The Semantic Web as a Semantic Soup

Simon Cox, Harith Alani, Hugh Glaser, Steve Harris
Electronics & Computer Science
University of Southampton, UK
simon.cox@gmail.com, {ha,hg,swh}@ecs.soton.ac.uk

Abstract

The Semantic Web is currently best known for adding metadata to web pages to allow computers to 'understand' what they contain. This idea has been applied to people by the Friend of a Friend project which builds up a network of who people know through their descriptions placed on web pages in RDF. It is here proposed to use RDF to describe a person and to have their RDF document follow them around the Internet. The proposed technique, dubbed Semantic Cookies, will be implemented by storing a user's RDF in a cookie on their own computer through the browser. This paper considers the concept of Semantic Cookies and investigates how far existing technology can be pushed to accommodate the idea.

1 Introduction

The current effort behind the Semantic Web (SW) is to add computer-readable information to web documents. However, the SW is not limited to describing documents. The Friend of a Friend (FoaF) project[1] is an example of describing personal information using the semantic markup language RDF[2]. It aims to provide a network of information about people and the way they are related to each other. RDF is a computer readable format in that a software agent can read in an RDF document and also make inferences based on the information contained in the document and in other sources of RDF referenced.

Almost all web sites that offer some interactive service store information about the people who use them. This data must be gathered during a registration process that can become repetitive for a person having previously registered for other sites. It is also difficult to keep the data up to date when person's circumstances change. Once a person is registered with a website cookies are used to match a computer with a user.

This report proposes the marrying of the Semantic Web with cookie technology to provide Semantic Cookies (SC). An example system has been created as a proof of concept, exploring how far existing technology can be pushed to provide this new technique.

The report will begin with a discussion of the motivations for semantic cookies and a brief background of similar technologies. The FoaF project will be described, showing how RDF can be used to describe people. Current cookie technology will be explained also noting limitations. Other existing methods of providing more enhanced user data will also be covered. Next, the proposed SCs will be described, stating what they will provide with advantages and limitations of using cookies. Some example applications will be mentioned, showing the varied scope for SCs. The report will finish with the observations and conclusions drawn from the implementation including what would be required in future technology to improve the viability of SCs.

2 Background

2.1 Friend of a Friend (FoaF)

The FoaF project is an application of semantic web technologies aiming to describe people and the links between them. Fundamentally, a person has an RDF fragment describing themselves which is stored in XML format and hosted somewhere on the internet where other RDF documents may use its URI to reference the person. Listing 1 shows a very

basic FoaF document saying that the person Simon Cox has a mailbox with a URI of mailto:samc100@soton.ac.uk.

```
<rdf:RDF
  xmlns:rdf=
    "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf=
    "http://xmlns.com/foaf/0.1/»

 <foaf:Person>
   <foaf:name>Simon Cox</foaf:name>
   <foaf:mbox
     rdf:resource=
       "mailto:samc100@soton.ac.uk" />
 </foaf:Person>

</rdf:RDF>
```

Listing 1: FoaF example in RDF

Listing 2 shows a fragment using the FoaF ontology saying that the person Hugh Glaser knows the person Simon Cox, and that they have a homepage as well. This example highlights the aim of RDF, as an application could merge the fragment with the one above, perhaps to show the homepages of all the people who know Simon Cox. [caption=FoaF knows example, label=lst:foaf-knows]

```
 <foaf:Person>

   <foaf:name>Hugh Glaser</foaf:name>
   <foaf:mbox
     rdf:resource="mailto:hg@ecs.soton.ac.uk"/>
   <foaf:homepage
         rdf:resource="http://ecs.soton.ac.uk/ hg/"/>

   <foaf:knows>
     <foaf:Person>
       <foaf:name>Simon Cox</foaf:name>
                   <foaf:mbox
                     rdf:resource=
                       "mailto:samc100@soton.ac.uk" />
     </foaf:Person>
   </foaf:knows>

 </foaf:Person>
```

Listing 2: FoaF knows example

FoaF was designed for decentralised networks where the users manage their own information. This is both a benefit and a disadvantage for general public use; it is necessary for a person to be in control of their own information, however, a user with little technical experience would find it extremely difficult to manage their information as RDF is not a trivial format and requires an in depth understanding to use well. It would be desirable for some other party to actually manage the information. Therefore a user with little technical expertise would not be able, or would find it extremely difficult to update

their own information.

2.2 Plain Cookies

A cookie is a small item of data placed on a client's computer during an HTTP interaction with a server as a result of a response [3].
Cookies were initially intended to create stateful sessions over several HTTP requests/responses. In design of the cookie mechanism, it was assumed that sessions would be relatively short lived and would only involve the client and a specific website. The specification for cookies states that user agents implementing cookie support should provide at least 4096 bytes per cookie, although session-based servers using cookies should strive to keep cookie size as small as possible. Due to this it is not possible to assume a user agent will be able to handle a cookie larger than 4 KB.
The most common use for cookies is store a session identifier for which a server can use to maintain a session with a client over a short period of time. However, cookies are also used for storing other information with a longer lifespan such as user names and site specific preferences.
As part of the intended design, cookies can only be accessed and manipulated by the server that created them. There are several reasons for this. Firstly, each cookie contains a URL specified at the time of creation and for each HTTP request that matches the URL the cookie is sent along with the request. As the vast majority of servers make use of cookies then there would be a huge amount of extra data sent over the Internet if all cookies were to be sent with all requests. Secondly, many sites use cookies to reduce the amount of information a user need provide to a server and if made publicly available could pose a security risk. For example, Amazon has a mechanism called '1-Click ordering'[4] where a user can store their payment details and need only be logged in (i.e. have a cookie with Amazon's session ID) to be able to click one button which purchases the item currently being displayed.
The data in cookies is held in arbitrary name value pairs with no semantic meaning. There may be a cookie with a name of `surname', but the value could mean anything, not necessarily the family name of the person using the computer. Therefore the values in cookies only bear meaning within the context of the domain that issued it. A standard naming convention could be applied so that all cookies with the same name had the same semantic meaning, but this would be completely impractical due to the nature of the Internet.

2.3 XML Cookies

Microsoft have introduced a form of XML cookies into their Internet Explorer browser under DHTML behaviors[5]. They are effectively a form of client-side only cookies, i.e. they can only be accessed through client-side scripts running in the browser. They are used for storing the state of a form and any other user data in an XML format. There are similar restraints to cookies, in that size is limited to per document (128KB) and per domain (1024KB).

2.4 Wallets

The term wallet, in the context of browser technology, broadly refers to a software agent that manages personal data on behalf of a user. A wallet can either be server-side or client-side. With a server-side wallet, all of the user's personal information is kept on a central server that the user can direct a website to if it requires information from the user. A client-side wallet operates on the user's own machine and interacts directly without the need for a third party server.
Generally, server-side wallets are used for financial transactions, storing a shipping/billing address and either the user's credit card information, or some kind of electronic money. Two examples of server-side wallets are the MSN Wallet[6] and PayPal[7]. The main advantage of a server-side wallet is that there is no need for any software to be installed on a client machine, i.e. only a browser is needed to be able to use the service. However, the user is required to tell any website that requires the information about the location of the wallet service. Also, there is usually some charge, either to the user or the website for using the service.
There is much more scope for the operation of client-side wallets. Current trends show them to be evolving into general stores of personal information for filling in HTML forms. For example, the browsers such as Mozilla have a wallet built in

[8] that operates by storing the user's information such as forename, surname or e-mail address, and attempts to fill in form fields that appear to relate to the information based on the name/id of the form element. For example, a form element named *pcode* would be filled with the user's post code. This a passive form of wallet requiring interaction from the user to make sure forms are filled in correctly. The advantage being that a server need not even know a wallet is being used.
Other more active client side wallets would interact directly with a server to provide information. These would require more trust from a user to be sure that any information given out was only given to authorised websites, and that the information could not be tampered with.
Disadvantages of both forms of client-side wallets are that they require software to be installed on the user's machine. However, it is this aspect that delivers the advantages therefore unavoidable.
Having a software agent managing a user's information, whether server-side or client-side, provides many advantages; mainly:

- access can be restricted to authorised sources,
- integrity of data can be maintained.

3 Semantic Cookies

The motivations for semantic cookies are the summation of the disadvantages of the different methods described above. Semantic cookies should provide a means of delivering personal information transparently to a website without the need for user interaction. However, the information held should be fully controllable by the user owning it.
Semantic cookies will provide:

- a means to update the personal information by a separate server that the user interacts with
- full control over the data if the user needs it
- semantically rich computer readable information
- no need to install extra software other than an Internet browser

The proposed solution is to store RDF fragments in public cookies. Public cookies here define a form of cookie that is sent to any server the client interacts with regardless of their domain. As mentioned in Section 2.2, there is no support in current browsers for public cookies; however, a viable temporary workaround is proposed in Section 4.
The basic mechanism will be that as a user browses a website, their RDF fragments will be sent to the web server. The web server can then process the user's personal details extracting as much information as needed and adapting the service it provides in response. The web server is also able to send new or changed RDF fragments back to the client to update their information. In this way a user need not even realise they have any RDF fragments relating to them. Since the data is stored on their machine they also have full control over the contents if they wish to modify it.
This model differs from the traditional paradigm where the whole Semantic Web is always accessible by following relationships throughout. In effect, a user's RDF fragment will be a part of the Semantic Web only as long as their HTTP session with the web server. This is a requirement to keep the user's information private as would be expected as the default behavior of the system. If a user required their information to be a permanent part of the Semantic Web, then they could visit a website which offers a persistence service. This web site would receive the user's semantic cookie and would provide a permanent URI that pointed to a copy of the RDF.
Storing a user's RDF on a client machine creates an issue if the user uses more than one machine. However, this allows the possibility of having a different semantic context for the user depending on the machine. Obviously certain parts of their description will remain the same, such as name and date of birth, but other parts may be dependent on the context, contact email for example, if given at all might differ between a person's home computer and their work computer. Using a persistence service, a user could swap in/out RDF fragments to the machine they are at and create a different semantic context for themselves.

4 Implementation

The aim of this research is to bring people into the semantic web without their realisation, or with as little interaction required. This means utilising existing Internet technology installed on home computers, therefore the implementation described as follows will not require *any* additional software to be installed on a user's computer.
The problem of a 3rd party website interacting with a user's cookie can be broken down into four separate cases:

- Retrieve RDF fragments from a user's semantic cookie
- Provide means to query/manipulate RDF
- Create RDF fragments from updated source
- Update/Create user's semantic cookie from updated/new RDF

4.1 Technology Used

The implementation created for this report uses the following architecture and any examples will be in the context of this architecture.

- *3rd party website* - website that will make use of a user's RDF. Uses JSP scripts on an Apache Tomcat server so that the Java API to the 3store can be used.
- *RDF Authority* - necessary for the mechanism used to retrieve a user's cookie. Uses PHP scripts on an Apache web server.
- *RDF Storage* - to manage storing, manipulating and querying of RDF. The 3store[9] server is used, requires MySQL to handle underlying data and the Redland Raptor library[10] to parse RDF.

4.2 Cookie Interface

As described above in Section 2.2 cookies are restricted to the domain a server is in. However, this limitation can be overcome by using a single server to manage cookies acting as an intermediary between the user and 3rd party web server. Fig. 1 shows the process for a 3rd party website to retrieve a cookie from a visiting user. The website can gain access to the cookie by including a JavaScript source file from the RDF Authority web server that receives the cookie and embeds it into the JavaScript source as a variable. The client-side redirect is not necessary, but provides a cleaner experience for the user. Once the website receives the RDF in the post header it can be used as desired.

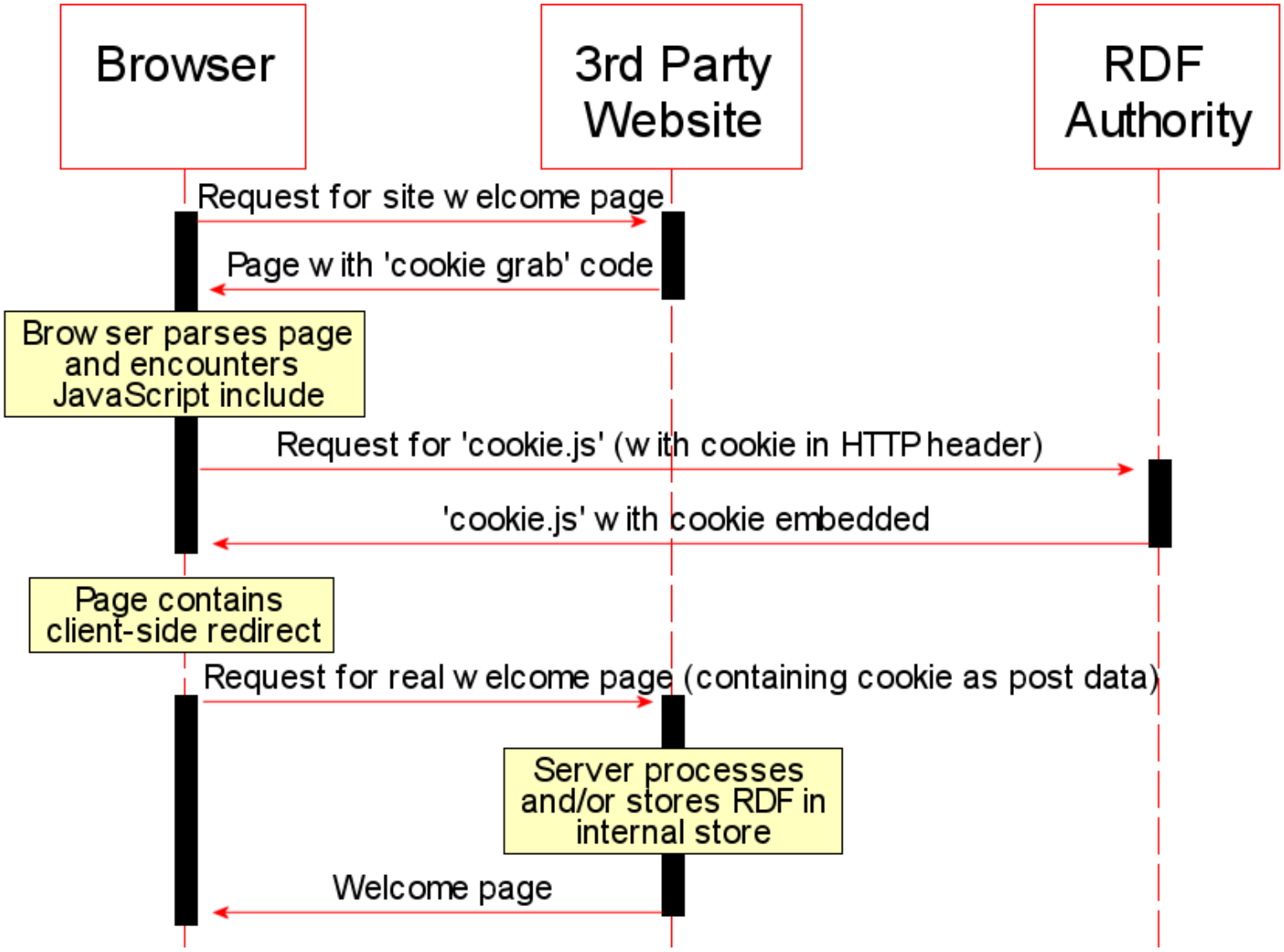

Figure 1: Retrieving RDF from a user's cookie

The RDF authority must be used again when the website wants to either create a new semantic cookie, or update an existing one. This is a similar process to receiving the cookie and is shown in Fig. 2. As the response for a request from the user's client, the website returns the RDF as part of a hidden form field. The action for the form is to post to the RDF authority which responds with the RDF in a cookie header along with a redirect to a specified page on the 3rd party website. Again, the form page contains automatic form submission code to reduce the number of clicks for the user.

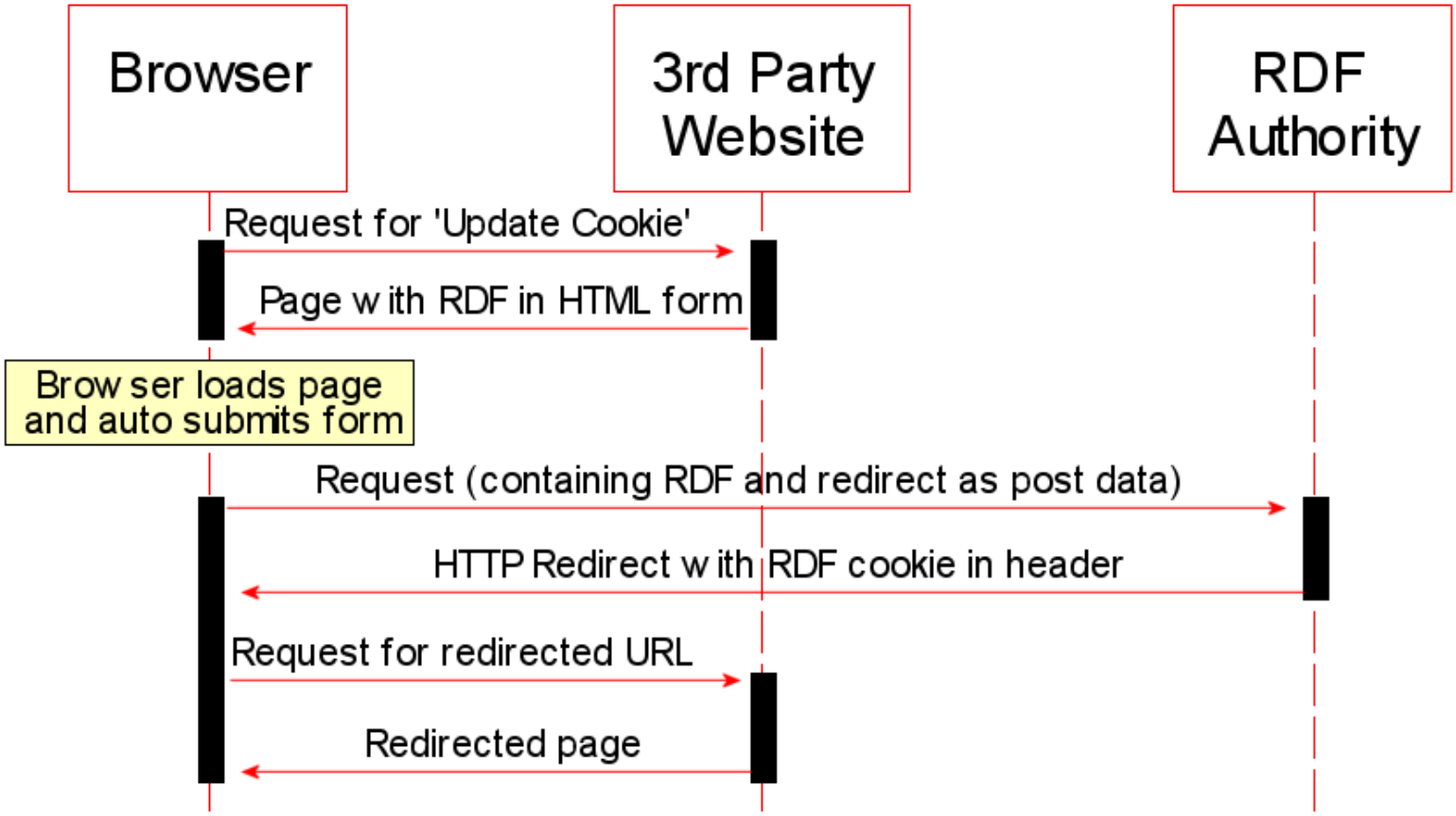

Figure 2: Updating RDF in a user's cookie

4.3 Querying/Updating User RDF

The RDF fragments need to be parsed into a format that can be queried and/or updated in order for the website to use them. For this implementation the 3store system was chosen to handle RDF, although any suitable RDF store could be used. The 3store is a database of triples built on top of MySQL and uses Raptor to parse the RDF. It can be queried in different ways, either using RDQL[11] or in a more programmatic way with the API provided. RDQL is a proposed W3C standard and has a similar syntax to SQL allowing for complex queries. Either way the 3store is queried, a set of triples is returned that match the criteria.
To update user information held in the 3store new triples can be asserted into the database and old triples can be removed.

4.4 Creating RDF Fragments

If a user's information has been changed in the 3store and the user wants to update their cookie with it then the relevant triples must be exported from the database to create the fragments that will be placed in the user's cookie. Unfortunately, the problem of deciding which triples are relevant to a user is not trivial.
RDF triples form a graph structure with either a URI or a literal as nodes and predicates as edges. By picking out a specific node to be root the structure can be traversed outwards to create an RDF document. To distinguish between users in the implementation it is assumed that all semantic cookies will contain a <foaf:mbox_sha1sum> predicate. This is the SHA1 hash of their e-mail address. The root node for a user can be found by using the RDQL in Listing 3.

```
SELECT ?person
WHERE (?person, <foaf:mbox_sha1sum>, ßha1 hash")
```

Listing 3: Selecting the root node

Naively, the rest of the RDF could be constructed using the query in Listing 4 recursively calling it for every object that is a URI.

```
SELECT ?predicate, ?object
WHERE ("current URI", ?predicate, ?object)
```

Listing 4: Recursive query

However, this approach could result in a very large number of triples being returned from a 3store with many relating triples.
The implementation avoids the problem by relying on the fact that the 3store assigns a model identifier to triples asserted at the same time. As long as each user is associated with a unique model then only triples asserted for that user will be retrieved in queries, thus limiting the scope. However, this is very much implementation specific and does not solve the problem for the general case.
One way of avoiding the problem would be to restrict the queries to specific ontologies. In this case only the predicates described in the ontologies would be followed. The drawback to this approach is that if a website does not know about an ontology used in a user's RDF, then when the cookie gets replaced information under the unknown ontology would be lost.

5 Usage

There are many possibilities of how semantic cookies could be used, although there are several models that most would fit into. These models are not mutually exclusive, for example a persistence store would probably offer some form of RDF editor.

5.1 3rd Party Website

The 3rd party website actually processes the information in the cookie and offers some kind of service to the user based on the content of the cookie. For example, the Audioscrobler service[12] collects user's listening habits by recording the songs they listen to. One service provided is to suggest which songs or artists a user may like which can be delivered as an RDF feed. When the user visits the Audioscrobbler website an option would be offered to `remember' their suggested artists. When the user then visits a website selling music, the Audioscrobbler triple would be discovered in their RDF which would be used to retrieve the list of suggested artists. By comparing this list to their database, music the user may be interested in could be suggested for purchase.
All of the above could be accomplished without using semantic cookies, however, they provide the means for it to happen without any user intervention, in fact the user need not even know they have any RDF associated with them.

5.2 Persistence Store

A persistence service would allow a user to visit their website and `deposit' their RDF for long term storage. The service may also allow the user's RDF to be made publicly viewable and in which case provide a URI for it. This would allow the user's personal information, or a subset of it, to be a permanent part of the semantic web.
The user may also choose to publish their RDF if it grew to a large size. Currently cookies have a maximum limit of 4KB, although this could change if browser developers bring in support for semantic cookies. However, for user's with a slow Internet connection it would be faster to simply send a URI to a permanent RDF store than to transmit the entirety of their RDF fragments.
The persistence store could also be used to hold different RDF contexts for a person. These would all contain the same subset of information such as the person's name and date of birth, but would allow for differing context specific information. For example, a person could have a `home' and `work' context. The home context would have (amongst other

things) their personal email address, a delivery address as their home and perhaps their music listening tastes. The work context would have work email address, the delivery address would be their place of work, and perhaps a link to their position in a company organisation chart. This could be performed by simply having the two different contexts on the user's home and work computer, however, a laptop or PDA may be used at home and at work and would need to be able to change context depending on how it was being used. To accomplish this the user would visit the persistence store, save their current context (probably not making it publicly visible) and select one of their other registered contexts.

5.3 RDF Editor

The user's RDF is stored locally and can be edited using a text editor or some other RDF tool. However, the majority of Internet user's would not have the ability, or inclination, to edit RDF themselves. A useful service would be for a website to provide a simple means to view and edit the information in a user's semantic cookie. The user would visit the website where they would be able to browse the contents of their RDF fragments and edit any information that had changed or add new information. The website could offer a wizard to update or add common ontologies to the user's RDF.

6 Observations & Conclusion

A working system has been implemented where a user can visit a website which can gain access to a cookie containing an RDF document. The RDF retrieved from the user can be inserted into a 3store which can be queried and updated. A new RDF document can be extracted from the 3store and used to replace a user's existing cookie.
The process of implementing a semantic cookie solution has highlighted many issues that would need to be addressed by a full prototype system.

6.1 Security

Security is very important when dealing with personal data and access to a user's semantic cookie should be controlled. There are no actual security procedures set in place in the current implementation as it is a proof of concept and the infrastructure is capable of providing access control.
With the current method of retrieving and updating user cookies the RDF authority would handle access control as it is the central point in the system. The user would specify in their RDF a security policy which stated the access rights to the RDF for different domains. Any access to the cookie must come via a request from the client to the RDF authority web server, in which case the RDF authority will always receive the user's RDF fragments before a 3rd party has had a chance to change it. The request from the client would always contain the address of the requesting website to allow the RDF authority to make access decisions.
Public key cryptography would be used as part of the cookie delivery mechanism to avoid eavesdropping and spoofing. When a user's RDF is requested the RDF authority would encrypt it using the website's public key. This stops spoofing as the website would be unable to decrypt the RDF. When attempting to update a user's cookie the website would encrypt the RDF with the public key of the RDF authority and sign it with their private key. This stops eavesdroppers and allows the RDF authority to confirm the RDF update request came from the specified website.

6.2 Cookie Size

As mentioned above in Section 5.2 for large RDF fragments a persistence store could be used and the cookie would just hold the URI for the store. A simpler intermediate approach would be to compress the RDF that is stored in the cookie. The easiest place to provide for this would be in the RDF authority as it reduces the complexity required for a 3rd party website to be able to use semantic cookies.

6.3 Understanding Ontologies

The RDF specification is just a syntax that allows items to be linked with predicates. For predicates to have a semantic meaning an ontology must be used. However, due to the nature of the Internet there are many different ontologies that describe the same things. It is impractical to expect all users of the Internet to use the same ontologies and equally impractical to expect all websites to be able to understand all of the different ontologies. Ontology mapping overcomes this problem by providing the notion of mapping items of one ontology onto another to show that they are semantically the same. For example, the two elements from different ontologies <O1:FirstName> and <O2:Forename> describe the same thing, a person's given name. When a user who uses the ontology O1 to describe themselves, visits a website that only understands the ontology O2, the website would need to use some form of ontology mapping lookup service to find a mapping between O1 and O2 so that they could make use of the user's personal data.

6.4 Other Issues

The benefit of RDF being machine readable also stands as a disadvantage if is not well formed. As the RDF is accessible by the user and open to modification it is possible that it may be left in an illegal state which would render it useless. It would be advisable to discourage inexperienced users from attempting to modify their semantic cookie, and instead use an online RDF editor as described in Section 5.3. This would ensure that the RDF is always well formed, assuming the editor service behaved correctly. This also gives rise to another possible use of an RDF editor to attempt to fix an invalid semantic cookie.

6.5 Proposed System

The system implemented proves that semantic cookies are possible with current technology, however the solution is not scalable due to the central RDF authority required to handle access to the cookie which would become a bottleneck. This could be overcome by allowing the user to choose a different RDF authority. However, this would then require them to inform websites of the URL for the authority, breaking one of the aims of semantic cookies to not require user intervention. The ideal solution would involve client-side support for either semantic cookies or at least public cookies. The HTTP protocol would be a good place for this architecture with the browser natively supporting semantic cookies. A possible implementation would add several new HTTP headers:

- *Has-Semantic-Cookie* - sent by the client with all requests. The presence of this header would indicate that the user has a semantic cookie. This is not strictly needed, but it removes the need to transmit the entire semantic cookie for each request which would be an unnecessarily large strain on Internet resources.
- *Semantic-Cookie*
  - if sent by the server in a response indicates that the server wishes to be sent the semantic cookie. Upon receiving the response the user's browser may respond with an HTTP request including the *Semantic-Cookie* header.
  - if sent by the client in a request, the header value would contain the user's RDF fragments which are visible to the server as specified in the user's RDF access policy.
- *Set-Semantic-Cookie* - sent by the server in a response. The header value would contain new RDF fragments to be stored in the user's semantic cookie. The user's browser would choose how to handle the new fragments depending on the current access policy.

The access policy would be embedded in the user's RDF as described above in Section 6.1.
It may be possible to create browser plugins to handle the extended HTTP headers above. Although this would require the user to install extra software it would be far simpler than to get browser manufacturers to actually change their own code to add support. The Mozilla Firefox browser offers good support for extending the browser through `extensions'[13], although its not clear whether user code can be inserted into the HTTP protocol handling section. This would be a good place to investigate a practical implementation for browser support of semantic cookies.

Acknowledgment

The work reported in this paper was supported in part by the Advanced Knowledge Technologies (AKT) Project (www.aktors.org), EPSRC Grant Number GR/N15764/01.

References

[1]
"The foaf project," RDFWeb.org. [Online]. Available: http://www.foaf-project.org/

[2]
E. Miller, "Rdf specification," 2000. [Online]. Available: http://www.w3.org/RDF/

[3]
D. Kristol, "Rfc 2109, http state management mechanism," Bell Laboratories, Lucent Technologies, February 1997. [Online]. Available: http://rfc.net/rfc2109.html

[4]
"Ordering via 1-click," amazon.com. [Online]. Available: http://www.amazon.com/exec/obidos/tg/browse/-/468480/ref=br_lr__2/104-1067665-4483110

[5]
"Introduction to dhtml behaviours," Microsoft. [Online]. Available: http://msdn.microsoft.com/library/default.asp?url=/workshop/author/behaviors/overview.asp

[6]
"Msn wallet," Microsoft. [Online]. Available: https://wallet.msn.com/home/home.aspx

[7]
"Paypal." [Online]. Available: http://www.paypal.com/

[8]
G. W. Bauer, "User data management," mozilla.org, February 2004. [Online]. Available: http://www.mozilla.org/projects/ui/communicator/browser/wallet/

[9]
D. N. Gibbins and S. Harris, "3store," AKT. [Online]. Available: http://inanna.ecs.soton.ac.uk/3store/

[10]
D. Beckett, "Raptor rdf parser toolkit," University of Bristol. [Online]. Available: http://www.redland.opensource.ac.uk/raptor/

[11]
A. Seaborne, "Rdql - a query language for rdf," HP Labs Bristol, January 2004. [Online]. Available: http://www.w3.org/Submission/RDQL/

[12]
R. Jones. [Online]. Available: http://www.audioscrobbler.com/

[13]
"Mozilla firefox extensions," Mozilla. [Online]. Available: http://texturizer.net/firefox/extensions/