Provenance of Inferences: A Trail of Proofs in Linked Open Data

Kanghao Lu, Oshani Seneviratne, and Tim Berners-Lee

MIT CSAIL, Cambridge
Massachusetts, USA
`( kennylu | oshani | timbl )@csail.mit.edu`

Abstract. With the recent uptake on the linked open data on the Web, there are many interlinked data sets on the Web today. Interesting results that might not be obvious from this *raw* data itself, can be derived by computing answers to rules using the data and essentially connecting the dots to see the big picture. We describe a novel method to crawl the Web of data and make inferences that satisfy rules of a particular domain and to re-inject the derived conclusions, along with their proofs in a proof ontology, back into the read-write Web. We have developed a "Computation Agent" to do this, using rules written in a formal Semantic Web rule language, and a user interface for viewing data along with their proofs that give the justification for that particular data in an intuitive manner.

1 Introduction

The linked open data cloud has seen a steady growth since DBPedia [2] project started in 2007. The DBpedia knowledge base alone currently describes more than 2.6 million things, and the knowledge base consists of 274 million pieces of information expressed in RDF triples [1]. Some Weblog hosting sites and social networking sites have already begun exporting user profiles in to RDF [10] using the Friend-Of-A-Friend (FOAF) vocabulary [9]. Many other data sets have linked up to this nucleus of DBpedia data set, and to each other, effectively increasing the intelligence of the Web and supporting large scale data integration. This allows people to use data browsers such as Tabulator [20] to surf across silos of data that have been converted into linked data.

Although much of the linked data cloud is interlinked there are many data sets that have no explicit semantic associations with each other even though these relationships might be obvious after a simple reasoning process. Once these relationships are exposed through a computation mechanism, the newly found relationships should be contributed back to the linked data cloud. This will allow reuse of these derived facts, without having to reason over the entire graph of data again. For someone who may be interested in knowing how a particular fact is derived, a proof tree detailing the derivation steps could be attached with the fact to enable provenance [1].

For example, in Fig 1, we have a simple linked data graph. There are several different nodes (encoded A - J), and different kinds of relations between these nodes and beyond. (One could assume that node A corresponds to the WebID of Alice, node B corresponds to the WebID of Bob, and the edge between A and B, i.e. 'rel:AB' corresponds to 'foaf:knows'. Likewise, other nodes and edges in this graph can be any other entity and relationship exposed as linked data). By merely looking at this graph of data, the relationship between the triple $A \rightarrow E$ and $H \rightarrow K$ may not be obvious at a first glance. But after the application of a specific rule we might discover that there is a relationship between these two triples. Assume that the derivation path of this fact is as outlined in the red arrows where new relationships between triples are created. It would be useful to make these new relationships on the linked open data cloud persistent, so that somebody else might be able to reuse this fact later. However, to reinforce the trustworthiness of the fact that $A \rightarrow E$ derives $H \rightarrow K$ we attach the derivation steps of the proof to it.

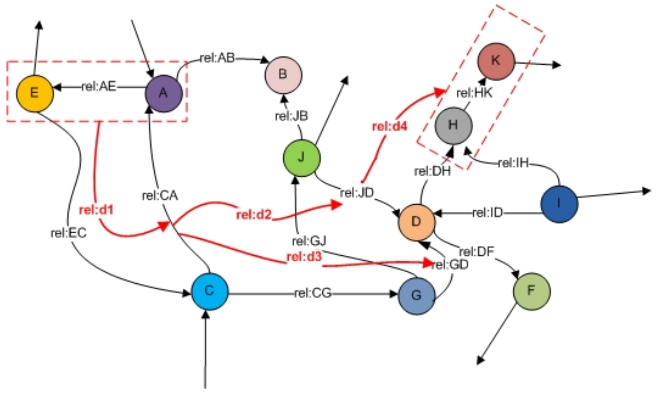


Fig. 1. Computations on a Linked Data Cloud (*Starting from the triple* $\{A$ *rel:AE* $E\}$ *we derive the triple* $\{H$ *rel:HK* $K\}$ *using the path outlined in red.*)

The recommended method of querying RDF, and hence extracting data from linked open data is by using SPARQL queries [16]. The SPARQL protocol [7] offers a standard way to serve and access structured data on the Semantic Web. However, SPARQL queries are designed to give an answer based on a particular data set at a given end point. We envision a slightly different system where answers to specific queries are computed using a *crawler based* "Computation Agent". These answers are updated using the SPARUL protocol [19] along with the method of derivation in the form of a proof using a specific attachment

[1] The impatient reader is encouraged to glance through [13] or to take a look at the implementation of the User Interface given in Section 4.5.

mechanism. This "Computation Agent" will act on the given rule written in the N3 (Notation 3) rule language [4] using any given URI as a seed to start the process of crawling the linked data cloud. Once it finds information that matches the criteria specified in the rule, it will update the derived facts from the computation and the proofs associated with these facts. In summary, this project has the following components:

- **Computation Agent** that crawls looking for data, makes deductions and sends the result along with the proof
- **Light weight Server** that accepts SPARQL Update + Proof
- **User Interface** for viewing proofs

The remainder of the paper is organized as follows: In Section 2 we describe the motivation behind this project and the desired features of such a system. Section 3 illustrates few use cases that could use a computation such as the one we have implemented. Section 4 outlines the protocols used in the implementation. Section 5 describes several related work from the Semantic Web world and from outside this area, especially where the inspiration for this system was drawn from. Section 6 discusses several future directions for this project. Finally in section 7 we provide a summary and outlook of the overall system.

2 Motivation

We find inspiration for such a system to provide solutions for maintaining the data web, particularly in creating new data, updating existing data and deleting unwanted data. We also believe attaching a proof with the derived result will make the fact more trustworthy, and hence give it a relative importance based on the number of proofs supporting the same derived fact.

2.1 Maintenance of the Data Web

Creating New Data and Updating Existing Data: The general mechanism in use for this is to use SPARQL Update [19]. After new triples are generated, the 'Computing Agent' sends SPARQL Update queries to 'Publication Agents', such as Web servers or SPARQL endpoints [2]. A 'Publication Agent' accepts this update query and modifies the corresponding RDF document or deny the request for update. It is relatively easy to modify pure RDF documents such as RDF/XML or RDF/N3 documents with the help of a decent serializer. But if the publication agent does not have write access, or if the documents in question are in RDFa or XML+GRDDL formats [3], the Publication Agent could send a notification to the person in charge making a change request.

[2] See section 4.3 for the technical details as to how this was achieved.

[3] It's relatively hard to programmatically find the correct piece to modify in RDFa or in XML+GRDDL documents.

Deleting Unwanted Data In general, we can treat the number of proofs of a triple to be the reference count of a triple, and do reference count-based garbage collection. That is, if a triple is deleted, all the proofs which has the triple as one of its antecedents should be removed. If all the proofs of a triple are removed, we delete this triple. In other words, if this triple is no longer supported by any evidence we delete it. Therefore, proofs or provenance data can help agents induce chained deletion.

2.2 Valuing Information

Attaching proofs to a triple gives rise to a naive way of giving relative weights to the triples. I.e. by counting the number of proofs supporting that triple we can deduce the relative importance of the triple. For example, in Fig 4, we can see that the statements like "Illaria Liccardi is a Tabulator developer" and "Albert Au Yeung is a Tabulator developer" are no longer supported with proofs, which suggests that they may be previous developers. A user who has a problem using Tabulator might not want to consult Illaria or Albert, but instead someone with a higher proof count such as KangHao Lu. We also experimented with a very controversial and a subjective example, in which we deduce *smart people* by using several criteria. As a matter of practical use, if an employer wants to select the smartest people from a group, she can use our user interface and select the one with the highest number evidence supporting the fact. Another scenario is when there are contradictory information. For example, if a video on *YouTube* has x number of 'good' ratings and y number of 'bad' ratings, then a user can naively compare the ratings and get a sense for the value of that video. Similarly with triples attached with contradictory proofs, we can get an estimate for the importance of the proof by naively comparing the respective 'good' and 'bad' proof counts.

2.3 Desired Features

Some of the desired features of the "Computing Agent" are as follows:

Never Ending Crawler: It needs to crawl data, generate new information, and update those information with proofs as long as data that satisfies the criteria are available. We imagine that in the future, people write opinions in machine readable rules. Then for example, a never ending crawler+updater would spread opinions of a person to places she doesn't even know. Notice that in this sense, crawlers+updaters are just like spamming programs. But since these crawlers+updaters also send proofs along with the data, we propose the idea of proof-based content filtering, as described in section 6.

Support Generalized Rules When the rules are general, it can be reused by others with very few modifications. For example, the rule mentioned in section 4.5, can be used to give notifications to all the absent-minded people all over the world.

Multilingual Information All the information updated by this agent, including the proofs, are in RDF. Hence it is readable by people who speak different languages. This is because at the UI level, the *rdfs:label* with the correct language tag of the URIs could be displayed instead of one specific language.

Persistence of Derived Results One of the major objectives of this system is that the results are made persistent for later reuse. By making these derived conclusions persistent on a read-enabled Web, we are extending the knowledge of the world. More importantly this prevents reinventing the wheel, which saves time and effort.

Provenance Preservation Provenance is an important metric to get an idea about where a piece of information came from, the stakeholders involved in generating that piece of information and the trustworthiness of the component data items used in deriving it. The proof entails where the data came from. Therefore, by attaching a proof to a particular data item we are preserving provenance.

3 Use Cases

Much of the adoption of the Semantic Web has resulted from a strong need to be able to integrate and analyze data across databases, applications, and communities. It has been fascinating to witness the breadth of solutions that have been implemented to meet these needs. In this section we outline several use cases that portray the utility of the Computation Agent that we propose.

3.1 Support Propagation of Changes in Facts

Rules would prove to be very useful if the pieces of data are cached or mirrored at several different places, and these rules are applied on such data automatically. For example, there used be a period when the de-facto content type of N3 was *text/rdf+n3*. After some discussion, it was changed to *text/n3*. This became confusing for new developers, as some user documentation still says the content type of N3 is *text/rdf+n3*. If the N3 format has been given a URI and the triple {*<N3 URI> format:content-type "text/rdf+n3"*} is stored with the URI and all the pages are decorated with RDFa, then, when the information changes to *<N3 URI> format:content-type "text/n3'*, running the above rule on such incorrect Web pages will identify the errors, update where possible, or notify the owners for updates for documents which it cannot update.

3.2 Biomedical Data Analysis

Diverse types of data about drugs, patients, diseases, proteins, cells, pathways, and so on are already being effectively integrated on the linked open data cloud.

Biomedical researchers often need to be able to ask questions that span many heterogeneous data sources in order to make well-informed decisions that may lead to important scientific breakthroughs. Once an important relationship has been found it would be very useful to other researchers to look for the result only without having to go through the burden of computing it from the beginning. Say for example, that a biomedical researcher finds out that a very specific symptom exhibited by patient X is subdued by a drug administered to him over a given period of time in a specific manner. This leads to the conclusion that the drug has an improved effectivity in fighting a certain disease. This fact obviously has some great benefit to other biomedical researchers. Of course one can issue a SPARQL query to obtain this same piece of information. However, whenever a researcher finds this fact, if he were to make an assertion and create some new data reflecting this discovery, it would relieve the others from the burden of doing the same thing over and over again. Also, for the skeptics of the effectiveness of this new derived fact of the drug, the derivation tree should be available with the data as a form of reinforcement.

4 System Design and Implementation

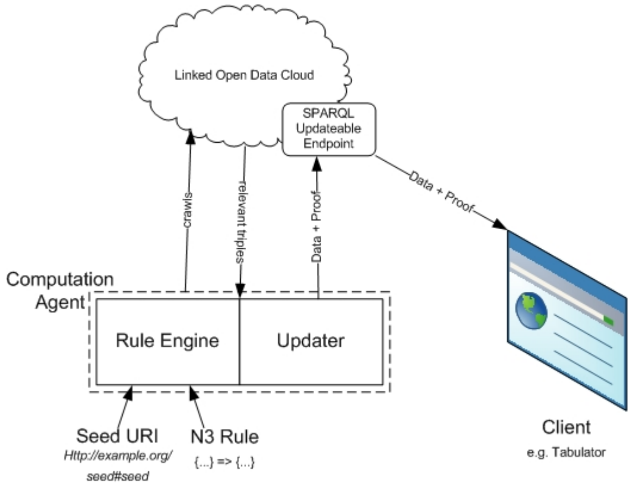


Fig. 2. System Design

The design of the current implementation is given in Fig 2. The 'Computation Agent' includes a 'Rule Engine' based on CWM (Closed World Machine) [5] and an 'Updater' which supports SPARQL updates. The User Interface is implemented on the Tabulator.

4.1 The Proof Vocabulary and Structure

Proof visualization relies upon the proof vocabulary which follows from the 'Truth Maintenance System *(tms)*' vocabulary developed by the TAMI project [4]. In order for Tabulator to understand the proof to render we use the following proof structure. This is highly customizable, and we envision support for languages such as Proof Markup Language (PML) [8] in the future.

```
@prefix tms: <http://dig.csail.mit.edu/TAMI/2007/amord/tms#> .
@prefix doap: <http://usefulinc.com/ns/doap#> .
@prefix doac: <http://ramonantonio.net/doac/0.1/#> .
@prefix swe: <http://dig.csail.mit.edu/2009/06/software_engineering#> .
@prefix ken: <http://dig.csail.mit.edu/People/kennyluck#> .

{ ken:I doac:skill "Javascript" . } #subject, the conclusion in formula
   tms:justification #predicate
   [ #object, a bnode
      tms:antecedent-expr  [
         a tms:And-justification; #a bnode of And-justification
         tms:sub-expr swe#doap2doac-skill , #the ruleURI is optional
            { #the antecedents in formula
               <http://dig.csail.mit.edu/2005/ajar/ajaw/data#Tabulator>
                  doap:developer ken:I;
               doap:programming-language "Javascript" .
            }
      ]
   ] .
```

The proof given above constitutes two formulae, namely the conclusion and the antecedent, two blank nodes, and effectively 5 triples. This is written using Turtle+Formula syntax, which is a subset of the full N3 allowing formulae or graphs, to be subjects or objects [5]. The client store of the proof visualization UI is required to support formulae as subjects or objects, but some of the full N3 constructs such as variables and quantification are not needed.

4.2 The Proof Update Protocol for the Computing Agent

The update protocol is adapted from the SPARQL Update protocol as described in the Tabulator-Redux [20]. This includes sending SPARQL queries via HTTP

[4] The current proof vocabulary may be revised in the future to make the proofs less verbose, and more human readable. However, even if the vocabulary changes, the changes expected in our system are at a bare minimum, and we expect the client implementation to be backwards compatible. (Project URI: `http://dig.csail.mit.edu/TAMI/`)

[5] As specified in the Design Issues Note for N3. See `http://www.w3.org/DesignIssues/diagrams/n3/venn`

POST with the MIME type “application/sparql-query”. For example, the following code snippet illustrates the update of the triple which says that the person identified by the URI `<http://dig.csail.mit.edu/People/kennyluck#I>` has Javascript skills.

```
POST /path-of-updatable-document HTTP/1.1
Content-Type: application/sparql-query

PREFIX ken <http://dig.csail.mit.edu/People/kennyluck#> .
PREFIX doac <http://ramonantonio.net/doac/0.1/#> .
INSERT { ken:I doac:skill "Javascript" . }
```

A consistent extension of SPARQL Update should allow subjects and objects to be graphs or formulae [6]. But SPARQL Update is not standardized and widely adopted yet. Therefore, we chose the following workaround in order to reduce the amount of work on the SPARQL Update server by introducing an additional HTTP Header called ‘X-Meta’:

```
POST /path-of-updatable-document HTTP/1.1
Content-Type: application/sparql-query
X-Meta: http://dig.cail.mit.edu/proof-tmp-storage/3deDQp

PREFIX ken <http://dig.csail.mit.edu/People/kennyluck#> .
PREFIX doac <http://ramonantonio.net/doac/0.1/#> .
INSERT { ken:I doac:skill "Javascript" . }
```

This means that we give the SPARQL Update server a URI link to the generated temporary proof. Notice that one disadvantage of this is that the computing agent has to have the power to write on the Web. As a multi-threaded application, the agent sends the data along with a pointer to the proof as long as a deduction is made. In the mean time, the crawling threads will be continuously fetching more data on the Web.

4.3 Providing Proofs from a Server and Retrieving Proofs from a Client

The current proof providing mechanism uses the standard HTTP content negotiation protocols. If the client store has the ability to store triples with formula subject or object, when fetching a document, the client's HTTP Accept header contains the content type representing the N3 format, namely *text/n3*, the server sends a big document containing proofs of some of the information in the document. If the client store only does flat RDF, the server sends a small document, with all the proofs removed from the big document. The proposed mechanism of proof transfer uses extra HTTP header as follows:

[6] See Section 6 for a more comprehensive discussion on this.

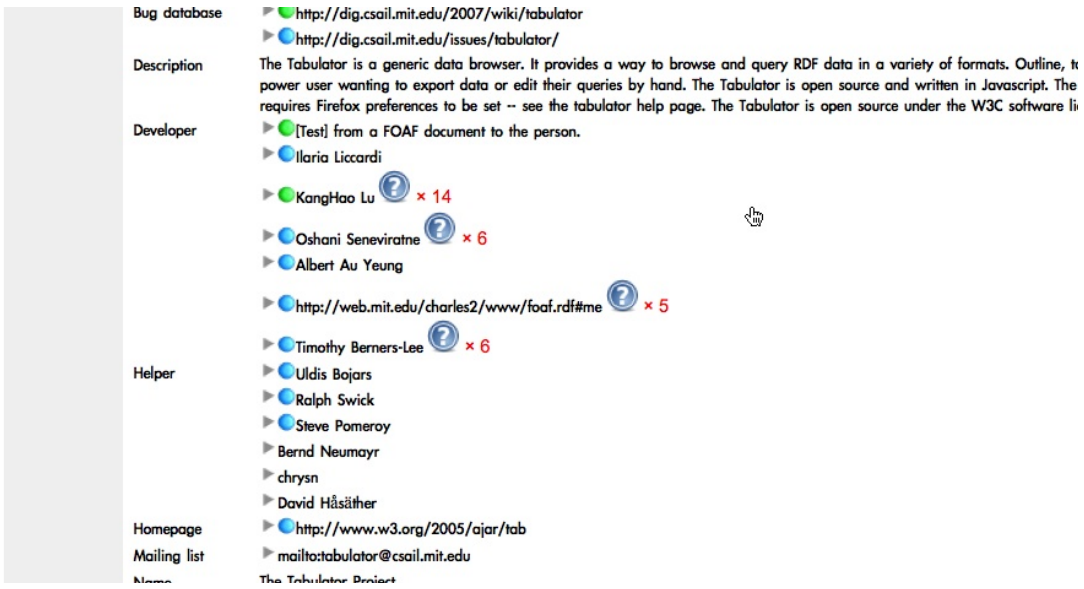
```
GET /data-file HTTP/1.1

HTTP/1.1 200 OK
Link: http://dig.csail.mit.edu/meta-file-for-data-file

<data-file-content>
```

The ‘Link’ header points to a meta file of the data file containing the information about the data file, and the client can fetch the meta file afterwards. The advantage of this approach is that it preserves the integrity of the data file. Notice that the Link header is not yet standardized and as far as we know, no existing HTTP server uses this mechanism yet. It is also not implemented in our system yet.

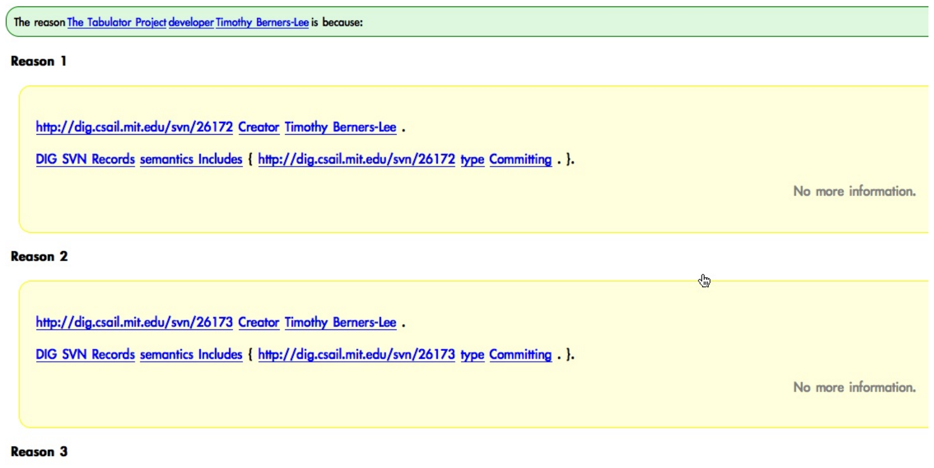
4.4 User Interface



Bug database	http://dig.csail.mit.edu/2007/wiki/tabulator
	http://dig.csail.mit.edu/issues/tabulator/
Description	The Tabulator is a generic data browser. It provides a way to browse and query RDF data in a variety of formats. Outline, t power user wanting to export data or edit their queries by hand. The Tabulator is open source and written in Javascript. The requires Firefox preferences to be set -- see the tabulator help page. The Tabulator is open source under the W3C software li
Developer	[Test] from a FOAF document to the person.
	Ilaria Liccardi
	KangHao Lu ? × 14
	Oshani Seneviratne ? × 6
	Albert Au Yeung
	http://web.mit.edu/charles2/www/foaf.rdf#me ? × 5
	Timothy Berners-Lee ? × 6
Helper	Uldis Bojars
	Ralph Swick
	Steve Pomeroy
	Bernd Neumayr
	chrysn
	David Håsäther
Homepage	http://www.w3.org/2005/ajar/tab
Mailing list	mailto:tabulator@csail.mit.edu
Name	The Tabulator Project

Fig. 3. User Interface Showing the Existence of Proof

Simple Proofs Fig 3 shows what the user interface of the system looks like on Tabulator. The ‘question mark’ symbol indicates that there is a proof associated with the corresponding triple. It is expected that the user will click on this icon and explore the proof using Tabulator’s ‘Justification UI’. The x n next to the question mark symbolizes how many proofs that particular triple has. Higher the number, more proofs supporting this particular triple, and hence more trustworthy the fact represented by the triple becomes.



The reason The Tabulator Project developer Timothy Berners-Lee is because:

Reason 1

http://dig.csail.mit.edu/svn/26172 Creator Timothy Berners-Lee .

DIG SVN Records semantics Includes { http://dig.csail.mit.edu/svn/26172 type Committing . }.

No more information.

Reason 2

http://dig.csail.mit.edu/svn/26173 Creator Timothy Berners-Lee .

DIG SVN Records semantics Includes { http://dig.csail.mit.edu/svn/26173 type Committing . }.

No more information.

Reason 3

Fig. 4. User Interface on Tabulator Showing the Reasons behind the Conclusion Given

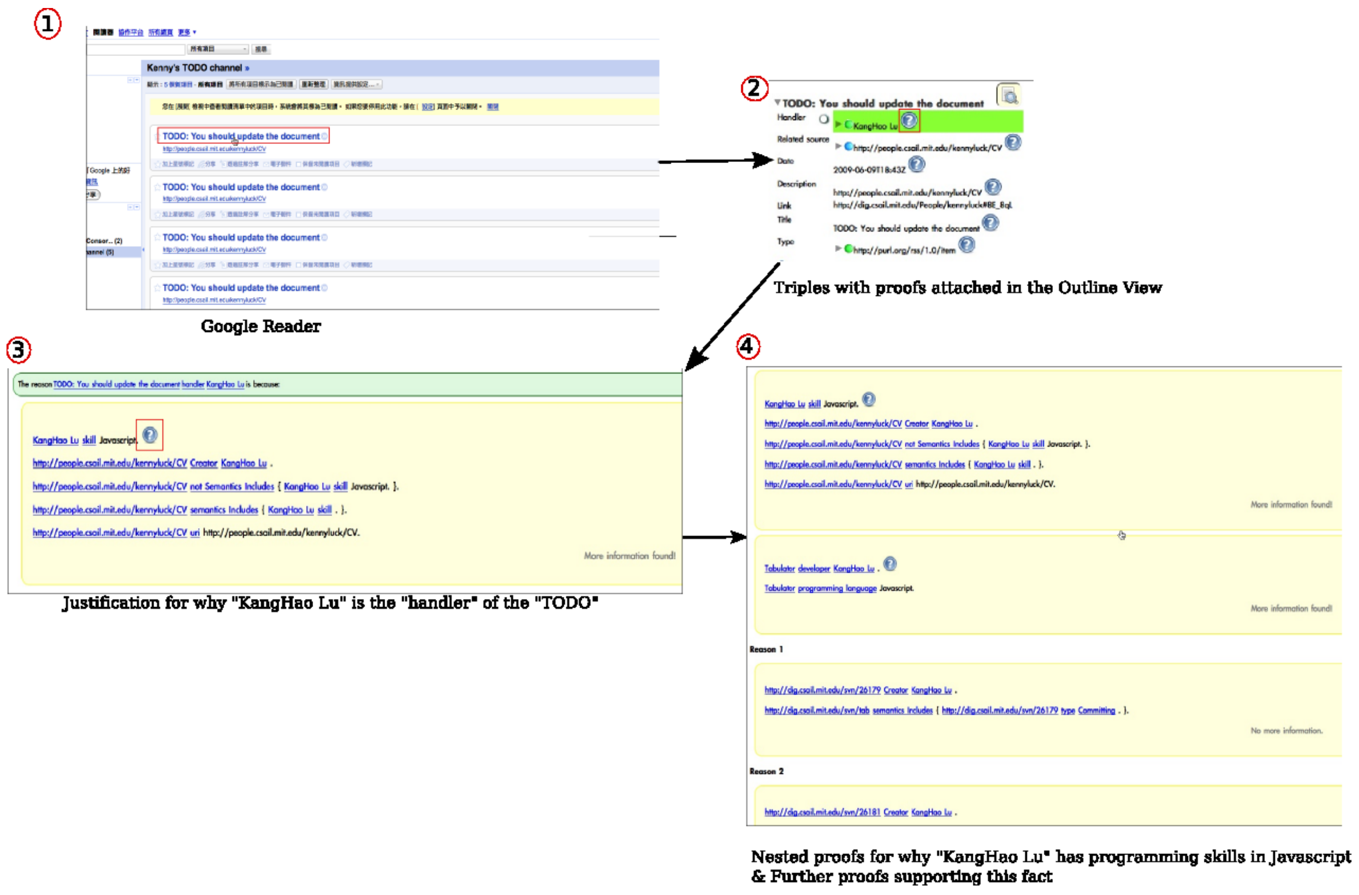
Once the question mark icon is clicked the justification of that particular triple will be displayed as shown in Fig 4. In this particular example, the user has clicked the question mark icon next to the triple which says 'Timothy Berners-Lee' is a 'Developer' of the Tabulator project given in Fig 3. The justification of this fact consists of recent commits that he has done to the Tabulator version control system. Reason 1 of the justification gives a particular commit revision of which 'Timothy Berners-Lee' is the creator of, and so on.

Chained Proofs Our system is capable of handling proof chaining as well. Let us illustrate this with a simple example. Suppose there are few rules as follows:

– If a person commits to a project, that person must be a developer of the project.
– If a person is a developer of a project and that project uses a programming language Y. Then that person must have the skill of programming in Y.
– If a person made a document, which lists some information but not complete, and there are some new information about a fact in the document, then that person should update the document.

The 'should' information in the last rule is encoded in the form of a RSS feed. Therefore once the last rule is fired, this feed would be sent to the person's personal RSS reader. Notice the fact that an RSS 1.0 channel is also a valid RDF/XML document. The scenario, as experienced by the user, is illustrated in Fig 5:

1. Our user will see that there is an event called "TODO: You should update the document".


1
Google Reader
2
Triples with proofs attached in the Outline View
3
Justification for why "KangHao Lu" is the "handler" of the "TODO"
4
Nested proofs for why "KangHao Lu" has programming skills in Javascript & Further proofs supporting this fact

Kenny's TODO channel

TODO: You should update the document
http://people.csail.mit.edu/kennyluck/CV

TODO: You should update the document
http://people.csail.mit.edu/kennyluck/CV

TODO: You should update the document
http://people.csail.mit.edu/kennyluck/CV

TODO: You should update the document
http://people.csail.mit.edu/kennyluck/CV

TODO: You should update the document
Handler KangHao Lu
Related source http://people.csail.mit.edu/kennyluck/CV
Date 2009-06-09T18:43Z
Description http://people.csail.mit.edu/kennyluck/CV
Link http://dig.csail.mit.edu/People/kennyluck#BE_8qL
Title TODO: You should update the document
Type http://purl.org/rss/1.0/item

The reason TODO: You should update the document handler KangHao Lu is because:

KangHao Lu skill Javascript.
http://people.csail.mit.edu/kennyluck/CV Creator KangHao Lu .
http://people.csail.mit.edu/kennyluck/CV not Semantics Includes { KangHao Lu skill Javascript. }.
http://people.csail.mit.edu/kennyluck/CV semantics Includes { KangHao Lu skill . }.
http://people.csail.mit.edu/kennyluck/CV uri http://people.csail.mit.edu/kennyluck/CV.
More information found!

KangHao Lu skill Javascript.
http://people.csail.mit.edu/kennyluck/CV Creator KangHao Lu .
http://people.csail.mit.edu/kennyluck/CV not Semantics Includes { KangHao Lu skill Javascript. }.
http://people.csail.mit.edu/kennyluck/CV semantics Includes { KangHao Lu skill . }.
http://people.csail.mit.edu/kennyluck/CV uri http://people.csail.mit.edu/kennyluck/CV.
More information found!

Tabulator developer KangHao Lu .
Tabulator programming language Javascript.
More information found!

Reason 1

http://dig.csail.mit.edu/svn/26179 Creator KangHao Lu .
http://dig.csail.mit.edu/svn/tab semantics Includes { http://dig.csail.mit.edu/svn/26179 type Committing . }.
No more information.

Reason 2

http://dig.csail.mit.edu/svn/26181 Creator KangHao Lu .

Fig. 5. Example of a Scenario which involves several Chained Proofs

2. Once he clicks on it, he will see all the information associated with it in the Tabulator outline view. Further, he notices that some of the triples have some derivations (question marks along with the triples).
3. He clicks on the question mark for the triple *<the event> handler 'KangHao Lu'*, and sees more information on the justification UI. This gives the reasons for why he is the handler of the event. This includes 5 facts. These facts are stacked in the order of derivation (and thus reads from bottom to top). The first fact indicates that KangHao Lu's CV has a particular URI. The second fact says that the CV includes a 'skill' category. The third fact indicates that these skills do not include 'Javascript'. The fourth fact says that the creator of the CV is KangHao Lu, and the final fact says that KangHao Lu is skilled in Javascript. Notice that there is a question mark next to the last triple.
4. Once you click on the question mark to learn why 'KangHao Lu' has skills in Javascript, the proofs for that is appended. In this case it happens to be

that he is a developer of Tabulator, and the primary programming language used in Tabulator is Javascript. There is again a question mark next to the triple *Tabulator developer 'KangHao Lu'*. By clicking on that, we learn the proof for this fact. in this case, it happens to be that 'KangHao Lu' has committed to the Tabulator project.

The last rule of this chain of deduction is implemented with the following simplified N3 rule:

```
@prefix log: <http://www.w3.org/2000/10/swap/log#> .
@prefix doape: <http://dig.csail.mit.edu/2007/wiki/doapextension#> .

@forAll :subject, :predicate, :object .
{   ?document dc:creator ?person .
    :subject :predicate :object .
    ?document log:semanticsIncludes { :subject :predicate [] . } .
    ?document log:notSemanticsIncludes { :subject :predicate :object .} . }
=>
{   [ a     rss:item;
      rss:title "TODO: You should update the document";
      doape:handler ?person;
      doape:relatedSource ?document ] . }.
```

This rule means that if a document has the triple *"{ s } { p } { o1 } ."*, but does not have the triple *"{ s } { p } { o2 } ."*, which we assume is available at another source, then the creator of this document is responsible for making this modification and a notification for an RSS item is generated.

5 Related Work

AIR (**A**ccountability **I**n **R**DF) [12] a policy language that exploits a dependency tracking approach to generate proofs with quasi-natural language explanations for a given policy and a log file. AIR policies are represented in Turtle [3], which is a human readable syntax for RDF, and include the quoting feature of N3Logic [4]. AIR consists of an ontology and a reasoner, which when given a set of policies and data in Turtle, attempts to compute compliance of the data with respect to the policies. Each computed compliance result has an associated explanation in Turtle outlining the derivation of the result from the inputs. The AIR reasoner is only capable of generating a proof at a document level, but not at the triple level. Also, it does not have the capabilities to update a corresponding file as in our method.

Proof Markup Language (PML) is a general proof language or "proof interlingua" that is used to describe proof steps generated by different kinds of reasoning engines. Inference Web (IW) provides a framework for displaying and manipulating proofs defined in Proof Markup Language (PML) [15]. IW concentrates on displaying proofs whereas our approach is for both generating and displaying proofs. We also allow chaining of proofs so that they can be explored recursively.

Although it is outside of the domain of Semantic Web, we can draw a strong analogy between our work and the the work done by Bill Lewis on the Omniscient Debugging concept [14]. Omniscient Debugging is the idea of collecting “time stamps” at each “point of interest” (setting a value, making a method call, throwing/catching an exception) in a program and then allowing the programmer to use those time stamps to explore the history of that program run. Similar to his idea, our “Social Computing Agent” performs computations and at a particular “point of interest” generates a useful triple and updates the document where the triple fits along with the proof of how it was derived.

SPARQL queries currently support for querying multiple graphs. Quilitz et al [17] describe an optimized method for querying distributed RDF data sources with SPARQL using their DARQ[7] implementation. Schenk et al [18] describe a way for linking user generated content on the Semantic Web. Yars2 (Yet Another RDF Store, Version 2) [11] presents an end-to-end semantic search engine that uses a graph data model to enable interactive query answering over structured and interlinked data collected from many disparate sources on the Web. Zhao et al [21] describe various design patterns for representing and querying provenance information relating to mapping links between heterogeneous data from sources in the domain of functional genomics. Their work illustrates the use of named RDF graphs at different levels of granularity to make provenance assertions about linked data. It also demonstrates that these assertions are sufficient to support requirements including data currency, integrity, evidential support and historical queries. While all the work mentioned above are analogous to the work we have done in gathering information from several sources, our ‘Computation Agent’ is able to find information that matches as long as such information is available. In addition, all of those works described above do not support SPARQL updates.

6 Future Work

We envision SPARQL/Update to be the most important communication protocol in the future. Here we briefly list some of the future work we consider interesting.

Proof-based Content Filtering: Different from normal machine learning-based content filtering schemes, which are more of less based on heuristics, SPARQL Update + Proofs provides a logical way of content filtering. We can implement a SPARQL Update receiver agent that validates proofs or checks whether the rule generating this new information is in its white list of rules. As rules have URIs and are part of the Linked Data, meta information of rules allows the agent to do more things, for example, building the white list of rules from rules that have more than certain number of supporters.

[7] http://darq.sf.net

SPARQL Update + Proofs is machine readable, traceable and multilingual. We believe it has the potential of replacing email in certain circumstances, for example in sending bug reports.

Linked Data Garbage Collector(s): Consider the following example which was briefly described in Sec **??**:

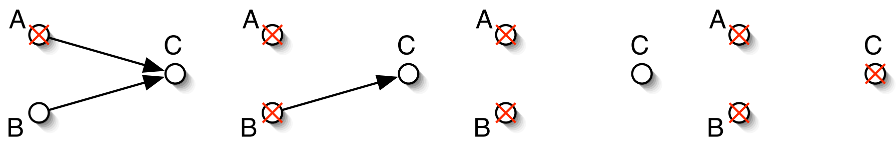


Fig. 6. Phases of a Linked Data GC example

Node A is the statement "Kenny is a member of the DIG group", Node B is "Kenny is a developer of Tabulator", Node C is "Kenny has the access right to the Tabulator repository" and the links represent the proofs. In the first phase, the statement "Kenny is a member of the DIG" is deleted, so in the second phase the proof that has Node A as an antecedent is also deleted. But at this point, Node C is still supported by a proof and hence remains existing. During the last phase, all the proofs of Node C, the statement "Kenny has the access right to the Tabulator repository", are no longer existing. Therefore we can remove this statement.

As Web is a decentralized system and there are a lot of information getting deleted at a particular time, the Web incorporated with this system can be imagined as a huge memory space with lots of garbage collectors. Few technical challenges remain:

1. Should servers or crawler clients send these removing notifications? Maybe both?
2. How does this system deal with cyclic dependency? [8]

Mechanism and User Interface for Grading Triples: As pointed out in section 2.2, the current implementation of user interface displays values of triples by naively counting the number of proofs. More grading algorithms should be created and provided to the end user in a customizable way. A a simple next step would be grading a proof with numbers of supporters of the corresponding rule. Google's Page Rank is also a possibility, namely strong statements proves strong results. However, it requires collaboration of different servers.

[8] This is also a well known problem in programming GC.

Other future works include:

– Extend SPARQL Update consistently to include updating formulae or graphs
– Implement the HTTP Link Header both on the server-side and the client-side
– Visualization for statements with cyclic dependency
– Upgrade Computing Agent with support for more rule formats, Including AIR [12] and RIF [?]

7 Conclusion

The ultimate vision of the Semantic Web is to make it an immortal brain where the computers work as if they were neurons [6]. Before wikis and blogs became popular, the Web had been basically a read-only medium and the machines did not resemble anything close to communicating neurons. This is perhaps due to the fact that HTML is not flexible enough for updating. This project investigated methods and protocols for updating the "Data Web", a Web with RDF as it's main data format. We described some protocols for achieving this. All these decisions are far from being standardized. Therefore, they are subject to change, but we expect later versions of the current implementation to be backwards compatible. The never ending crawler and updater demonstrates that the Web can be designed to think and learn by itself. The garbage collecting paradigm, as described in section 6, demonstrates a reasonable way in which certain parts of the Data Web could be programmed to forget. We believe that proofs on the Web will eventually weave into a Truth Maintenance System and that updating the proofs back to the Data Web will be a key point for achieving the Web Intelligence.

References

1. About dbpedia, 2009.
2. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. DBpedia: A Nucleus for a Web of Open Data. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007*, pages 722–735, 2007.
3. D. Beckett. Turtle - Terse RDF Triple Language. `http://www.dajobe.org/2004/01/turtle/`.
4. T. Berners-Lee, D. Connolly, L. Kagal, Y. Scharf, and J. Hendler. N3logic: A logical framework for the world wide web. *Journal of Theory and Practice of Logic Programming*, 2007.
5. T. Berners-Lee and D. Connoly. Turtle - Terse RDF Triple Language. `http://www.w3.org/2000/10/swap/doc/cwm.html`.
6. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. `http://www.scientificamerican.com/article.cfm?id=the-semantic-web`.
7. K. G. Clark, L. Feigenbaum, and E. T. (editors). SPARQL protocol language for RDF, w3c recommendation 15 january 2008.

8. P. P. da Silva, D. L. McGuinness, and R. Fikes. A proof markup language for semantic web services. *Information Systems*, 31(4-5):381 – 395, 2006. The Semantic Web and Web Services.
9. Dan Brickley and Libby Miller. FOAF Vocabulary Specification.
10. Dave Beckett and Brian McBride. RDF - Resource Description Framework.
11. A. Harth, J. Umbrich, A. Hogan, and S. Decker. Yars2: A federated repository for querying graph structured data from the web. In *6th International and 2nd Asian Semantic Web Conference (ISWC2007+ASWC2007)*, pages 211–224, November 2007.
12. L. Kagal, C. Hanson, and D. Weitzner. Using dependency tracking to provide explanations for policy management. *IEEE Policy 2008*, 2008.
13. KangHao Lu. Slides on this project presented at the Cambridge Semantic Web Meeting on 9 June 2009. `http://dig.csail.mit.edu/2009/Talks/0609-tmsweb-kennyluck/`.
14. B. Lewis. Debugging backwards in time, 2003.
15. D. L. Mcguinness and P. Pinheiro. Explaining Answers from the Semantic Web: the Inference Web Approach. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(4):397–413, October 2004.
16. E. Prudhommeaux and A. S. (editors). SPARQL query language for RDF, w3c recommendation 15 january 2008.
17. B. Quilitz and U. Leser. Querying distributed rdf data sources with sparql. In *5th European Semantic Web Conference (ESWC2008)*, pages 524–538, June 2008.
18. S. Schenk and S. Staab. Networked graphs: A declarative mechanism for sparql rules, sparql views and rdf data integration on the web. In *17th International World Wide Web Conference (WWW2008)*, pages 585–594, April 2008.
19. A. Seaborne and G. Manjunath. SPARQL/Update a language for updating rdf graphs.
20. Tim Berners-Lee and James Hollenbach and Kanghao Lu and Joe Presbrey and Eric Prud'ommeaux and mc schraefel. Tabulator Redux: Browing and Writing Linked Data . In *Linked Data on the Web Workshop at WWW08*, 2008.
21. J. Zhao, A. Miles, G. Klyne, and D. Shotton. Linked data and provenance in biological data webs. *Brief Bioinform*, 10(2):139–152, 2009.