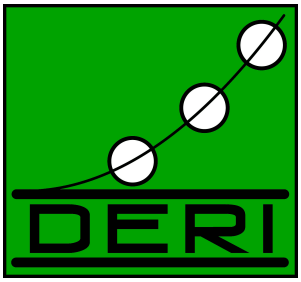
DERI – DIGITAL ENTERPRISE RESEARCH INSTITUTE



On Integration Issues of Site-Specific APIs into the Web of Data

Tim Berners-Lee
Richard Cyganiak
Michael Hausenblas
Joe Presbrey
Oshani Seneviratne
Oana-Elena Ureche

DERI Technical Report 2009-08-14
August 2009

DERI – DIGITAL ENTERPRISE RESEARCH INSTITUTE

DERI Galway
IDA Business Park
Lower Dangan
Galway, Ireland
http://www.deri.ie/

DERI Technical Report
DERI Technical Report 2009-08-14, August 2009

On Integration Issues of Site-Specific APIs into the Web of Data

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

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. Starting from the experience we have gathered with Tabulator Redux—a single-triple update system based on a data Wiki—we review necessary components to realize a read-write Web of Data. We propose a form-based editing approach for RDF graphs along with the integration of site-specific APIs. Further, we present a concept of a uniform architecture for a read-write Web of Data, including a demonstration. Eventually, our work reveals issues and challenges of the proposed architecture and discusses future steps.

Keywords: Web of Data, read-write Web, update RDF data, site-specific APIs

DISCLAIMER *We note that this is work in progress and should be cited as such. This is an intermediate report to capture the current state of our work (mid 2009). For further information and ongoing activities, the interested reader is invited to visit* `http://esw.w3.org/topic/WriteWebOfData`.

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

Acknowledgements: This work has been carried out in the Linked Data Research Centre (LiDRC).

Contents

1 Introduction

With the recent uptake of the linked data movement, we are witnessing the publication of huge amounts of valuable structured data on the Web. Further, major search engines (Google and Yahoo!) started to support indexing structured data such as RDFa and microformats. Hence, the incentive to publish these formats has increased dramatically. The current Web of Data allows, for example, data integration, faceted browsing and structured queries over large datasets. Additionally, it drives the development of applications on top of structured and interlinked data [Hau09]. Very likely, 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 Web of Documents featured editable pages since the very beginning on, which happened to be continued in Wiki environments. The current *Web of Data* is essentially a read-only Web. When we talk about the Web of Data, we mean structured and possibly interlinked data in RDF, such as hinted in Fig. 1. We mainly focus on the linked data part of the Web of Data in this paper. Linked data [BL] has changed the way RDF is deployed nowadays. Linked data is 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. The starting

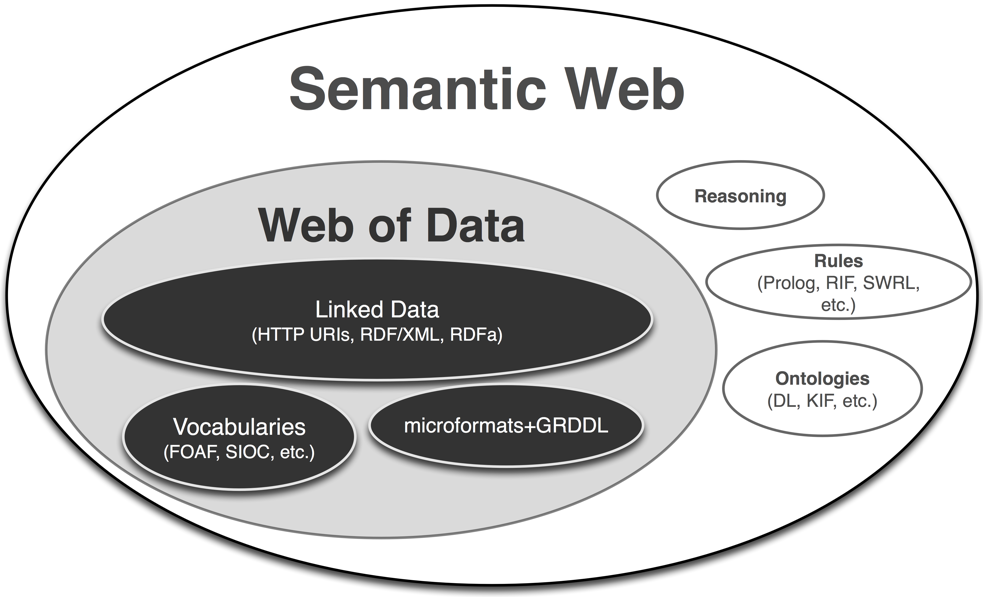


Figure 1: The Web of Data—structured and interlinked data in RDF.

point of this work is the experience we have gathered with Tabulator Redux [BLHL$^+$08], which extends the linked data browser Tabulator with a single-triple update system based on a data Wiki. Our contribution on the one hand is a form-based editing approach for RDF graphs called RDForms, along with the integration of legacy systems such as site-specific APIs, realized through write-wrappers (pushbacks), somehow comparable to the linked data read-wrappers. On the other hand, we present our concept of a uniform architecture for a read-write Web of Data, including a demonstration.

A read-write Web of Data has potentially many application areas. Reviewing the current areas of Web update applications suggests that one has to deal with manifold dimensions such as structured vs. unstructured content, granularity, etc.—in Fig. 2 an overview on Web update applications is shown. The x-axis represents the degree of *freedom regarding the content structure*

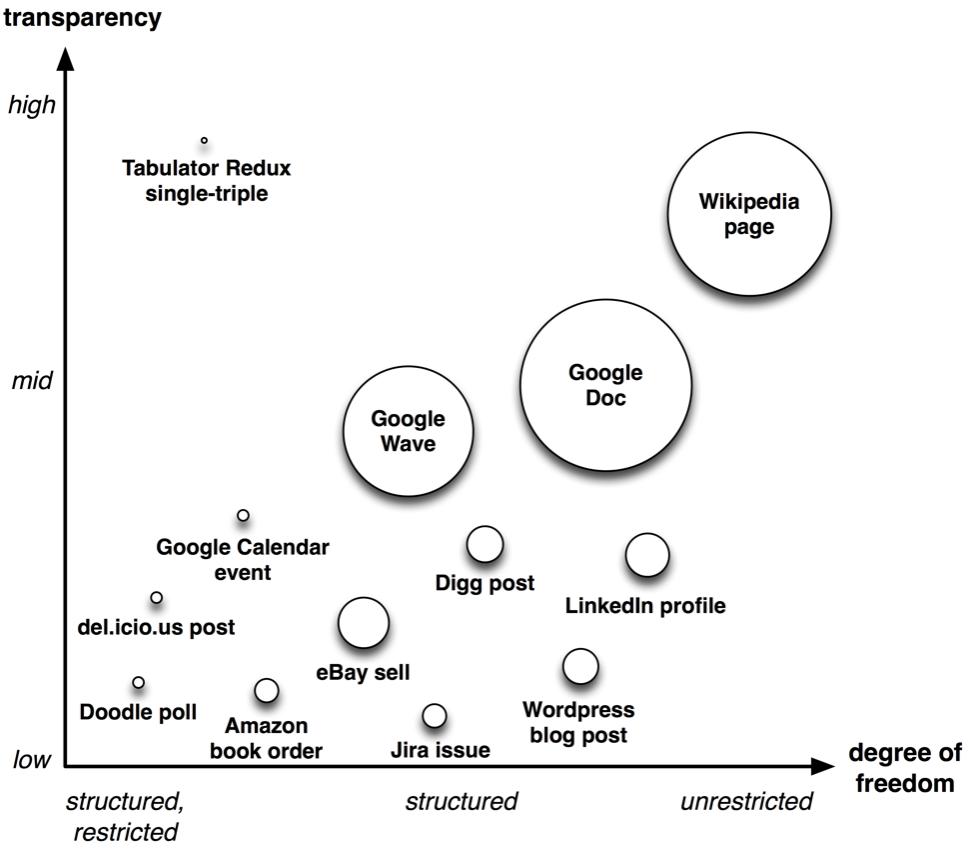


Figure 2: Web update applications.

and the *transparency of the editing process* is shown on the y-axis. With transparency of the editing process we mean how present the editing is to the human user. Whereas in case of an update process of a page on Wikipedia, one has apparently the impression of editing the actual page, however, when one orders a book at Amazon, the update (of Amazon's database) is much more hidden and of indirect nature. Note that the size of the bubbles in Fig. 2 denote the volume of a typical update operation, ranging from a single RDF triple in case of Tabulator to couple of sections in a Google Doc. Eventually, there are many more update application scenarios on the Web, not depicted in Fig. 2.

We will now present use cases and state requirements for a read-write Web of Data (Section 2). Then, Section 3 introduces the overall concept of a read-write Web of Data and shows how the components play together. In Section 4 we review necessary core components and in Section 5 we present our proposal for a form-based approach for editing RDF graphs, including the integration of legacy systems. We describe a demonstration of the proposed read-write Web concept in Section 6 and outline the next steps in Section 7.

2 Use Cases and Requirements

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. A human agent would typically navigate through linked datasets using Tabulator[1] or the OpenLink Data Explorer[2], while a machine agent (a crawler, aggregator, etc.) would use some HTTP library such as Ruby's `net/http.rb`[3]. While operating in the semantic space, at some point in time the agent might want to perform an action on it, such as adding a bookmark or updating a calendar.

2.1 UC1: Calendars

Finding a common time slot is often hard. Given two calendars and a certain event, how can a time slot be found and updated in the respective calendars of the individual participants?

2.2 UC2: Blogging

Microblogging services such as Twitter, identi.ca, etc. allow people to broadcast and exchange short messages on the Web. Lara, browsing a SIOCified blog post[4], wants to post a twit from there. The other day Lara wants to bookmark, for example, a URI in a microblogging post from Twitter in delicious. Can she do these things without leaving her environment (blog, etc.)?

2.3 UC3: Software Development

Bob, a software project manager has three developers available for a project, as well as an issue tracker. Based on an urgent feature request Bob wants to change the assignments of bugs to certain developers, based on their availability. Thanks to D2R server[5] the issue tracker data is available in RDF, same is the calendar data. Bob figures out who of his developers is available when by merging the three developer's calendars. Then, he needs to go back to the issue tracker and change the assignments and additionally may need to confirm or block certain time slots in the calendars. Why can't he do this in-place, where he has all the merged RDF data already available?

2.4 UC4: Photo Annotation

Sarah consumes an RDF graph where data from the flickr wrapper and Geonames is mashed up. She wants to update the geo-location of a flickr photo (e.g., because it is inaccurate) with the data from Geonames. She needs to log into flickr, copy and paste the data from Geonames and update the annotation there, though she has all the data needed for the task already in the initial RDF graph.

[1] `http://www.w3.org/2005/ajar/tab`
[2] `http://linkeddata.uriburner.com/ode/`
[3] `http://www.ruby-doc.org/stdlib/libdoc/net/http/rdoc/`
[4] `http://esw.w3.org/topic/SIOC/EnabledSites`
[5] `http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/`

2.5 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*: Uniform Resource Identifiers (URIs) [BLFM05] for identifying things of interest, the Hypertext Transfer Protocol (HTTP/1.1) [FGM+99] for accessing things and description of things, and the Resource Description Framework (RDF) [KCM04] as a data model to relate and describe things;
- Provide for a *uniform interface*, that is, treat RDF and non-RDF back-ends, such as site-specific APIS, the same way;
- *Scale* to the size of the Web;
- Support the *social constraints and expectations* human users have regarding the update of the underlying data in terms of integrity, 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. Eventually, the solution should be *usable* in currently deployed Web environments, such as xAMP, JavaScript.

3 Concept of a Read-Write Web of Data

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 realized through linked data, the *write* path to the left relies on SPARQL Update. Wherever necessary, wrapper allow non-RDF sources to participate, offering the same interface as native RDF systems.

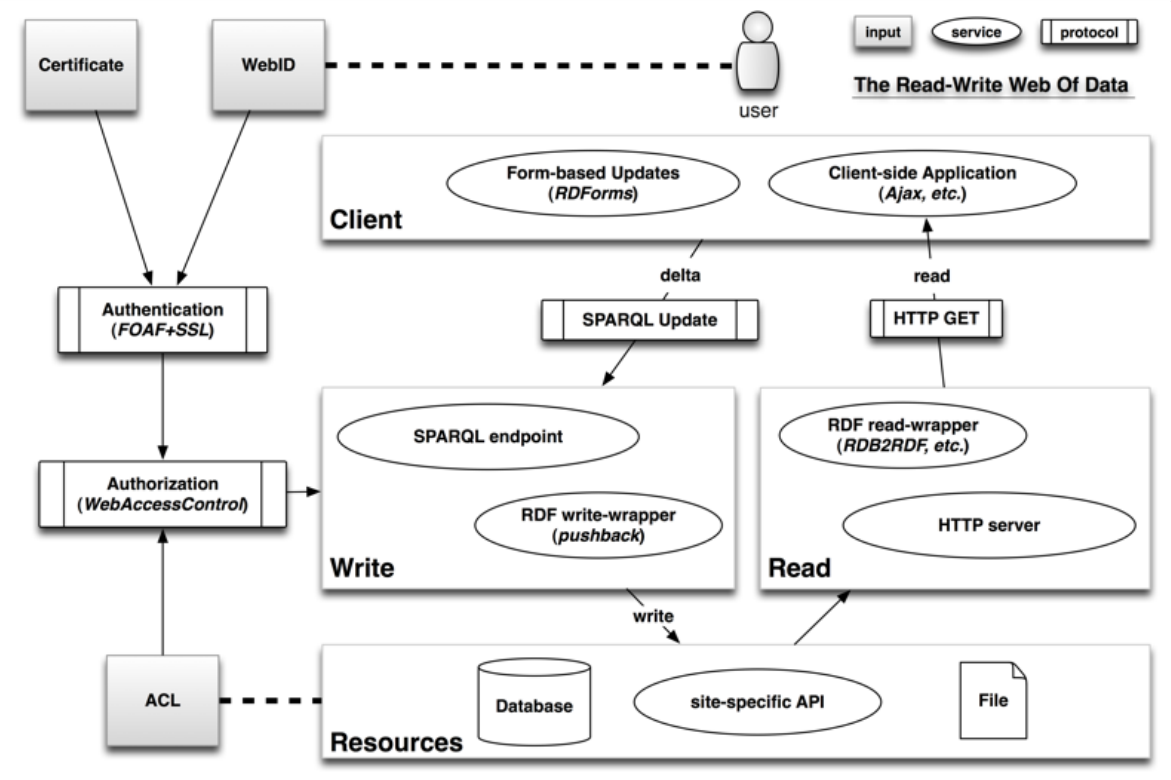


Figure 3: Concept of a read-write Web of Data.

Fig. 3 depicts the read-write Web of Data concept comprising the following components:

- A read protocol and a write protocol (Section 4.1);
- An agent identifier and an authentication scheme and protocol (Section 4.2);
- An authorization scheme and protocol (Section 4.3);
- For the integration of non-RDF sources, read/write-wrapper (Section 5).

The read/write process (Fig. 4) is as follows: an agent, authenticates against a service with and the authorization is checked. If the desired operation on the resource is allowed for the agent, it is executed, potentially engaging wrapper for legacy data sources.

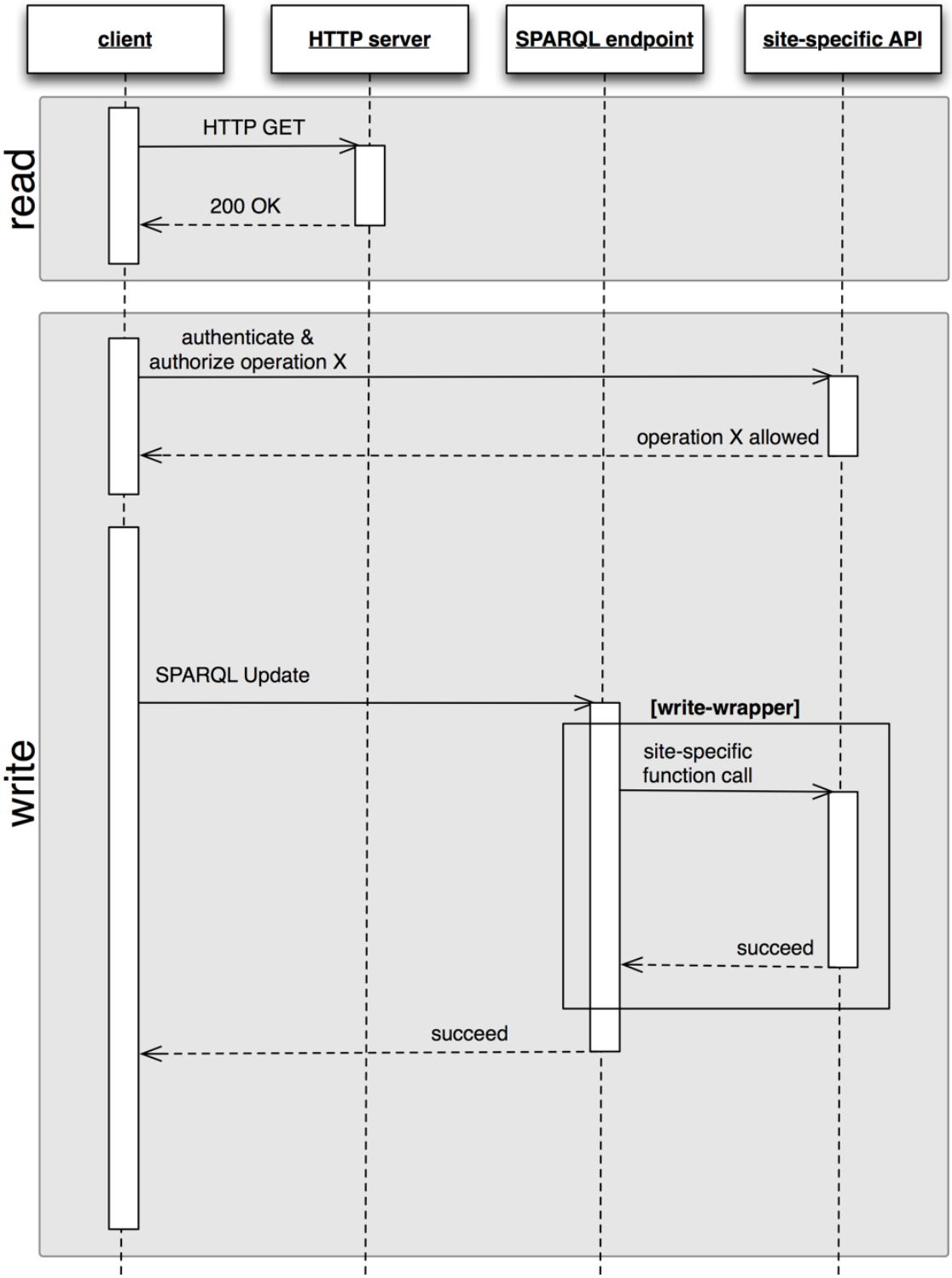


Figure 4: Sequence diagram for read and write operations in the Web of Data.

4 Core Components

In order to realize a read-write Web of Data, we require an update protocol for writing linked data resources, authentication and authorization mechanisms to secure linked data access and management, and implementations supporting these protocols and mechanisms for servers and clients. Additionally, wrappers for integrating legacy data as RDF are necessary for maintaining compatibility with the Web at large. In the following, we discuss these components and highlight their interdependencies.

4.1 Update Protocols

4.1.1 Existing Solutions

The HTTP protocol [FGM+99] specifies general methods for querying or managing a single document or resource by URI, e.g., HEAD, GET, POST, PUT, DELETE, OPTIONS. The Web Distributed Authoring and Versioning protocol (WebDAV) [Dus07] extends HTTP with additional methods for collaborative authoring of documents and management of document collections, for example COPY, MOVE, LOCK. WebDAV support is included in the freely distributed Apache web server and several widely-deployed client interfaces including OS X and Microsoft desktop and office products.

However, HTTP with WebDAV is not optimal for editing linked data due to the coarse granularity of HTTP request URIs. For example, to update to a single value in some document using WebDAV, the full document source must be serialized and retransmitted using PUT during every commit since the request URI may not reference individual data within the document (commonly a URI fragment). Further, a client that uses the LOCK method on a resource before editing (for collision avoidance) prevents updates by other clients to all data in the entire document. Thus, WebDAV editing latency increases with document size and collaborating authors.

4.1.2 Web of Data

In [BLHL+08] 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 base in a Wiki environment. About the same time, the authors of [DHP08] 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 [Ink09], and only recently Gray et. al. [GLA09] 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[6].

One influential activity can be detected around SPARQL. The respective W3C Working Group has recently been chartered[7] 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*. Currently, only few secure SPARQL endpoint implementations are known. One example is OpenLink's Data Space[8].

[6] `http://n2.talis.com/wiki/Changesets`
[7] `http://www.w3.org/2009/01/sparql-charter`
[8] `http://www.openlinksw.com/OdbcRails/main/ODS/VirtODSFOAFSSL`

4.2 Authentication

The first step in the process of allowing a read/write operation is called *authentication*. In the context of REST [FT02], authentication of a client is performed by a server to determine the identity of the requesting agent. 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*[9], a URI which identifies an agent and typically exposes a description of that agent when dereferenced. Using the Web ID as the principal identifier for the user avoids the per-domain boundaries of the username/ID token of most centralized authentication models.

4.2.1 Existing Solutions

In [FHBH+99] the HTTP Authentication is defined. It includes the specification for the Basic Authentication scheme which transmits a clear-text username and password and is considered insecure unless used with an encryption scheme such as HTTPS as provided by SSL. OpenID Authenticaion[10] 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 be approved by a central authority. However, its non-standard consumer authorization implementation requires clients to handle provider-specific HTML confirmation pages. This constraint requires all clients of OpenID services to facilitate HTML rendering for the user. For example, WebDAV services deploying OpenID authentication would additionally require modification of WebDAV client software and user upgrades. Another authentication mechanism widely used are API keys, where one registers for a key which is then used in the read/write operation.

4.2.2 Web of Data

As described in [SHJJ09], FOAF+SSL is an authentication protocol that links a Web ID to a public encryption key using X.509 Public Key Infrastructure [HFPS99] to provide a global, decentralized and open yet secure social network by building a "Web of Trust" using FOAF profile data. The user claims a Web ID by self-issuing a client SSL certificate including the Web ID URI in the "Subject Alternative Name" component of the certificate and publishing the cryptographic properties of the certificate in their FOAF file. Normal SSL client-server negotiations occur providing message privacy but a trusted Certificate Authority is not required to validate SSL identity and thus decentralizing authentication. Instead, the server dereferences the Web ID and validates the certificate data with that published in the linked data to provide message authenticity. All clients supporting client SSL certificates inherit support for FOAF+SSL. This includes many widely-deployed browsers such as Mozilla Firefox, Microsoft Internet Explorer, Apple Safari (and even iPhone) and others. We have provided server support for the Apache web server by extending its existing support for SSL with the `mod_authn_webid`[11] module. As discussed in [Pre09], the Apache implementation is modular, high-performance, and scalable. Two high-level language implementations of FOAF+SSL authentication exist (Perl[12] and PHP[13]) that can be integrated into servers, already.

[9] `http://esw.w3.org/topic/WebID`
[10] `http://openid.net/specs/openid-authentication-1_1.html`
[11] `http://dig.csail.mit.edu/2009/mod_authn_webid/`
[12] `http://search.cpan.org/~tobyink/CGI-Auth-FOAF_SSL/`
[13] `https://foaf.me/testLibAuthentication.php`

4.3 Authorization

After authenticating an agent's identity, a mechanism called *authorization* determines whether the identity may perform the requested operation.

4.3.1 Existing Solutions

In [TAPH05] existing access control models as applied to collaboration are examined. In a similar vain in [Zha08] three access control models for collaborative environment are introduced and compared using several evaluating criteria.

4.3.2 Web of Data

Giunchiglia et. al. [GZC09] have recently presented an ontology-driven community access control mechanism. We have recently introduced the *WebAccessControl* (WAC)[14], a decentralized system for allowing different users and groups various forms of access to resources, where users and groups are identified by HTTP URIs. WAC is similar to the access control system used within many file systems, except that the protected documents, as well as the users and the groups are identified by URIs.

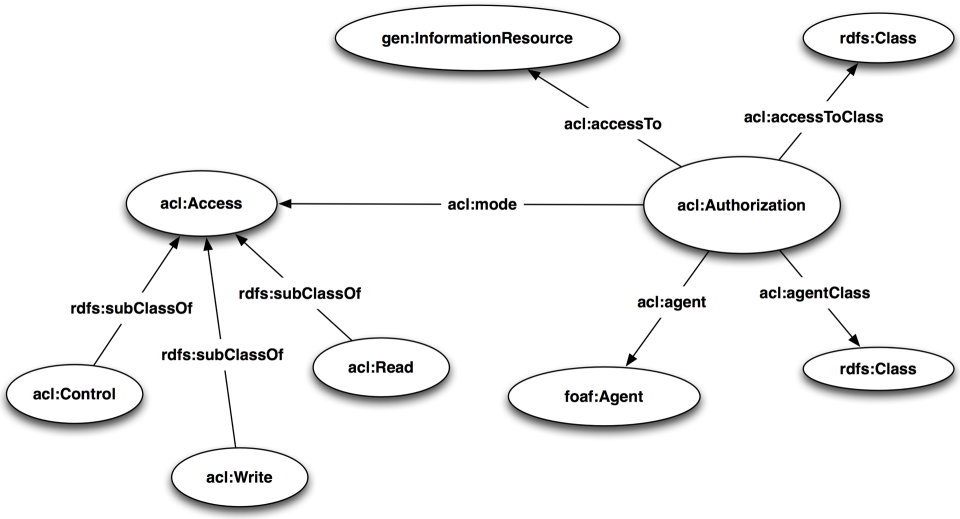


Figure 5: The WAC vocabulary.

Fig. 5 shows the WAC vocabulary[15], providing the terms necessary for access control lists to be used on the Web. Note the following prefix assignments:

```
acl: <http://www.w3.org/ns/auth/acl#>
gen: <http://www.w3.org/2006/gen/ont#>
```

[14] http://esw.w3.org/topic/WebAccessControl
[15] http://www.w3.org/ns/auth/acl

We have provided the `mod_authz_webid`[16] Apache module to demonstrate secure, collaborative authoring of linked data on a WebDAV-enabled filesystem [Pre09] authorized using WAC. Further, a PHP implementation[17] for WAC is available, as well as an experimental WAC explorer for embedded WAC policies[18].

5 A Form-Based RDF Graph Editing Approach

As already noted in [BLHL+08], there are challenges regarding the editing of RDF graphs with regards to the Web of Documents. In order to better understand the underlying issues, we will first have a deeper look into common data models, how they correspond and how the human user on the client side may be supported or hindered by the interface and interactions offered.

5.1 Data Model Correspondency

Motivated by the *Relational Databases on the Semantic Web* design note[19] we can identify correspondences between a relational database table, the RDF data model, graph patterns (in the sense of SPARQL[20]) and forms, as shown in Table 1.

Relational DB Table	RDF data model	Graph pattern	Form
a table	type of the resource s in subject position	`:s rdf:type ?type`	N.A.
a record in the table	all triples where resource s is in subject position	`:s ?p ?o`	a form
a column in the table	a property p	`?s :p ?o`	a field's key
a cell in the table	the value of object o of a certain property p of s	`:s :p ?o`	a field's value

Table 1: Correspondency between relational data, RDF, and forms.

The Fig. 6 introduces the concept of an *information unit*, a set of related data items in a certain context. Referring to Table 1, in case of a relational data base table, the information unit is the record, in an RDF graph the information unit can be understood to be all triples where a certain resource s is in subject position, and in case of a form, it is the entire form itself. Though forms are often perceived as sheer UI or interaction vehicles, one can assign explicit semantics to its components (fields). From Fig. 6 we further learn that there are potentially data items, such as a start date of an event, etc. that are required (marked with an "R"). The entire information unit does not make sense if this data item is not provided or removed. This forms an integrity constraint, which one must address when updating information units.

[16] `http://dig.csail.mit.edu/2009/mod_authz_webid/`
[17] `http://lists.foaf-project.org/pipermail/foaf-protocols/2009-May/000599.html`
[18] `http://ld2sd.deri.org/wacup/`
[19] `http://www.w3.org/DesignIssues/RDB-RDF.html`
[20] `http://www.w3.org/TR/rdf-sparql-query/#defn_TriplePattern`

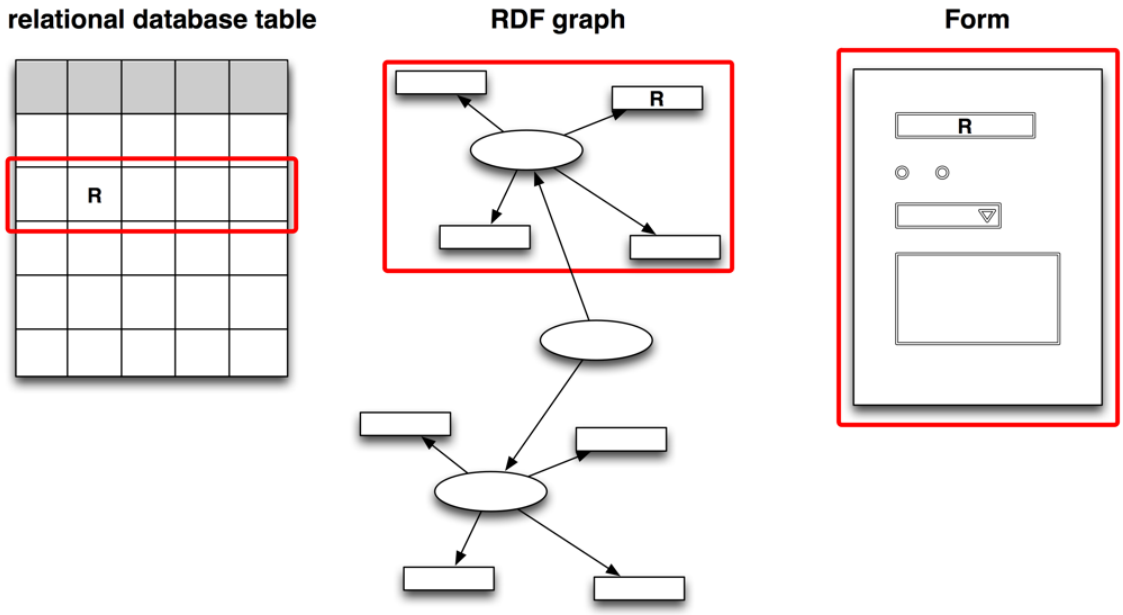


Figure 6: An information unit across different models.

Another characteristic of data items is, that they can be structured (such as dates, ZIP codes, etc.) or unstructured (free-text) regarding their content. The Table 2 illustrates these data item characteristics by example of an information unit from the iCalendar schema [DS98].

	structured	**unstructured**
required	`cal:dtstart`	`cal:summary`
optional	`cal:Valarm`	`cal:description`

Table 2: Data item characteristics examples for an iCalendar information unit.

Eventually, from an interaction perspective, we understand that information units should be first-class objects: Tullis [Tul97] states that:

> ... user should be able to assume that the elements within a group relate to each other semantically.

We hence advocate a form-based editing approach for RDF graphs, which we will elaborate on in greater detail in the next section.

5.2 HTML form + RDFa = RDForms

There are some earlier partial attempts known that proposed a form-based editing of RDF data[21] or mapping[22], alike. However, to the best of our knowledge, no fully specified and implemented proposal for form-based editing of RDF graphs exists.

[21] `http://www.markbaker.ca/2003/05/RDF-Forms/`
[22] `http://www.dehora.net/journal/2005/08/automated_mapping_between_rdf_and_forms_part_i.html`

In order to allow form-based editing of RDF graphs we introduce *RDForms*, somehow comparable to the mapping of field semantics in [DKP+07]. An RDForm is a way for a User Agent to solicit and communicate structured updates to a SPARQL endpoint. An RDForms consists of (i) an HTML form, annotated with the RDForms vocabulary[23] in RDFa [ABMP08], and (ii) an RDForms processor, gleaning the triples from the form in order to create a SPARQL Update statement, which is then sent to a SPARQL endpoint. Additionally, the RDForm can be used on the client-side to check integrity constraints or automatically fill-in values.

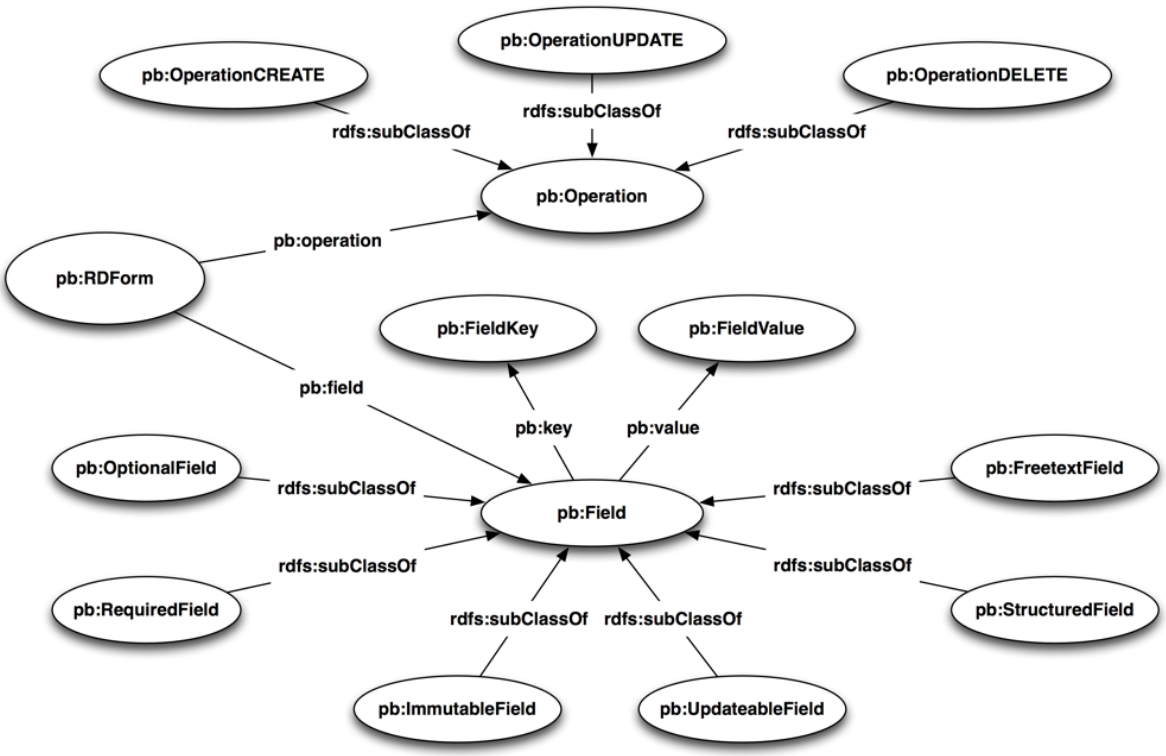


Figure 7: The RDForms vocabulary.

The RDForms vocabulary (Fig. 7) defines the semantics of a key-value pair set of fields in an HTML form. The main reason for introducing a dedicated vocabulary to capture the values is that this allows us to write a generic RDForms processor. However, this requires knowledge about how to map the original vocabulary terms (such as FOAF, SIOC, etc.) to the terms of the RDForm vocabulary. Further, the RDForms vocabulary defines *Create-Update-Delete* (CUD)[24] operations, where a *CUD operation is a declaration of an intended operation on an RDForm.*

[23] `http://rdfs.org/ns/rdforms`
[24] `http://en.wikipedia.org/wiki/Create,_read,_update_and_delete`

In a first step, we have implemented a client-side, JavaScript based RDForms processor using `rdfquery`[25]. The code for the RDForms processor as well as source code examples are available online[26]. One concrete example of an RDForm is depicted in Fig. 8 (as well as parts of the HTML+RDFa source code in Listing 1). This is an RDForm for manipulating data in the Jira issue tracker[27]; it defines an update operation (that is, changing existing values of the information unit *Issue*), and, more specifically Listing 1 shows the RDFa markup of a certain field, such as its key and value definition, as well as its characteristic (*required*).

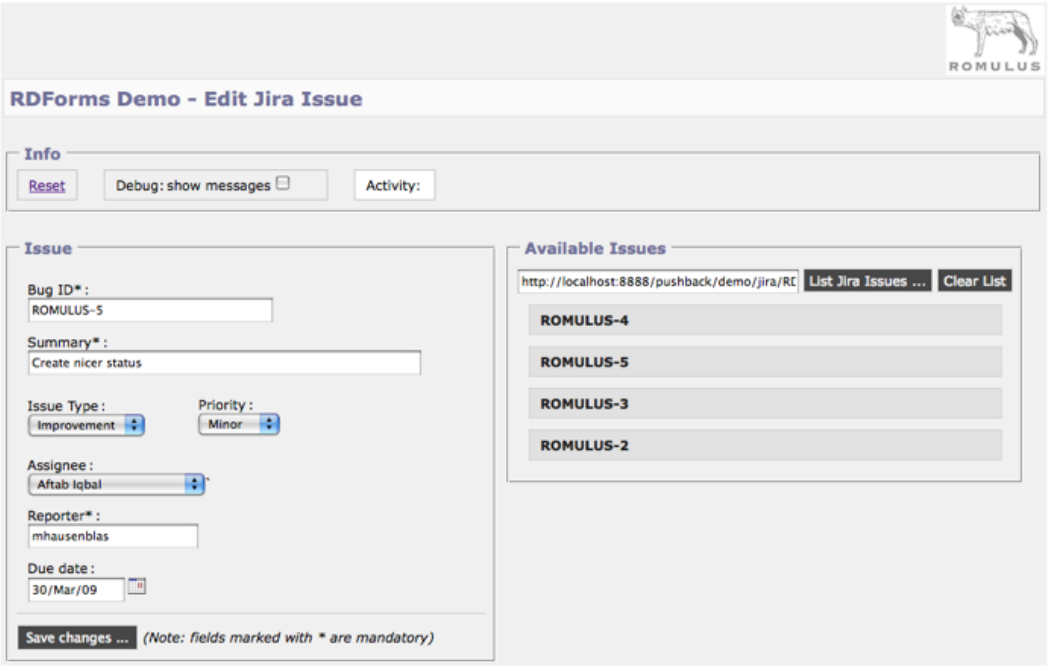


Figure 8: Screen-shot of an RDForm for the Jira issue tracker.

Concerning other exemplary RDForms (such as for Twitter), the reader is referred to the online demo page[28] and a screen-cast demonstrating the Jira RDForm[29].

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, a service that implements a client-side (JavaScript-based) fusion, is available online[30].

[25] `http://code.google.com/p/rdfquery/`
[26] `http://code.google.com/p/pushback/`
[27] `http://www.atlassian.com/software/jira/`
[28] `http://ld2sd.deri.org/pushback/`
[29] `http://vimeo.com/3663028`
[30] `http://ld2sd.deri.org/pushback/fusion/`

```
<form id="fo1" action="" about="#bugform" typeof="pb:RDForm">
  <div rel="pb:operation">
   <div about="#update-op" typeof="pb:CRUDOperationUPDATE" />
   <fieldset>
    <legend>Issue</legend>
     <div rel="pb:field">
      <div about="#summary" typeof="pb:UpdateableField pb:RequiredField">
       <label for="summary.val" rel="pb:key" resource="#summary.key"
           property="dcterms:title">Summary</label>:
       <div rel="pb:value">
        <div about="#summary.val" typeof="pb:FieldValue">
         <input id="summary.val" type="text"
                property="rdf:value" content="" value="" size="40" />
        </div>
       </div>
      </div>
     </div>
   </fieldset>
</form>
```

Listing 1: Jira RDForm source code.

5.3 Wrapper

Most of the data on the Web to-date is based on formats such as XML, JSON, etc.[31], though linked data is acting as the backbone on the Web of Data, providing a common data model through RDF and a universal data space through the usage of URIs and HTTP. In order to enable non-RDF data sources (such as Atom feeds or Web services, etc.) to participate in the RDF-based Web of Data, we need to wrap them.

In the read-only case of a wrapper, we refer to the linked data publishing process[32]. Many read-wrapper are already available (URIBurner[33], etc.). In fact, linked data can be understood as a read-only RESTful architecture [RR07]. Equally, the majority of write-interfaces resides in site-specific API environments; examples of so called *site-specific APIs* range from bookmarks[34] to the wide range of Google products[35]. A leading API directory lists more than 1300 APIs at time of writing[36]. A 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? We propose *RDF write-wrapper*, which conceptually accept SPARQL Update as input and map to calls of site-specific API functions. We will give an example of an implementation in the next section.

[31] http://webofdata.wordpress.com/2009/07/20/what-else/#comment-132
[32] http://linkeddata.org/docs/how-to-publish
[33] http://linkeddata.uriburner.com/
[34] http://delicious.com/help/api
[35] http://code.google.com/apis/
[36] http://www.programmableweb.com/apis

6 Demonstrator

To demonstrate our concept of a read-write Web of Data, we have implemented a use case where calendar data is updated using the Google calendar API (Fig. 9).

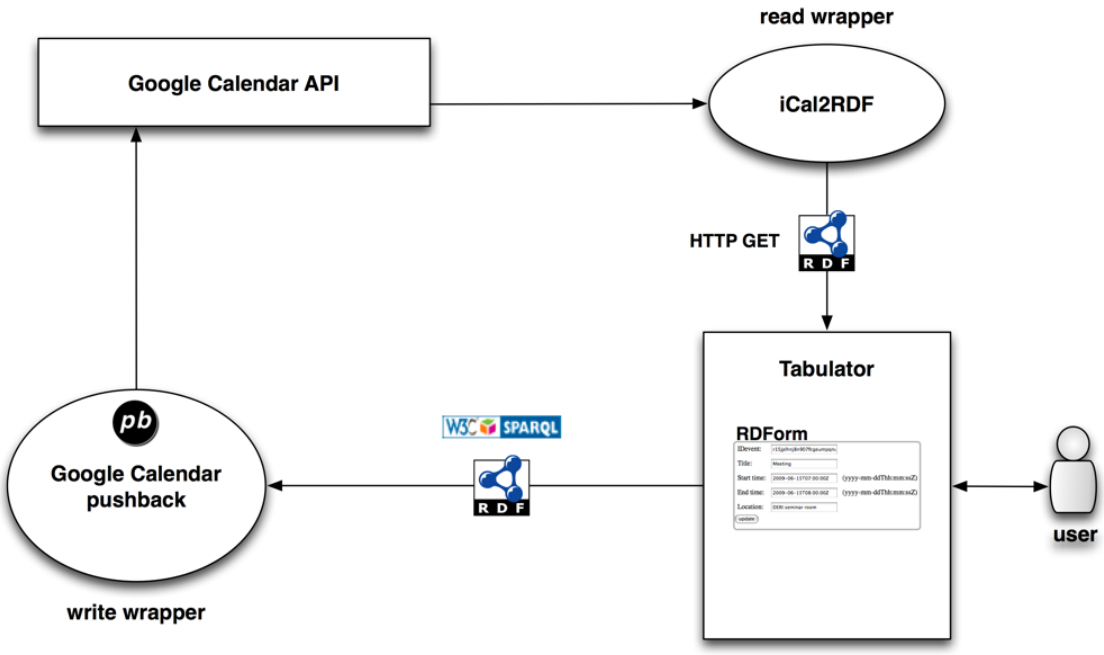


Figure 9: The Google calendar read-write Web of Data demo.

Tabulator can readily discover calendar data from RDF data sources such as FOAF files. Calendar data from non-RDF sources such as Google iCal files can be converted to RDF using simple tools[37] to enable Tabulator to browse the calendar data. Once such calendar data are available, the "Pushback Pane" on Tabulator (Fig. 10) can be used to modify the event data. The "Pushback Pane" implements an RDForm, which sends a SPARQL Update of the corresponding change to the pushback. The "Pushback Pane" is implemented in a modular manner, so that other API specific updates could be implemented easily.

On the server-side we have implemented a pushback that wraps the Google Calendar API. We have set up an ARC2[38] RDF triple store, which acts as a SPARQL endpoint towards the RDForm, caching the RDF triples sent from the RDForm. A server module polls the RDF store to detect changes and maps the RDForm fields to function calls of the Google calendar API. Note that the demonstrator is available at the Tabulator development version control system at `https://svn.csail.mit.edu/dig/2007/tab/` (username and password for the public account are "dig").

[37] By using tools such as ics2rdf, `http://torrez.us/ics2rdf/`.
[38] `http://arc.semsol.org/`

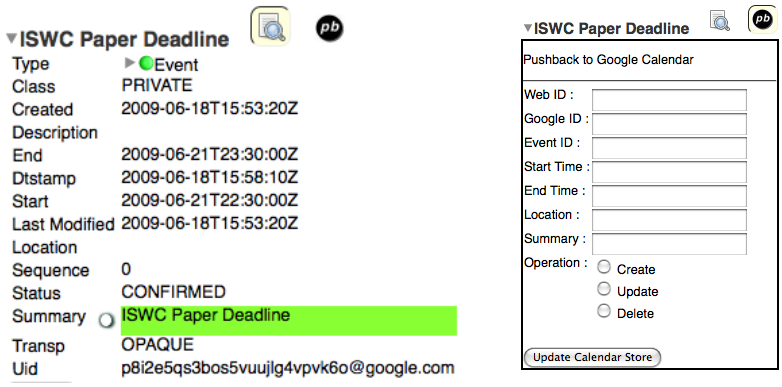


Figure 10: **Left**: Outline view in Tabulator displaying the calendar information. **Right**: "Pushback pane" to modify the calendar information and send updates to the pushback.

	HTML form/POST	**Tabulator single-triple**	**Tabulator RDForms**
		Architectural Properties	
1.1 AWWW	yes	yes	yes
1.2 RESTful	yes	no	no
1.3 Semantics	no	yes	yes
		Complexity	
2.1 Client	low	middle	middle
2.2 Server	low	middle	high
		Characteristics	
3.1 Agnostic	no	no	yes
3.2 Information Unit	yes	no	yes

Table 3: Comparison of the different update interfaces.

Table 3 is a comparison of the RDForm-based approach with a conventional HTML Form using HTTP POST and the original Tabulator Redux (single-triple edit), using following keys:

1. Architectural Properties
   (a) AWWW ... compliant to the *Architecture of the World Wide Web* [JW04]
   (b) RESTful ... compliant to the REST architectural style as of [FT02]
   (c) Semantics ... compliant to SPARQL Update semantics
2. Complexity
   (a) Client ... effort to develop client side of the implementation
   (b) Server ... effort to develop server side of the implementation
3. Characteristics
   (a) Agnostic ... data source agnostic (same interface for RDF and non-RDF back-ends)
   (b) Information Unit ... supports information units

7 Conclusion & Outlook

We have argued in this paper that we lack a general concept for a read-write Web of Data and motivated the need to understand how to create, update and delete RDF data in a secure, reliable, trustworthy and scalable way. Combining existing components such as FOAF+SSL and adding RDForms as well as pushbacks, we have proposed a concept of a read-write Web of Data along with a demonstrator for editing Google calendar entries.

After reviewing the available core components, we have, building on our experience with Tabulator Redux, introduced RDForms. Though, regarding integrity constraints, HTML forms offer support for information units, they do not convey explicit field semantics. With RDForms, however, we have proposed a solution to this issue, as one is able to use form-based editing of RDF graphs over a uniform interface (SPARQL Update). With SPARQL Update one has a fine-granular, efficient way to update RDF graphs. We hence understand, that the role that HTTP plays in the RESTful architecture [FT02] of the Web of Documents is filled by SPARQL Update in the Web of Data. In order to integrate the numerous site-specific APIs, we have introduced write-wrapper, called pushback.

While working on the demonstrator we have identified certain issues. It is a non-trivial task to harmonize and integrate the various, currently used authentication mechanisms. For example, using Google's authentication together with FOAF+SSL and WAC is subject to further research. Further, editing data sources may potentially happen in parallel. We need yet to clarify how to handle concurrency edits in our concept. Equally, to ensure data integrity, we might need to consider supporting transactions. Eventually, the error handling in case of a write operation has not yet been explicitly addressed in our concept; this issue should be resolved as the standardisation of SPARQL Update advances.

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). The authors thank Jürgen Umbrich, Krystian Samp, and Cosmin Basca for their valuable input.

References

[ABMP08] B. Adida, M. Birbeck, S. McCarron, and S. Pemberton. RDFa in XHTML: Syntax and Processing. W3C Recommendation 14 October 2008, W3C Semantic Web Deployment Working Group, 2008.

[BL] T. Berners-Lee. Linked Data, Design Issues. `http://www.w3.org/DesignIssues/LinkedData.html`.

[BLFM05] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. Request for Comments: 3986, January 2005, IETF Network Working Group, 2005.

[BLHL+08] T. Berners-Lee, J. Hollenbach, Kanghao 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.

[DHP08] 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.

[DKP+07] A. Dix, A. Katifori, A. Poggi, T. Catarci, Y. Ioannidis, G. Lepouras, and M. Mora. From Information to Interaction: in Pursuit of Task-centred Information Management. In *Second DELOS Conference on Digital Libraries*, Tirrenia, Italy, 2007.

[DS98] F. Dawson and D. Stenerson. Internet Calendaring and Scheduling Core Object Specification (iCalendar). Request for Comments: 2445, November 1998, IETF Network Working Group, 1998.

[Dus07] L. Dusseault. HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV). Request for Comments: 4918, June 2007, IETF Network Working Group, 2007.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Request for Comments: 2616, June 1999, IETF Network Working Group, 1999.

[FHBH+99] 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.

[FT02] R. Fielding and R. Taylor. Principled design of the modern Web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, 2002.

[GLA09] 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.

[GZC09] 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.

[Hau09] M. Hausenblas. Exploiting Linked Data To Build Web Applications. *IEEE Internet Computing*, 13(4):68–73, 2009.

[HFPS99] 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.

[Ink09] T. Inkster. Inav the Terrible. Buzzword.org.uk Draft 7 February 2009, buzzword.org.uk, 2009.

[JW04] I. Jacobs and N. Walsh. Architecture of the World Wide Web, Volume One. W3C Recommendation 15 December 2004, W3C Technical Architecture Group (TAG), 2004.

[KCM04] G. Klyne, J. J. Carroll, and B. McBride. RDF/XML Syntax Specification (Revised). W3C Recommendation, RDF Core Working Group, 2004.

[Pre09] J. Presbrey. RDF Policy-based URI Access Control for Content Authoring on the Social Semantic Web. Technical report, Massachusetts Institute of Technology, 2009.

[RR07] L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly Media, Inc., 2007.

[SHJJ09] 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.

[TAPH05] W. Tolone, G. Ahn, T. Pai, and S. Hong. Access control in collaborative systems. *ACM Comput. Surv.*, 37(1):29–41, 2005.

[Tul97] T. S. Tullis. *Screen design*, pages 503–531. Elsevier Science, 1997.

[Zha08] B. Zhao. Collaborative Access Control. *Telecommunications Software and Multimedia*, 2008.