Realising A Read-Write Web of Data

v2009-06-11

Tim Berners-Lee[1], Richard Cyganiak[2], Michael Hausenblas[2], Joe Presbrey[1], Oshani Seneviratne[1], and Oana-Elena Ureche[2]

[1] MIT CSAIL, Cambridge
Massachusetts, USA
(timbl | presbrey | oshani)@csail.mit.edu
[2] DERI, National University of Ireland, Galway
IDA Business Park, Lower Dangan, Galway, Ireland
firstname.lastname@deri.org

Abstract. The current Web of Data, including linked datasets, RDFa content, and GRDDL-enabled microformats is a read-only Web. Although this read-only Web of Data enables data integration, faceted browsing and structured queries over large datasets, we lack a general concept for a read-write Web of Data. That is, we need to understand how to create, update and delete RDF data in a secure, reliable, trustworthy and scalable way. Attempting to change this situation, this paper reviews available components, presents our vision of a uniform architecture for a read-write Web of Data as well as a proof of concept. The paper exposes issues and challenges of the proposed architecture and discusses the next necessary steps.

1 Introduction

With the recent uptake of the linked data movement, as well as RDFa content and GRDDL-enabled microformats being used widely, we are witnessing the publication of huge amounts of valuable structured data on the Web. Additionally, major search engines (Google and Yahoo!) started to support indexing structured data such as RDFa. Hence, the incentive to publish these formats has increased dramatically. This in itself is a valuable development; the read-only Web of Data allows data integration, faceted browsing and structured queries over large datasets. Additionally, it drives the development of applications on top of structured and interlinked data [8]. However, these applications will sooner or later not only want to consume data, but also perform changes to the state of the resources they are operating on.

The current read-only Web of Data is the starting point of our work, its baseline. The contribution of this paper is on the one hand a general concept how to create, update and delete RDF data in a secure, reliable, trustworthy and scalable way. On the other hand, based on our proof of concept, we expose issues and challenges for realising a read-write Web of Data.

In order to put our contribution in a context, we will elaborate the background for this work in the next section. Section 3 presents use cases and states requirements for solutions to enable a read-write Web of Data. Section 4 then lists core components of our vision of a read-write Web of Data, reviewing existing and related work for each components. In Section 5, we propose the overall concept, show how the core components fit in and discuss issues in terms of existing non-semantic technologies and integration. Finally we conclude our contribution in section 6 and outline the next steps.

2 Background

The Web of Documents featured editable pages from the very beginning on, which happened to be continued in Wiki environments. The current *Web of Data*, however, is a read-only Web allowing for data integration, faceted browsing and structured queries over large datasets. However, we are still lacking a general concept for a read-write Web of Data. When we talk about the Web of Data, we mean structured and possibly interlinked data in RDF, such as hinted in Figure 1.

We mainly focus on the linked data part of the Web of Data in this paper. Linked data [1] has changed the way RDF is deployed nowadays. Linked data is essential RDF technology that uses HTTP URIs to denote things, and provides useful information about a thing at the thing's URI. Additionally, links to other RDF datasets are included, allowing machines to traverse the Web of Data similar to what humans can do in the Web of Documents.

To the best of our knowledge only few attempts to realise a true read-write Web of Data can be identified, nowadays. In the following we will review them as well as contrast them to solutions from the Web 2.0.

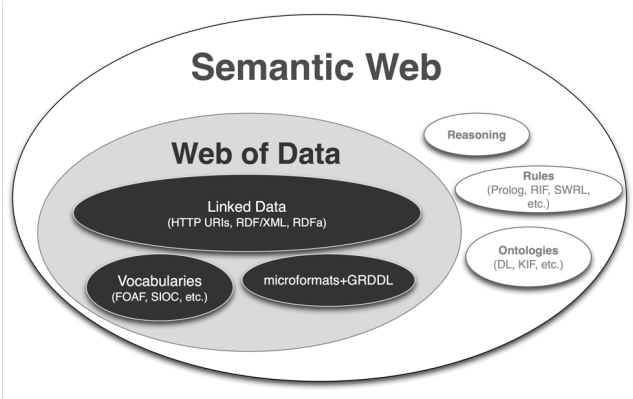

Fig. 1. The Web of Data—structured and interlinked data in RDF.

2.1 Existing Work in the Web of Data realm

In [2] we have described Tabulator Redux, one of the first read-write Web of Data implementations. We extended Tabulator to write back changes on a per-triple bases in a Wiki environment. About the same time, the authors of [3] described a method based on an extension of another Wiki, using RDFa to establish in-place edits. In early 2009, a RESTful method to contribute triples to an RDF file has been proposed [10], and only recently Gray et. al. [7] presented a RESTful read-write Web of Data service, giving application authors access to simple persistence, simple (social) sharing, and lightweight semantics, based on the Changesets specification[3].

One influential activity can be detected around SPARQL. The respective W3C Working Group has recently been chartered[4] to standardise the dialects—Jena's SPARUL, ARC2's SPARQL+, etc.—to define a uniform SPARQL Update interface. In the following, we will refer to the currently different implementations as SPARQL Update (neglecting the minor differences between them and assuming that we will soon have a standard language, here). Currently only view secure SPARQL endpoint implementations are known. One example is OpenLink's Data Space[5].

@@TODO: WebDAV

Finally, we started to investigated and implement a form-oriented update mechanism called *pushback/RDForms*[6] only recently. The insights we gained from this straw men proposal influenced the herein presented work.

To this end, we tried to highlight the fact that, though there exist partial attempts to enable a true read-write Web of Data, we lack a general concept for a true read-write Web of Data, addressing security, privacy, trust, reliability and scalability.

2.2 Existing Work in the Web 2.0

While the above discussed approaches and technologies target RDF environments, the majority of write-interfaces reside in Web 2.0 environments. Examples of so called *site-specific APIs* range from photo sites[7] over bookmarks[8], to the wide range of Google products[9]. A leading API directory lists more than 1300 APIs at time of writing[10].

An additional question arises, now: until the wide availability of RDF-based back-ends (such as SPARQL endpoints), how can the majority of the site-specific APIs be accessed and hence integrated in the Web of Data.

---

[3] `http://n2.talis.com/wiki/Changesets`
[4] `http://www.w3.org/2009/01/sparql-charter`
[5] `http://www.openlinksw.com/OdbcRails/main/ODS/VirtODSFOAFSSL`
[6] `http://esw.w3.org/topic/PushBackDataToLegacySources`
[7] `http://www.flickr.com/services/api/`
[8] `http://delicious.com/help/api`
[9] `http://code.google.com/apis/`
[10] `http://www.programmableweb.com/apis`

3 Use Cases and Requirements

We have seen so far that, though there are certain attempts around partial addressing the realisation of a read-write Web of Data, no general concept is available. In the following we will have a closer look into use cases for a read-write Web of Data and state our requirements.

3.1 Use Cases

The use cases described below have in common that they happen in the *semantic space*. By semantic space we mean that an agent (human or machine alike) consumes some linked data in RDF. Most likely the agent would operate on linked data, but also data mashups may be the source of the RDF data. A human agent would typically navigate through linked datasets using Tabulator[11] or the OpenLink Data Explorer[12], while a machine agent (a crawler, aggregator, etc.) would use some HTTP library such as Ruby's `net/http.rb`[13] or PHP's `curl`[14].

While operating in the semantic space, at some point in time the agent might have gathered some information and wants to perform an action on it. This can be adding a bookmark, annotating a photo, ordering a book, updating a calendar, or posting a comment to a blog. This *write operation* should cause the original resource to be changed or some effects to take place, such as the event appearing in the calendar or the book to be shipped.

UC1: Event Synchronisation Calendar data of individuals along with event data from sites such as upcoming.org (IMPLEMENTED)

UC2: Link Management Jim browses DBpedia (`http://dbpedia.org/resource/Fixed-wing_aircraft`) and detects a broken link in this resource[15]. He is a good Web citizen and wants to fix this link.

UC3: Microblogging . Twitter, identi.ca, etc. provide base for exchange, discussion. Lara, browsing a SIOCified blog post want to post a twit from it (IMPLEMENTED) or wants to bookmark for example a URI in a microblogging post from Twitter in delicious.

UC4: Software Development . Bob, a software project manager has three developers (and their calendars), and an issue tracker. Based on an urgent feature request he wants to change the assignments of bugs to certain developers, based on their availability (IMPLEMENTED).

[11] `http://www.w3.org/2005/ajar/tab`
[12] `http://linkeddata.uriburner.com/ode/`
[13] `http://www.ruby-doc.org/stdlib/libdoc/net/http/rdoc/`
[14] `http://ie2.php.net/curl`
[15] `http://upload.wikimedia.org/wikipedia/commons/6/6f/air_india_b747-400_vt-esn_arp.jpg`

UC5: Profile Information Reuse . In FOAF profiles we have data such as name or address readily available. Joe wants to, rather than re-entering the data always in each HTML form (booking flight, order book, etc.) to use it from his FOAF profile directly (IMPLEMENTED - pac - but note that this is actually RDForms-specific, as currently no actual update happens there).

UC6: Photo Annotation . Sarah consumes an RDF graph where data from both the flickr-wrapper and Geonames is present and wants to update the geo location data (as it is for example taken on wrong position or inaccurate).

3.2 Requirements

Based on the use cases outlined above and taking into account our experience with earlier systems, we require that a read-write Web of Data must:

- Be based on the *Web of Data core standards*: URIs, HTTP and RDF;
- Provide for a *uniform interface*, that is, treat RDF and non-RDF back-ends the same way;
- *Scale* and provide for *high-performing* operations;
- Support the *social constraints and expectations* human users have regarding the update of the underlying data in terms of trust, provenance, etc.

Additionally, the solution should be *representation agnostic*, meaning whatever RDF serialisation is supplied (RDF/XML, Turtle, RDFa, microformats+GRDDL), the system should be able to handle it. Finally, the solution should be *usable* in currently deployed Web environments, such as xAMP, JavaScript. Ideally, it requires little configuration effort and provides for a framework defining interfaces and terms on an abstract level along with a reference implementation.

4 Core Components

In order to realise a read-write Web of Data, several components are necessary. In the following we discuss these components and highlight their interdependencies.

4.1 Authentication

The first step in the process of allowing a write operation is to learn who the party that requests it is. This is called *authentication*.

Identity In order to perform authentication, one needs to know about the identity of an agent. In the context of the Web of Data we speak of a *Web ID*[16], a URI which identifies an agent and typically exposes a description of that agent when dereferenced.

[16] `http://esw.w3.org/topic/WebID`

Existing Solutions In [5] the HTTP Authentication is defined. It includes the specification for a Basic Access Authentication scheme not considered to be a secure method of user authentication—unless used along with an external secure system, such as SSL—as the credentials are send over the network in clear text.

OpenID Authenticaion[17] provides a way to prove that an agent owns a Web ID without passing around the agent's credentials. It is a completely decentralized system in the sense that anyone can choose to be a consumer or identity provider without having to register or be approved by any central authority.

@@TODO: API keys

Web of Data As described in [12], FOAF+SSL, is an authentication protocol that links a Web ID to a public key, thereby enabling a global, decentralized and open yet secure social network. It uses X.509 Public Key Infrastructure [9] standards where the "Web of Trust" is built using FOAF profile data. There exist already many implementations ranging from Apache [18] to iPhone mobile phones.

4.2 Authorisation

Along or after the authentication of an agent, the authorisation takes place. Given a resource R and an agent's Web ID URI_{WebID} the question is if the agent is allowed to access the resource and if so, which operations the agent is allowed to perform on it.

Existing Solutions In [13] existing access control models as applied to collaboration are examined. In a similar vain in [14] three access control models for collaborative environment are introduced and compared using several evaluating criteria.

Web of Data WebAccessControl (WAC) [19] is a decentralized system for allowing different users and groups various forms of access to resources, where users and groups are identified by HTTP URIs.

Figure 2 shows the WAC vocabulary[20]

Note that the following prefix assignments have been used: `acl:` ... `<http://www.w3.org/ns/auth/acl#>` and `gen:` ... `<http://www.w3.org/2006/gen/ont#>`[21]

@@TODO: Describe Joe's Apache module here

Similar to WAC, Giunchiglia et. al. [6] have recently presented an ontology-driven community access control mechanism.

[17] `http://openid.net/specs/openid-authentication-1_1.html`
[18] `http://foaf.me/Enabling_SSL_Client_Certificates_on_Apache.php`
[19] `http://esw.w3.org/topic/WebAccessControl`
[20] `http://www.w3.org/ns/auth/acl`
[21] `http://www.w3.org/DesignIssues/Generic.html`

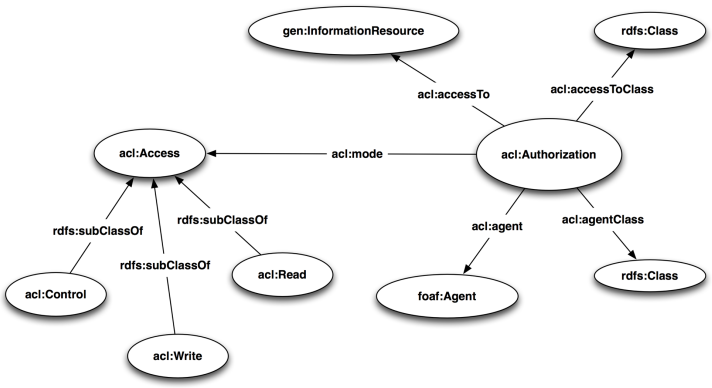

Fig. 2. The WAC vocabulary.

4.3 Update Protocols

Existing Solutions WebDAV, proprietary REST/POST interface etc.,

Web of Data SPARQL Update, changesets (?)

4.4 Wrapper

Reason: most data out there to-date is non-RDF, hence we need to wrap it.

Read-Wrapper Read: linked data publishing process

Write-Wrapper A pushback is a possible realisation of a RDF write-wrapper.

4.5 Client-side mechanisms

possible approaches, such as single-triple update, form-based, visual (?), etc.

RDForms An RDForm is a way for a User Agent to solicit and communicate structured updates to a SPARQL endpoint that supports SPARQL Update. It consists of:

- An (X)HTML form, decorated with RDFa, using the RDForms vocabulary[22];
- An RDForms processor, gleaning the RDF from the form and turning it into a SPARQL Update statement.

[22] `http://rdfs.org/ns/rdforms`

We note that the main reason for introducing a dedicated vocabulary to capture the values is that this allows us to write a generic RDForms processor which only needs to understand this vocabulary. However, this requires knowledge about how the original terms map to the terms in the RDForm vocabulary. The RDForms vocabulary (Figure 3) defines the semantics of a key-value pair set of fields in an HTML form along with allowed operations on them.

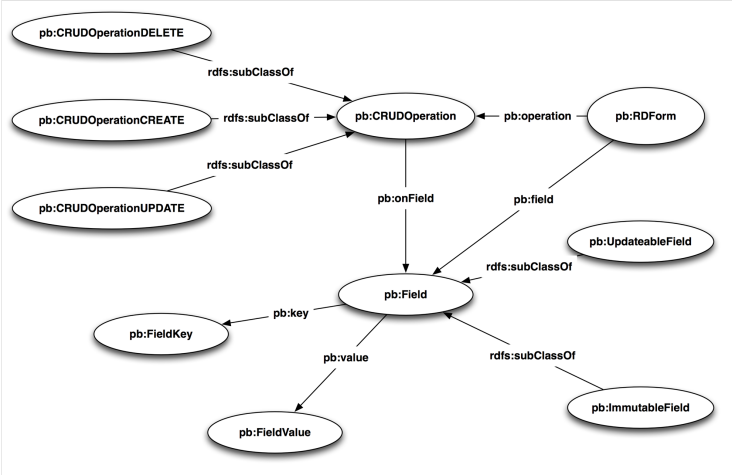

Fig. 3. The RDForms vocabulary.

For each RDForm field a CUD[23] operation, such as "add" a field or "update the field's value", may be permitted. In order to understand which CUD operation is allowed and intended on which RDForm field, we must flag it somehow. The process of defining which CUD operation should be applied on which field(s) is called "binding" in RDForms. A *CUD operation in RDForms* is a declaration of an intended operation on an RDForm field. An intended operation can be one of the following: CREATE, UPDATE or DELETE, along with a binding to an RDForm field, defineing to which field the operation applies.

RDForms can be dynamically generated from an RDF graph along with an existing (X)HTML form for the data present in the graph. The process of generating an RDForm on-the-fly is called "fusion". The RDForms generator that implements the fusion is online available[24]. To this end, we have implemented a client-side, JavaScript based RDForms processor using jQuery/rdfquery and several RDForms (Twitter, Jira issue tracker, etc.) [25]. Parts of the source code of the exemplary Jira RDForm are shown in Listing 1.1. Note that the usage of @resource and @id in the case (within RDForms) is actually desired and fully

[23] `http://en.wikipedia.org/wiki/Create,_read,_update_and_delete`
[24] `http://ld2sd.deri.org/pushback/fusion/`
[25] `http://esw.w3.org/topic/PushBackDataToLegacySourcesRDForms`

```
<form id="fo1" action="" about="#fo1" typeof="pb:RDForm">
  <div rel="pb:operation">
   <div about="#crud-op1" typeof="pb:CRUDOperationUPDATE">
    <span rel="pb:onField" resource="#fo1.f1" />
   </div>
   ...
   <fieldset>
    <legend>Issue</legend>
     <div rel="pb:field">
      <div typeof="pb:UpdateableField" about="#fo1.f1" >
       <label for="fo1.f1" rel="pb:key" resource="#bug-id"
            property="dcterms:title">Bug ID*</label>:
       <div rel="pb:value">
        <div about="#fo1.f1.val" typeof="pb:FieldValue">
         <input id="fo1.f1.val" type="text" property="rdf:value" content=""
              value="" size="40" />
        </div>
       </div>
      </div>
     </div>
     ...
   </fieldset>
</form>
```

Listing 1.1. Jira RDForm source code.

in-line with the semantics; we are actually talking about the elements of the HTML form.

5 Concept

After introducing the different components above, we now discuss how they play together from an architectural point of view.

We propose a three-tiered architecture: an agent (top) has a Web ID associated with it, same as resources have access control information attached (bottom). The *read* path to the right is realised through linked data, the *write* path to the left relies on SPARQL Update. Wherever necessary, wrapper allow non-RDF sources or sinks to participate, offering the same interfaces as native RDF systems.

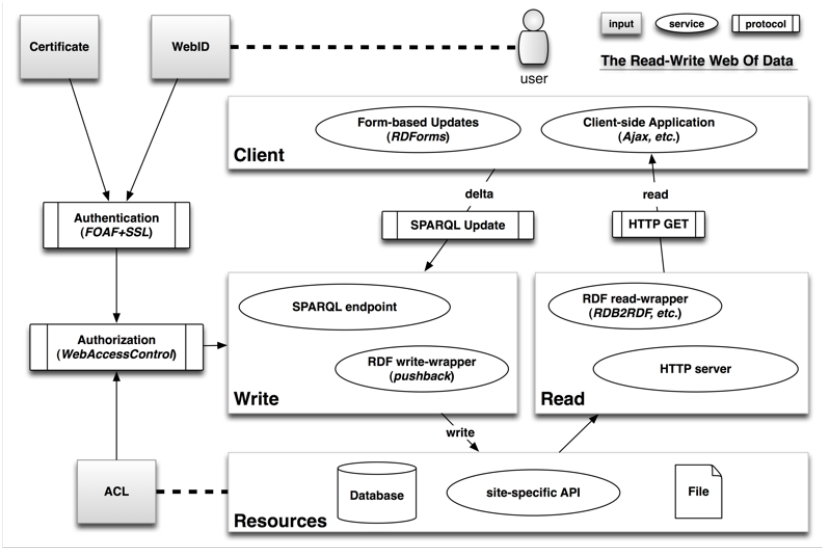

Fig. 4. Concept of a Read-Write Web of Data.

Fig. 4 shows the read-write Web of Data concept comprising the following components:

- Agent identifier (Web ID)
- An authentication scheme and protocol (FOAF+SSL)
- An authorization scheme and protocol (WAC)
- An update protocol (SPARQL Update)
- Read-wrappers around Web 2.0 sites (linked data)
- Write-wrappers around Web 2.0 sites/data (pushback)
- Client-side mechanism to send updates (RDForms)

5.1 Proof of Concept

In order to demonstrate how our concept of a read-write Web of Data works, we have implemented one of our use cases where calendar data is updated using the Goolge calendar API.

@@TODO: Oana to describe setup on the server side (ARC2, Google API)

@@TODO: Oshani to describe setup on the client side (invoke of RDForm in Tabulator, update creation)

5.2 Evaluation

We have in total implemented the following three interfaces:

HTTP POST RDFa processor parses RDForm on client-side, key-value pairs are extracted, proprietary HTTP POST sent to pushback that calls according site-specific APIs functions.

SPARQL Update with RDForms RDFa processor parses RDForm on client-side, an INSERT operation is with the gleaned triples from the RDForm is used throughout all operations, the operations semantics are conveyed in the RDForms, pushback acts as a SPARQL endpoint, has to poll and interpret RDForm semantics to call according site-specific APIs functions.

RDForms to SPARQL Update RDForms processor on client-site creates according SPARQL Update operations from the RDForm and sends the pushback the SPARQL Update operations which has again to interpret RDForm semantics to call according site-specific APIs functions.

What follows in the Table 5.2 is a comparison of the three different implemented approaches described above with the following keys:

1. Architectural Properties
   - 1.1 AWWW ... compliant to the *Architecture of the World Wide Web* [11];
   - 1.2 RESTful ... compliant to the REST architectural style as of [4];
   - 1.3 Semantics ... compliant to SPARQL semantics;
2. Complexity
   - 2.1 Client ... effort to develop and maintain the client side of the implementation;
   - 2.2 Server ... effort to develop and maintain the server side of the implementation;
3. Implementation Issues
   - 3.1 Uniform Interface ... treating RDF and non-RDF back-ends uniformly;
   - 3.2 Scalability ... expected scalability w.r.t. the Web;
   - 3.3 Performance ... collective measure for HTTP round-trips, cache-ability, etc.

	HTTP POST	SPARQL with RDForms	RDForms to SPARQL
Architectural Properties			
1.1 AWWW	yes	yes	yes
1.2 RESTful	yes	no	no
1.3 Semantics	N.A.	no	yes
Complexity			
2.1 Client	low	middle	high
2.2 Server	low	high	high
Implementation Issues			
3.1 Uniform Interface	no	yes	yes
3.2 Scalability	good	unknown	unknown
3.3 Performance	good	good	good

Table 1. Comparison of the three implemented update interfaces.

5.3 Issues

While working on the proof of concept as described above, we have identified certain issues: authentication, simple vs. uniform interface, etc.

5.4 Challenges

We are not able to handle transactions, only one shots. Error handling not addressed. Further we need to clarify how to handle concurrency edits.

6 Conclusion & Outlook

Proof of concept there, improve details, implement more UC, extend support for authentication, etc.

Acknowledgements

Our work has partly been supported by the European Commission under Grant No. 217031, FP7/ICT-2007.1.2, project "Domain Driven Design and Mashup Oriented Development based on Open Source Java Metaframework for Pragmatic, Reliable and Secure Web Development" (Romulus).

References

1. T. Berners-Lee. Linked Data, Design Issues. `http://www.w3.org/DesignIssues/LinkedData.html`.
2. T. Berners-Lee, J. Hollenbach, K. Lu, J. Presbrey, E. Prud'hommeaux, and mc schraefel. Tabulator Redux: Browsing and Writing Linked Data. In *WWW 2008 Workshop: Linked Data on the Web (LDOW2008)*, Beijing, China, 2008.

3. S. Dietzold, S. Hellmann, and M. Peklo. Using JavaScript RDFa Widgets for Model/View Separation inside Read/Write Websites. In *Proceedings of the 4th Workshop on Scripting for the Semantic Web*, Tenerife, Spain, 2008.
4. R. T. Fielding and R. N. Taylor. Principled design of the modern Web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, 2002.
5. J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. Request for Comments: 2617, June 1999, IETF Network Working Group, 1999.
6. F. Giunchiglia, R. Zhang, and B. Crispo. Ontology Driven Community Access Control. In *Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2009)*, 2009.
7. N. Gray, T. Linde, and K. Andrews. SKUA retrofitting semantics. In *Proceedings of the 5th Workshop on Scripting for the Semantic Web*, Heraklion, Greece, 2009.
8. M. Hausenblas. Exploiting Linked Data For Building Web Applications. *IEEE Internet Computing*, 13(4):NN, 2009.
9. R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. Request for Comments: 2459, January 1999, IETF Network Working Group, 1999.
10. T. Inkster. Inav the Terrible. Buzzword.org.uk Draft 7 February 2009, buzzword.org.uk, 2009.
11. I. Jacobs and N. Walsh. Architecture of the World Wide Web, Volume One. W3C Recommendation 15 December 2004, W3C Technical Architecture Group (TAG), 2004.
12. H. Story, B. Harbulot, I. Jacobi, and M. Jones. FOAF+TLS: RESTful Authentication for the Social Web. In *Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web*, 2009.
13. W. Tolone, G.-J. Ahn, T. Pai, and S.-P. Hong. Access control in collaborative systems. *ACM Comput. Surv.*, 37(1):29–41, 2005.
14. B. Zhao. Collaborative Access Control. *Telecommunications Software and Multimedia*, 2008.