Tabulator Redux: Browsing and Writing Linked Data

T Berners-Lee, J. Hollenbach, Kanghao Lu, J. Presbrey, E. Prud'ommeaux, mc schraefel

MIT CSAIL, Cambridge, MA, USA

Electronics and Computer Science, University of Southampton, UK

{timbl | eric | mc } @ csail.mit.edu

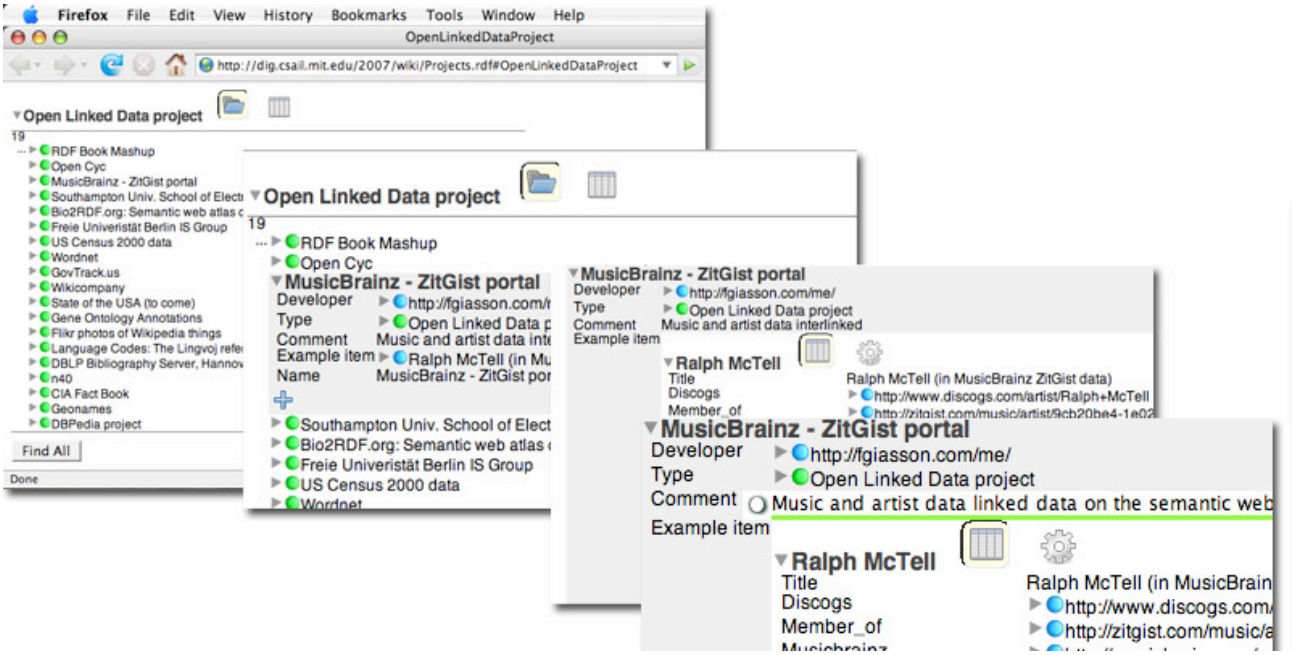

Figure 1. The Tabulator. The first frame shows the Tabulator with an RDF source, the Open Linked Data Project open. The second frame shows information within that source expanded, the third frame shows another source within that source expanded, and finally, the last frame shows that the label of that source has been edited from "Music and artist data interlinked" to "Music and artist data linked on the Semantic Web"

ABSTRACT

A first category of Semantic Web browsers was designed to present a given dataset (an RDF graph) for perusal in various forms. These include mSpace, Exhibit, and to a certain extent Haystack. A second category tackled mechanisms and display issues around presenting linked data gathered on the fly. These include Tabulator, Oink, Disco, Open Link Software's Data Browser, and Object Browser. The challenge of once that data is gathered, how might it be edited, extended and annotated has so far been left largely unaddressed. This is not surprising: there are a number of steep challenges for determining how to support editing information in the open web of linked data. These include the representation of both the web of documents and the web of things, and the relationships between them; ensuring the user is aware of and has control over the social context such as licensing and privacy of data being entered, and, on a web in which anyone can say anything about anything, helping the user intuitively select the things which they actually wish to see in a given situation. There is also the view update problem: the difficulty of reflecting user edits back through functions used to map web data to a screen presentation. In the latest version of the Tabulator project, described in this paper we have focused on providing the write side of the readable/writable web. Our approach has been to allow modification and addition of information naturally within the browsing interface, and to relay changes to the server triple by triple for least possible brittleness (there is no explicit 'save' operation). Challenges that remain include the propagation of changes by collaborators back to the interface to create a shared editing system. To support writing across (semantic) Web resources, our work has contributed several technologies, including a HTTP/SPARQL/Update-based protocol between an editor (or other system) and incrementally editable resources stored in an open source, world-writable 'data wiki'. This begins enabling the writable Semantic Web.

Classification

H.3.5 Online Information Services: Web Based Services

General Terms

Documentation, Performance, Design, Security, Human Factors.

Keywords

Tabulator, semantic web, read/write, provenance.

1. INTRODUCTION

While the Semantic Web has been developed much as a data integration technology for the last few years, it has lacked an essential element which the hypertext WWW had from the start: the immediate gratification for information providers of seeing the results of their efforts on a screen. The viral spread of the HTML web was largely powered by the process of seeing a web page, viewing the source, copying it with small changes, and then having one's own page to show off to others immediately. For the first few years, however, Semantic Web development focused on back-end technologies. Many large sources of Semantic Web data were largely consumed off-line, and not generally available to others. Worse still, off-line processing reduced the social pressure to use dereferencable URLs for Semantic Web identifiers, and to back them with useful, machine and human-redable web pages.

Recently, collections of offline or zipped RDF data have increasingly been replaced by Linked Data [3]. Linked Data is data using RDF technology that (i) uses HTTP URIs to denote things; (ii) provides useful information about a thing at that thing's URI; and (iii) includes in that information other Linked Data URIs.

The Tabulator [5] was originally written as a linked data browser (Figure 1), designed to navigate the web of links, without any domain-specific programing by the user or the information provider. It has the inherent knowledge of a few common global concepts, such as time and geographical location, to give it the power of typical Web 2.0 applications such as on-the-fly calendar and mapping mashups. Using the Tabulator, anyone publishing e.g. a personal FOAF [8] document can see their own information on the screen and follow links from it to the FOAF descriptions of their friends, not to mention their publications and projects. They become part of an open social network. Since the inception of the Tabulator project, a number of linked data projects [18] have emerged, including several similar data browsers: Oink [17], Open Link Software's Data Browser [20], and Object Browser [19].

While these developments have been satisfying, the authors were concerned that a major potential of the system was unimplemented: the web of things (i.e., the Semantic Web), like much of the web of documents, was a read-only web from the point of view of the user. Given the goal of making the web in general a read-write space, surely it is important that a linked data application allow editing as well as browsing. Adding write functionality, however, introduced a number of technical and user interaction design challenges.

One challenge, faced by the read-only Tabulator and exacerbated by the read-write requirement, is that the semantic web provide an extra level of abstraction -- the graph of connected things -- above the web of documents with which the web browser user is familiar. We refer to those features that complicate things by introducing dependencies or connections between otherwise clean architectural layers as "Level-breakers". We explain why they are needed to allow operation in both web spaces where necessary, for social reasons and for helpful error reports. Another challenge is to enable the user to express themselves with relationships and fields selected from a portion of a potentially unbounded web. Also, there is the View Update problem making it less than straightforward to understand what affect and on which RDF document is implied by a given user change to the display.

We will present and motivate these choices, and describe the design and the underlying network protocol and software architecture. We will describe a 'data wiki' space that allows remote editing, and the technology used to support it on the server side. We will then discuss plans for future work.

2. Writing as (Mainly) Editing

The Semantic Web is two structures, at different levels. There is a space, we call here the 'web', of directed, untyped links between documents, and there is a space we call here the 'graph', of directed, typed of relationships between the things described by the documents. The goal of the project is that the user of the interface should work effectively with co-workers by exploring, analyzing, and collaboratively co-authoring the shared graph of knowledge. We do this in a domain-independent way so that the tool can be used on new fields without programming.

2.1 The Web of documents vs Graph of things

In the Semantic Web, primarily, users read aggregated information in the graph, ignoring the fact that the data about them may have been assimilated from many sources, possibly with inference. The original tabulator experience [5] demonstrated that readers must also be able to determine the source documents, and so understand the provenance of the data (we use the term document, though the source may be the sort of thing more often referred to as a store, and may be accessed using SPARQL rather than a simple HTTP dereferencing; the same social aspects of the information apply in either case). The reader can then ask questions such as: Who wrote this? Who is maintaining it? Can I trust it? May I re-use it? and related social questions. These attributes follow from the source of the data. Just as, to trust a document on the web, one peeks at the domain name of the web site, so to trust a statement in the graph, one peeks at the URI of (and metadata about) the document.

This peeking between levels breaks the consistency of the user interface that would have been possible at a single homogeneous level. This level-breaking is also necessary to make errors understandable. Just as, when a web error occurs in a web browser, the user checks the URI and may check the network connectivity to the host, so the reader at the graph level must be able to understand what document or network operation produced an error. A strength of web browsers when compared with many distributed systems built of less familiar components, is that they allow the user to understand the nature of network errors. We therefore assumed that an editor of the graph must allow users to understand the nature of errors at the document level and below. One must be able to distinguish, for example, between data which is missing in a file, files which have syntax errors, and network errors which prevent us reading them at all.

The tabulator handles these breaks by representing the document layer by coloured balls near each concept as shown on left side of images in Figure 1. The color of the ball indicates the state (unfetched, fetching, ok, error) of documents holding information about the concept. Clicking or hovering over the balls provides more information, and a cogwheel 'under the hood' button provides access to details of HTTP transactions, parsing, etc in case the user needs to explore further. Likewise, a list of all sources is maintained in another window, and clicking on any fact (data field or table cell) causes the source of that fact to be highlighted.

2.2The Writing/Editing Process

When considering editing or writing RDF data, most people will have social concerns beyond and complimentary to those of

reading data. These concerns include who will make sure this data is stored persistently; who will be able to read it; who will they be allowed to re-use it, and if so under what terms. For example, when entering certain information, one must be aware of whether it will be part of a personal address book or a public resource. A challenge for an editing application is to ensure that these questions are answerable, without themselves distracting from the main purpose of editing existing or creating new data.

Though the graph a person may wish to edit by changing existing or adding new data is effectively an aggregation of many graphs from different sources, a simple design of a semantic web editor would be to allow the user to edit one graph at a time. This would obviate the need for connections between graphs and documents. Several single graph editors exist including RDFAuthor [25] and IsaViz [21]. We considered two ways to apply this working model. One was the model in which a given single document is selected for editing, and changes are only allowed to be made to that graph of that document. The interface becomes a single document editor, effectively like an HTML document editor such as Amaya in normal editing mode [2]. Another way is to allow the entire graph to be browsed in a read only mode, but annotations made on it and stored on a specific annotation document. This is like the Amaya browser operating in annotation mode [16]. Both modes are evidently useful, and will be considered for future work, but did not, we feel, meet the goal of allowing the user to operate at the abstract level of the giant global graph.

Neither single-graph solution allows the granularity necessary for the social questions of understanding the provenance and controlling the destiny of data; nor do they scale across a web where anyone must be able to buy, rent, borrow or be given storage space under all kinds of arrangements in an open market. We decided to allow users to edit data, even if derived from multiple sources, as simply as if it were a single graph, making changes to different documents throughout the web.

The interface to support this approach must therefore determine where in the web to store a user's addition to the graph. The algorithm we chose for deciding where to store a triple is as follows:

- When a triple is modified, the revised data is stored in place of the old.
- When a triple is added, it is stored in the same place as the triple immediately above it in the property/value list. Successive additions with the same subject will be consistently written to the same place.
- If a statement is added to an item which has no other statements, if it has a URI like x#y where x is the URI of an editable document, then the triple is added to that document.

In general when creating a new project from scratch, a user must be able to define a new data file and its social properties. Where a new data file is started, it must have well-defined properties. In many Web 2.0 sites, such as Facebook or Google Groups, the policies are set by that site. In general, our approach is that users should be both aware of the policy, but also able to create and select new ones. We cannot yet create policies in Tabulator, but people can select data sources that use particular policies such as creative commons [6] policies.

In this iteration of Semantic Web editing in tabulator, we have avoided the complexities of access control, and out of interest in the wiki model of open collaboration, we chose to open an experimental area of URI space as a form of data wiki. This is a space of data documents that anyone may edit as linked data using the Tabulator or compatible client. As a test site for Tabulator, for example, within the data wiki URI space, any URI starting with http://dig.csail.mit.edu/2007/wiki/ identifies a document that the server considers existent, though possibly empty. A fetch to a document which has not been previously stored returns an empty RDF document, flagged editable by an HTTP header. Any data added to such a document causes the actual file to be created to hold the data. Looking up, for example, http://dig.csail.mit.edu/2007/wiki/foo/fruit#A if http://dig.csail.mit.edu/2007/wiki/foo/fruit does not exist, will return no error, and an item 'Apple' with no data. Adding information about Apple, for instance, that it is a Class, would cause the directory foo and the file fruit to be created, and a triple `<http://dig.csail.mit.edu/2007/wiki/foo/fruit#apple> rdf:type rdfs:Class.` stored in it.

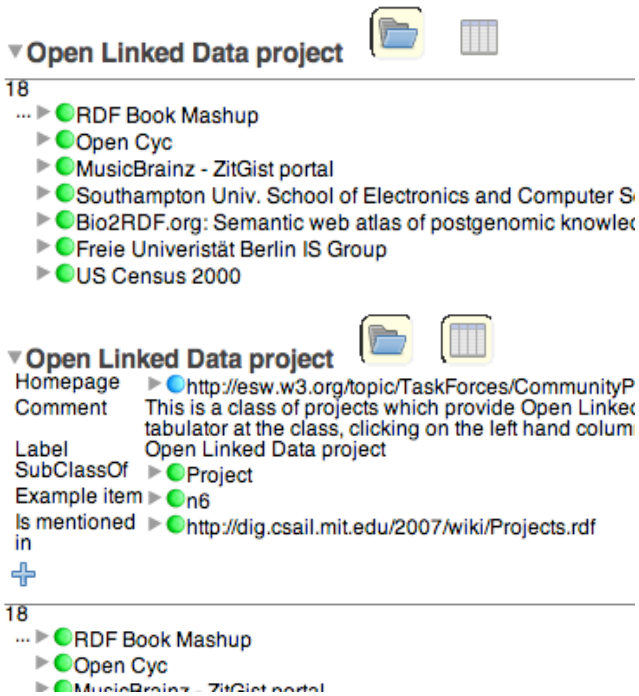

Figure 2. The membership pane (above) and properties pane (below) for a class.

3.TABULATOR INTERFACES

Reviewing the basic interfaces provided by the tabulator for editing, recall that, as described in [5] it is designed to support two interconnected user modes of operation: exploration, to see what information is available, and querying to gather similar subgraph patterns into tables similar to a spreadsheet table presentation for analysis. Exploration is done in a mode in which a given thing is presented using a table of predicate/object pairs. In the case that the object is something about which more is known, the user may recursively open a nested view of its property objects in turn. We refer to this nested hierarchical form as outline mode, by analogy with outline writing systems. This is strictly a tree view, but like many trees views, it is used for what is in fact a graph, and the same node can in principle be found more than once. The icons chosen mimic the (Mac OS X) nested directory interface, analogous to tree-like navigation aids in web sites which actually

have many cross-links, and hierarchical file systems which have soft links.

The user, then, explores sources by opening up related things, occasionally refocusing by restarting a new tree at any given point. The jump to analysis mode is made by selecting a number of fields in outline mode, and pressing a "Find All" button. The linked data graph is then searched for subgraphs matching the given fields. The results form a table, and, if geospatial or time coordinates are include in the columns, a map or a timeline respectively. The jump back is made by selecting any item in the analysis display and opening as a new outline mode display.

Note that whether exploring under user control in outline mode or performing a graph-matching query, the Tabulator store looks up the URIs of any objects which are opened in outline view, or matched as part of a subgraph matching algorithm. It also looks up any property and class, recursively, as ontologies help with inference and user interface. All the data retrieved in this process if kept in the local store.

The description of outline mode above is a slight simplification. In fact, at each level, various styles of predicate/object table may be available. These are called *panes*. If more than one is available then they are stacked vertically and each may be turned on an off by icon-decorated buttons. If only one is available, then no icons are shown (see Figure 2).

A class has a special pane to list instances. A document may have panes for inspecting the network transactions involved in fetching it, its human-readable content, or its RDF content reserialized. Other user interfaces for exploration used elsewhere include a circles-and-arrows graph (Isavix, Foafnaut, Object browser, etc), which tend to be insufficiently compact on the screen for practical quantities of data [14] and property linked predicate/object tables without outlining [17], which tabulator supports as a special case. The former could be used for selection of a subgraph query, whereas the latter could not as only the arcs from a given node are available on the screen at one time.

Other modes of analyzing similar datasets are many and varied, and include the faceted browser of mSpace [24], Longwell [13], Piggybank [10], Exhibit [9] slideshows, photo contact sheets, and multidimensional visualizations [26]. These styles could all be used just as well as the table, map and timeline modes of tabulator, could link back just as easily to other start new explorations, and indeed could be added as alternative views.

3.1Types of Editing

Three forms of editing are possible in outline mode: the modification of a object, the addition of a new object with an existing predicate, and the addition of a new predicate/object pair for an existing subject. Consider first the modification of an object cell that contains a literal value (non-string datatypes are not currently supported). Cell modification is done by clicking once, or pressing Return, when a cell is highlighted. The field becomes editable (Figure 3). Pressing return (etc) again causes the edit to be committed to the appropriate destination.

3.1.1Object Selection

If the object of the predicate/object pair in question is not a literal value but something identified by a URI, then it may be selected by name or by drag-and-drop. Following the goal of primarily enabling the user to stay at the knowledge level rather than the document level, URIs are not be shown nor does the user need to type them in. Whenever possible, the tabulator uses an appropriate name for something instead of its URI (specifically, any subproperty of rdfs:label is used, with preference for dc:title or foaf:name). To refer to something, the user can simply type in its name. An auto completion dialog box allows selection of the appropriate object without having to type the entire name. An alternative is to drag an object from any object the tabulator view, or the URI icon from any browser navigation bar or tabbed browsing tab. Note that in both these cases, the system must have already have seen the thing in question in some form. Various hacks allowed the expression of a URI explicitly if necessary, but in general the modus operandi is to first get both things visible somewhere before recording a relationship between them.

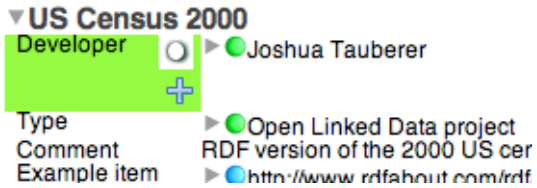

Figure 3 Addition of another developer. Selection of the predicate cell causes the plus button to appear.

A special item in the dialog box is "New...". This makes up a URI in the target document local namespace, one which the document does not use already. This creates a new nested property/object list, and the user is free to add more properties. Once a suitable name has been added to its properties, the generated URI is no longer visible. This creation of new nodes in a tree does mimic outline writing aids, as the user can chose to offload knowledge into the graph in any order, as it comes to mind Compare this to a "Wizard" system of cascading forms, for example, which forces a certain sequence.

An attempt is made to restrict the items in the dialog box to be those appropriate for a given situation. As the tabulator currently only has limited OWL inference, without disjoint classes, it is not easy to establish that, say, a given document is not a candidate as a friend of a person. In fact, we note, there are currently few ontologies such as FOAF, which declare classes as being disjoint with other classes in other ontologies.

Consider the addition of a new value to the predicate/object table, using the same predicate. When this is possible, when the source of the existing property/object statement is editable by the user, a blue plus sign shows in the predicate cell whenever it is selected. Clicking on this icon adds a new predicate/object pair, with the same predicate and an object selected by the user as above.

3.1.2Predicate Selection

Now consider the need to add a new fact to the property/object table, with a predicate not currently in the table. For this purpose, if there is an appropriate editable source, a blue plus is displayed on the left at the end of the whole table. Pressing this causes a new pair to be added, prompting with an auto-completion box for the predicate, and then selecting the object as above.

In object-oriented or frame-based systems, of course, there is a finite set of slots for any type of (software) object. This is not so in the Semantic Web, where RDFS and sometimes OWL constraints exist, but "Anyone can say anything about anything" remains effectively true at the user interface. The tabulator can prompt from a list of all the predicates it has encountered in the session, either in instance data or in ontologies. The user must explore enough to expose the tabulator session to see the

necessary predicates before using them to write. Often there is a large set of valid predicates. Further, some consider it bad form to use RDFS' domain and range constraints, preferring to OWL restrictions that for example the friend of a person should be a person, but not constraining a non-person from having a friend. This may lead to greater re-use of ontologies, but it also makes it more difficult to unclutter the interface. In future work, we would like to add inference to include awareness of disjoint classes.

An alternative design choice that we considered and, while unimplemented, is still appealing, is to select a similar object nearby in the graph and provide a form which prompts explicitly for those properties connected to the those objects. While the usermust always be able to escape into use of new predicates, much data is repetitions, so it is useful to optmize for its entry. In an address book, for example, one typically uses a small set of all the very many properties one could in principle record about a person.

3.1.3Editing in Table Mode

Recall that the table is formed by performing a query for a sub-graph pattern across the graph. Row insertion involves constructing a new subgraph which will match the query template. The destination store for each arc is copied from that of the arc for (arbitrarily) the last row in the table. Therefore, if a table is made from a join of several sources, they can all be updated by adding a new row. The operation of cell value editing, as in outline mode, involves removing a statement and inserting a replacement in the same document.

4.NETWORK PROTOCOL FOR WRITING

Driving the design of the network update protocol is the desire to create a web of editable resources, and to allow the user to naturally interact with the data. The user should not have to set up preferences such as 'up-load addresses' or 'publish location', which are very typical of web hosting services. A subgoal therefore was to make the system self-configuring. To this end, we send updates to the URI of the destination document itself. We use two protocols, the standard WevDav [28] (not completely implemented at time of writing) and a version of SPARQL/Update, the Semantic Web query language, extended to allow update. [1]

An HTTP server may advertise that a given document is editable by sending an HTTP header when the document was fetched. We noticed that servers supporting WebDAV authoring often send a non-standard header "MS-Author-Via: WebDAV". Feeling that one big pile was, as it were, better than two little ones, we adapted this to send "MS-Author-Via: SPARQL" to indicate that the server supports incremental update by SPARQL.

Other systems, such as the HTTP PUT method (like Amaya [2]) or the WebDAV protocol [28] also communicate using the URI from which the document was read. With these systems, though, a typical editing session involves more or less off-line editing, followed by an explicit save user action. This can result in lost data if the client system crashes or is closed down before the edits can be written back. While offline/sync systems such as IMAP clearly have their advantages when disconnected, we decided to implement a real-time online system with small change granularity. A user immersed in the community knowledge would ideally be allowed to directly update all the collaborator's screens; immediate update is a step towards this goal.

Tabulator's collaborative editing protocol is based on a server-side document store potentially shared by many clients following a strategy of optimistic concurrency. When any edited field loses user focus or is changed and deemed savable, Tabulator uses the URI of the 'appropriate destination' document to be edited as described above. It assembles an update message to send to the document's server. At this point, the modified field is grayed out, and locked for user input, so no conflicting changes can be made before the update process completes. This graying out also serves as feedback to the user that their changes are being saved. Tabulator submits these statements in the body of a POST request to the update URI. When an acknowledgment is received from the server (a "200 OK" HTTP response) confirming that the change has been made to the document, the edited field will unlock.

If on the other hand, an error occurs, the user is alerted with a dialog box requiring acknowledgment, and the change in the user interface is backed out. In a collaborative environment the error could be a user-level concurrency error that incompatible changes have been made by another client to the same document. However, network errors, server unavailability, and so on, may also have to be explained to the user. The update message, and un-graying of the field is performed asynchronously so that the user is free to perform more editing, possibly with several modifications pending server acknowledgment.

The protocol builds on HTTP and SPARQL with as few arbitrary design decisions as possible. It is hoped that the resulting protocol is largely uncontentious and will gain wide adoption. The convention of treating each document on a web server as a SPARQL endpoint is not typical ; most SPARQL endpoint access one large store, possibly containing many individual graphs from different files. Our design is, however, it is quite consistent with the SPARQL semantics. The extensions used for update, INSERT and DELETE, take a syntactic form based on the existing CONSTRUCT production, and so are not particularly novel. This update protocol design also inherits useful functionalities of HTTP implemented by the client browser. Document permissions can be implemented and access can be limited as specifically as for any other URI on the web, using the standard HTTP authentication mechanisms.

This is not perfect: it would be nice if they HTTP response distinguished between an empty document and a non-existent one, but we would have to have a way of saying that the 'Not Found' error was merely advisory during a write operation. It is not obvious how many hoops the user should be made to jump though to create a new file, whether just to reference it, or confirm their intentions, or specifically ask to create a new file with a given URI. HTTP PUT could of course be used for creating a new file, though our server does not currently support it.

This approach should be extended to a collaborative system: when concurrent editing results in a clash, the response form the server (or the peer-peer system) should be a series of patches (from other clients), which cause localdata to roll-back to a state consistent with the server. This roll-back has been implemented in principle, but not the patch distribution protocol.

[1] The update extension proposed in SPARUL and SPARQL/Update [19] is not standardized, but we we derive comfort from the fact that we successfully used the intersection of the two current proposals.

4.1 Current Implementation

As stated, to explore the social assumptions of a wiki at the graph level, we set up a sandbox for anyone to create new data by deploying a data wiki. Any RDF data file could be uploaded to the wiki, but of course it will be reserialized, losing any comment. The system is designed to integrate very smoothly with a filestore-based web server. The data is all stored in RDF files. Setting up a read/write access to an arbitrary file should not be complicated.

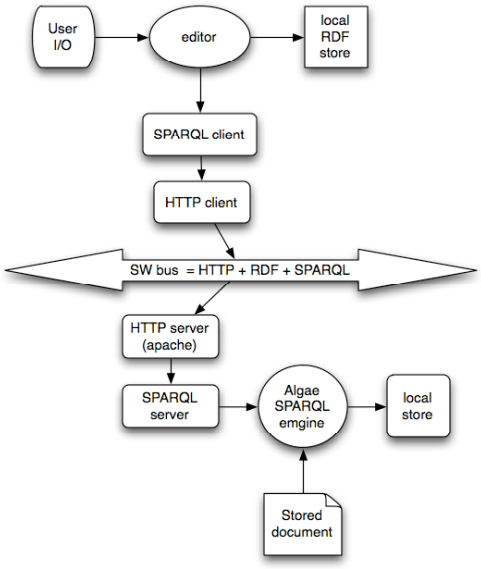

Figure 4. The client side is implemented in the asynchronous Javascript environment of a Firefox extension. A local provenance-aware triple store aches all RDF data seen in the session. When a change is made, the editor uses the SPARQL-Update client

In our implementation (Figure 4), we hold the data in each document in a file in the file system, represented in the data wiki. Since every update request is posted its respective document URI, the server trivially locates the destination of the update request, parses it, and attempts to apply the update. The DIG RDF wiki runs Apache and PHP that parses out the update payload. It instantiates an Algae [1] RDF store, which reads the file's contents, applies the update, and writes the file back to generate the document's revised edition.

5. Challenges, Future Work

While we have made good progress in enabling real-time editing of semantic web resources, a number of challenges remain that are part of our agenda for Tabulator, described below.

Browser integration. The integration of the tabulator data browser-editor and the Firefox browser posed some technical difficulties due to the assumptions that the Firefox design made. The Firefox browser assumes that one document is displayed in one window. As a matter of security, it makes sure that the URI in the bar always matches that of the page being shown. This user interface guarantee makes no sense when the URIs the user is interested in are those of things in the graph, not items in the web. This is one of the tensions between the user interfaces at the graph and web level.

Updating Information. There are many ways in which the existing implementation needs rounding out to have simply the power that a conventional application: the handling of datatypes, explicit or implicit; the implementation of offline working mode; update using WebDav for those who need to source editable RDF but have ISPs who do not support SPARQL (yet). The table view should have the facilities of a typical spreadsheet. All views should allow update, the map view and the time line view for example should allow the dragging of objects whose coordinates are editable. And so on.

Collaboration. Improving the collaborative aspects of the system could involve the subscription by clients to streams of and changes to any sources which currently affect the display seen by the user. Peer-peer distribution on differences for editing of data between local network neighbors without a common server would be another possibility.

Predicates. We discussed above the need for better selection of predicates and objects for user input. If the number of predicates could be cut down to something of order 10, then a form (as a tabulator pane) could be created for every new object, which would mimic typical applications more easily. Obviously, the provision of forms languages such as Xforms would allow tailored user input experience, but we wanted in this project to push the boundaries of what could be built up from ontologies, with forms seeming to emphasize the application domain boundaries which we had wished to dissolve.

Social Policy. In the longer term, we are interested in adding user interfaces for creating an awareness of policy, in adding workflow actions in the style **of** papertrail [4]**.**

User Interface (UI). The goal of Tabulator is to make it easy for non-semantic web specialists to be able to explore and now edit RDF data. To that end, how to communicate RDF graphs for querying and editing to such neophytes is non-trivial. Many approaches may be possible: present graph visualization like IsaViz or database style interfaces like Microsoft Access.

In Tabulator, we have leveraged two familiar models: (1) an outline style of interaction to enable information points to be expanded or collapsed on demand, and (2) form editing similar to an address book applications where existing fields can be edited or new instances of a field added, like pressing a plus sign to add a new Work phone number. This hybrid approach of Outliner + Field Editor has let us share a prototype for exploring both requirements elicitation for the user interface and for the back end protocols to support the interaction.

We do not claim that these UI approaches are the optimal interface for exploring and editing RDF data. These approaches do however provide a basis for exploring the *implementation* of the concepts we have described here. We look forward to using the findings from this work to develop a variety of UI prototypes in the near future for effective usability design.

One of the key advantages of the Semantic Web approach is that once we have the data and the protocols, a variety of interfaces can be applied to these data sets. Likewise, we encourage interaction designers to leverage our back end work to support innovative front end designs for exploration and editing.

Longer term developments. In the future, we plan to address the prompt update of all users' displays when one user changes the data, to make collaboration clearer. This will require changes to the network protocols, and an upgrade of the local store to a full

Truth Maintenance System. We would like to allow system sheets, possibly in the style of Fresnel (but for editing) to define forms (tabulator panes) appropriate to different data patterns.

6. Conclusion

Recent years have seen an explosion in user-generated content on the web, which can be divided into two categories. On the one hand, the blogs and wikis are human-readable content which thrive by being linked together globally. On the other hand are the social networking sites, where users add relationships between people, but where linking is only site-wide. We set a goal to create an editable data space not limited to a particular domain (not just friends, photos or events), and linked across domains, to break it open into a globally linked system linked across websites; to make it collaboratively editable as a shared store of knowledge and thus to bring about a step change in the power of an individual.

We have shown that live semantic web editor is a non-trivial design challenge, but capable of providing a collaborative editing environment in at a level of abstraction above that of the web of documents: the graph of things. Though the Tabulator prototype lacks some usability features and polish, it demonstrates the feasibility of direct editing of semantic web data across multiple servers and interconnected domains of discourse. It does this adapting many familiar interface metaphors from current hum interface practice. Unlike in object oriented and frame-oriented system, there is no fixed set of slots for each object for the user to fill in. There are no forms: instead, we explored the balance between ontology and existing data to help guide the user when adding more data. Just as semantic web readers need to be aware of the provenance of the data they read, and its social implications, so writers must be aware of the destiny of the data they write - and its social implications.

The system works. Its greatest value we feel is as a basis for other things. We encourage others to experiment with different styles of client and of server built to the same HTTP/SPARQL network protocol. We hope to tackle many of the large set of request for enhancement. A hope is that it will become sufficiently intuitive for, say, a spreadsheet user to use effectively. Already at this stage, though, we feel that the feasibility of this architecture has been conclusively demonstrated. We have resolved a number of design questions. We have created an application-independent architecture in which application-specific features can be smoothly blended. We demonstrate that there is no good reason why the semantic web should not be collaboratively writable, such that the fusion of the ideas of humanity and machine-processable knowledge of machines becomes ever closer.

7. ACKNOWLEDGMENTS

This work has been supported by Nokia Research Center Cambridge, MIT/CSAIL's UROP Summer Student Program, and by a Royal Academy of Engineering Global Research Award.

8.REFERENCES

[1] Algae How To. http://www.w3.org/1999/02/26-modules/User/Algae-HOWTO.html

[2] Amaya. http://www.w3.org/Amaya/

[3] Berners-Lee, T. Linked Data. http://www.w3.org/DesignIssues/LinkedData

[4] Berners-Lee, T. PaperTrail http://www.w3.org/DesignIssues/PaperTrail.

[5] Berners-Lee, T. Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., Lerer, A., Sheets, D. Tabulator: Exploring and Analyzing linked data on the Semantic Web. SWUI06 Workshop at ISWC06, Athens, Georgia.

[6] Creative Commons. http://creativecommons.org/

[7] Cunningham, Ward and Leuf, Bo (2001): The Wiki Way. Quick Collaboration on the Web. Addison-Wesley.

[8] Friend of a Friend. http://www.foaf-project.org/.

[9] Huynh, D. Exhibit http://simile.mit.edu/exhibit/.

[10] Huynh, D., Mazzocchi, S., Karger, David. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. International Semantic Web Conference (ISWC) 2005.

[11] Kagal, L, Berners-Lee, T., Connolly, D., Weitzner, D. Using Semantic Web Technologies for Policy Management on the web. AAAI 2006.

[12] Kagal, L, Berners-Lee, T., Connolly, D., Weitzner, D. Self-describing Delegation Networks for the Web, IEEE Workshop on Policy for Distributed Systems and Networks (POLICY 2006).

[13] Karger, David R., Bakshi, K., Huynh, D., Quan, D., and Sinha, V. Haystack: A General Purpose Information Management Tool for End Users of Semistructured Data. Conference on Innovative Database Research (CIDR) , 2005: 13--26.

[14] Karger, D. and schraefel, m.c.. The Pathetic Fallacy of RDF. SWUI06 Workshop at ISWC06, Athens, Georgia.

[15] Kolovski, V, Katz, Y, Hendler, J., Weitzner, D. Berners-Lee, T. Towards a Policy-Aware Web},The Semantic Web and Policy Workshop at ISWC,2005.

[16] Koivunen, M., Swick, R., and Prud'hommeaux, E. (2003) Annotea shared bookmarks. KCAP 2003 Knowledge Markup and Semantic Annotation workshop. http://www.w3.org/2001/Annotea/Papers/KCAP03/annoteabm.

[17] Lassila, O: "Browsing the Semantic Web", 17th International Conference on Database and Expert Systems Applications (DEXA'06), 5th International Workshop on Web Semantics, pp.365-369, Krakow (Poland), September 2006.

[18] Linked Data Project. http://linkeddata.org.

[19] Object Browser. http://webseitz.fluxent.com/wiki/ObjectBrowser.

[20] Open Link Software's Data Browser. http://demo.openlinksw.com/DAV/JS/rdfbrowser/index.html.

[21] Pietriga, E. IsaViz. http://www.w3.org/2001/11/IsaViz/.

[22] Prud'hommeaux,E., Seaborne, A., eds, SPARQL Query Language for RDF http://www.w3.org/TR/rdf-sparql-query/.

[23] Seaborne, A., Manjunath, G. SPARQL/Update: A Language for Updating RDF Graphs. Version2: 2007-08-09. http://jena.hpl.hp.com/~afs/SPARQL-Update.html.

[24] schraefel, m. c., Smith, D. a., Owens, A., Russell, A., Harris, C. and Wilson, M. L. (2005) The evolving mSpace platform:

leveraging the Semantic Web on the Trail of the Memex. In Proceedings of Hypertext, 2005, Salzburg.

[25] Steer, D. RDFAuthor. http://rdfweb.org/people/damian/RDFAuthor/

[26] Tufte, Edward R. Envisioning Information. Graphics Press, Michigan, USA, 1990.

[27] W3C ACL System. http://www.w3.org/2001/04/20-ACLs.

[28] Whitehead, Jr., E. J. World Wide Web Distributed Authoring and Versioning (WEBDAV) -- An Introduction. ACM StandardView, Vol 5., No. 1, March 1997, p. 3-8.

[29] Weitzner, D., Hendler, J., Berners-Lee, T.. Connolly, T. Creating a policy-aware web: Discretionary, rule-based access for the world wide web. Web and Information Security, Elena Ferrari and Bhavani Thuraisingham, eds, IRM Press, 2006.