. Lowet and D. Goergen, "Co-Browsing Dynamic Web Pages," in *The 18th international conference on World wide web (WWW '09)*, Apr. 2009.

[21] N. Mehta, J. Sicking, E. Graff, A. Popescu, and J. Orlow (eds), "Indexed Database API," *W3C Working Draft*, Sep. 2011. [Online]. Available: http://www.w3.org/TR/2011/WD-IndexedDB-20110419/

[22] D. Osinga. (2009, May) Introducing the Google Wave APIs: what can you build? [Online]. Available: http://googlewavedev.blogspot.com/2009/05/introducing-google-wave-apis-what-can.html

[23] L. Rasmussen. (2009, May) Went Walkabout. Brought back Google Wave. [Online]. Available: http://googleblog.blogspot.com/2009/05/went-walkabout-brought-back-google-wave.html

[24] C. Reis and S. D. Gribble, "Isolating Web Programs in Modern Browser Architectures," in *the 4th ACM European Conference on Computer systems (EuroSys '09)*, Apr. 2009.

[25] T. Ruottu and K. Markus, "BrowserSocket API," 2010. [Online]. Available: http://browsersocket.org/api.html

[26] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core," *RFC 6129*, 2011.

[27] M. Seki, "dRuby and Rinda: Implementation and Application of Distributed Ruby and its Parallel Coordination Mechanism," *International Journal of Parallel Programming*, vol. 37, no. 1, pp. 37–57, 2009.

[28] Y. Shinjo, K. Matsui, T. Sugimoto, and A. Sato, "An Anti-Spam Scheme Using Capability-Based Access Control," in *Proceedings of IEEE 34th Conference on Local Computer Networks, 5th IEEE LCN Workshop on Security in Communication Networks (SICK)*, 2009, pp. 907–914.

[29] Skype Limited. (2011) Skype Public API. [Online]. Available: http://developer.skype.com/accessories

[30] A. S. Tanenbaum and R. Van Renesse, "Distributed Operating Systems," *ACM Computing Surveys*, vol. 17, pp. 419–470, December 1985.

[31] S. Tang and H. Mai, "Trust and Protection in the Illinois Browser Operating System," *the 9th USENIX Conference on Operating Systems Design and Implementation*, 2010.

[32] H. S. Tømmerholt and D. Davis, "Opera Unite developer's primer," *Dev.Opera*, Oct. 2009. [Online]. Available: http://dev.opera.com/articles/view/opera-unite-developer-primer-revisited/

[33] T. Triebel, B. Guthier, and W. Effelsberg, "Skype4Games," in *the 6th ACM SIGCOMM workshop on Network and system support for games (NetGames '07)*, Sep. 2007.

[34] C. von der Weth and A. Datta, "COBS: Realizing Decentralized Infrastructure for Collaborative Browsing and Search," in *IEEE Advanced Information Networking and Applications (AINA 2011)*, 2011, pp. 617–624.

[35] M. Zer-Aviv. (2010, Sep.) ShiftSpace Developer Tutorial. [Online]. Available: https://github.com/ShiftSpace/shiftspace/wiki/Developer-Tutorial