Hypermodal

Dynamic Media Synchronization and Coordination between WebRTC Browsers

Li Li[1], Wen Chen[2], Zhe Wang[3] and Wu Chou[1]
[1]*Shannon Lab, Huawei Technologies, Bridgewater, New Jersey, U.S.A.*
[2]*Contractor, Global Infotech Corporation, Kearny, New Jersey, U.S.A.*
[3]*CS Dept., Rutgers University, Piscataway, New Jersey, U.S.A.*

Keywords: Temporal Linkage, Synchronization Tree, RDF, Media Fragments URI, WebRTC, REST API.

Abstract: This paper describes a Web based real-time collaboration system, Hypermodal, based on the concept of temporal linkage between resources. The system allows the users to construct, manipulate and exchange temporal linkages organized as synchronization trees. The temporal linkage is defined by RDF <sync> predicate based on a novel use of Media Fragments URI and permits on-the-fly tree updates while the resources in the tree are playing. We propose RDF <mirror> predicate and a new protocol to correlate and initialize distributed synchronization trees without requiring clock synchronization. Moreover, we develop a new REST API optimized for efficient tree updates and navigations based on super nodes. The preliminary test results on a prototype system show the approach is feasible and promising.

1 INTRODUCTION

The true power of the Web is to organize distributed information in an unconstrained way through hypertext (Berners-Lee, 2000). With the vast amount of multimedia resources on the Web and the advent of WebRTC, there is an acute need to be able to link these real-time multimedia resources in an accurate and meaningful way.

The continuous nature of multimedia resources makes it insufficient to link them the way we link discrete documents or images. What we need is a new type of temporal linkage that can link intervals, regions or objects within multimedia resources. The application domains of such technology are wide open. We can link a person in a video stream to his home page so that the conference participants can find more about him without asking. When discussing a trip to Barcelona Spain, we can link the conversation to a Google map, a Wikipedia page, and a public transportation page about the city. Users of MOOCS websites can link part of an online video lecture to relevant segments of another video during a live discussion such that students can learn the same concepts from different professors. The agents that link the resources can also be machine programs, such as Speech Recognition, Machine Translation, or Face Tracking and Detection engines. For example, a moderator can schedule conference topics and a topic search engine can link resources relevant to the topics on time into the conference.

Temporal linkages can even link discrete resources without a temporal dimension, by treating them as continuous resources whose content does not change in small time scale. They can also link abstract resources that have a temporal dimension but no intrinsic content, such as a session. This generalization gives us the ability to temporarily link any types of resources in anywhere in a uniform way.

This paper describes a real-time collaboration system Hypermodal based on the concept of temporal linkage. The system allows the users to construct, manipulate and exchange temporal linkages in a meaningful way in real-time. Our goal is to create a mutual feedback loop between the system and the Web: any Web resources can be linked to the system and the links created by the system become part of the Web. To achieve this goal, we use as many standards as possible such that the components processing the temporal linkages can be developed independently but fully interoperate. Under this guideline, we address the following research issues in this paper.

If not constrained, the temporal linkages can

Li L., Chen W., Wang Z. and Chou W..
Hypermodal - Dynamic Media Synchronization and Coordination between WebRTC Browsers.
DOI: 10.5220/0004856100740081
In *Proceedings of the 10th International Conference on Web Information Systems and Technologies* (WEBIST-2014), pages 74-81
ISBN: 978-989-758-024-6

form an arbitrary directed graph which is difficult to manage for both users and machines. To solve this problem, we propose RDF <sync> predicate to define a generic temporal linkage based on a novel use of W3C Media Fragments URI standard (Troncy, 2012), such that we can construct synchronization trees, instead of directed graphs, to represent temporal linkages.

During a collaboration session, the synchronization trees are not static but are constructed incrementally and may change at any time when the resources in the tree are playing. To support on-the-fly updates, we develop a mechanism to play individual <sync> linkages without interrupting the resources in play.

In our system, whenever a user modifies a synchronization tree, the modification may propagate to other remote trees so all users have the same view. However, WebRTC Web browsers assign different URIs to the same multimedia resource, making it impossible to exchange <sync> linkages between synchronization trees that share the same resource. To address this problem, we propose RDF <mirror> predicate and a new protocol to correlate and initialize distributed synchronization trees without requiring clock synchronization between browsers and servers.

The components in our system need a protocol to communicate their updates to the synchronization trees. However, current RDF query and update languages are not designed for efficient updates to synchronization trees. To address this problem, we develop a novel REST API to navigate and update synchronization trees based on the concept of super node.

The rest of this paper is organized as follows. Section 2 surveys the related work. Section 3 defines the synchronization tree based on the <sync> and <mirror> relations. Section 4 describes how to initialize synchronization trees using WebRTC call control protocol. Section 5 introduces the REST API for managing the synchronization. Section 6 describes our prototype implementation and experimental results, and we conclude the paper with Section 7.

2 RELATED WORK

The approach described in this paper is different from the screen/application sharing offered by many Web conference systems. In screen sharing, shared content is read-only to all users except the owner, whereas in our approach, shared resources are interactive to all users. Screen sharing requires more network bandwidth to send the encoded video than the REST API to coordinate distributed synchronization trees. For example, Skype (Skype, 2013) requires at least 128 kbits/s for screen sharing, whereas we estimate the bandwidth required by our REST API is lower than 5 kbits/s. Furthermore, screen sharing creates a loophole for the Same Origin Policy (Rescorla, 2013), whereas our approach enforces this policy.

WebRTC (Bergkvist, 2013) is an ongoing joint effort between W3C and IETF to develop Web (JavaScript) and Internet (codec and protocol) standards to enable P2P real-time communication between Web browsers without any external plug-ins. Several open source Web browsers, including Chrome (WebRTC Chrome) and Firefox (WebRTC Firefox), already support some WebRTC functions and demonstrate certain degree of interoperability.

SMIL (Bulterman, 2008) is a XML dialect to express temporal synchronization between multimedia resources. However, SMIL does not support dynamic media synchronization. Once a SMIL document begins to play, we cannot modify the document to add new media or change the synchronization relations between existing media.

Nixon (Nixon, 2013) describes a research agenda for Linked Media, inspired by the Linked Data initiative, where the main approach is to link multimedia based on the conceptual relations between the fragments of the multimedia. Li et al (Li, 2012) describes Synote, a system to interlink multimedia fragments based on RDF and Media Fragments URI. Oehme et al (Oehme, 2013) describes a system to link a video to Web resources and overlay the resources on top of the video. Mozilla Popcorn Maker (Popcorn Maker) is an open source JavaScript library that allows a user to create multimedia presentations by layering Web information, e.g. Google map or Wikipedia page, at different intervals and regions of an online video.

However, all these approaches are for single user presentations or based on ad-hoc languages, not for multi-user and on-the-fly resource synchronization based on temporal linkage.

3 SYNCHRONIZATION TREE

Figure 1 illustrates the smallest Hypermodal system with two Web browsers connected by a Web server, where the synchronization trees are stored in the browsers and the <mirror> linkages are stored on the server.

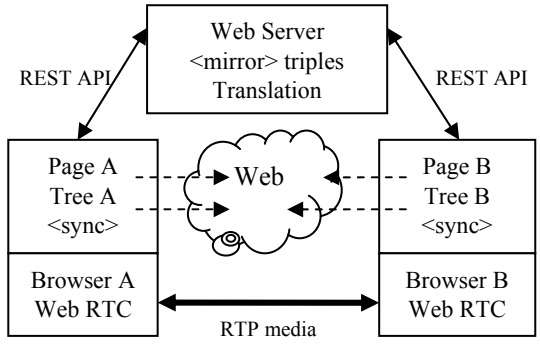

Figure 1: Basic Hypermodal system architecture.

A synchronization tree consists of nodes that define the intervals of resources based on Media Fragments URI, and edges that define temporal linkages between the nodes based on our RDF <sync> predicate, as illustrated in Figure 2, where the nodes are rendered as horizontal lines, whose lengths indicate the intervals, and the edges as vertical arrows whose positions indicate synchronization points. Both trees are rooted at the same session resource identified by URI0. The <mirror> linkages that link the same resources shared between the trees are rendered as the curved arrows.

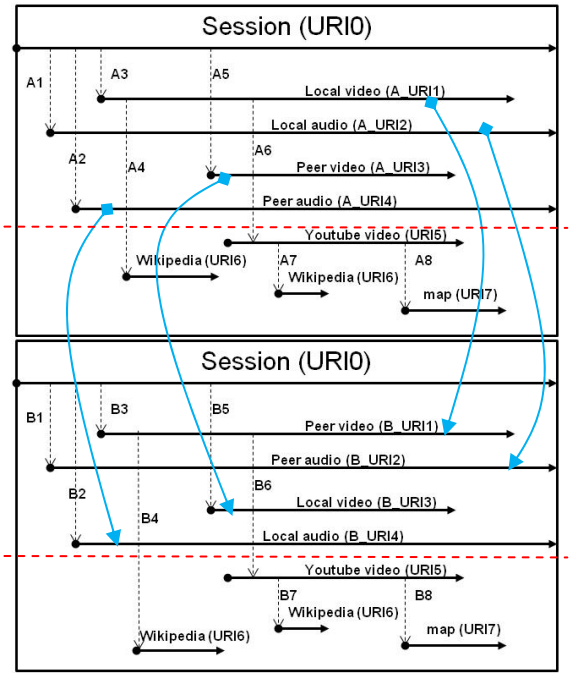

Figure 2: Correlations between tree A (top) and tree B (bottom).

Synchronization tree does not constrain the resources in any way as long as they can be identified by Media Fragments URI. For example, we can even synchronize two different intervals of the same video. If we treat the unique resources, not the intervals of the resources, as nodes, they can form a directed graph linked by the <sync> linkage. However, a tree is a more accurate representation of <sync> linkages, because different intervals of the same resource can be updated independently as distinct resources. For instance in Figure 2, the same Wikipedia page (`URI6`) occurs as two nodes in two <sync> linkages (`A4` and `A7`). When one linkage is changed, the other one is not affected. But if the page is changed, it will be reflected on both linkages.

Synchronization trees are constructed dynamically when the resources in the tree are playing: 1) new edges can be added anywhere by inserting RDF <sync> triples (edges); 2) the intervals and synchronization points of a resource can be changed by modifying the Media Fragments URI (nodes). This process is detailed in Sections 4 and 5.

3.1 The <Sync> Predicate

We propose <sync> RDF predicate whose subject and object are Media Fragments URIs. If `xs,xe,ys,ye` are nonnegative integers that denote the start and end time of a resource in second, then the canonical triple:

```
<URI_X#t=xs,xe> <sync> <URI_Y#t=ys,ye>
```

instructs the user agent to play the interval `[sy,ey)` of resource `URI_Y` within the interval `[sx,ex)` of resource `URI_X`. For example, the following triple:

```
<A_URI1#t=50> <sync> <URI5#t=30,120>
```

plays the interval `[30, 120)` of resource `URI5` when resource `A_URI1` reaches the 50th second and before it ends.

The <sync> linkage assigns different meanings to Media Fragments URI based its role in a <sync> triple: the subject interval `[xs,xe)` defines synchronization points on `URI_X`, not a new resource extracted from `URI_X`, whereas the object interval `[ys,ye)` defines a new resource extracted from `URI_Y`. This is why we can simultaneously attach many resources to intervals of `URI_X` while `URI_X` is playing, but at the same time maintain the tree structure by breaking `URI_Y` into independent portions.

The canonical <sync> triple requires the user agents to know the absolute play time of resources. This is difficult in distributed system when the machine clocks are not synchronized. To address this problem, we introduce two extensions to Media Fragments URI: relative delay and event

synchronizations. If `d` and `r` are positive numbers, then the relative delay triple:

`<URI_X#t=+d,+r> <sync> <URI_Y#t=ys,ye>`

plays an interval of `URI_Y` within the interval `[now+d, now+d+r)` of resource `URI_X`, where `now` denotes the normal play time of resource `URI_X` when the triple is to be played.

Event synchronization is borrowed from SMIL (Bulterman 2008). If `E1 and E2` denote events generated by resource `URI_X`, then the triple:

`<URI_X#t=E1+d,E2+r> <sync> <URI_Y#t=ys,ye>`

defines the resource play interval based on the time of the events. For example, to automatically display the PowerPoint at `URI_Y` when topic `t1` is being discussed in the session that generates event `t1_s` when `t1` starts and event `t1_e` when the topic ends, we could use the following triple:

`<session#t=t1_s,t1_e> <sync> <URI_Y>`

To play a canonical <sync> triple while the subject resource is playing, we use the process depicted in Figure 3, which takes the current subject play time and a <sync> triple as input, calculates the actual play time of the object resource, and produces a task as output. A <sync> triple can specify any start and end times, and it can be played at any time, but the process makes sure the actual play interval of the object resource is always within the current play time and the specified end time of the subject resource. If this is impossible, the object resource will not be played.

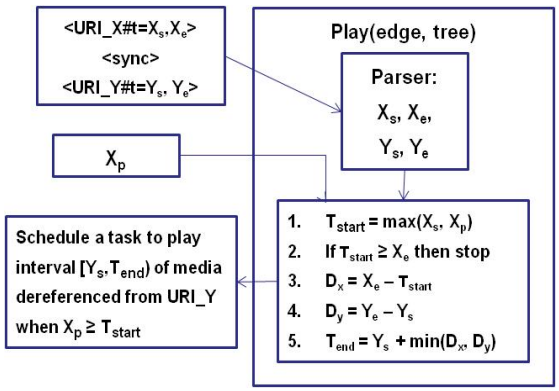

Figure 3: Process to play a <sync> triple.

To play a relative delay triple, the process translates `+d` to absolute play time `now+d`. To play an event triple, the process translates `E+d` to absolute play time `time(E)+d`. For example, if event `t1_s` happens at the 600th second, and `t1_e` happens at the 1200th second, then `<session#t=t1_s,t1_e>` is translated to `<session#t=600,1200>`.

The play process automatically compensates the network delay by adjusting the play interval. To illustrate this effect, suppose browser B sends browser A the <sync> triple: `<URI5#t=20,30> <sync> <URI6>` when the current play time of `URI5` at browser A is 19. When browser A receives the triple in 2 seconds, the current play time of `URI5` has advanced to 21=19+2. Browser A will then play `URI6` only for 9 seconds. This approach never rewinds a subject resource in play although it may abbreviate the play intervals of object resources. It is a reasonable approach if the network delay is relatively small compared to the play intervals. An alternative approach outlined in (Pan, 2012) is to "rewind" `URI5` back to 20 to play `URI6` for 10 seconds. However, such approach will not work if there are multiple conflicting "rewind" actions.

3.2 The <Mirror> Predicate

We propose <mirror> RDF predicate whose subject and object identify resources from the same source, as illustrated in Figure 4.

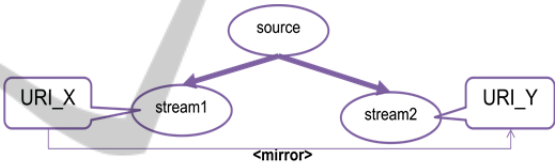

Figure 4: Resource model for <mirror> linkage.

For example, `<A_URI1> <mirror> <B_URI1>` indicates the two URIs in Figure 2 identify two resources whose streams come from the same video camera. The <mirror> linkage is not equivalent to RTP SSRC (Perkins, 2008), because two resources in the <mirror> linkage can have different SSRC values. For example, a media device may mix a video stream1 with an advertisement stream2 to create a picture-in-picture stream3. Although stream1 and stream3 have different SSRC, they are still regarded as the same by users because their main contents are the same.

The <mirror> closures can be computed based on the following properties:

1. Commutative: `X <mirror> Y => Y <mirror> X`
2. Transitive: `X <mirror> Y, Y <mirror> Z => X <mirror> Z`

With these closures, the translation between <sync> triples is straightforward as shown in Figure 5.

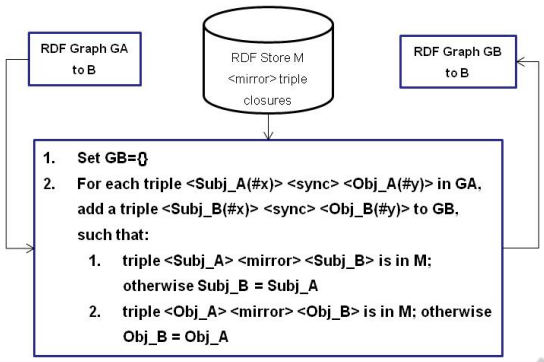

Figure 5: translation based on <mirror> closures.

4 TREE INITIALIZATION

When a user joins a collaboration session, the synchronization tree is initialized automatically by the system with the session and whatever media streams the user chooses to send and receive. To avoid performance overhead, the best approach is to embed the tree initialization protocol within the regular session establishment protocol. Figure 6 illustrates this technique using WebRTC offer/answer protocol (Rosenberg, 2002) with two additional messages `ack` and `ok`. The server sends the browsers the session URI and play time in regular `answer` (`t1` at step 6) and `ack` (`t2` at step 8) messages so that browsers can attach local media resources to the session at the correct time. The browsers send the server the correlation relations in `ack` (step 7) and `ok` (step 11) messages so that the server can derive the <mirror> relations between local URIs from the correlations. The correlation is established from media stream identifier (msid) maintained by WebRTC API.

When a browser wants to attach a resource to the session, it must know the current session time. However, the session time is maintained by the server clock, whose rate is unknown to the browser. One solution is to synchronize the clocks of the browsers and the server. But this requires additional protocol stack (e.g. NTP), which is often not available on the browsers and servers. This paper proposes two alternative approaches without requiring any dedicated time synchronization protocol.

In the *server-based* approach, the browsers use relative delay URI and let the server to figure out the session play time. For example, browser A can send `<URI0#t=+0> <sync> <A_URI1>` to the server, which calculates the session play time `tx` by: `tx=now-d(S,X)` according to the current session time `now` and the network delay `d(S,X)≥0` between the browser and the server. The advantage of this approach is that the browser does not need to know the session time maintained by the server. The constraint is that the server needs to store the synchronization trees for the browsers.

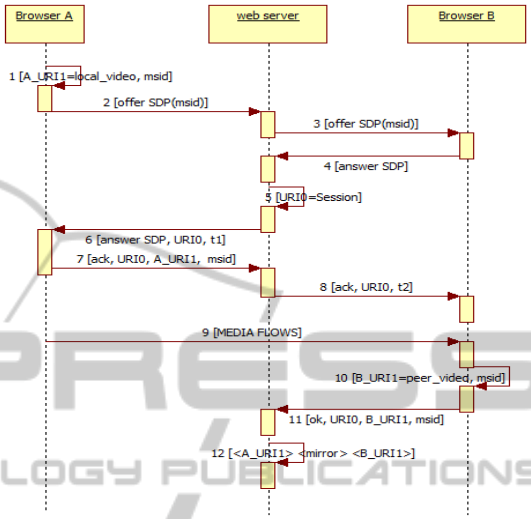

Figure 6: Tree initialization and correlation process.

In the *client-based* approach, each browser estimates the current session play time using its own clock. When a browser first receives the session time `t0` at its local time `t1`, it records these numbers. When it sends a <sync> triple at time `t2≥t1`, it estimates the current session time `tx` by: `tx=t0+t2-t1` and uses `tx` in the <sync> triple. The advantage of this approach is that server does have to store the synchronization trees. The disadvantage is that the estimated session time may be inaccurate when the client clock differs from the server clock.

5 SYNCHRONIZATION TREE REST API

In our system, users can frequently add <sync> triples, update the triples, or delete a triple. Users may also navigate and explore synchronization trees. Different user agents may accept different formats of a synchronization tree, e.g. XML, JSON or RDF Turtle (Manola, 2007). Servers and user agents need to cache and store the trees at different locations and formats. These use cases led us to choose REST API to encapsulate the synchronization trees.

There are several approaches (Gearon, 2013, Sesame REST API, Berners-Lee 2001, Tummarello 2007) to update RDF graphs using REST API (ref).

However, these approaches treat the entire RDF repository (i.e. a synchronization tree in our case) as a resource. To locate a RDF triple in the repository, a client must specify the subject, predicate and object of the triple. However, in a concurrent system, a client's knowledge about a triple will be out of date if other clients have changed the triple.

To address this problem and to reduce message size, our REST API treats each <sync> triple as a resource and assigns it a unique URI that does not change, which can be easily achieved because many RDF packages such as Jena (Jena) assigns unique internal identifiers to RDF triples. With this URI, a client can locate any <sync> triple without specifying its current state. This idea is illustrated in Figure 7 where each rectangle enclosing a <sync> triple represents a REST resource with a unique URI. For example, the triple: `<URI_0#T01> <sync> <URI_X#TX0>` is a REST resource identified by `URI_triple1`.

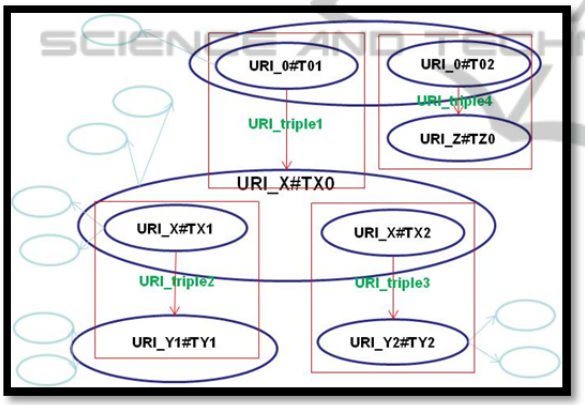

Figure 7: Resource model of synchronization tree.

However, these REST resources are not connected as the rectangles do not intersect. To address this problem, we introduce the concept of super node, represented by ovals that enclose the objects and subjects of different <sync> triples which share the same base URI (URI without fragment). For example, the super node `URI_X` contains `URI_X#TX0` of `URI_triple1`, `URI_X#TX1` of `URI_triple2`, and `URI_X#TX2` of `URI_triple3`. Because `URI_X#TX1` and `URI_X#TX2` are the children of `URI_X#TX0`, we map these relations to the corresponding REST resources to connect them as follows:

```
<URI_triple1> <child> <URI_triple2>
<URI_triple1> <child> <URI_triple3>
```

At the top of Figure 7 is a root super node that contains a set of "sibling" URIs: `URI_0#T01` of `URI_triple1` and `URI_0#T02` of `URI_triple4`. Similarly, we map this relation to the REST resources to connect them:

```
<URI_triple1> <sibling> <URI_triple4>
```

The members of a super node are updated whenever a new <sync> triple X is attached to an existing <sync> triple Y by the following rule: assert relation: <Y> <child> <X> if the subject of X has the same base URI as the object of Y. For new triple without parents, the REST API attaches them to an internal subject, so they can be checked for <sibling> relations.

5.1 Create Triple

Figure 8 shows POST request and response to add new triples to a tree resource identified by `URI_tree`, or to attach new <sync> triples to an existing one, by replacing `URI_tree` with the URI to the triple.

Figure 8: create a <sync> triple.

5.2 Retrieve Triple

Figure 9 shows the GET request and response that returns the requested triple and its neighbors connected by super nodes.

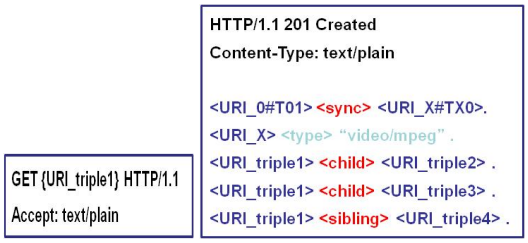

Figure 9: retrieve a <sync> triple.

5.3 Update and Delete Triple

Figure 10 shows three ways to update a <sync> triple: 1) subject interval; 2) object interval; or 3) both subject and object intervals.

We use DELETE message to a <sync> resource to delete it, which will stop its playback and remove all its child <sync> triples.

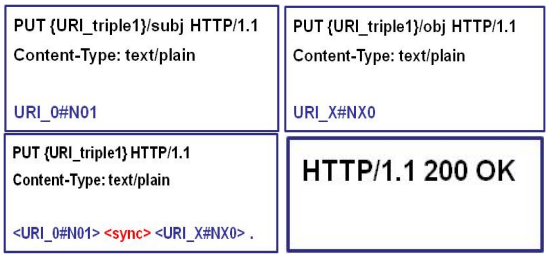

Figure 10: 3 update a <sync> triple.

6 PROTOTYPE SYSTEM

A prototype Hypermodal system was implemented based on the open source Mozilla Popcorn Maker engine (Popcorn Maker) and Jena RDF package (Jena).

Mozilla Popcorn Maker engine is implemented in JavaScript and runs in Web browsers. The engine allows a user to layer multimedia resources, such as YouTube video, Google map and Twitter stream, on a time axis. Users can change the start time and duration of the layered media during playback, and control the playback of these media with start, stop, seek, pause and resume actions.

We integrated a WebRTC call control module into Popcorn Maker so that users can make audio/video calls through the Web server. During the call negotiation, the Web server and the Web browsers initialize the synchronization trees as described in Section 4. We modified the Popcorn Maker to detect relevant events at one Popcorn Maker instance, translate these events to the REST API messages, and send these messages over WebSocket to the Web server as described in Section 5. The Web server will translate the messages based on <mirror> relations and broadcast them to the Popcorn Maker instances in other browsers.

Two screenshots of the modified Popcorn Maker interface are shown in Figure 11 for Alice (top) and Bob (bottom). Here Alice added a Google map beneath her video and the map is displayed on Bob's screen under Alice's video. Similarly, Bob added a recorded video to his video, and this video is shown on Alice's screen below Bob. Other Web resources, including Wikipedia page, Twitter page, chat window and images, can be added to the synchronization trees by the users as well.

We tested the performance of the system at the Web browsers (Lenovo Thinkpad 420 with Intel Core i5 2520M 2.50GHz (Dual Core) and 4.0GB RAM, 32-bit Windows 7 Professional) and the Web server (Dell OptiPlex 990 Mini Tower with Intel® Core™ i7 2600 Processor (3.4GHz, 8M), 16GB RAM, 64-bit Windows® 7 Professional) in a LAN environment, when users add new <sync> triples or modify them. For each operation, the following 4 time measurements were recorded.

1. BT: browser translates UI action to RDF triples and sends them in REST API request.
2. SP: the server parses the triples in request and stores them in Jena RDF models.
3. ST: the server translates the request triples and broadcasts them to other browsers.
4. BR: round-trip time at the browser from sending REST API request to receiving response.

Figure 11: Screenshots of Hypermodal prototype system.

The following tables summarize the results (in millisecond) for adding the triples (top) and updating the triples (bottom) respectively, each averaged over 20 runs.

Table 1: Task time for adding and updating triples.

Time/task	BT	SP	ST	BR
mean	47.55	0.66	0.10	18.45
std	7.98	0.34	0.01	1.93

Time/task	BT	SP	ST	BR
mean	14.05	0.21	0.24	6.9
std	1.93	0.04	0.05	0.85

These experimental results indicated that the proposed approach is feasible and promising since the total server processing time is less than 1 ms and

the total round-trip delay at browsers is 66 ms for adding triples and 21 ms for updating triples.

7 CONCLUSIONS

The contributions of this paper are summarized as follows.

1. A synchronization tree model based on temporal linkage defined by RDF <sync> predicate to allow dynamic modifications to the tree while the resources in the tree are playing.
2. A RDF <mirror> predicate and a new protocol to correlate and initialize distributed synchronization trees so that updates to one tree can be correctly translated to another tree without clock synchronization.
3. A novel REST API to support efficient updates on synchronization trees by treating <sync> triples as REST resources and connect them through super nodes.

For future work, we plan to extend the temporal linkage to spatial regions and objects in resources, study multimedia resource cache mechanisms for efficient constructions of synchronization trees, and security mechanisms to prevent unauthorized and malicious updates to synchronization trees, and apply the described Hypermodal system to more complex real-time collaboration applications.

REFERENCES

Bergkvist, A. et al (ed): WebRTC 1.0: Real-time Communication Between Browsers, W3C Editor's Draft 30 August 2013, http://dev.w3.org/2011/webrtc/editor/webrtc.html, Last Access: October 10, 2013.

Berners-Lee, T.: Weaving the Web, Harper, 2000.

Berners-Lee, T. et al: Delta: an ontology for the distribution of differences between RDF graphs, 2001, http://www.w3.org/DesignIssues/Diff, Last Access: October 10, 2013.

Bulterman D. et al (ed): Synchronized Multimedia Integration Language (SMIL 3.0), W3C Recommendation 01 December 2008, http://www.w3.org/TR/SMIL3/, Last Access: October 10, 2013.

Gearon, P. et al (ed): SPARQL 1.1 Update, W3C Recommendation 21 March 2013, http://www.w3.org/TR/sparql11-update/, Last Access: October 10, 2013.

Jena: http://jena.apache.org/, Last Access: October 10, 2013.

Li, Y. et al: Synote: Weaving Media Fragments and Linked Data, LDOW2012, April 16, 2012, Lyon, France, http://events.linkeddata.org/ldow2012/papers/ldow2012-paper-01.pdf, Last Access: October 10, 2013.

Manola, F. et al (ed): RDF Primer — Turtle version, http://www.w3.org/2007/02/turtle/primer/, Last Access: October 10, 2013.

Nixon, L. J. B.: The Importance of Linked Media to the Future Web, WWW 2013 Companion, May 13–17, 2013, Rio de Janeiro, Brazil, http://www2013.wwwconference.org/companion/p455.pdf, Last Access: October 10, 2013.

Oehme, P. et al: The Chrooma+ Approach to Enrich Video Content using HTML5, WWW 2013 Companion, May 13–17, 2013, Rio de Janeiro, Brazil. pages 479-480.

Pan, J., Li, L., Chou, W.: Real-Time Collaborative Video Watching on Mobile Devices with REST Services, 2012 Third FTRA International Conference on Mobile, Ubiquitous, and Intelligent Computing, pages 29-34, Vancouver, Canada, June 26-28, 2012.

Perkins, P.: *RTP, Audio and Video for the Internet*, Addison-Wesley, 2008.

Popcorn Maker: https://popcorn.webmaker.org/, Last Access: October 10, 2013.

Rescorla, E.: Notes on security for browser-based screen/application sharing, March 11, 2013, http://lists.w3.org/Archives/Public/public-webrtc/2013Mar/0024.html, Last Access: October 10, 2013.

Rosenberg, J. et al: RFC3264: An Offer/Answer Model with the Session Description Protocol (SDP), June 2002, http://www.ietf.org/rfc/rfc3264.txt, Last Access: October 10, 2013.

Sesame REST API: http://openrdf.callimachus.net/sesame/2.7/docs/users.docbook?view, Last Access: October 10, 2013.

Skype: https://support.skype.com/en/faq/FA1417/how-much-bandwidth-does-skype-need, Last Access: October 10, 2013.

Troncy, R. et al (ed): Media Fragments URI 1.0 (basic), W3C Recommendation 25 September 2012, http://www.w3.org/TR/media-frags/, Last Access: October 10, 2013.

Tummarello, G. et al: RDFSync: efficient remote synchronization of RDF models, The Semantic Web, Lecture Notes in Computer Science, Volume 4825. ISBN 978-3-540-76297-3. Springer-Verlag Berlin Heidelberg, 2007, p. 537, http://iswc2007.semanticweb.org/papers/533.pdf, Last Access: October 10, 2013.

WebRTC Chrome: http://www.webrtc.org/chrome, Last Access: October 10, 2013.

WebRTC Firefox: http://www.webrtc.org/firefox, Last Access: October 10, 2013.