2013 IEEE/WIC/ACM International Conferences on Web Intelligence (WI) and Intelligent Agent Technology (IAT)

Hyperlink Pipeline: Lightweight Service Composition for Users

Li Li
Huawei Shannon Lab
li.nj.li@huawei.com

Wu Chou
Huawei Shannon Lab
wu.chou@huawei.com

Tao Cai
Huawei Shannon Lab
t.cai@huawei.com

Zhe Wang
Rutgers University
zhewang@cs.rutgers.edu

***Abstract*—This paper describes a lightweight service composition mechanism in web browsers, called hyperlink pipeline, to allow users to interactively chain distributed atomic web services into composite services similar to using Unix pipelines. This mechanism addresses the limitation in current Web, which confines users to following predefined hyperlinks and forms. We present the functional composition framework to facilitate recursive hyperlink pipeline construction and execution based on REST services and the HTML5 Microdata model to efficiently implement the framework. The proposed mechanism has been implemented and used in several web applications for real-time communication and collaboration based on WebRTC. The initial tests indicate the approach is feasible and promising.**

Hyperlink Pipeline, REST service, WebRTC, Microdata

I. INTRODUCTION

The Web is a distributed informational space, in which a piece of information, e.g. a HTML page, is linked to many other pieces of information, e.g. images, videos and texts, usually located somewhere else. The primary way to interact with the Web today is to follow predefined hyperlinks in HTML pages, to retrieve, create, delete or update information. Users can only create new information by filling forms, which are again predefined.

This way of using the Web facilitates access to well-structured information and applications, but does not allow users to compose web services in unexpected ways as their needs grow. Without service composition, a web service will either not satisfy a user's needs, or become too complicated by trying to satisfy all users. Moreover, web services designed this way cannot interoperate with each other except in a very limited manner, i.e. following the predefined hyperlinks and forms.

In other words, the current Web has more or less achieved the goal that "any data that are related can be linked," but it has not achieved the goal that "any services that are related can be composed."

To address this problem, we adopt an alternative way of using the Web, called hyperlink pipeline, modeled after the Unix pipeline system [1], in which a user can create a composite program by chaining the atomic programs into a pipeline. In the hyperlink pipeline system, an atomic web service implemented by a resource is equivalent to an atomic Unix program, and a user can create a composite web service by chaining the atomic web services. Since any REST service is identified and described by a hyperlink in web browsers, what a user needs to do is to chain relevant hyperlinks into a new hyperlink that identifies and describes the composite service, which can be used in new hyperlink pipelines recursively.

Many tasks can be achieved quickly by such a hyperlink pipeline system. For example, a user can chain a hyperlink to his online pictures and a hyperlink to a map to display the pictures on the map by their locations. A user can chain a hyperlink to his personal checks at his bank and a hyperlink to his online calendar to sort the checks by dates. A user can also chain a hyperlink to his phone and a hyperlink to his friend's phone to make a call. To streamline the tax return process in US, a user can chain his employer W2 form service, his favorite only tax service, and the IRS tax return service.

Such hyperlink pipelines become reusable automatically when the referenced resources change without breaking the service interfaces. For example, the hyperlink pipeline for tax return this year would work without change when the W2 form changes next year. They can also be reused through modifications, much like we tweak a Unix pipeline for different tasks.

If we provide users an intuitive and interactive way to create such hyperlink pipelines, we may change the way the web applications are developed and the way we interact with the applications. The focus of web development will shift from isolated applications to interoperable services and the users will also become service creators without having to learn any programming languages.

However, to achieve this new vision, we are still faced with many technical challenges. Unlike in Unix pipeline where all programs are collaboratively developed by a community, interact through the standard Unix I/O stream interface and are executed by the same operating system, web services are independently developed, distributed over the network and have more complex interfaces and interactions. Furthermore, we cannot expect regular users to type Unix shell commands into web browsers.

The key problem is therefore to allow users to chain hyperlinks from any websites. On one hand, we need some standard language to describe hyperlink such that hyperlinks from different website can be chained successfully. On the other hand, we need to allow each website to customize their hyperlink pipelines for various purposes.

To address this problem, we combine semantic web technologies, in particular Microdata [2], with the REST design principles [3], to provide a functional composition framework to tie hyperlink pipelines with REST web services and an object-oriented mechanism to implement the interactive hyperlink pipeline composition in web browsers without any plug-in.

978-1-4799-2902-3/13 $31.00 © 2013 IEEE
DOI 10.1109/WI-IAT.2013.71

The rest of this paper is organized as follows. Section II provides a survey on some related work. Section III discusses the design decisions and the architecture of hyperlink pipeline. Section IV describes the prototype system and some web applications based on hyperlink pipeline and we conclude with Section V.

II. Related Work

There are many web mash-up tools, such as Yahoo Pipes [4], Intel Mash Maker [5], and EMML [6], which allow users to combine existing Web data and functions into new ones. These mash-up tools separate design and execution phases, such that a deployed application cannot be changed on the fly. Applications designed by one tool cannot be ported to another tool without change.

Intel Mash Maker [5] is a browser extension that allows users to combine a HTML page (e.g. flight schedule) with a web widget (e.g. map) into a new page (where flight itinerary is plotted on the map). Although this tool supports mash-up within a web browser, it requires a special browser plug-in and is not designed for service composition.

Web Intents [7] under development at W3C adopts the Intent-Service design pattern from the Android OS to web browsers, such that JavaScript modules can invoke each other through an Intent object that they agree upon beforehand. Although Web Intents can be used to compose services, the composition has to be defined by web applications and cannot be changed by web users.

Several Web Service Composition architectures and systems [8][17][18][19][20] have been developed to support manual or automated composition of SOAP or REST services, following a two phased process: design the composition workflow in a language and then deploy it to a web server for execution. However, these systems are not browser based but designed for web developers who have sophisticated engineering skills.

In summary, the current composition tools and languages are either too restrictive or too powerful for regular web users, compared to hyperlink pipeline:

- These systems require special composition servers or web browser plug-ins to understand and execute the composition workflows, whereas hyperlink pipeline does not require any special web servers or web browser plug-ins;
- These systems separates design phase from runtime phase, whereas hyperlink pipeline interleaves the two phases within web browsers;
- These systems uses complex service description languages, such as WSDL [9], whereas hyperlink pipeline describes services with lightweight Microdata in HTML5;
- These systems are designed for service developers whereas hyperlink pipeline is designed for regular web users;

III. Hyperlink Pipeline Architecture

The basic concept of hyperlink pipeline can be illustrated by the following example. In Figure 1, the user (Alice) is presented with some hyperlinks pointing to existing web services (e.g. A for Alice's phone service). If Alice wants to call Bob and save the call to her favorite cloud storage service (L), she uses hyperlink pipelines as follows: she drags source hyperlink A and drops it to destination hyperlink B to create a hyperlink pipeline B(A), representing a call from A to B. Then she drags source hyperlink B(A) and drops it to destination L to create hyperlink pipeline L(B(A)), representing saving the call to the cloud. The result of this process is a tree whose root is the hyperlink pipeline and whose children are hyperlinks denoting either atomic or pipeline services (Figure 1). To invoke the chained services in the pipeline, Alice just needs to click hyperlink L(B(A)).

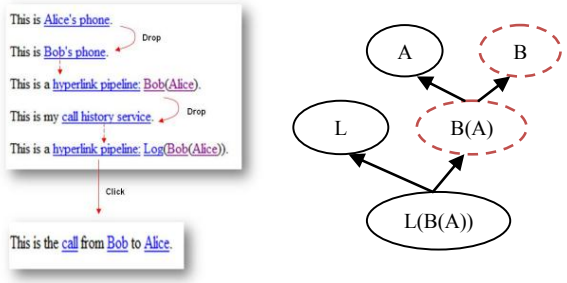

Figure 2: (left) create and execute a hyperlink pipeline; (right) the tree representation of the hyperlink pipeline (sources in dashed circle)

A. Functional Composition Framework

In the REST based hyperlink pipeline system, each hyperlink denotes a function (resource) that maps some incoming representation to some outgoing representation [3] when the hyperlink is dereferenced. A destination hyperlink D accepts a source hyperlink S to create a hyperlink pipeline P denoted by function D(S), if and only if the outgoing representation of S matches the incoming representation of D.

This functional framework can represent arbitrary recursive hyperlink compositions in pure URI syntax [10], as illustrated in Figure 2. At first, hyperlink C accepts hyperlink A to create pipeline C(A), then hyperlink C(A) accepts hyperlink B to create pipeline C(A)(B). Finally, hyperlink D accepts hyperlink C(A)(B) to create pipeline D(C(A)(B)).

The functional composition framework also provides a distributed method to execute hyperlink pipelines, as illustrated in Figure 3. When the hyperlink pipeline D(C(A)(B)) is clicked, the browser dereferences this URI by contacting resource D providing argument C(A)(B). Next, resource D then dereferences its argument by contacting resource C, providing arguments A and B. Finally, resource C dereferences URI A and B accordingly. After all the URIs are dereferenced, the resources can exchange necessary messages to finish the service composition, and the final result is sent back to the browser.

In this framework, the critical task is to efficiently test if a hyperlink accepts another hyperlink as a valid argument based on the service description of the hyperlinks. If we store the service description for each hyperlink on the server, and follow the above dereference process to do the tests, it will not be very efficient because the process involves 2(N+1) network trips in the worst case, where N is the number of

edges of the hyperlink pipeline tree. For this reason, this paper explores client-side service description based on Microdata, such that the descriptions are embedded inside HTML pages and all the tests are carried out within a web browser without any network traffic.

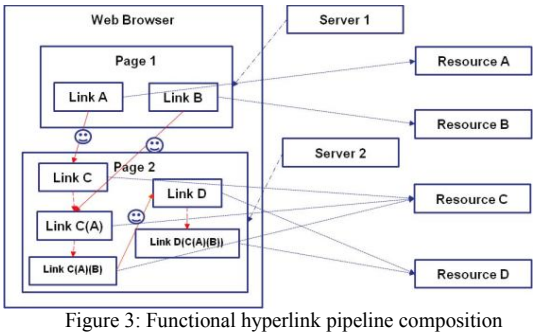

Figure 3: Functional hyperlink pipeline composition

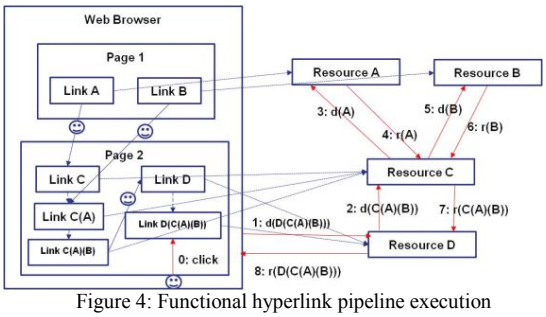

Figure 4: Functional hyperlink pipeline execution

B. *Describe Hyperlink Pipeline in Microdata*

The client-side service description and test mechanism has two goals:

- Indicate where and what pipelines can be formed from which hyperlinks on a HTML page, since not all hyperlinks can form valid hyperlink pipelines;
- Allow web applications to customize hyperlink pipeline presentation, execution and management without sacrificing interoperability between hyperlink pipelines;

To achieve these goals, we develop a HTML5 Microdata [2] language to annotate hyperlinks with a "pipeline factory" following the UML class diagram in Figure 4.

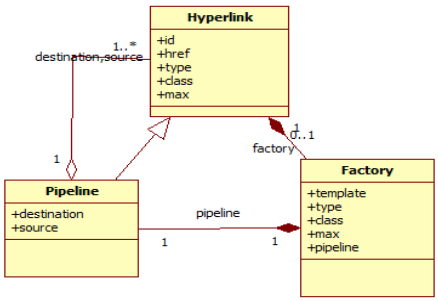

Figure 5: Class relations between hyperlink, pipeline and pipeline factory

In this modal, each destination hyperlink uses a factory to check source hyperlinks and produce hyperlink pipelines. All hyperlink pipelines produced from the same destination hyperlink will inherit its factory, type and class properties, but will overwrite the other properties.

The Microdata "pipeline factory" consists of the following properties:

- template: a URI template [13] with a destination variable {dest} to be replaced by the destination URI and a source variable {src} to be replaced by the source URI when they are composed;
- type: list of accepted media types for the outgoing representations of the sources;
- class: list of accepted semantic classes for the outgoing representations of the sources;
- max: the maximum number of accepted sources (default=1, unbounded = -1);
- pipeline: the URI pointing to a HTML5 DOM element which stores the pipeline element as a child element. The DOM element may also contain a special element which defines the pipeline template for the pipeline elements in it. The default pipeline template is the HTML5 <a> element;

The template property allows an application to customize the URI representation of functions if it does not want to use the default parentheses characters reserved by URI syntax [10]. The pipeline template allows applications to use JavaScript and CSS to customize the presentation and execution of the created pipelines for consistent look-and-feel. For example, an application can use a pipeline template:

```
<a onclick="execute()" id="template">
    <img src="pipe.png">…</img>
</a>
```

to display the pipeline as an image and invoke some JavaScript code when the pipeline is clicked.

An example Microdata is as shown in Figure 5 (top), with a fake standard namespace URI "`http://www.pipe-dict.com`". The extracted factory specification is illustrated in Figure 5 (bottom).

```
<p>
This is <a href="http://www.site1.com/alice" id="alice"
itemscope itemtype="http://www.pipe-dict.com/factory">Alice's phone
<span itemprop="http://www.pipe-dict.com/type" hidden="hidden">text/html</span>
<span itemprop="http://www.pipe-dict.com/class"
hidden="hidden">http://www.example.com/tags#phone</span>
<span itemprop="http://www.pipe-dict.com/max" hidden="hidden">1</span>
<span itemprop="http://www.pipe-dict.com/template" hidden="hidden">{dest}{{src}}</span>
<span itemref="http://www.pipe-dict.com/pipeline" hidden="hidden">#pipeline</span>
</a>.
</p>
```

```
http://www.pipe-dict.com/factory
    http://www.pipe-dict.com/type=text/html
    http://www.pipe-dict.com/class= http://www.example.com/tags#phone
    http://www.pipe-dict.com/max=1
    http://www.pipe-dict.com/template={dest}{{src}}
    http://www.pipe-dict.com/pipeline=#pipeline
```
Figure 6: (top) hyperlink Microdata; (bottom) extracted factory properties

A similar factory is also attached to hyperlink B in Figure 1. After hyperlink B accepts hyperlink A, hyperlink pipeline

B(A) is created as a HTML5 <a> element with properties represented in Microdata (Figure 6).

```
<p>
This is <a href="http://www.site2.com/bob(http://www.site1.com/alice)" id="pipeline"
itemscope itemtype="http://www.pipe-dict.com/pipeline">a pipeline
<span itemprop="http://www.pipe-dict.com/destination" hidden="hidden">#bob</span>
<span itemprop="http://www.pipe-dict.com/source" hidden="hidden">#alice</span>
<span itemprop="http://www.pipe-dict.com/max" hidden="hidden">0</span>
</a>
</p>
```

```
http://www.pipe-dict.com/pipeline
  http://www.pipe-dict.com/href=http://www.site2.com/bob(http://www.site1.com/alice)
  http://www.pipe-dict.com/destination=#bob
  http://www.pipe-dict.com/source=#alice
  http://www.pipe-dict.com/max=0
```

Figure 7: (top) hyperlink pipeline Microdata; (bottom) extracted pipeline properties

C. *Construct Hyperlink Pipeline Interactively*

Since all hyperlinks that can create hyperlink pipelines are annotated with the standard Microdata, a generic constructor, called Factory Module, can be used to construct the hyperlink pipelines as illustrated below (Figure 7).

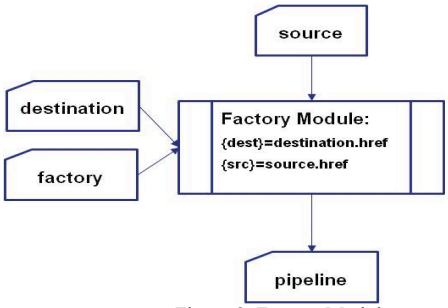

Figure 8: Factory Module

The construction algorithm is outlined below:

Construct (D: destination, F: factory, S: source, P: pipeline):

1. If D.max > 0, continue; otherwise stop;
2. If S.type is subtype of F.type according to Internet Media Type [11], continue; otherwise stop;
3. If S.class is subclass of F.class according to rdfs:subClassOf defined by RDFS [12], continue; otherwise stop;
4. Create a DOM element P under F.pipeline element according to its template;
5. Set P.href = F.template where {dest}=D.href and {src}=S.href,
6. P.type = D.type, P.class=D.class, P.max = D.max – 1, P.destination = D, P.source = S;

D. *Execute Hyperlink Pipeline*

Within the general execution plan in Figure 4, hyperlink pipelines can be executed in two ways as illustrated in Figure 9: 1) by the hyperlink dereference mechanism in web browser; 2) by custom JavaScript in web page.

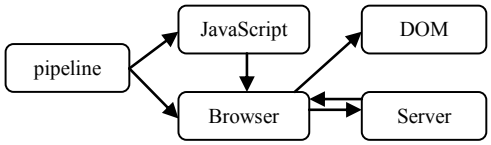

Figure 9: Possible execution flows of pipeline

For example, to execute the pipeline in Figure 2, the page invokes WebRTC JavaScript API [14] and Websocket JavaScript API [15]. To execute a hyperlink pipeline with Google Map, the page calls Google Map JavaScript API.

Another design consideration is hyperlink pipeline state. Some hyperlink pipelines, such as those in Figure 2, have distinct states. The hyperlink pipelines are in an "initial" state when created, and transition to some intermediate state, like "calling" when clicked, then to a final "success" or "failure" state based on backend service executions. Other hyperlink pipelines have only one state. For example, when we drop some pictures to a map, we expect to see the pictures on the map immediately. In this case, the created hyperlink pipeline `map(pictures)` has only a final state.

Executing a hyperlink pipeline may create a new hyperlink pointing to the new resource. For example, executing a `phone(phone)` hyperlink pipeline (Figure 10) creates a call resource, while executing a `map(pictures)` pipeline (Figure 11) creates a new map resource.

E. *Modify, Save and Retrieve Hyperlink Pipeline*

A user can delete a source hyperlink from a hyperlink pipeline. A user cannot delete the destination hyperlink without deleting the entire hyperlink pipeline.

A user can save the hyperlink pipelines on a HTML page to local or remote storage according to the pipeline data model. When a HTML document is loaded into the browser, the associated hyperlink pipelines can be loaded into memory and revalidated. A hyperlink pipeline is valid if its destination and source URIs can be dereferenced through preflight HTTP (e.g. GET) requests to the identified resources. The validation procedure should handle resource moves and update the pipeline data models accordingly.

IV. PROTOTYPE SYSTEM AND EXPERIMENTS

The described hyperlink pipeline framework is implemented as a JavaScript library based on HTML5 drag&drop JavaScript API widely available in many major desktop web browsers, including Internet Explorer 9+, Firefox, Opera, Chrome, and Safari. The library consists of the Pipeline Factory Module and the Microdata Module.

Whenever a user drags a source hyperlink and drops it to a destination hyperlink, the `ondragstart` event handler on the source hyperlink captures the necessary source data, and the `ondrop` event handler on the destination hyperlink captures the necessary destination data to call the Factory Module, which will call the JavaScript Microdata Module.

The library is used in several web applications for real-time communication and collaboration based on WebRTC and Google MAP JavaScript API. Although these JS APIs are widely supported, only Chrome [16] has full support for

WebRTC JS API [14]. Therefore, the web applications that require WebRTC are only tested on Chrome at this point.

We used a Tomcat 7.0.27 server which provides the built-in support for WebSocket Java API needed for two-way communication between the web browsers in a LAN environment.

A. *One-to-One Video Call*

This application allows users to perform 3rd party call controls based on WebRTC API by creating a `phone(phone)` hyperlink pipeline from two phone hyperlinks. **Figure 2** shows the hyperlink pipeline in green on the left side and the two video views: self-view and peer-view on the right side.

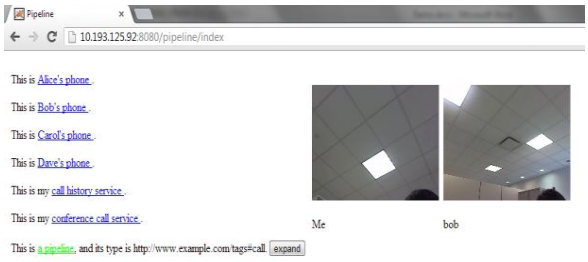

Figure 10: A phone(phone) pipeline

B. *Resource Compositions*

This application uses hyperlink pipelines to sort web resources by space and time.

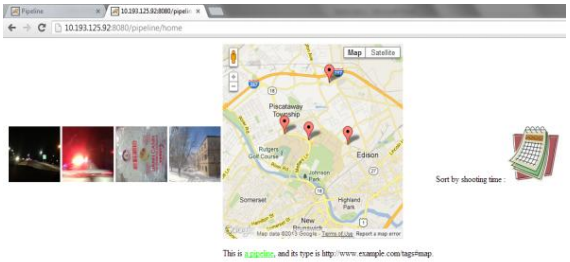

Figure 11: A map(picture) pipeline

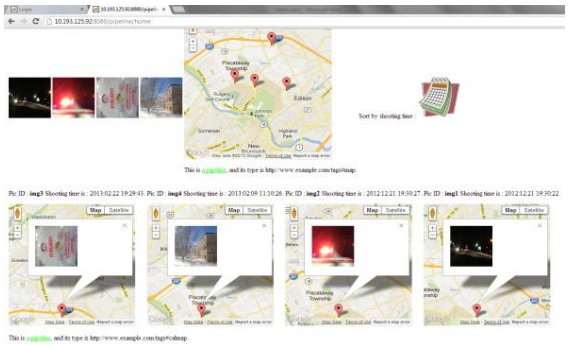

Figure 12: A calendar(map(pictures)) pipeline

To sort images by their shooting locations (Figure 11), the user creates a `map(pictures)` hyperlink pipeline from some images (top left) and a Google Map (top middle).

To sort images by their shorting time (Figure 12), the user creates a `calendar(map(pictures))` hyperlink pipeline from the previous pipeline and a calendar hyperlink (top right).

C. *Share Resources between Browsers*

Users in real-time communication and collaboration can share web resources, such as map, by creating hyperlink pipelines from the resources. Figure 13 shows a hyperlink pipeline `call(map)` from the call (Figure 10) and a Google map. The shared map automatically shows up in peer browser and both users can manipulate the map, such as move, zoom and add a picture, at the same time. All map manipulations are synchronized over the WebSocket between the browsers in real-time.

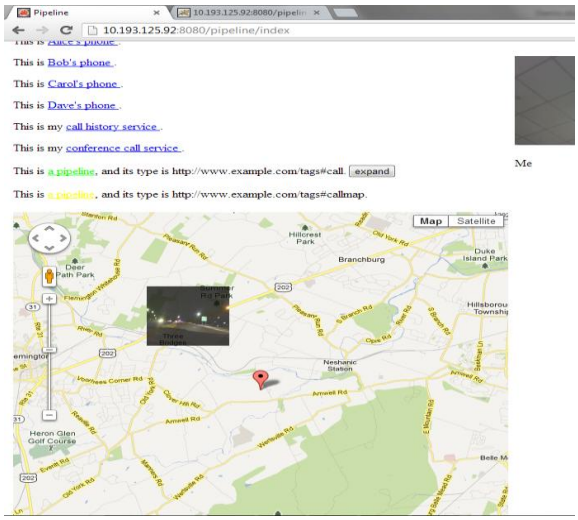

Figure 13: A call(map) hyperlink pipeline

D. *Merge Calls to Conference*

Figure 14 shows a hyperlink pipeline `call(call)` to merge two 2-party calls into one 4-party call from two call hyperlinks.

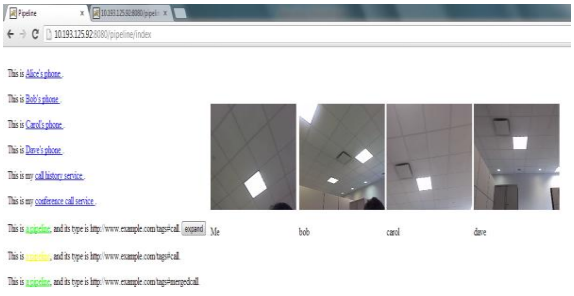

Figure 14: A call(call) hyperlink pipeline

E. *Delete Hyperlinks from Pipeline*

Figure 15 shows that deletion of source hyperlink from a hyperlink pipeline can itself be represented as hyperlink pipeline `trash(link)` from the link to be deleted and a special trash hyperlink.

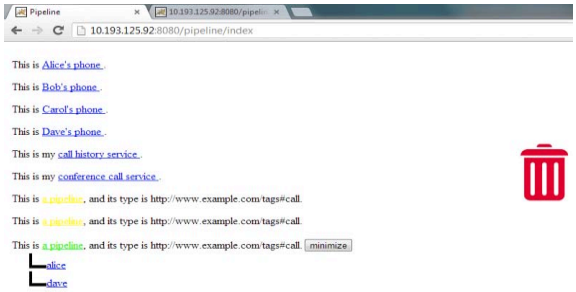

Figure 15: The trash(link) hyperlink pipeline

F. Performance Tests

We measure the delay (d) between the time the user releases a source hyperlink (t1) and the time the hyperlink pipeline is displayed on screen (t2), i.e. d=t2–t1 (milliseconds). The t1 and t2 are obtained by JavaScript function `windows.performance.now()`.

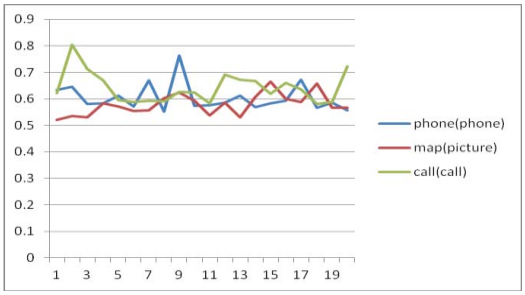

Figure 16: Hyperlink pipeline construction performance

The chart in Figure 16 shows 20 delay measurements for each of the three applications (IV.A, IV.B, IV.D) on a Firefox 20.0.1 browser on a TP410 computer with Intel Core i5 M560 2.67GHz (Dual Core) and 4.0GB RAM running Windows 7 Professional. The following table shows the averages (ms) and standard deviations for the measurements, which indicate the performance is stable and satisfactory.

TABLE I. AVERAGE AND STD OF PERFORMANCE

	phone(phone)	map(picture)	call(call)
AVG	0.60	0.57	0.64
STDEV	0.05	0.04	0.058

V. CONCLUSIONS

This paper describes the hyperlink pipeline architecture and its applications in real-time communication and collaboration. The contributions of this approach are summarized below:

- Proposed hyperlink pipeline as a new way to interactively compose reusable web services in web browsers without plug-ins;
- Developed a functional composition framework to tie hyperlink pipelines to distributed web services to support recursive construction and execution of hyperlink pipelines;
- Developed an efficient client-side Microdata model and format to standardize the construction, modification, and storage of hyperlink pipelines while allowing customized executions and presentations;
- Implemented several web applications based on hyperlink pipeline to demonstrate the idea is feasible and promising;

For future work, we will study more applications and efficient service negotiation, execution and authorization.

REFERENCES

[1] Pipeline (Unix): http://en.wikipedia.org/wiki/Pipeline_%28Unix%29.

[2] HTML5 Microdata: http://www.whatwg.org/specs/web-apps/current-work/multipage/microdata.html.

[3] Roy Fielding: Architecture Styles and the Design of Network-based Software Architectures, 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[4] http://pipes.yahoo.com/pipes/

[5] Rob Ennals, et al. Intel Mash Maker: join the web, ACM SIGMOD, vol 36 Issue 4, December 2007, pages 27-33.

[6] Enterprise Mashup Markup Language, http://www.openmashup.org/omadocs/v1.0/

[7] Greg Billock, et al. (editors): Web Intents, W3C Editor's Draft 23 April 2013, https://dvcs.w3.org/hg/web-intents/raw-file/tip/spec/Overview-respec.html.

[8] Diane Jordan, John Evdemon: Web Services Business Process Execution Language Version 2.0, 11 April 2007, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

[9] Erik Christensen, et al. (editors): Web Service Description Language (WSDL) 1.1, W3C Note 15 March 2001, http://www.w3.org/TR/wsdl.

[10] T. Berners-Lee, et al: Uniform Resource Identifier (URI): Generic Syntax, RFC 3986, January 2005, http://tools.ietf.org/html/rfc3986.

[11] N. Freed, et al: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, RFC 2046, November 1996, http://www.ietf.org/rfc/rfc2046.txt.

[12] Dan Brickley, R.V. Guha: RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation 10 February 2004, http://www.w3.org/TR/rdf-schema/.

[13] J. Gregorio, et al: URI Template, RFC6570, March 2012, http://tools.ietf.org/html/rfc6570

[14] Adam Bergkvist, et al: WebRTC 1.0: Real-time Communication Between Browers, W3C Editor's Draft 22 March 2013, http://dev.w3.org/2011/webrtc/editor/webrtc.html.

[15] Ian Hickson: The WebSocket API, W3C Candidate Recommendation 20 September 2012, http://www.w3.org/TR/websockets/.

[16] http://www.webrtc.org/.

[17] Cesare Pautasso: RESTful Web service composition with BPEL for REST, Journal of Data & Knowledge Engineering archive, Volume 68 Issue 9, Pages 851-866, September 2009

[18] Rosa Alarcon, et al: Hypermedia-driven RESTful service composition, ICSOC'10 Proceedings of the 2010 international conference on Service-oriented computing, Pages 111-120, 2010

[19] Pautasso, C.: RESTful Service Composition with JOpera, http://www.jopera.org/node/435, May 21, 2010

[20] Xia Zhao, et al: RESTful web service composition: Extracting a process model from Linear Logic theorem proving, 2011 7th International Conference on Next Generation Web Services Practices (NWeSP), Pages 398 - 403, October 19-21, 2011.