The Web Browser Personalization with the Client Side Triplestore

Hitoshi Uchida[1,2], Ralph Swick[2], and Andrei Sambra[3]

[1] Canon Inc., Tokyo, Japan,
[2] World Wide Web Consortium, MIT CSAIL, Cambridge, MA USA
{uchida,swick}@w3.org,
[3] Decentralized Information Group, MIT CSAIL, Cambridge, MA USA
asambra@mit.edu

Abstract. We introduce a client side triplestore library for HTML5 web applications and a personalization technology for web browsers working with this library. The triplestore enables HTML5 web applications to store semantic data into HTML5 Web Storage. The personalization technology enables web browsers to collect semantic data from the web and utilize them for enhanced user experience on web pages as users browse. We show new potentials for web browsers to provide new user experiences by personalizing with semantic web technology.

Keywords: semantic web, HTML5, triplestore, inference, web browser

1 Introduction

1.1 Silos of Current Web Services

It is becoming common to manage our personal data on diverse web services in which we can create documents and presentation materials, manage personal schedules, send and receive emails, manage personal photo albums not only with web browsers on laptop PCs but also dedicated smart phone applications. It is also becoming common to synchronize PC data to cloud storage services which enable users not only to backup but also to open the synchronized files with smart phone applications on the go. Thanks to social network services, our daily lives became more communicative with friends and family by instantaneously sharing messages and schedules and postings.

However, current web systems don't provide enough options for users to mash up and utilize personal data which are distributed among those services users depend on in daily lives. A practical way to reuse our personal data among services is to exchange them with one of authorization protocols such as OAuth [4]. Through a handshake between OAuth client and OAuth provider, they exchange an access token which grants a permission to allow OAuth client to access user data in the OAuth provider. However, the traditional approach using the authorization protocol causes privacy issues for reuse of user data on other services.

[4] http://tools.ietf.org/html/rfc6749

Without disclosing our private data to 3rd party services, we can't reuse and mash up among them. It is also difficult to understand how securely our privacy is protected. After granting the access for user data to the 3rd party services, in general, users don't pay attention to which user data are still opened and accessible for 3rd party services they previously authorized. In addition it is difficult for users to understand what is happening during the handshake because the architecture of general authorization protocols depends on HTTP redirection through client web browsers. If our favorite services don't support the authorization protocol, we can't reuse personal data between services without migrating or copy-and-pasting them manually.

1.2 The User Data Centralization on 3rd Party Services

Web browsers are becoming more functional on not only rendering rich graphical web pages but also using latest innovative technologies such as real-time transfer protocols of WebSocket[1] and SPDY[2], video streaming interface WebRTC[3] and client side Web Storage[4]. However, we strongly depend on 3rd party services to use and manage our personal web data. By disclosing our private data to those 3rd party services, we get the benefits of reuse of our web data. We depend on their functionality to control our web data and security levels because current web browsers lack functionality for us to control personal web data. Though Web Storage can store user data in web browsers, this capability is used by web applications and not directly by users. We are always tied up with the architecture of current web system where web browsers mainly work as web application execution engine. Web browsers are users' personal tools and they should give more control for our personal data on the web without strongly depending on 3rd party services and disclosing our privacy.

1.3 What We Want to Achieve

The prior use cases and applications using semantic web were mainly server side. The major architecture was to store application data as triples with standardized formats like RDF[5] and provided endpoints which enabled client applications to retrieve the stored application data with a dedicated query language such as SPARQL[6]. However, there are a few challenges to apply semantic web to client side web applications, especially for the new HTML5[5] platform which is dramatically changing the existing web infrastructure.

In this paper we introduce a client side triplestore library, *triplestoreJS*[6], and semantic web browser plug-in, *Semantic Spider* [7], working with the triplestore. The *Semantic Spider* site [8] describes the detailed architectures and demonstrates

[5] http://www.w3.org/TR/html5/
[6] http://www.w3.org/2013/04/semweb-html5/triplestoreJS/
[7] https://chrome.google.com/webstore/detail/semantic-spider/ckdnmkbanbampnifpddcfdphonmfibkb
[8] http://www.w3.org/2013/04/semweb-html5/spider/

how it works. The source code of *triplestoreJS* and *Semantic Spider* is available in a Github public repository[9]. The triplestore is a wrapper application programming interface (API) for HTML5 Web Storage and enables HTML5 web applications to store semantic data triples into a web browser local store and search these stored triples with a dedicated triplestore API. The triplestore is expected to meet enough processing performance to enable web applications to work with it at reasonable speed. The semantic web browser plug-in is an HTML5 application working with the triplestore and currently works on Google Chrome as an extension. The fundamental architecture is to extract semantic data from web pages which a user visits in daily web browsing and save these data into the triplestore of the plug-in. In addition to the semantic data extracted from the web pages, the plug-in also collects personal semantic data from major social networking services (SNS). The collected semantic data represents the user interests precisely, and allows the web browser integrating the plug-in to work with the personal semantic data. This architecture to centralize user data into the web browser local storage has the potential to resolve privacy issues caused by the traditional approaches of permitting services to share data among themselves. We hope this challenge to apply semantic web technology into an actual web browser will provide new inspirations and expand the use cases.

2 Related Works

One of the well-known cases using semantic web for knowledge bases is *DBpedia* [10]. *DBpedia* is a crowd sourced community effort to extract knowledge information from Wikipedia and make the information accessible on the Web in structured form. Client applications can retrieve the extracted information with semantic web tools using RDF/JSON/CSV/HTML as data format and SPARQL as query language and can become more intelligent by integrating the knowledge bases. Currently the English version of *DBpedia* describes 4 million things, out of which 3.22 million are classified in a consistent ontology, including 832,000 persons, 639,000 places, 372,000 creative works, 209,000 organizations and so on. DBpedia is also available in localized versions in 119 languages. Because Wikipedia is growing and maintained by contributors from all over the world, *DBpedia* will be one of the central knowledge sources for intelligent applications. It is easy to create an encyclopedia application continuing to support new words and keep up to date with the internationalization support.

Another well-known case using semantic web for knowledge bases is search engines. Though a search engine crawler analyzes web pages to identify embedded data for web search, it also extracts semantic data RDF and RDFa[7] and microdata[8] which annotate the web contents. The semantic data provides machine readable information which helps client applications like the crawler to precisely understand the data types and composing properties of the web contents from the standardized structure format. All of latest major web sites

[9] https://github.com/shishimaru/triplestoreJS
[10] http://dbpedia.org/

integrate semantic data into their HTML pages. They expect that the crawler analyzes their web sites more precisely and collects higher quality information which will be useful for web search processing. Currently the major consumer of the semantic data integrated into web sites is the crawlers of the search engines. Though the number of web pages continues to increase with the additional semantic data to annotate the web contents, general users don't directly feel the full benefit in web browsing.

The Tabulator Extension[9] is a browser plug-in that visualizes RDF semantic data in tabular form which is retrieved from a server. Users can browse and edit the visualized RDF data in the web browser and reflect the modification against the originating server with SPARQL update messages. *Piggy Bank*[10] is also a browser plug-in which stores extracted semantic data during web browsing into the web browser and provide a user interface to review them on any web sites. However, these prior tools are for semantic web engineers; general users of web browsers don't get clear benefits how semantic web can change our lives on the web. Therefore, we demonstrate how we can utilize potential semantic web for enhancing browsing experiences as section 4.2-4.5.

There are existing development efforts in semantic web JavaScript libraries for web applications. *Green Turtle*[11] and *microdatajs*[12] are RDFa and microdata parsers in JavaScript, respectively. *sparql.js*[13] is a JavaScript SPARQL library which enables web applications to retrieve semantic data with sending SPARQL messages to SPARQL endpoints. *rdf-store.js*[14] is a comprehensive semantic web JavaScript library which supports JSON-LD/Turtle/N3 parsers and a persistent storage using HTML5 LocalStorage and a SPARQL query. Because the persistent storage is based on W3C RDF Interfaces API which is a set of basic primitives and a low level interface, it is for advanced developers who understand the semantic web well. In the other hand, because our *triplestoreJS* is based on an extension of RDFa API [15] whose architecture integrates the W3C DOM API general web developers are familiar with, the learning curve is gentler and it is easier to start web application development with the library.

3 A Triplestore for HTML5 Web Storage

We developed a triplestore wrapper library *triplestoreJS* in JavaScript which stores subject-property-value triples into HTML5 Web Storage[16]. Web Storage is a new persistent data storage of key-value pairs for web applications and enables to store application data into local storage of a web browser. The API of *triplestoreJS* is an extension on the RDFa API and provides operations to store and search triples. Though Web Storage is based on key-value and isn't

[11] https://github.com/alexmilowski/green-turtle
[12] https://github.com/foolip/microdatajs
[13] http://www.w3.org/2001/sw/wiki/SPARQL_Javascript_Library
[14] https://github.com/antoniogarrote/rdfstore-js
[15] http://www.w3.org/TR/rdfa-api/
[16] http://www.w3.org/TR/webstorage/

optimized for storing triples, *triplestoreJS* is organized for storing and searching the subject-property-value model. *triplestoreJS* also conceals the routine work to resolve CURIEs within RDFa. Therefore, it can reduce the development cost for web applications which store triples into a browser storage. Performance measurements of the triplestore are described in section 5.

3.1 A Save Operation

Web applications can store specified triples into Web Storage with a dedicated triplestore API. Because the Web Storage is based on the key-value model using string data type, the triplestore stores a subject as a key and a JSON string of the corresponding RDF properties and values as a value.

```
var st = new Triplestore();
/*
 * 'setMapping(prefix, URI)' registers a pair
 * of prefix and URI for CURIE processing.
 */
st.setMapping('foaf', 'http://xmlns.com/foaf/0.1/');
/*
 * 'add(subject, property, value)' method saves a triple.
 * If the subject already has a value for the property,
 * the new value is appended as additional values of the property.
 */
st.add('http://example.org/people#bob', 'foaf:name', 'Bob');
st.add('http://example.org/people#bob', 'foaf:homepage',
       'http://old.org');
/*
 * 'set(subject, property ,value)' method overwrites
 * all old values of the property with new one.
 */
st.set('http://example.org/people#bob', 'foaf:homepage',
       'http://new.org');
```

3.2 A Search Operation

Web applications can search the stored triples with simple APIs, getProperties(subject), getValues(subject, property) and so on. If the property parameter of getValues(subject, property) is null, all values which associate with the subject are returned.

```
//returns ['http://xmlns.com/foaf/0.1/name',
//         'http://xmlns.com/foaf/0.1/homepage']
var properties = st.getProperties('http://example.org/people#bob');
//returns ['Bob']
var name = st.getValues('http://example.org/people#bob', 'foaf:name');
```

```
//returns all values ['Bob', 'http://new.org']
var values = st.getValues('http://example.org/people#bob', null);
```

4 A Web Browser enhanced with Personal Semantic Data

We integrated *triplestoreJS* into Google Chrome as a Chrome extension to evaluate the potential to apply semantic web technology into the actual web browser experience. Especially, our challenge is to address how to utilize the personal semantic data collected during web browsing to enhance current and future user experience during browsing operations. The Chrome extension is an HTML5 application and has its own Web Storage.

4.1 Collecting Personal Knowledge Bases from The Web

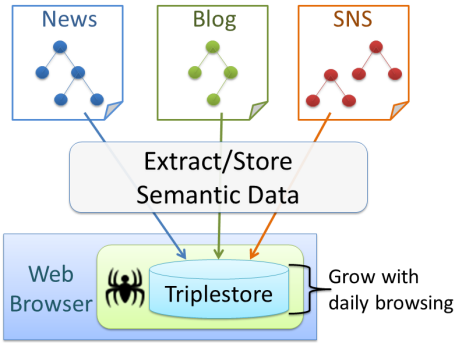

Fig. 1. Architecture for collecting personal knowledge bases

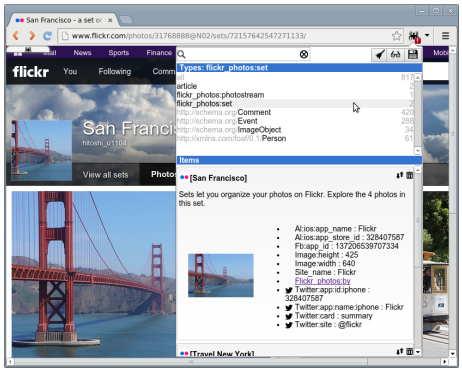

Fig. 2. Screenshot of a search function for stored semantic data

Any web contents annotated with RDFa or microdata are extracted from web pages users visit and are stored into the web browser automatically via the *triplestoreJS* as shown Figure 1. An automatic save function works when one of following conditions is met:

- When a user stays at a page longer than a specific period, e.g. 5 minutes
- When a user visits a page more than a specific frequency, e.g. 5 times

If a subject and the properties are already stored and now new additional properties of the subject are found in other web pages, the additional properties are appended to the subject. In this manner, the users' database of semantic data will grow based on their web browsing and the plug-in can provide more personalized functions with the stored personal semantic data.

Besides semantic data extraction from web pages, the plug-in supports login to Google and Facebook to gather user profile information, contact lists of friends, personal schedules and postings by the user and friends. Those user data and SNS data are converted into triples with standard vocabularies from schema.org [17] and Friend-of-a-Friend (FOAF) [18]. Therefore, after storing the user data from those services, the internal representations can be equally combined with general semantic data stored from web pages.

If a stored triple has an expiration date and time, it will be removed automatically from the triplestore as the browser does for cookie or cache expiration. When a user visits a web page, the plug-in monitors HTTP traffic and handles *Expires* header in HTTP response. When the semantic data is stored, the expiration information is also stored at the same time. Even if the auto-save function is always enabled by the user, this auto-remove function reduces the growth of stored semantic data in the triplestore.

The stored semantic data can be synchronized among browsers. If a user would like to copy stored favorite semantic data into another browser, the user can indicate which items she would like to synchronize. The plug-in stores the specified items into a dedicated Chrome synchronization storage and a plug-in working in another browser merges them into the local triplestore. The plug-in executes the synchronization process only when the activity on the browser is idle so as not to slow the browsing operation.

Figure 2 is a screenshot of the visualized stored semantic data. The user interface has three components: keyword search field, item type search field, and the search result field. The figure shows an example in which a user searched items whose types were '*flickr_photos:set*' and semantic data stored from online photo album Flickr is shown in the search result. The type is a service oriented name or a standardized URI or a term defined in schema.org or FOAF.

4.2 Suggesting Related Semantic Data

The plug-in detects related stored semantic data by calculating the similarities using inference processing based on the *Jaccard* similarity coefficient algorithm [13]. Suppose that A is a list of words composing a stored item X and B is a list of words composing a new item Y found in a web site the user is browsing. The plug-in calculates the similarity of A and B by equation (1) after sanitizing them by eliminating noise words such as numbers and determiners. If the similarity is above a pre-defined threshold, the plug-in recognizes the item X has a relationship with item Y. If the web site has several items of semantic data, the plug-in calculates the similarity for all combination of item X and Y. We used 0.5 as the pre-defined threshold for *Jaccard*.

$$J(A, B) = |A \cap B| / |A \cup B| \tag{1}$$

[17] http://schema.org/
[18] http://www.foaf-project.org/

One use case for this similarity function is to make a personal online photo album by mashing up user data distributed on the web. Figure 3 represents a possible architecture for the use case. Generally the online photo album should contain some relationships with other user data. For example, if the photo was taken while traveling, then the corresponding schedule would be also registered in a calendar service. If the photo has persons, some of them may be friends who are registered in SNS services. An SNS friend may post a new message for the online album representing her impression.

In step 1 of figure 3, the plug-in collects those personal user data from the web and stores into the local triplestore. In step 2, the plug-in finds related semantic data from the personal triplestore by calculating the *Jaccard* similarities and suggests this data to mash up with the online multimedia. Then the plug-in generates an HTML fragment including the detected semantic data and inserts this fragment into the web page. Figure 4 is a screenshot of the behavior of this use case on a Google+ photo album. The plug-in suggests the related schedule item of the trip containing the date and location and creator, a comment item for the album posted by a Facebook friend, the friend's SNS profile item containing the name and account id and organization, the album item containing the title and date and number of photos. The suggestion window created by the plug-in is minimized by default and toggled with the ESC key. One common issue for photo albums is how to easily add annotations to photos because this is a tedious work and we need to consider the contents of the annotation itself. In section 4.3, we introduce an annotation function which assists users to annotate photos with the suggested semantic data.

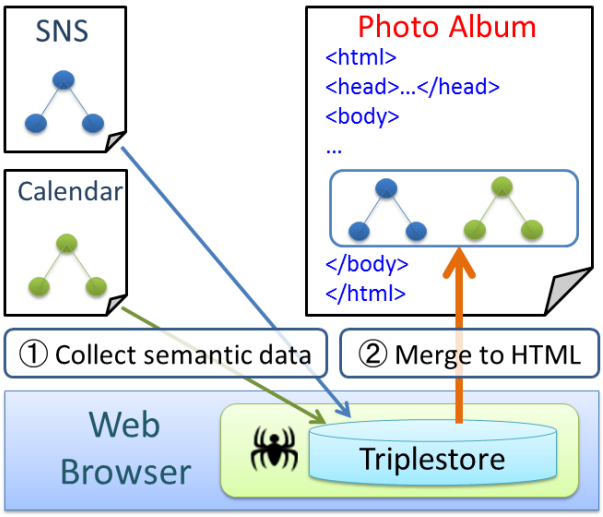

Fig. 3. Architecture of the personalized photo album

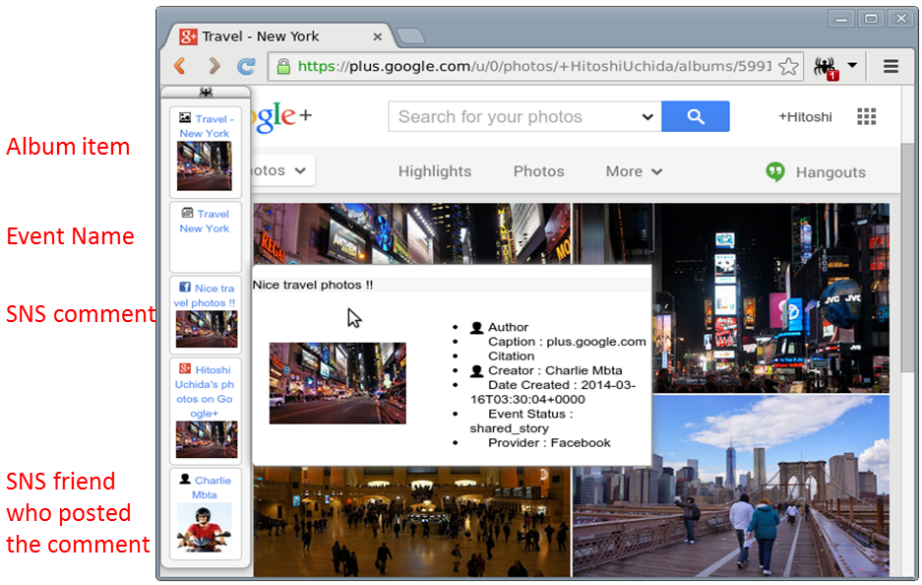

Fig. 4. Screenshot of the augmented Google+ photo album

In the current architecture, to mash up user data on several services users needed to authorize those services to allow access to user data. The access to user data always raises underlying privacy issues. If some of the services don't support an authorization protocol, users can't mash up and utilize their personal web data. The challenge for mashing up personal data continues to grow as users manage data in more dedicated services and the kinds of data become diverse: from an office domain like documents and calendars and emails to a social network domain like friend networks and published postings and photos and videos. The web browser itself has a potential to help the users to resolve the data access situation and we facilitate that with semantic web technology.

4.3 Assisting Media Sharing Operations

Our Chrome plug-in assists users to annotate and share online photos/videos with their friends and supports users' SNS activities by utilizing a stored contact list from FOAF. When a user Paul wants to send photos stored in online photo album services such as Flickr to a new friend whose contact information isn't registered to the service, if Paul visits the friend's homepage or SNS page including the friend's contact information as FOAF, this information is stored into the triplestore and the plug-in works as personal contact list manager without depending on and disclosing the private contact information to 3rd party services. Figure 5 illustrates this architecture. In step 1, the plug-in collects FOAF data from SNS services and Blog sites and stores the data into the triplestore. In step 2, when a user indicates to the plug-in to share a specified online photo

with the SNS friends, the plug-in shows the stored contact information of the SNS friends overlaid on the photo album site. If the user selects one of contacts, then the plug-in sends the photo using the specified contact information.

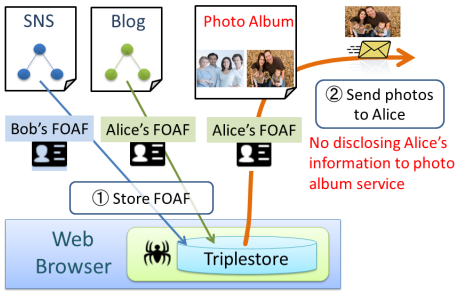

Fig. 5. Architecture of the assisted media sharing

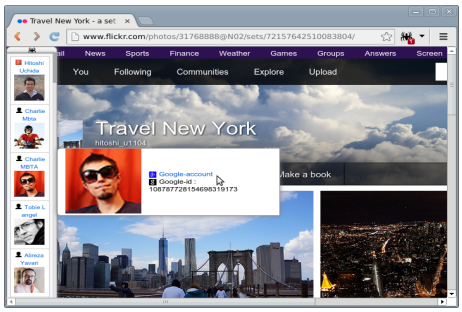

Fig. 6. Screenshot of showing a contact list

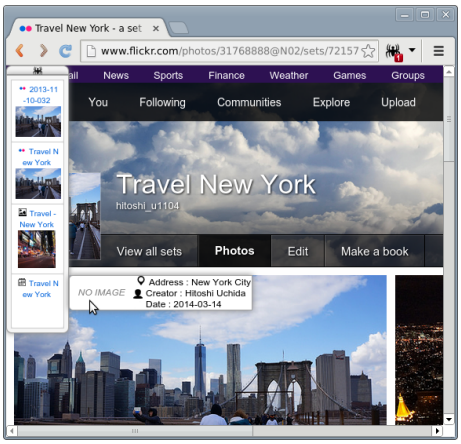

Fig. 7. Screenshot showing suggested tags for annotating a photo

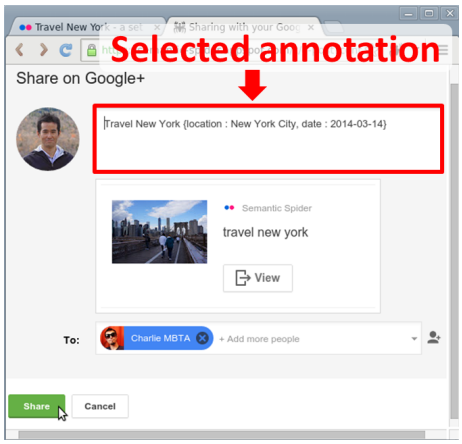

Fig. 8. Screenshot showing the photo sharing with a Google+ friend

For example, in a personal photo album of a trip to New York City on Flickr, when a user selects a photo with right click to share with an SNS friend and select 'share' from the menu provided by the plug-in then a stored contact list is overlaid on the web site as showed in Figure 6. When the user selects one of these contacts, the plug-in asks the user to select related semantic data to annotate the photo to be shared. In Figure 7, a schedule name of a trip to New York stored in Google Calendar and photo album items stored in Flickr are suggested

for annotating the photo. If the schedule name is selected, the user can send the photo through Google+ or Facebook or email with the annotated travel schedule as showed in Figure 8. The receiver can see the shared photo with the annotated travel schedule.

4.4 Assisting Text Input Operations

When a user is inputting a search keyword into text fields in web sites, the plug-in suggests candidates from stored semantic data. Though newer online services also suggest popular keywords or users' prior input histories, keyword suggestion provided by this plug-in is independent of any 3rd party services and derived from the users' personal semantic data collected from their web browsing and representing their interests. It is difficult for 3rd party services to collect this personalized data, however the plug-in learns them from the user's activities and we utilize this stored data for the keyword suggestion.

The keyword suggestion works in any text fields on web pages. Figure 9 shows how this works on online photo albums mashed up with personal schedules of stored semantic data. In step 1, the plug-in stores the user's personal schedules from calendar services into the triplestore. In step 2, when the user starts to input a search keyword in a search field on the online photo album, the plug-in finds items whose name are matched with the search keyword and generates an HTML fragment including the names of the matched items and appends it close to the search field the user inputs.

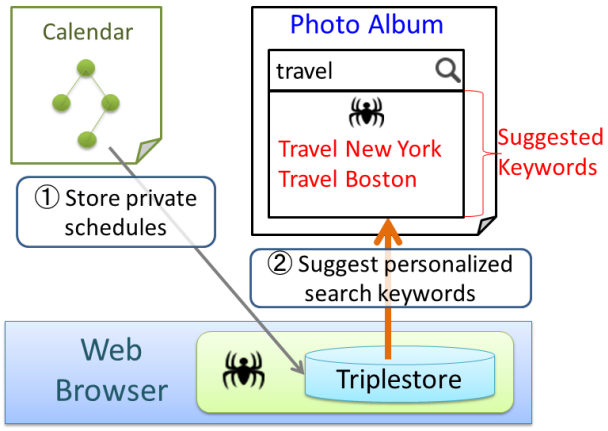

Fig. 9. Architecture of the keyword suggestion combining calendar and photo album

Figure 10 is the screenshot of the behavior on an online photo album Flickr. If a user manages his photo albums based on travel names like 'Travel New York' and starts to search with a keyword 'Travel' on a search field in Flickr, then because personal schedules from Google Calendar can be collected as semantic

data by the plug-in, the corresponding matched travel names are suggested for the candidate.

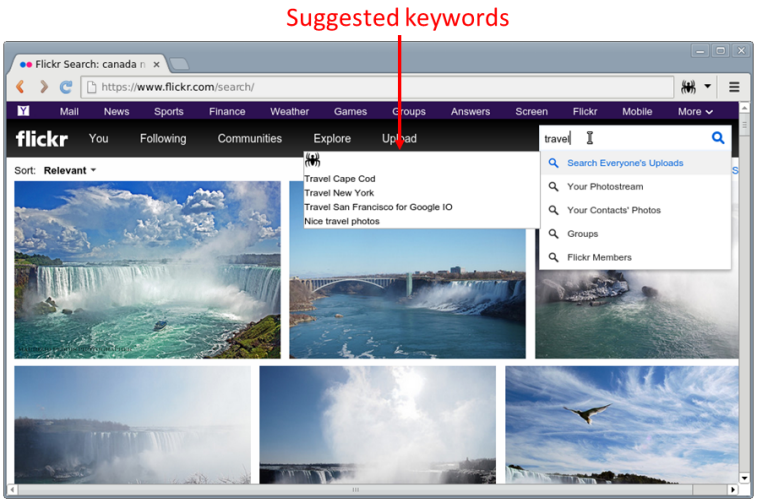

Fig. 10. Screenshot of the search keyword suggestion on Flickr

The plug-in recognizes semantic tagging on HTML input fields; If a text field in a web page is annotated with an attribute *@itemtype* with microdata or *@type* with RDFa which constrains the type of semantic data, the plug-in understands the annotation and suggests only keywords whose types of semantic data are matched with the specified type. For example, if the text field is described with the following markup, only keywords which are related to information whose type is 'http://schema.org/Event' are suggested.

```
<input type="text" itemtype="http://schema.org/Event">
```

It is difficult for a general web service to suggest such keywords which are derived from other web services because the service needs to support at least one authorization protocol to get permission to access user data and collect user interests and preferences from other web services beforehand. Our plug-in architecture allows the web browser to securely collect user interests from web browsing without changing services, therefore this keyword suggestion can be realized and works on any web pages.

4.5 Annotating Online Photos

Major online photo album services provide face annotation functionality which shows the name of the identified person if his or her name and face image are registeredin the services beforehand. Some online photo album services support

online machine learning which enables users to register new face images on their online photos and learn with the new faces and improve the accuracy of the face annotation.

Our plug-in supports a face annotation function which works on any online photo with the stored personal semantic data without disclosing the user's private information to the photo album services. The plug-in collects profile information of friends and family from any web pages distributing FOAF data and from SNS services like Google+ and Facebook through a login functionality the plug-in provides. If a user visits a blog site managed by a friend distributing FOAF data, the plug-in can easily collect profile data when visiting the blog. After storing FOAF data, the plug-in annotates online photos with the stored FOAF data using face identification processing we developed. For face detection, we used open source libraries *ccv.js*[19] and *face.js*[20] which enable web applications to detect the face locations on an online photo. We developed a face identification JavaScript library to annotate online photos with the stored semantic data.

Matthew A. Turk and Alex P. Pentland[14] describe a fundamental identification algorithm by comparing characteristics of the face to known individuals using principal component analysis (PCA). We trained our recognizer offline using a face database [21] of 13233 images and created a training result *Eigenfaces*. Figure 11 is the visualized *Eigenfaces* we acquired. We serialized this into JavaScript codes to integrate into the face identification processing of the plug-in. The plug-in compares each face of an online photo with SNS profile images from the stored semantic data using the serialized *Eigenfaces*.

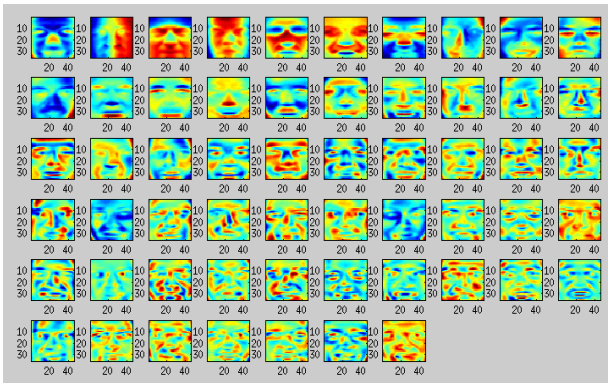

Fig. 11. Screenshot of visualized *Eigenfaces* trained with 13233 face images

Figure 12 is a screenshot of the face annotation functionality on an online photo. The gray rectangle represents the detected face location obtained from

[19] http://libccv.org/
[20] https://github.com/wesbos/HTML5-Face-Detection
[21] http://vis-www.cs.umass.edu/lfw/

face.js. If a user moves a mouse pointer to one of the gray rectangles, then the corresponding FOAF data collected from the web is overlaid on the photo. In figure 12, a Google+ account is suggested for the selected SNS friend and the user can share the online photo with the friend through Google+. Without depending on the functionality of 3rd party online services and disclosing our SNS friends' information to them, the face annotation can be realized with stored personal semantic data within client side scripts in real-time.

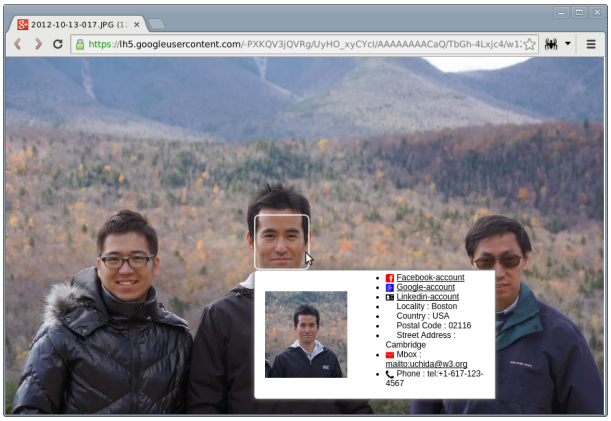

Fig. 12. Screenshot of face annotation with stored FOAF data

5 Evaluation

We measured the processing performance of the client side triplestore library used by the plug-in. For performance measurement we used semantic data collected from Google+ and Facebook and web sites including RDFa or microdata. The average number of properties per item was 9.5.

For the 'save' operation, we measured the performance by saving triples, a subject and properties and corresponding values into the triplestore. For the 'search' operation, we measured the performance by searching all values of specified subjects and properties. The searched data is the data stored by 'save' operation. Graph 13 shows the measured performance based on the number of triples. The horizontal axis is the number of triples and the vertical axis is the total time to complete each operation. For example, in case of 7698 triples, the average time to save was 1125 milliseconds, and the average time to search was 245 milliseconds. As we can see from the graph 13, the performance is linear with the number of triples. We think the core operations 'save' and 'search' using semantic data collected from actual web services meet sufficient performance for web applications to work on a general HTML5 platform.

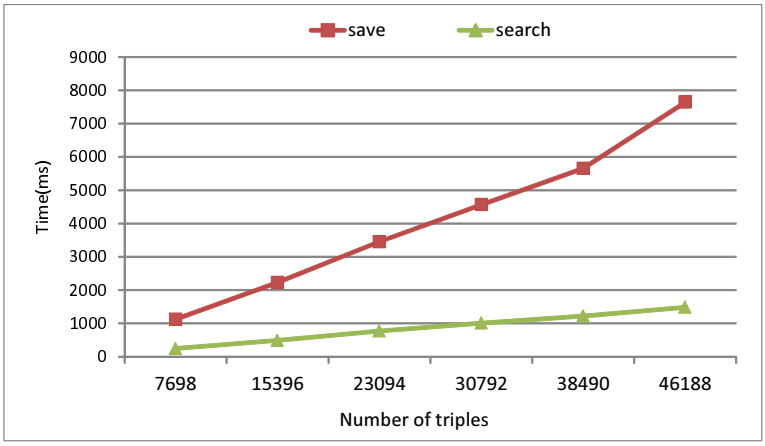

Fig. 13. The result of performance measurement based on the number of triples

6 Conclusion and Future Work

In this paper we introduced a client side triplestore library *triplestoreJS* for the HTML5 platform and a web browser personalization technology working with this library. Where existing examples of semantic web development were mainly server side applications and used for background development tools, general users didn't directly see benefits in daily web browsing. The motivation for general web developers to integrate semantic data into their web sites was to expect higher ranking on search results. We directly enhance user experience in the web browser by utilizing personal semantic data collected during prior browsing activity.

One of the features we achieved is to make online services more informative by mashing up with personal semantic data discovered on the web and securely collected into the web browser local storage. Another feature we achieved is to enhance user experiences in the web browser by suggesting candidates for text input operation and assisting to share and annotate online multimedia. This feature is especially helpful for small devices to improve user experiences whose user interface area are small and the input features such as hardware buttons are limited.

A future work is to replace the underlying Web Storage with IndexedDB [22] in the triplestore library. When we started the development, the standardization progress of IndexedDB was in W3C Candidate Recommendation and there was still a risk to change the API or behavior. And because we thought supporting mobile web platforms was also important and these didn't support IndexedDB at all, we selected Web Storage for the underlying storage of the triplestore. However, because IndexedDB can directly store semantic data as JavaScript objects without converting the objects to a string value to store into Web Storage,

[22] http://www.w3.org/TR/IndexedDB/

it is expected the performance for storing and searching semantic data can be improved. A performance comparison of the triplestore between Web Storage and IndexedDB will be also addressed.

We hope the work introduced in this paper will inspire new applications of semantic web technologies in HTML5 and will expand the use cases and be a promising bridge between them.

References

1. Pimentel, V., Nickerson, B.G.: Communicating and Displaying Real-Time Data with WebSocket. Internet Computing, vol. 16, pp. 45-53. IEEE (2012)
2. Cardaci, A., Caviglione, L., Gotta, A., Tonellotto, N.: Performance Evaluation of SPDY over High Latency Satellite Channels. In Personal Satellite Services, vol. 123, pp. 123-134. Springer International Publishing (2013)
3. Singh, V., Lozano, A. A., Ott, J.: Performance Analysis of Receive-Side Real-Time Congestion Control for WebRTC. Proc. of IEEE Packet Video, vol. 2013, (2013)
4. West, W., Pulimood, S. M.: Analysis of privacy and security in HTML5 web storage. Journal of Computing Sciences in Colleges, vol. 27, pp. 80-87. (2012)
5. Broekstra, J., Kampman, A., Van Harmelen, F.: Sesame: A generic architecture for storing and querying rdf and rdf schema. In The Semantic WebISWC, pp. 54-68. Springer Berlin Heidelberg (2002)
6. Quilitz, B., Leser, U.: Querying distributed RDF data sources with SPARQL. In The Semantic Web: Research and Applications, pp. 524-538. Springer Berlin Heidelberg (2008)
7. Dietzold, S., Hellmann, S., Peklo, M.: Using javascript rdfa widgets for model/view separation inside read/write websites. In Proceedings of the 4th Workshop on Scripting for the Semantic Web. (2008)
8. Heinrich, M., Gaedke, M.: WebSoDa: a tailored data binding framework for web programmers leveraging the WebSocket protocol and HTML5 Microdata. In Web Engineering, pp. 387-390. Springer Berlin Heidelberg (2011)
9. Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., Sheets, D.: Tabulator: Exploring and analyzing linked data on the semantic web. In Proceedings of the 3rd International Semantic Web User Interaction Workshop (Vol. 2006). (2006)
10. Huynh, D., Mazzocchi, S., Karger, D.: Piggy bank: Experience the semantic web inside your web browser. In The Semantic WebISWC 2005, pp. 413-430. Springer Berlin Heidelberg (2005)
11. Cai, M., Frank, M.: RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. In Proceedings of the 13th international conference on World Wide Web, pp. 650-657. ACM (2004)
12. Buraga, S. C., Panu, A: A Web Tool for Extracting and Viewing the Semantic Markups. In Knowledge Science, Engineering and Management, pp. 570-579. Springer Berlin Heidelberg (2013)
13. McAuley, J.: Machine grouping for efficient production. Production Engineer, vol. 51, pp. 53-57. (1972)
14. Turk, M., Pentland, A.: Eigenfaces for recognition. Journal of cognitive neuroscience, vol.3, No. 1, pp. 71-86. (1991)