Using semantics and user participation to customize personalization

Davide Eynard

HP Laboratories
HPL-2008-197

Keyword(s):
customization, personalization, semantics, RDF, Web browser

Abstract:

Personalization and customization are two techniques that allow systems to provide users an experience which is tailored to their preferences. Looking at the huge success many customization tools had, not just on a particular set of expert users but also on the wide Internet public, we decided to make personalization customizable, that is to allow users access their own personal profile and build custom applications that exploit it. To do this, we chose a specific use case: the information that users leave when they use their browsers, that is the history of the visited websites and the collection of their bookmarks. We then converted this information from the proprietary format used by Firefox 3 in an open, RDF-based format. We built tools to do the conversion offline or in realtime and a browser extension that uses these data to provide new, potentially interesting, information. The whole set of tools is built to be extensible, so it makes heavy use of configuration files that can be easily modified by users to provide additional features. As an example, we developed two plugins: one for the visualization of user history, using Google's MotionChart, and one for enhancing browsing with related information which are downloaded in realtime from Freebase. Finally, we evaluated our approach to enhanced browsing using both Freebase and Nielsen datasets.

External Posting Date: December 5, 2008 [Fulltext] - External Approved for External Publication
Internal Posting Date: November 21, 2008 [Fulltext]

Using semantics and user participation to customize personalization

Davide Eynard*

Hewlett-Packard
Palo Alto, CA, USA
davide.eynard@hp.com

Abstract. Personalization and customization are two techniques that allow systems to provide users an experience which is tailored to their preferences. Looking at the huge success many customization tools had, not just on a particular set of expert users but also on the wide Internet public, we decided to make personalization customizable, that is to allow users access their own personal profile and build custom applications that exploit it. To do this, we chose a specific use case: the information that users leave when they use their browsers, that is the history of the visited websites and the collection of their bookmarks. We then converted this information from the proprietary format used by Firefox 3 in an open, RDF-based format. We built tools to do the conversion offline or in realtime and a browser extension that uses these data to provide new, potentially interesting, information. The whole set of tools is built to be extensible, so it makes heavy use of configuration files that can be easily modified by users to provide additional features. As an example, we developed two plugins: one for the visualization of user history, using Google's MotionChart, and one for enhancing browsing with related information which are downloaded in realtime from Freebase. Finally, we evaluated our approach to enhanced browsing using both Freebase and Nielsen datasets.

1 Introduction

Personalization, under a general interpretation, means tailoring a product or a medium to a user, according to some personal details they provide. More specifically, on the Web it is the process of providing relevant content based on individual user preferences. These preferences can be provided to the system either implicitly or explicitly. Examples of explicit data collection include user ratings (i.e. stars on YouTube or diggs on Digg.com[1]), rankings, or wish lists (such as Amazon's wish list[2]). Examples of implicit data collection include observing which Web pages are viewed by users and the time they spend on them,

* Written under the supervision of Craig Sayers (craig.sayers@hp.com)
[1] `http://www.youtube.com`, `http://www.diggg.com`.
[2] `http://www.amazon.com`.

keeping a record of the items that a user purchases online, reading playlists of multimedia contents, and analyzing users' social networks.

Once user preferences have been collected into what, from now on, we will call a *user profile*, personalization engines can access them to provide a tailored Web experience with personalized contents or interfaces. Of course, there always needs to be someone who provides material which can be tailored on each person's profile.

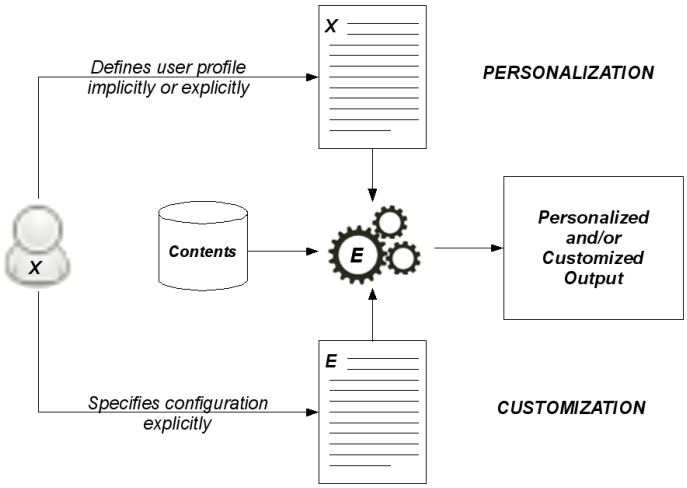


Fig. 1. Personalization and Customization.

Customization takes place when users directly affect the behavior of the engine which provides the contents, by specifying (usually explicitly) different parameters inside a *configuration*. A schema of how both personalization and customization work is shown in Figure 1. Looking at the figure, it appears quite evident how much the two are similar, and it also explains why they are often confused with each other: basically, for the tailoring engine (which takes care of personalization, customization, or both) the profile and the configuration are nothing but input parameters that tell how it has to operate; once it reads them, it can tailor contents to the user's desire. However, from the user perspective personalization and customization work in very different ways: in the former case, the only knowledge users have to provide is about themselves (i.e. who they are, what they like, and so on); in the latter, as users have to provide information about how the engine has to operate they need to know something about the system itself.

The fact that customization is so dependent on the knowledge of the system makes it much more complex for beginners: this is the reason why many

applications often have some settings enabled by default, so that unexperienced users will basically have to know nothing about the program to be able to run it. However, the more users get experienced the more they appreciate the possibility of customizing the behavior of their applications so that it better reflects their preferences. The result is that customization allows users to know what to expect from the programs they have configured: as an example, a media player can be configured not only to play the songs a user likes, but it can also play them following a particular order which is specified inside a playlist.

Even from this example it might seem that customization is preferrable to personalization, as it allows users to have full control of their systems, this is also useful to understand why sometimes personalization could be much better than customization. After all, who does not ever get bored of always listening to the very same sequence of songs? Systems like Last.fm or Pandora[3] have a great success lately as they provide personalization on radio streams: users can choose the bands or the genres they like most and, thanks to a system that learns their preferences, they listen not only to the songs they already know, but also to other unknown ones that they might like given their profile. In an interconnected world where the songs we know will always be much less than the ones we do not (and, by analogy, where all the Web pages we can read will be just a tiny part of the ones that are published), personalization has the advantage of providing us with new, interesting information which suits our taste.

A huge limit of personalization, however, is that it highly relies on the user profile: on the one hand this affects the system precision, as profiles often have to be built from scratch and it takes time for the system to learn users' preferences; on the other hand, giving away their personal information is often a problem for users, who want to protect their privacy and security as much as they can.

Summarizing, both customization and personalization have their pros and cons:

- *customization* allows to have a finer grained control on the system, allowing users to know what to expect; however, already knowing what to expect is sometimes boring and users might need to know too many technical details, which could make actual customization unfeasible for beginners. Also, as configurations usually show only the details of the system which are exposed to users, they might give them the false feeling of having control on something they can only see in part.
- *personalization* is less predictable and shows potentially new and interesting information, requiring virtually no technical knowledge on the user's side. However, it takes time to train and it works on a model of the user preferences (the profile) which can be wrong, limited, or incomplete. Also, the way the profile is used to generate the final results, and often the profile itself, are usually hidden to users, so they can only take advantage of what they are given, hoping it is going to provide something useful for them. The impossibility of choosing how their own profiles are used, together with concerns

[3] `http:/www.last.fm`, `http://www.pandora.com`.

about their privacy, often make users choose for the easiest choice, that is eliminate personalization[4].

Starting from these assumptions, we decided to give users the chance of *customizing personalization*, allowing them to access their own profiles with different layers of abstraction and providing them with an easy way to develop, share, and run applications which use their profiles to do something useful. Even if the number of users which are also programmers is small, both literature [1] and real-world examples (see Section 2.1) show that it is possible to exploit the different levels of user engagement to have different qualities of participation.

For our experiments we chose a very specific context: user history and bookmarks within the Firefox browser. We then developed an ontology to describe this knowledge and a set of tools that provide the following functionalities:

- offline conversion of Firefox 3.0 history and bookmarks to RDF;
- a framework for the development of personalization plugins, in the form of a Firefox extension which updates the ontology in realtime (i.e. while the user is browsing the Web) and uses information taken from it to customize the Web experience.

Finally, we developed some example personalization plugin which allow for information visualization, augmented browsing, and website customization.

The report is organized as follows: in Section 2 we describe the technologies and the software we used and the related work; in Section 3 we describe our approach and motivate our choices; in Section 4 we describe the architecture of our project and show the most important details of our software; in Section 6.1 we evaluate the project and in Section 7 we conclude and suggest future developments.

2 Prior work

Semantics and personalization have already been studied from different point of views. As an example, [2] provides some papers collected for the Semantic Web Personalization Workshop that took place at the 3rd European Semantic Web Conference in Budva, Montenegro.

Semantics can be used to provide better recommendations. [3] uses an ontology do describe a domain and maps keywords extracted from Web pages to concepts inside the ontology. The ontology terms are used to annotate the Web content and users' navigational patterns, and at the same time to do word sense disambiguation. The personalization framework built onto this ontology is then used to enhance the recommendation process with content semantics.

[4, 5] use ontologies as a base to calculate similarity between items on a website (respectively, Web pages or documents in a bibliographic peer-to-peer

[4] A very interesting real-world example of "customization to disable personalization" is shown at `http://googlesystem.blogspot.com/2007/04/how-to-disable-google-personalized.html`.

system). In the first case, a domain ontology is used and the similarity depends on the relations that connect concepts inside the ontology. In the second, the SWRC (Semantic Web for Research Communities) ontology[5] and the ACM topic hierarchy[6] are used as a base for different similarity functions.

Semantics can be used to improve search by filtering search results. [6] shows how information gathered from social bookmarking services can be used to provide better search results. A similarity metric has been suggested that uses the tags to describe the web results, and has been used to rerank search results.

Semantics can be used to describe and make the user profile portable. [7] uses Semantic Web technologies and the two ontologies SIOC[7] and FOAF[8] to model user information and user-generated contents in a manchine-readable way. The advantage is the possibility of reusing user data and having her network information saved in a standard and well known format.

Semantics can be used to personalize the tool. [8] describes the use of FOAF and a modification to the HTTP protocol to create personalized Web pages on the server side. The same authors in [9] show a similar approach, however suggesting the very interesting alternative of using personal information for personalization on the client side. [10] also shows how Web history can be used to customize the user interface on the client side, filtering or re-ranking the Web applications suggested by many social websites depending on which ones have been accessed more in the past. Finally, all the recent browsers (Firefox 3, Google Chrome, and Flock) provide some way to show users their most accessed pages: Firefox shows them as "Smart Bookmarks" in the bookmark bar, while both Chrome and Flock provide a personalized start page with the favorite pages. Flock currently even goes a step further, with a start page which has content gathered from different social Web applications and the possibility for the user to customize it.

2.1 Client-side Customization

The idea of customizing Web pages on the client is not new, but began to have a wide success only lately. The concept at the base is pretty simple: all the information entering a machine can be customized by the machine itself before they are shown to users: this allows them to have control over what they see, rather than using the computer as a TV[11].

Some of the fist programs which allowed users to do client-side customization on Web contents were Scott R. Lemmon's Proxomitron[9] and Philtron[10]. They worked as local proxy servers and were able to filter incoming HTML pages before they were loaded by the browser for rendering; most of the customization was done exclusively on the page contents.

[5] `http://ontoware.org/projects/swrc`
[6] `http://www.acm.org/class/1988`
[7] `http://sioc-project.org`
[8] `http://www.foaf-project.org`
[9] `http://www.proxomitron.info`
[10] `http://philtron.sourceforge.net`

While client-side customization tools did not seem very useful at their origins (and surely their complexity was much higher than the actual advantage of using them, at least for the large majority of users), they are now much more useful as they can customize not only page contents, but also user interfaces. This is partly due to the fact the Web is currently used not only as a repository of information, but also as a provider of online applications, which share with Web pages the same client-side modifiable representation. Moreover, latest browsers like Firefox are built to be customized: with its *extensions*, which are basically Javascript applications that are embedded into the browser and can access its APIs, it is possibile both to operate on the interface (i.e. adding functionalities) and on the page contents (i.e. filtering advertisements).

An example of Firefox extension, which acted as an inspiration to this project, is Greasemonkey[11]. It provides a set of functions, which allows to easily customize the content of Web pages, and a script manager, which allows users to write and share their scripts easily. As a proof of the interest of users towards this kind of products, this extension is one of the most popular ones on the Firefox Add-ons website with about 9.5 million downloads, and provides more than 20,000 scripts on its community website[12].

3 Our Approach

Our project is based on the assumption that personalization could provide much more useful results and be better accepted by users if they could have access to their own profiles. As a specific example, we decided to develop a tool which works on the browser history and bookmarks to provide personalized experiences to users.

There are different reasons why we chose this particular application. First of all, as both computer applications and multimedia entertainment are becoming more and more web-driven, there are going to be more and more chances to have a quite detailed personal profile just by browsing our history (that is, what websites we have visited) and our bookmarks (that is, what websites we want to remember). Then, as the quantity of information available on the Web increases, the classic hierarchical bookmarking paradigm is going to become old and very difficult to manage (supposing it is not already). Social bookmarking (i.e. del.icio.us), Web search (i.e. Google's search history), and starred items (such as in Firefox 3) are some possible solutions to this problem, but we believe that allowing users to easily access their own history could provide a bigger advantage to them. Finally, browser bookmarks and history are already available on every computer and, even if there are some tools which actually use them[13], they are not shared with a standard format.

[11] `http://www.greasespot.net`

[12] Data gathered from `https://addons.mozilla.org/it/firefox/addon/748` and `http://userscripts.org/scripts` on September, 10th 2008.

[13] See, for instance, the default start page in Google Chrome, which shows users their most visited websites.

The immediate advantage of our approach is that personal information such as history and bookmarks are available to the user in a standard and simple format; moreover, data is updated in realtime and can be accessed in any moment by any application, either internal or external to the browser. This makes a big difference, as Firefox 3 locks its database while it is running and the only chance to access it is from within the browser (i.e. programming an extension) or working offline when it is closed.

3.1 The Data Model

To build our ontology, we decided to study the schema of an existing history profile: we chose Firefox 3 `places.sqlite`, a SQLite 3 database file which contains both history visits and bookmarks. The file is usually saved together with Firefox user profile information: for instance, under Windows Vista it can be found in

```
<user home>\AppData\Roaming\Mozilla\Firefox\Profiles\<profile>\places.sqllite
```

Under Linux, instead, it is available in

```
~/.mozilla/firefox/<profile>/places.sqllite
```

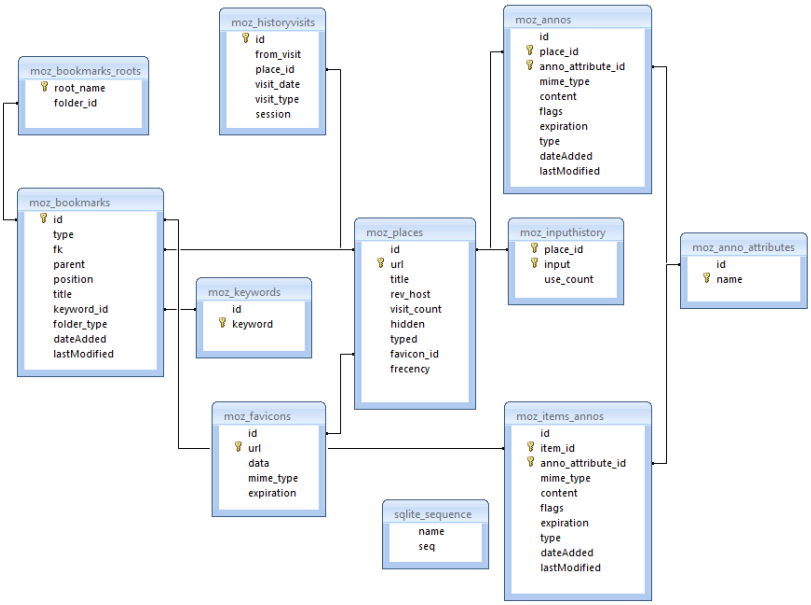


Fig. 2. The schema for Firefox 3 places.sqlite file.

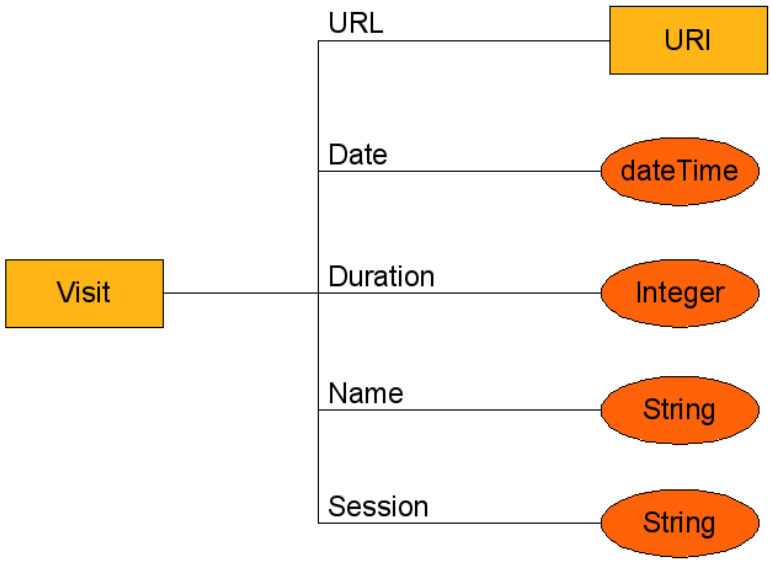


Fig. 3. Data model for visits.

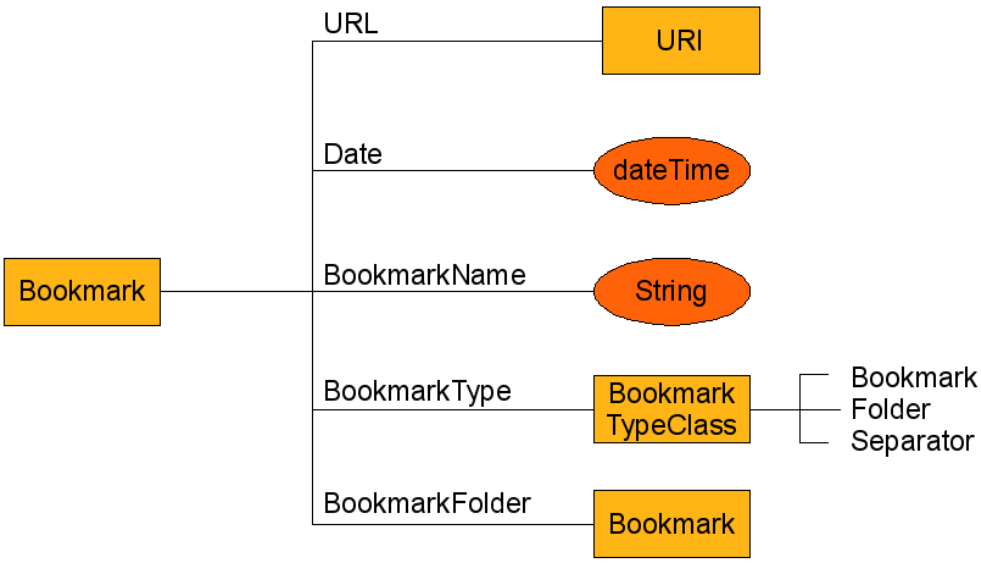


Fig. 4. Data model for bookmarks.

The database schema is shown in Figure 2. The tables we are currently gathering data from are:

- `moz_places`, which provides information about URLs. For instance, together with an id and the actual URL, it also has a field containing the page title, one with the number of time it has been visited, and one linking to the favorite icon;
- `moz_historyvisits`, which contains the history. It links to places from `moz_places` and provides information about the referrer, the visit date, the type of visit (i.e. a direct connection, a page redirection, etc.), and the browsing session;
- `moz_bookmarks`, which contains the bookmarks collection. For every bookmark different information are provided, such as the type (bookmark, folder, or separator), the name, the parent folder, and the dates of creation and modification.

The ontology we built (see Appendix B) has a simpler structure, taking into account only two main classes:

- **Visit** (see Figure 3) describes browser history and its properties are *URL*, *Date*, *Name*, *Session*, and *Duration*. Currently only URL points to a resource, while all the other properties point to literals (datatypes are xsd:DateTime for Date, xsd:String for Name and Session, and xsd:Integer for Duration). The last property describes the quantity of time that a user spent on a specific page and is not provided by Firefox history;
- **Bookmark** (see Figure 4) describes bookmarks and its properties are *URL*, *Date*, *BookmarkName*, *BookmarkType*, and *BookmarkFolder*. URL and the last two properties point to resources, while Date and BookmarkName point to literals whose datatypes are, respectively, xsd:dateTime and xsd:String.

One of the main advantages of having these data saved in RDF format is that the knowledge base can be expanded very easily, just by asserting new triples about the existing resources[14]. For instance, the one shown in Figure 5 is an RDF description of an element from the history: the last two lines represent the top tag used for that particular URL on del.icio.us and the total number of posts at that particular time. These two pieces of information have been added by another application (explained more in detail in Section 4.1) and are used together with the main data for visualization purposes.

4 Framework Architecture and Implementation

A very modular architecture has been decided for the project, allowing users to access the profile with different levels of abstraction and complexity (see Figure 6). Basically, user browsing profile (history and bookmarks) is saved in RDF inside an HSQLDB database; SDB provides the layer of abstraction that allows us to have a database-backed RDF store and offers access to data in (at least) the three following ways:

1. a generic Java application can directly access the RDF store using Jena and the SDB library;
2. Joseki can be configured to access the RDF store using SDB, then any external application can manipulate the store via HTTP requests, using the SPARQL endpoint;

[14] For a brief description of RDF, see Appendix A

```
PREFIX HPLBrowseData: <http://hpl.hp.com/schemas/FFontology#>
PREFIX mozh:          <http://hpl.hp.com/schemas/history#>
PREFIX xsd:           <http://www.w3.org/2001/XMLSchema#>

mozh:826 rdfs:type                HPLBrowseData:Visit ;
         HPLBrowseData:Session    "45524179520808800"
         HPLBrowseData:Referrer   mozh:0
         HPLBrowseData:URL        <http://delicious.com/>
         HPLBrowseData:Name       "Delicious"
         HPLBrowseData:Date       "2008-08-11T23:04:11Z"^^xsd:dateTime
         HPLBrowseData:DeliTopTag "delicious"
         HPLBrowseData:DeliPosts  741
```

Fig. 5. A history item in RDF, with related del.icio.us information attached to it.

3. information could be exposed through a (Java+Jena+SDB) API, then any external application can manipulate the store exchanging JSON data over HTTP.

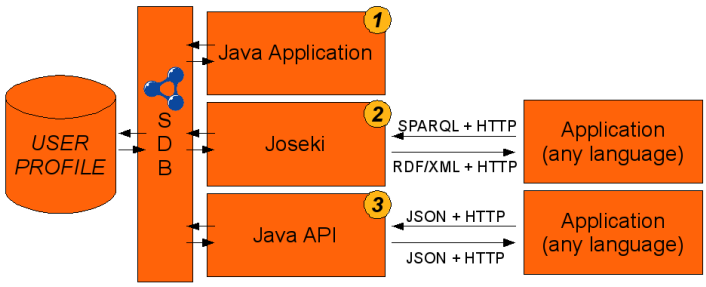


Fig. 6. Three possible architectures to access the RDF store.

Note that the second and the third approach, using Java applications that use Jena and SDB, are particular cases of the first one. However, each one provides a different interface to the profile and has its pros and cons:

- the first solution gives full control over the ontology through the Jena API and provides the best performances. However, it is so low-level that it also leaves programmers the burden of dealing with database connection details (i.e. the authentication parameters) and requires an in-depth knowledge about the ontology structure, RDF, and SPARQL. Moreover, it constrains programmers to use Java, Jena, and SDB.
- Joseki, allowing programmers to work at a higher level of abstraction, removes the constraints on programming language and does not require users to know any detail

about the database connection: basically, any application able to send SPARQL requests over HTTP can thus access the ontology. The advantages of this solution are clear: it relies on well known standards, it is very fast to deploy, and it works out of the box with other applications that already support the used protocols. However, it still requires programmers to know SPARQL and the structure of the underlying ontology.
– the API solution uses JSON, which is another well known standard and is already used in many large-scale projects (such as in Freebase or del.icio.us APIs). Its main limit is that it offers less control over the ontology, but its main advantage is that it does not even require programmers to know that an ontology exists one underneath the interface. Applications can be built with any programming language and do not require any knowledge about SPARQL.

4.1 Developed tools

Places2RDF is an example of type 1 application: it is a command-line tool that loads the `places.sqlite` file from a Firefox profile and converts it to RDF. Output is shown as text or can be directly saved inside an RDF store using SDB. We have bundled Places2RDF with a local, self contained, platform independent SPARQL endpoint built with the previously described technologies (Joseki, Jena, SDB, and HSQLDB), so that information could be immediately available to SPARQL-enabled applications.

The program has been built to be flexible enough to support different databases and ontologies. All the details about the data structure are saved inside a configuration file, which is built as an ontology: a sketch of its structure is shown in Figure 7, while the full schema and an example are shown in Appendix C.

The application relies, in a way which is very similar to the SquirrelRDF program (more details in [12]), on the concept of *mapping* between fields inside a table and properties. In the configuration file users can specify:

– the path of the database file;
– the output format for the ontology (i.e. "N3");
– a list of regular expressions for urls that have to be filtered away (for instance, secure http connections or selected domains);
– a list of *queries*: every query contains the actual SQL query, the default subject for the triples that will be created, and a list of *mappings*. Every mapping matches a field with a particular property in the output ontology, specifies the format of the data and allows to specify a default prefix that is currently used to convert table values to custom IDs.

The final results are similar to the ones shown in Figure 5. One of the main advantages of this tool is that it does not depend on the particular Firefox data file, so it could be adapted to become a general conversion tool for different formats (just to name one, Google Chrome's history and bookmarks file) and different ontologies.

RDFMonkey Customization Extension is an example of type 2 application. It is a Firefox extension that accomplishes the following tasks: first of all, it checks in realtime which websites the user is browsing or saving as bookmarks and gathers related data; then, it passes this information to its plugins, which can use it for different purposes.

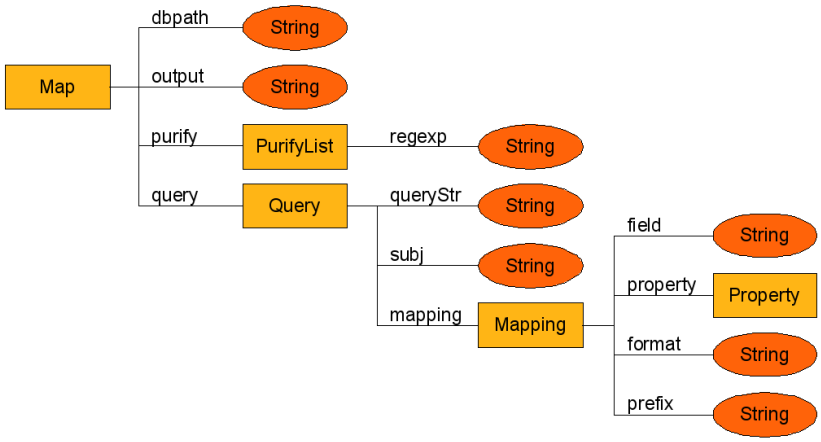


Fig. 7. Data model for Places2RDF configuration file.

The default action is to save data as RDF inside the ontology: this is done sending SPARQL UPDATE queries to the Joseki server. An example of such a query is the following, which updates the ontology stating that a bookmark has just been saved:

```
PREFIX HPLBrowseData:   <http://hpl.hp.com/schemas/FFontology#>
PREFIX mozb:    <http://hpl.hp.com/schemas/bookmarks#>
PREFIX mozh:    <http://hpl.hp.com/schemas/history#>
PREFIX xsd:     <http://www.w3.org/2001/XMLSchema#>

INSERT {
    "{23592b76-e4ea-4d93-81e3-bea0d2ed9cd1}0" a HPLBrowseData:Bookmark;
        HPLBrowseData:URL <http://www.google.com>;
        HPLBrowseData:BookmarkName "Google";
        HPLBrowseData:BookmarkType <HPLBrowseData:BookmarkClass>;
        HPLBrowseData:Folder "{a6358e91-aa9d-4c36-8626-8a4bc61ebaa1}1";
        HPLBrowseData:Date "2008-09-17T17:36:54Z"^^xsd:dateTime;
}
```

To test what the advantages of having user's browsing information exposed and accessible, we developed two plugins that exploit the data saved by RDFMonkey: the former takes advantage of the open and extensible nature of RDF to enrich it with information taken from del.icio.us and provides a visualization of user's browsing history; the latter augments Web browsing, using Freebase to find information related to the current page and providing links to other potentially interesting related data sources.

5 Example plugin: del.icio.us and Google's MotionChart

Our first plugin takes advantage of the open structure of RDF to merge history information with data downloaded on the fly from del.icio.us. Its architecture is shown in Figure 8. For every visited page (a), it accesses del.icio.us API[15] and (b) gathers information about it: if the URL has been saved in the social bookmarking website, then its popularity (i.e. how many people have saved it) and its most popular tag are retrieved. This information is then saved together with the history item inside the RDF store. In this way, the downloaded data is bound to the particular date the visit took place: it is thus possible to keep track of the evolution of this information during time.

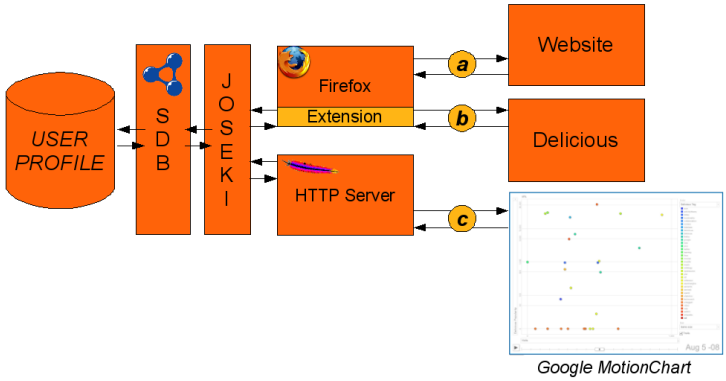


Fig. 8. Architecture of the RDFMonkey visualization plugin.

To visualize (c) the information we saved, we use Google's MotionChart[16]. This tool allows to manage different kind of information at the same time by drawing bubbles on a plane: data can affect their position, their size, their color, and can also change in time.

MotionChart access information saved in a particular format, that is the same one used by Google Spreadsheet to save its data. To make our history (augmented with del.icio.us information) available to MotionChart, we set up a service that accesses the RDF store and automatically converts the available information in the desired format. The conversion is made easy thanks to the fact that RDF properties also describe their own datatypes, so it is possible to set the right format for the different fields automatically. An example of this conversion is shown in Appendix D.

The final result inside MotionChart is shown in Figure 9 and 10. Every bubble is a visited URL and its color matches the color of the most common tag that has been used for it. A list of all the tags appears on the right and whenever the mouse pointer hovers on either a URL or a tag, their matching elements blink. For each axis it is possible to choose one of the following data sets: del.icio.us popularity, del.icio.us

[15] More info available at `http://delicious.com/help/api`.
[16] See `http://www.gapminder.org/graphs.html`.

number of posts per date, and total number of visits by the user. The same data can be used to change the bubbles' size. A scroll bar below the graph allows to change the data across the timespan in which information has been gathered. A "play" button allows to automatically see the evolution of someone's history.

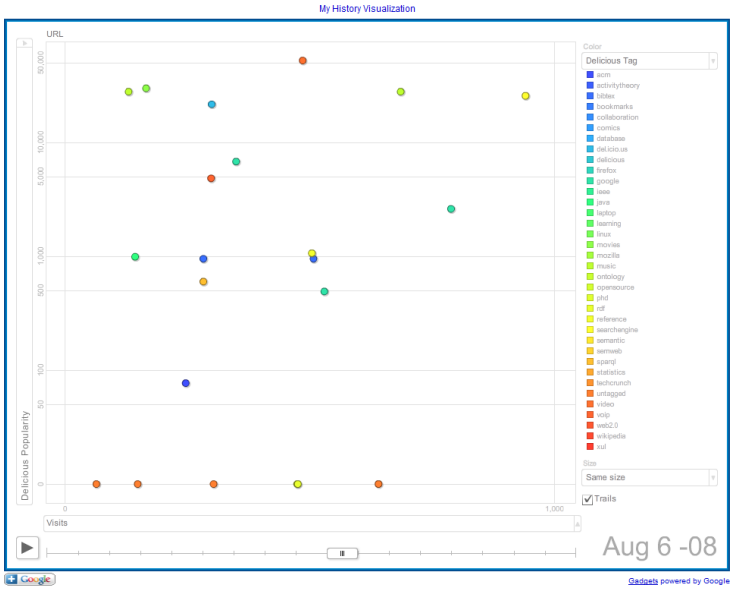


Fig. 9. History visualization through Google's MotionChart.

Having a chance to see all this information at a glance allows users to give many different interpretations to what they have visited in a period of time, depending on what's more interesting for them. For instance, Figure 9 shows the static information for a particular day, with number of visits on the x axis and del.icio.us popularity on the y axis: from this, a user might know wheter she has visited more or less websites than in the other days; she could easily understand which websites she has visited most (as they appear on the right) and if they are also popular (as they appear on the top) or not; watching at the evolution of her history in time, she could see whether the main topics she is interested in are changing or not, and whether she is a trend follower or not.

In Figure 10, instead, the x axis still matches the total number of visits at a particular day but on the y axis del.icio.us posts per day are mapped, while the size of the bubbles increases with del.icio.us popularity. From this view a user could see if trends change independently from the actual popularity of a website: in Figure 10, for instance, it is possible to see how delicious.com, the new URL for del.icio.us, is becoming more trendy than the old address, despite the fact that its overall popularity (i.e. the number of people who actually saved it) is still smaller.

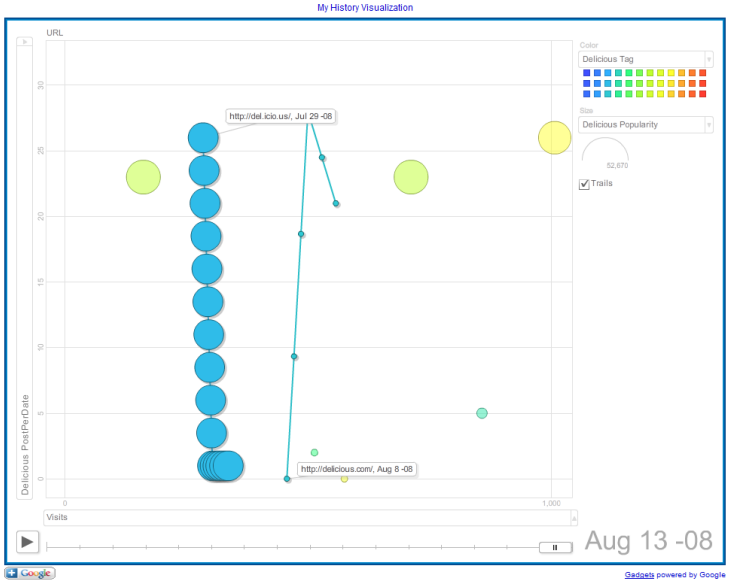


Fig. 10. Delicious.com versus del.icio.us on MotionChart.

6 Example plugin: Freebase

Our second plugin tries to enhance users' browsing experience by adding some links and information which are related to a webpage while the user is accessing it. The starting point for this small project is the following: a lot of topics in Freebase contain links that point to external websites which could provide additional information for the topics themselves. For instance, the topic about the band "Metallica" has links that point out to Wikipedia, Musicmoz, the New York Times "Times Topics" devoted to the band, and, of course, their official homepage.

Thus, whenever users visit one of these particular webpages, it is possible to follow the links backwards and see what topic contains them. From the topic, then, it is possible to get the matching Freebase types and some of their properties. Finally, it is possible to use these values to run other Web services that provide related information, in a way which is very similar to the one described in [13]. The architecture for this plugin is shown in Figure 11.

The tool shows the additional information inside a sidebar in the Firefox browser. The way information is presented depends on XUL[17] templates, which are stored in JSON format and can be easily modified by users themselves: thus, anyone can choose what kind of related information is shown by the browser and how it is presented to the user. As an example, the following code provides the information to show when a page related to a "/music/artist" is shown:

[17] XML User Interface Language. It is a language made by Mozilla to allow programmers build rich cross-platform applications which are very easy to customize.

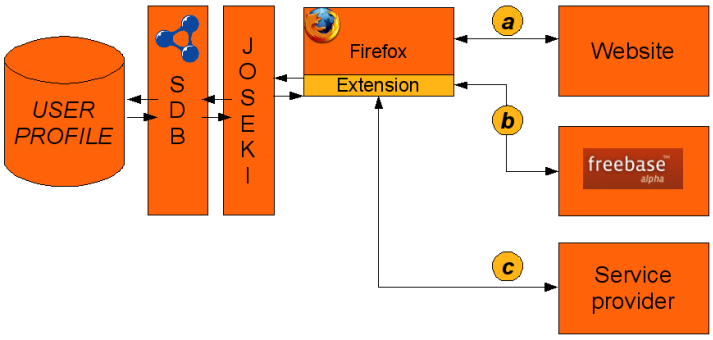


Fig. 11. Architecture of the RDFMonkey sidebar plugin.

```
"/music/artist" : [
                {
                "href"   : "http://www.last.fm/music/{{{NAME}}}",
                "image"  : "chrome://rdfmonkey/skin/lastfm.png",
                "text"   : "Listen to this artist on Last.fm",
                "xul"    : "link",
                "NAME"   : "eval(obj.source.name)"
                }
]
```

The previous code is used to load a XUL template containing an image, a link to Last.fm and a simple text. Thus, whenever a musical artist page is found, users are automatically shown a link that points to that artist's radio on Last.fm. This is just a very simple example, which takes information already available (the name of the Freebase topic which is linking to the current page) and just links to another website. However, with more advanced templates and additional queries on freebase it is possible to create something more useful and appealing for users: for instance, Figure 12 shows (a) how it is possible to directly embed the Last.fm radio into the sidebar, (b) a link to Amazon that appears whenever a book-related page is shown, and (c) a map that appears for anything that is described as a location, obtained by first querying Freebase for the object's coordinates and then passing them to the Google's Maps widget which is embedded in the sidebar.

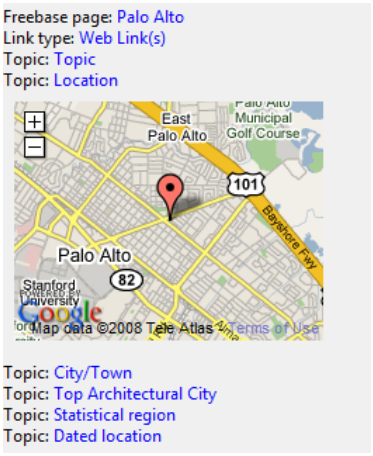
6.1 Plugin Evaluation

For our evaluation we mainly focused on the information we could gather from Freebase. For the tool to be actually useful, we need it to work with as many pages as possible and, at the same time, to provide information which could be considered useful by the users.

Our first task was to extract URLs appearing in Freebase as outgoing links (the ones that from now on we are going to simply call *Freebase URLs*, even if they are not Freebase's): our dataset has been built downloading the July 2008 "/common/webpage"

(a)

(b)



(c)

Fig. 12. Related content provided by the RDFMonkey Sidebar plugin.

data dump (Updated July 9, 2008) and searching for all the HTTP URLs present in the file. The total number of URLs extracted from Freebase was 310278.

Of course this number is ridiculously small when compared to the size of the whole Internet, but it has a completely different value when it's compared to the subset of the webpages that are actually visited by users. To prove this, we decided to calculate the incidence of Freebase URLs inside a dataset provided by Nielsen[18]. These data describe the anonymized browsing history of hundreds of thousands of users, together with a classification of the visited websites and some additional information about user census. For our work we chose to use only a subset of this dataset, related to the month of January 2008: this amounts to a total of more than 200 million page hits, done by a set of more than 61000 users.

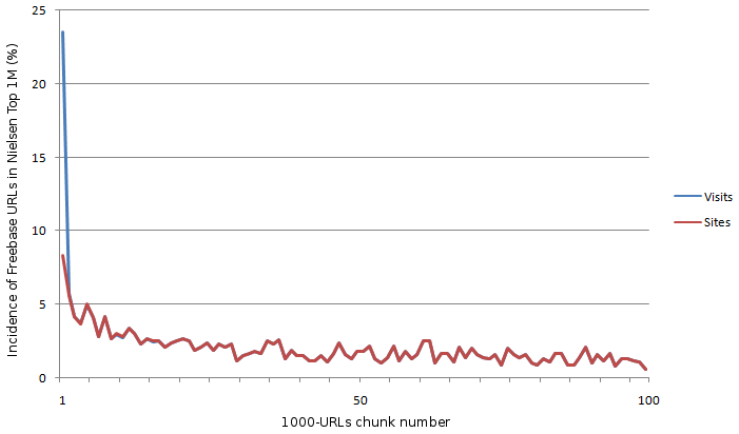
We ordered the set of visited URLs depending on how many times they had been accessed by users. We then kept the top million URLs, with the first one having 2.6 million hits and the last one only six. Inside that set, the total incidence of Freebase URLs is 0.77%: this means that out of the top million web pages only 7722 were also appearing in Freebase. This value, however, becomes much higher if we weight the URLs by the times they have been actually visited: out of about 59 millions visits, about 5.7 millions were directed to web pages that also appear in Freebase, with an incidence of 9.75%.

Given how much the number of visits can influence the final results, we also checked how the average incidence changed moving from the most popular URLs to the less visited ones. Thus, we divided the dataset in chunks of 1000 URLs each and we calculate incidence (both on sites and on visits) for each one of them. Then we plotted the resulting values both for each single chunk and as a global average that changes in time. The first plot is shown in Figure 13(a), where the top 100 chunks appear with their respective incidences: only the very first ones have significative values, then they stabilize around a very low threshold. After the first few chunks, the values for sites and visits also appear to be very similar as the weights tend to be distributed in a much more homogeneous way (i.e. the difference between the first URL and the 1000th is more than 2.6 million hits, the one between the 1000th and the 2000th is only 1800 hits). The second plot is shown in Figure 13(b) and shows how the total incidence changes while more chunks are taken into account. As expected, it decreases quickly at the very beginning and then almost stabilizes around the final value.

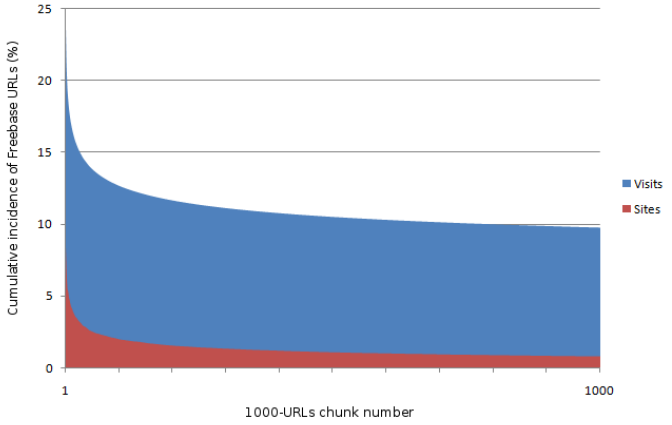
Checking the top URLs in Nielsen database we noticed that particular categories of links (such as search engines and online email providers) were more common than others. As we are interested in just particular classes of pages (i.e. we are not going to find our personal email messages linked into Freebase), we decided to take advantage of Nielsen's website categorization to run the same kind of statistics for each available class. The result is shown in Figure 14 and presents the incidence of Freebase URLs in each category, weighted by the number of visits. Out of the 15 categories, the one named "Search Engines/Portals & Communities" has the highest percentage (more than 23% of the visits are directed towards websites that are also linked inside Freebase); it is then followed by "News & Information" with more than 11% and then by "Finance/Insurance/Investment" with about 7%; all the other categories have an incidence which is below 5%.

Knowing which classes of URLs have a higher incidence of freebase URLs help us give an interpretation to the previous results: for instance, despite having high ranked URLs in Nielsen data, the category "Telecom/Internet Services" does not

[18] `http://www.nielsen.com`



(a)



(b)

Fig. 13. Incidence of freebase URLs inside the Top 1 million visited urls: values for single 1000-URLs chunks (a) and behavior of the overall value as the set grows bigger (b).

have a very high incidence, while News—which could provide more interesting related information—has a higher value.

Finally, it is worth noting that there are some websites which almost do not appear in the list of Freebase URLs, but that could be easily linked automatically. These are all those websites that are used as *authorities*, that is the ones that provide resources with a unique id such as Wikipedia, IMDB, Netflix and so on. As these IDs are also imported into Freebase, it is possible to link to a related Freebase topic starting from any page in these systems. This means that our tool could be particularly useful for specific classes of users, such as Wikipedia contributors or movie aficionados.

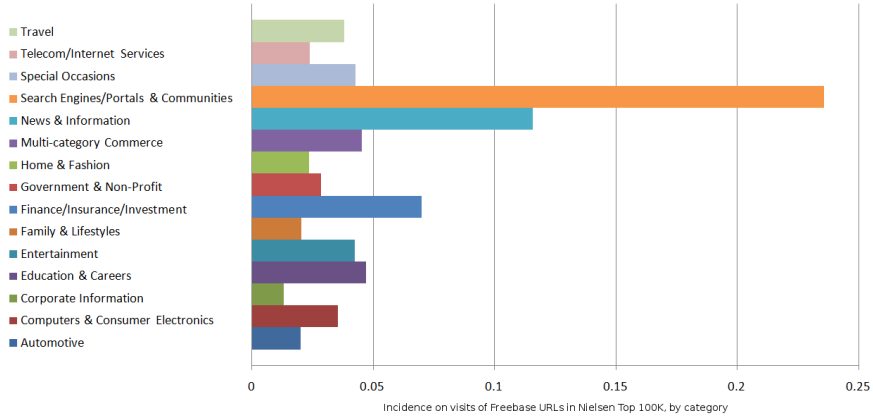


Fig. 14. Incidence of Freebase URLs inside visits, divided into Nielsen categories.

After checking the incidence of URLs, we decided to focus on Freebase types. The reason is that we are not just interested into having a particular page linked into Freebase, but also into having interesting related information about it. Basically we assumed that the more the types and the properties the topic has, the highest amount of potentially interesting related information it can provide.

Thus, we first ranked Freebase types according to their usage. Freebase has more than 4000 types and their distribution is shown in Figure 15. In the top 15 types (see Table 1) there are of course "common topics", that is one of the most generic (and from our point of view unuseful) Freebase types, but also musical tracks, artists and albums, people, and locations. In the top 100 a lot of other potentially useful types are present, such as cities, films and actors, books and book authors, restaurants, companies, and videogames.

Thus, if it is true that there are a lot of topics whose type is "/common/topic", it is also true that there are many which share other types. Moreover, even in the worst case, they can provide some additional information through their own links, for instance to Wikipedia: in fact, out of all the topics which share the common type, only 21810 do not have links to Wikipedia, while the remaining 1.1 million link at least to their matching Wikipedia article.

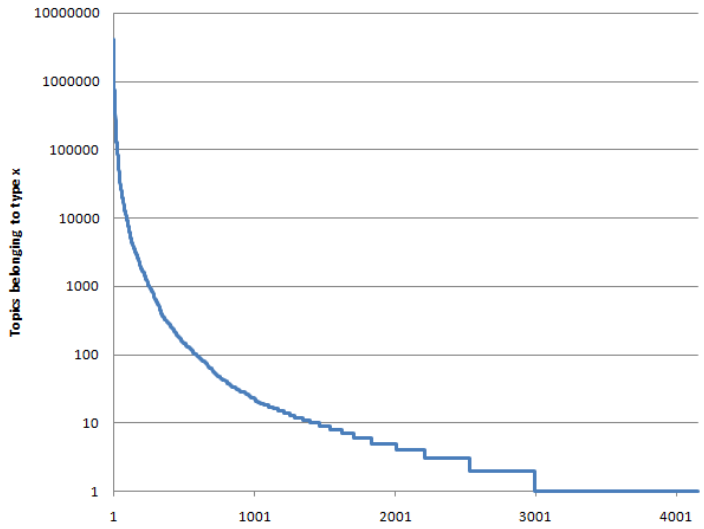


Fig. 15. Distribution of Freebase types: the most used one (/music/track) is associated to 4.1 million topics.

Type	Topics
/music/track	4110740
/common/topic	3911182
/common/document	2607367
/people/person	775876
/type/content_import	773153
/type/content	747299
/common/image	705154
/media_common/creative_work	511159
/music/release	505623
/location/location	500534
/music/album	464239
/music/artist	365760
/common/webpage	331361
/location/geocode	262923
/business/employer	235032

Table 1. The top 15 types in Freebase, ordered by number of topics.

7 Conclusions

Both personalization and customization are techniques which can bring value to a system, making the user feel like it is more "personal". Currently, however, while more and more systems allow users themselves to apply their own customizations (such as UI skins, security settings, or personal preferences), there are still few which allow users to directly access and use their personal information.

Looking at the huge success many customization tools had, not just on a particular set of expert users but also on the wide Internet public, we decided to make personalization customizable, that is to allow users access their own personal profile and build custom applications that exploit it.

To do this, we chose a specific use case: the information that users leave when they use their browsers, that is the history of the visited websites and the collection of their bookmarks. We then converted this information from the proprietary format used by Firefox 3 in an open, RDF-based format. We built tools to do the conversion offline or in realtime and a browser extension that uses these data to provide new, potentially interesting, information. The whole set of tools is built to be extensible, so it makes heavy use of configuration files that can be easily modified by users to provide additional features.

As one of our prototypes aims at enhancing the browsing experience by providing related information gathered from Freebase, our evaluation focused on Freebase links and types. Using the Nielsen database, we drew statistics about the incidence of Freebase URLs inside the top million visited websites and we found promising results: while the percentage of top Nielsen links that also appear in Freebase is still pretty low (0.77%), the percentage of *visits* that take to a link belonging to freebase is rather high (9.75%). We repeated the same calculations over 15 different groups of URLs, belonging to the different website categories provided by Nielsen, and were able to detect how the incidence is distributed across the groups. Finally, trying to detect among the Freebase topics which ones were more useful from our point of view, we ordered Freebase types by usage and analyzed the results: even if the most generic types appear in the most used ones, there are also many potentially interesting types in the top 100 that could be exploited to have useful related information.

One thing that is missing in this project, due to its time constraints, is an analysis of the potential security problems of such a system. We are aware of the fact that merging our personal data with external services is a potential risk, as whenever we get related information we are also providing ours to all the services we query. We are currently leaving the decision of sharing the profile to users themselves, but the problem is definitely worth our attention and will be studied with more detail.

A Tools and Technologies

A.1 RDF

(Resource Description Framework[19]) is a standard model for data interchange on the Web. RDF is built on the following definitions:

- a *Resource* is anything that can have a URI[20]. This includes, for instance, all the world's Web pages which are identified by a URL;
- a *PropertyType* is a resource that has a name and can be used as a property. For instance, speaking about a bookmark, Name and Date could be a couple of related properties. The fact that a property type is a resource too allows it to have its own properties: for instance, we can say that the range for the Date property we defined for bookmarks is a string that conforms to the XML DateTime datatype (so we know how to parse it), has a human-readable description that says it is the date the bookmark was saved, and has a label that reads like "Date" in English and "Data" in Italian.

Using these building blocks, RDF allows to express assertions in the form of *triples* (subject, predicate, object), where the both the subject and the predicate are resources while the object can be either a resource or a constant string, called *literal*. As an example, if we wanted to say that the page at URL `http://www.example.org/index.html` has the title "My Page" and its author is John Smith, identified by the URL `http://www.example.org/people/123456`, we could write:

```
<http://www.example.org/index.html>              (Subject)
  <http://purl.org/dc/elements/1.1/creator>      (Predicate)
  <http://www.example.org/people/123456> .       (Object)

<http://www.example.org/index.html>              (Subject)
  <http://www.example.org/terms/page-title>      (Predicate)
  "My Page" .                                    (Object)
```

Typed properties and the triple structure facilitate data merging even if the underlying schemas differ and support the evolution of schemas over time without requiring all the data consumers to be changed. This is one of the main reasons why we chose to describe our user profile in RDF: in fact, this format allows us to easily use the same profile across different applications, being able at the same time to extend it with new information.

A.2 Freebase

Freebase has been defined as "an open database of the worlds information". It has been built drawing information from large open data sources, such as Wikipedia and MusicBrainz[21], and reconciling it so that a particular topic appears only once in Freebase

[19] See `http://www.w3.org/RDF/`

[20] Uniform Resource Identifier. URLs, which are a subset of URIs, are probably the most common URIs used to refer to a resource. For more information, see `http://en.wikipedia.org/wiki/Uniform_Resource_Identifier`.

[21] `http://musicbrainz.org`

and contains all the information related to it. The result is a collection of structured data about many popular topics such as movies, music, people and locations, which can be easily edited online within a wiki-like interface or queried by anyone with the provided API. The underlying structure of Freebase allows users to run complex queries using a query language called MQL (Metaweb Query Language), which uses JSON (JavaScript Object Notation) syntax, making it ideal for JavaScript and Python based clients. The typical query is made *by example*, that is providing a template of the structure of the desired data, where all the information we do not know are simply left blank; the query response will simply fill the template with the missing information. An example, asking for George Lucas films starring Harrison Ford, is shown in Figure 16.

```
[                                       | [
  {                                     |   {
    "directed_by" : "George Lucas",     |     "directed_by" : "George Lucas",
    "name" : null,                      |     "name" : "Star Wars Episode IV: A New Hope",
    "starring" : [                      |     "starring" : [
      {                                 |       {
        "actor" : "Harrison Ford"       |         "actor" : "Harrison Ford"
      }                                 |       }
    ],                                  |     ],
    "type" : "/film/film"               |     "type" : "/film/film"
  }                                     |   },
]                                       |   ...
```

Fig. 16. An MQL query asking for all the films by George Lucas starring Harrison Ford (left) and the first of its results (right).

We decided to use Freebase to provide us information related to the browsed pages, for different reasons: first of all, being based on Wikipedia knowledge, it contains data which is not limited to a specific domain; then, it provides some structure and semantics that we decided to exploit for our needs; finally, it is a free source of information which will continuously grow thanks to the direct contributions of its own users or the indirect ones coming from the users of other open data communities.

A.3 Software

HSQLDB [22] is a relational database engine completely written in Java. It is multiplatform, small, and fast and supports both in memory and disk based tables. We chose this engine as it can easily be run in a standalone mode as part of an application program, in its same Java Virtual Machine.

SDB [23] is a component of Jena[24] which provides scalable storage and query of RDF datasets using conventional SQL database as a backend. It can be used in standalone applications and is designed specifically to support the SPARQL[25] query language.

[22] `http://hsqldb.org`
[23] `http://jena.sourceforge.net/SDB`
[24] `http://jena.sourceforge.net`
[25] `http://www.w3.org/TR/rdf-sparql-query`

Joseki [26] is an HTTP engine that supports the SPARQL Protocol and the SPARQL query language. It can be configured to work with SDB as a backend and provides a SPARQL endpoint to the RDF store. We decided to use Joseki as it provides a layer of abstraction on the ontology which allows programmers to access it just by using SPARQL; moreover, there are already many other applications that work with SPARQL endpoints, so this choice allows us to automatically make our data useful for other projects too.

Firefox [27] is a well-known, opensource browser, which with about 650 million downloads[28] is getting more and more used not only by computer-savvy people, but also by many common internet users. Being opensource, it has also become a basis for other projects such as Flock[29] (a "social Web browser", specialized in interfacing with many social applications) and Google Chrome[30]. We chose to build part of our tools as Firefox extensions, so they can easily access history and bookmarks in realtime and provide an integrated interface within an environment which is already well-known to the user.

[26] `http://www.joseki.org`
[27] `http://www.mozilla.com/firefox`
[28] Data gathered from `http://www.spreadfirefox.com` on September, 8th 2008.
[29] `http://www.flock.com`
[30] `http://www.google.com/chrome`

B Browser history and bookmarks ontology

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix HPLBrowseData: <http://hpl.hp.com/schemas/FFontology#> .
@prefix mozb: <http://hpl.hp.com/schemas/bookmarks#> .
@prefix mozh: <http://hpl.hp.com/schemas/history#> .

HPLBrowseData:Bookmark a rdfs:Class ;
rdfs:label "bookmark"@en ;
rdfs:comment "A bookmark (or a folder) saved inside Firefox bookmarks."@en .

HPLBrowseData:Visit a rdfs:Class ;
rdfs:label "visit"@en ;
rdfs:comment "A visit item in the browser history of Firefox."@en .

HPLBrowseData:BookmarkTypeClass a rdfs:Class ;
rdfs:label "bookmarkType"@en ;
rdfs:comment "Specifies the bookmark type."@en .
HPLBrowseData:BookmarkClass a rdfs:Class ;
rdfs:subClassOf HPLBrowseData:BookmarkTypeClass ;
rdfs:label "bookmark"@en ;
rdfs:comment "Specifies this bookmark element's type is 'bookmark'."@en .
HPLBrowseData:BookmarkSeparatorClass a rdfs:Class ;
rdfs:subClassOf HPLBrowseData:BookmarkTypeClass ;
rdfs:label "separator"@en ;
rdfs:comment "Specifies this bookmark element's type is 'separator'."@en .
HPLBrowseData:BookmarkFolderClass a rdfs:Class ;
rdfs:subClassOf HPLBrowseData:BookmarkTypeClass ;
rdfs:label "folder"@en ;
rdfs:comment "Specifies this bookmark element's type is 'folder'."@en .

HPLBrowseData:URL a rdf:Property;
rdfs:label "url"@en ;
rdfs:comment "The URL for the resource. This will be the URL that is opened
                      when a user double-clicks on the entry in a tree.
                      (see <http://home.netscape.com/NC-rdf#URL>)"@en ;
rdfs:domain HPLBrowseData:Bookmark ;
rdfs:domain HPLBrowseData:Visit ;
rdfs:range rdfs:Resource .

HPLBrowseData:Date a rdf:Property;
        rdfs:label "date"@en ;
        rdfs:comment "Holds the date when this object was added or updated."@en ;
        rdfs:domain HPLBrowseData:Bookmark ;
        rdfs:domain HPLBrowseData:Visit ;
        rdfs:range xsd:dateTime.
```

```
HPLBrowseData:Duration a rdf:Property;
        rdfs:label "duration"@en ;
        rdfs:comment "The time in ms the user spent on the page.
                      (see <http://home.netscape.com/NC-rdf#URL>)"@en ;
        rdfs:domain HPLBrowseData:Visit ;
        rdfs:range xsd:integer.

HPLBrowseData:Referrer a rdf:Property;
        rdfs:label "referrer"@en ;
        rdfs:comment "The referrer of the URL, which is, in general, the URL of the
                      page which contained the link to this one.
                      (see <http://home.netscape.com/NC-rdf#Referrer>)"@en ;
        rdfs:domain HPLBrowseData:Visit ;
        rdfs:range rdfs:Resource .

HPLBrowseData:Name a rdf:Property;
        rdfs:label "name"@en ;
        rdfs:comment "The name of the resource. This will be the title of the page.
                      (see <http://home.netscape.com/NC-rdf#Name>)"@en ;
        rdfs:domain HPLBrowseData:Visit ;
        rdfs:range xsd:string .

HPLBrowseData:Session a rdf:Property;
        rdfs:label "session"@en ;
        rdfs:comment "A Firefox browsing session."@en ;
        rdfs:domain HPLBrowseData:Visit ;
        rdfs:range xsd:integer .

HPLBrowseData:BookmarkName a rdf:Property;
        rdfs:label "bookmarkName"@en ;
        rdfs:comment "The name of a bookmark (it defaults to the page title, but
                      might be a custom string)."@en ;
        rdfs:domain HPLBrowseData:Bookmark ;
        rdfs:range xsd:string .

HPLBrowseData:BookmarkType a rdf:Property;
        rdfs:label "bookmarkType"@en ;
        rdfs:comment "The bookmark type. It can have three values:
                      HPLBrowseData:BookmarkClass, HPLBrowseData:BookmarkSeparatorClass
                      and HPLBrowseData:BookmarkFolderClass."@en ;
        rdfs:domain HPLBrowseData:Bookmark ;
        rdfs:range HPLBrowseData:BookmarkTypeClass .

HPLBrowseData:BookmarkFolder a rdf:Property;
        rdfs:label "bookmarkFolder"@en ;
        rdfs:comment "The bookmark folder: matches the ID of the parent folder or 0 if
                      it comes from the root."@en ;
        rdfs:domain HPLBrowseData:Bookmark ;
        rdfs:range HPLBrowseData:Bookmark .
```

C Example configuration files

C.1 Configuration schema for Places2RDF

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix pmap: <http://hpl.hp.com/schemas/pmap#> .

pmap:Map a rdfs:Class ;
rdfs:comment "PMAP map"@en .

pmap:Query a rdfs:Class ;
rdfs:comment "Provides both SQL query and fields-properties mapping"@en .

pmap:Mapping a rdfs:Class ;
rdfs:comment "Mapping between database fields and RDF properties"@en .

pmap:PurifyList a rdfs:Class ;
rdfs:comment "List of regular expressions used to purify URIs"@en .

pmap:regexp a rdf:Property;
rdfs:comment "Regular expression used to purify URIs"@en ;
rdfs:domain pmap:PurifyList ;
        rdfs:range xsd:string .

pmap:dbpath a rdf:Property;
rdfs:comment "Database path"@en ;
rdfs:domain pmap:Map ;
        rdfs:range xsd:string .

pmap:query a rdf:Property;
rdfs:comment "Specifies query"@en ;
rdfs:domain pmap:Map ;
        rdfs:range pmap:Query .

pmap:output a rdf:Property;
rdfs:comment "Specifies the output format"@en ;
rdfs:domain pmap:Map ;
        rdfs:range xsd:string .

pmap:queryString a rdf:Property;
rdfs:comment "Actual SQL query string"@en ;
rdfs:domain pmap:Query ;
        rdfs:range xsd:string .

pmap:subject rdf:Property;
rdfs:comment "Specifies which field in the query results will become the
                    subject of the triples"@en ;
rdfs:domain pmap:Query ;
        rdfs:range xsd:string .
```

```
pmap:mapping a rdf:Property;
rdfs:comment "Each specifies a mapping between one field and one
                    RDF property"@en ;
rdfs:domain pmap:Query ;
        rdfs:range pmap:Mapping .

pmap:field a rdf:Property;
rdfs:comment "Specifies the field in one mapping"@en ;
rdfs:domain pmap:Mapping ;
        rdfs:range xsd:string .

pmap:property a rdf:Property;
rdfs:comment "Specifies the property in one mapping"@en ;
rdfs:domain pmap:Mapping ;
        rdfs:range rdf:Property .

pmap:format a rdf:Property;
rdfs:comment "Specifies the object format in one mapping"@en ;
rdfs:domain pmap:Mapping ;
        rdfs:range xsd:string .

pmap:prefix a rdf:Property;
rdfs:comment "Specifies a prefix to paste before the object value"@en ;
rdfs:domain pmap:Mapping ;
        rdfs:range xsd:string .
```

C.2 Example Places2RDF configuration for visits

```
@prefix pmap: <http://hpl.hp.com/schemas/pmap#> .
@prefix HPLBrowseData: <http://hpl.hp.com/schemas/FFontology#> .
@prefix mozb: <http://hpl.hp.com/schemas/bookmarks#> .
@prefix mozh: <http://hpl.hp.com/schemas/history#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<> a pmap:Map ;

 # dbpath is the path where the places.sqlite DB is found
 pmap:dbpath "C:/Users/.../places.sqlite";

 pmap:purify [ a pmap:PurifyList;
               pmap:regexp        "^https.*" ;
               pmap:regexp        ".*?#.*?#.*" ;
             ];

 # output allows to choose the output format for the triples
 pmap:output "N3" ;  # can be "RDF/XML", "RDF/XML-ABBREV", "N-TRIPLE", "TURTLE", and "N3"


 # every query has a queryString and a sequence of field-property mappings
```

```
pmap:query  [ a pmap:Query ;
              pmap:queryString "select mh.id, mh.from_visit, mh.visit_date, mh.visit_type,
                                mh.session, mp.url, mp.title from moz_historyvisits as mh,
                                moz_places as mp where mh.place_id=mp.id;";

              pmap:subject "mozh:id" ;

              pmap:mapping [ a pmap:Mapping ;
                             pmap:field    "from_visit" ;
                             pmap:property HPLBrowseData:Referrer ;
                             pmap:format   "URI" ;
                             pmap:prefix   "mozh" ;  # in this case we use a prefix to
                                                     # convert a literal value to a URI
                           ];

              pmap:mapping [ a pmap:Mapping ;
                             pmap:field    "visit_date" ;
                             pmap:property HPLBrowseData:Date ;
                             pmap:format   "DateTime" ;
                           ];

              pmap:mapping [ a pmap:Mapping ;
                             pmap:field    "visit_type" ;
                             pmap:property HPLBrowseData:visitType ;
                           ];

              pmap:mapping [ a pmap:Mapping ;
                             pmap:field    "session" ;
                             pmap:property HPLBrowseData:Session ;
                           ];

              pmap:mapping [ a pmap:Mapping ;
                             pmap:field    "url" ;
                             pmap:property HPLBrowseData:URL ;
                             pmap:format   "URI" ;
                           ];

              pmap:mapping [ a pmap:Mapping ;
                             pmap:field    "title" ;
                             pmap:property HPLBrowseData:Name ;
                           ];
.
```

D Example data file for TouchGraph

```
google.visualization.Query.setResponse(
{
  requestId:'0',
  status:'ok',
  signature:'7439076922257713675',
  table:{
    cols: [ {id:'A',label:'URL',type:'t',pattern:''},
            {id:'B',label:'Date',type:'d',pattern:'yyyy-MM-dd HH:mm:ss'},
            {id:'C',label:'Visits',type:'n',pattern:'#0.################'},
            {id:'D',label:'Delicious Popularity',type:'n',pattern:'#0.################'},
            {id:'E',label:'Delicious PostPerDate',type:'n',pattern:'#0.################'},
            {id:'F',label:'Delicious Tag',type:'t',pattern:''}
          ],
    rows: [
            [{v:'http://www.google.com/'}, {v:new Date(2008,6,28,22,58,6)},
             {v:1.0}, {v:25798.0}, {v:0.0}, {v:'searchengine'}],
            [{v:'http://airwiki.elet.polimi.it/mediawiki/index.php/Writing_a_Firefox_extension'},
             {v:new Date(2008,6,28,23,1,4)}, {v:1.0}, {v:0.0}, {v:0.0}, {v:'untagged'}],
            [{v:'http://www.last.fm/'}, {v:new Date(2008,6,29,3,2,31)}, {v:1.0}, {v:28031.0},
             {v:0.0}, {v:'music'}],
            ...
          ]
  }
}
);
```

References

1. Jean Lave and Etienne Wenger. *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press, New York, 1991.
2. *International Workshop on Semantic Web Personalization, Budva, Montenegro*, 06 2006.
3. Magdalini Eirinaki, Dimitrios Mavroeidis, George Tsatsaronis, and Michalis Vazirgiannis. Introducing semantics in web personalization: The role of ontologies. In Markus Ackermann, Bettina Berendt, Marko Grobelnik, Andreas Hotho, Dunja Mladenic, Giovanni Semeraro, Myra Spiliopoulou, Gerd Stumme, Vojtech Svtek, and Maarten van Someren, editors, *EWMF/KDO*, volume 4289 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2005.
4. Christina Tziviskou and Marco Brambilla. Semantic personalization of web portal contents. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *WWW*, pages 1245–1246. ACM, 2007.
5. Peter Haase, Marc Ehrig, Andreas Hotho, and Bjrn Schnizler. Personalized information access in a bibliographic peer-to-peer system. In Steffen Staab and Heiner Stuckenschmidt, editors, *Peer-to-Peer and SemanticWeb, Decentralized Management and Exchange of Knowledge and Information*, pages 143–158. Springer, 2006.
6. Michael Noll and Christoph Meinel. Web search personalization via social bookmarking and tagging. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon J B Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Guus Schreiber, and Philippe Cudr-Mauroux, editors, *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC2007), Busan, South Korea*, volume 4825 of *LNCS*, pages 365–378, Berlin, Heidelberg, November 2007. Springer Verlag.
7. Uldis Bojars, Alexandre Passant, John G. Breslin, and Stefan Decker. Social network and data portability using semantic web technologies. In *2nd Workshop on Social Aspects of the Web (SAW 2008) at BIS2008*, pages 5–19, 2008.
8. Anupriya Ankolekar, Markus Krtzsch, Thanh Tran, and Denny Vrandecic. The two cultures: mashing up web 2.0 and the semantic web. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 825–834, New York, NY, USA, 2007. ACM Press.
9. Anupriya Ankolekar and Denny Vrandecic. Kalpana - enabling client-side web personalization. In Peter Brusilovsky and Hugh C. Davis, editors, *Hypertext*, pages 21–26. ACM, 2008.
10. Niall Kennedy. Sniff browser history for improved user experience, 2008. `http://www.niallkennedy.com/blog/2008/02/browser-history-sniff.html`; accessed 08-Sep-2008.
11. Davide Eynard. Powerbrowsing, 2005. `http://davide.eynard.it/malawiki/PowerBrowsingEn`. Accessed 10-Sep-2008.
12. Davide Eynard, John Recker, and Craig Sayers. An imap plugin for squirrelrdf. Technical report, HP Labs, 2007.
13. Davide Eynard and Marco Colombetti. Exploiting user gratification for collaborative semantic annotation. In *Semantic Web User Interaction at CHI 2008: Exploring HCI Challenges*, 2008.