sparqlPuSH: Proactive notification of data updates in RDF stores using PubSubHubbub

Alexandre Passant¹ and Pablo N. Mendes²

 ¹ Digital Enterprise Research Institute National University of Ireland, Galway alexandre.passant@deri.org
 ² Kno.e.sis Center, CSE Department Wright State University, Dayton, OH - USA pablo@knoesis.org

Abstract. With the growing numbers of status update websites and related wrappers, initiatives modelling sensor data in RDF, as well as the dynamic nature of many Linked Data exporters, there is a need for protocols enabling real-time notification and broadcasting of RDF data updates. In this paper we present a flexible approach that provides such notifications to be delivered in real-time to any RSS or Atom reader. Our framework enables the active delivery of SPARQL query results through the PubSubHubbub (PuSH) protocol upon the arrival of new information in RDF stores. Our open source implementation can be plugged on any SPARQL endpoint and can directly reuse PuSH hubs that are already deployed in scalable clouds (*e.g.* Google's).

1 Introduction

Since the Semantic Web is "an extension of the current Web" [5], it has to deal with the different paradigm shifts happening on the Web. In particular, more and more streamed information is available online, ranging from microblogging updates to sensor data, often being combined with trends in ubiquitous computing — e.g. status and geolocation updates from mobile phones, related to the two aforementioned aspects. From the Semantic Web side, this entails the ability to capture this streamed information in RDF, through efforts such as semantic microblogging [12] [9] or representation of sensor information as Linked Open Data [1]. In addition, many data sources in the Linking Open Data cloud are build from user-generated content, as DBpedia, Freebase, FOAF profiles, etc [6]. Consequently, there is a need to tackle this dynamic generation aspects which lead to an area of constantly evolving RDF data available at Web scale.

These dynamic aspects of RDF data entail various issues, including changes management [13], stream querying [4], etc. In this paper we focus on how to enable real-time notifications of data updates in RDF stores. We provide a way to let users subscribe to a subset of content available within an RDF store (defined as a SPARQL query) and get a notification message each time some content within that subset changes in the store. To achieve this goal, we define a complete framework for such notifications and broadcasting, based on:

- 2 Alexandre Passant and Pablo N. Mendes
- the representation of data updates in RDF stores through RSS/Atom feeds and a registration system to map SPARQL queries to these feeds;
- the use of the PubSubHubbub protocol³ to proactively broadcast the previous feeds and inform clients about data updates in real-time;
- an open-source implementation of the aforementioned principles, build in PHP and flexible enough to be adapted on the top of any SPARQL endpoint supporting SPARQL Query SPARQL Update⁴.

The rest of this paper is organised as follows. In Section 2, we discuss our motivations for real-time notification of data updates, before discussing related work in Section 3. In Section 4, we present how we use the PubSubHubbub protocol to broadcast changes happening in RDF stores. In particular, we discuss how we map SPARQL query results to RSS feeds and how they are broadcasted to interested parties. In Section 5, we discuss the implementation of the previous principles in sparqlPuSH. Finally, we conclude the paper.

2 Motivations

On the Web, particularly in the post-Web 2.0 era, there is an ubiquitous feed of socially created content around the globe that may or may not find its way to an interested user. Imagine users interested in monitoring a developing news story (e.g. the 2010 Chile earthquake), keeping up-to-date on media placements for a brand, or getting the latest news on the stock market. In such cases, content updates may contain actionable information that is useful only if delivered in real-time, especially in emergency scenarios such as earthquake monitoring [11].

Google Alerts, a content monitoring service, automatically notifies users when there are new Google results for a set of keywords. It delivers updates through e-mails (text and HTML), and RSS feeds. However, monitoring alerts are limited to keyword-based queries. From the information retrieval point of view, in comparison to keywords, SPARQL queries provide a more expressive language to describe a user's information need — *i.e.* complex constraints and unambiguous references. A Semantic Web counterpart to Google Alerts should allow users to register a SPARQL query and get updates pushed to the users as new matching triples arrive in an underlying RDF store containing relevant data.

As such, Semantic Search engines that allow persistent searches and real time updates could support the brief use-cases that we now briefly describe.

2.1 Monitoring competitors information in a corporate context

A product manager interested in following the competitors of his company could ask a semantic search engine to "select user-generated content mentioning companies that compete with mine", using DBpedia to identify such competitors, as depicted in Fig. 1. Using a pull approach, his RSS aggregator would have to

³ http://code.google.com/p/pubsubhubbub/

⁴ http://www.w3.org/TR/sparql11-update/

constantly fetch the feeds to identify new content. However, with a push model, relevant data is simply delivered as soon as it comes into the original system(s).

```
PREFIX ex: <http://example.org/>
PREFIX moat: <http://moat-project.org/ns#>
PREFIX company: <http://dbpedia.org/ontology/Company>
SELECT ?document
WHERE {
   ?document moat:taggedWith ?competitor .
   ?competitor company:industry ?industry.
   ex:MyCompany company:industry ?industry.
}
```

Fig. 1. Example SPARQL query selecting documents that mention competitors (*i.e.* working in the same field) of a fictitious company identified as ex:MyCompany.

2.2 Following updates on the Health Care Reform

In the context of the Health Care Reform in the U.S.A., we present the following fictitious use case: Otto is a balanced congressman that is concerned primarily with practical considerations, rather than moralistic premises. He would like to follow, as the discussion unfolds, how does the public perception change across states. Especially, he would like to compare the trending topics in states with Republican versus Democratic majorities.

Many data sources can be applicable to Otto's use case, including local news and microblog posts. Otto would like to "select all entities mentioned in microblog posts from democratic states", while in another window he would like to "select all entities mentioned in microblog posts from republican states". Looking at the windows side by side he would be able to quickly glance over the differences. However, a more complex query could provide a direct answer, if he choses to "select all entities mentioned in microblog posts from democratic states that were not mentioned in microblog posts from republican states". Once again, in order to get an accurate perception of the current trends, this information should be updated as soon as new data comes in one of these original systems.

3 Related work

Various work recently focused on the representation of changes in RDF data sources and related datasets, notably in the Linked Data realm⁵. These include

⁵ http://www.ldodds.com/blog/2010/04/rdf-dataset-notifications/

the Talis Changesets⁶ and Triplify update vocabularies⁷ [3], as well as the dady (Dataset Dynamics) vocabulary⁸, that can be combined with voiD [2]. While the two first ones provide the ability to represent atomic changes (*e.g.* new triples being added to a resource), dady focuses mainly on representing characteristics of changes in dataset, such as their expected frequency. In addition, atomic changes can be transmitted in Atom⁹. Other efforts include the Web of Data Link Maintenance Protocol¹⁰ [14], which notifies linked resource when links are added to and removed from them, as well as — to some extent — the Semantic Sitemap extension [7] that defines how often data can be re-crawled from a website to get fresh information, while it relies on clients regularly fetching it, not directly solving the notification issue.

Regarding real-time notification, XMPP messaging [10] can be used as a way to transport SPARQL queries¹¹ [8], while Semantic Pingback¹² focuses on informing remote sources of new links, as soon as a link is created from a seed source.

Finally, outside the Semantic Web world, both rssCloud¹³ and PubSubHubbub¹⁴ (PuSH) address the notification issue. Both focus on a push approach to broadcast notifications in feeds transmitted *via* hubs that push information proactively from one service to the various clients interested in following this service changes. However, rssCloud focuses only on RSS 2.0 feeds, while PuSH can be adapted to both RSS and Atom. In addition, it provides a public infrastructure that can be used by implementers, notably by using the Google public PuSH hub¹⁵.

4 Distributing data updates as RSS and Atom feeds

4.1 Shifting from a *pull* to a *push* approach

As we presented in the use-cases, our main motivation is to enable data changes notification to shift from a *pull* to a *push* approach, *i.e.* letting people being notified of changes in the information they are interested to, rather than forcing them to constantly pull sources to identify new data.

In particular, our goal is to enable proactive notification of changes happening in RDF stores, whatever they deal with: new data of a particular type being added, updated statements about a given resource, etc. To do so, we rely on the aforementioned PubSubHubbub protocol to broadcast these updates, combined

¹⁰ http://www4.wiwiss.fu-berlin.de/bizer/silk/wodlmp/

¹³ http://rsscloud.org/

⁶ http://n2.talis.com/wiki/Changeset

⁷ http://triplify.org/vocabulary/update

⁸ http://vocab.deri.ie/dady

⁹ http://linkeddatacamp.org/wiki/LinkedDataCampVienna2009/DatasetDynamics

¹¹ http://danbri.org/words/2008/02/11/278

¹² http://aksw.org/Projects/SemanticPingback

¹⁴ http://pubsubhubbub.googlecode.com/

¹⁵ https://pubsubhubbub.appspot.com/

with a two-steps approach: (i) registering the SPARQL queries related to the updates that must be monitored in a RDF store and (ii) broadcasting changes when data mapped to these queries is updated in the store.

This workflow has been implemented in sparqlPuSH, a system that can be plugged on the top of any SPARQL endpoint to achieve this goal, and that we will further describe in Section 5.

4.2 Registering SPARQL queries for data updates



Fig. 2. Workflow of the sparqlPuSH query registration, see text for details.

The registration of a SPARQL query to be notified of updates happening in a RDF store works as follows¹⁶ (Fig. 2):

- a user sends a SPARQL query to the sparqlPuSH interface (compliant with the SPARQL protocol), e.g. http://example.org/sparqlPuSH/;
- the sparqlPuSH interface registers the query locally and maps it to a new feed, also indicating its creation date (Fig. 3). The information is stored in a particular graph, e.g. http://example.org/sparqlPuSH/feeds;
- 3. the interface generates a feed (RSS or Atom) corresponding to the query, and registers it to a PuSH hub (using the public Google's one by default);
- the feed, containing a link to the PuSH hub URL in its header according to the PuSH specification¹⁷ — is send to the client;
- 5. the client parses the feed in order to get the PuSH hub URL, and registers its interest to the feed at this particular hub.

¹⁶ The last two steps of this workflow are simply the adaptation of the PuSH principles to our Semantic Web use-case.

¹⁷ http://code.google.com/p/pubsubhubbub/wiki/RssFeeds

6 Alexandre Passant and Pablo N. Mendes

```
@prefix sp: <http://vocab.deri.ie/sparqlpush#> .
@prefix dct: <http://purl.org/dc/terms/> .
<http://example.org/feed/34562738> a sp:Feed ;
sp:query "
SELECT ?uri ?author ?label ?date
WHERE {
    ?uri a sioc:Post ;
        sioc:has_creator ?author ;
        dc:title ?label ;
        dct:created ?date .
} ORDER BY ASC(?date)" ;
    dct:modified "2010-03-29T09:18:23Z" .
```

Fig. 3. Mapping RSS feeds to SPARQL query results.

In order to register the query, any system can send a HTTP POST query to the sparqlPuSH interface, and the query has to be passed using the query parameter¹⁸. The system automatically interprets some common prefixes, and additional ones can be easily added in an appropriate configuration file. Moreover, in order to provide relevant feeds (*e.g.* appropriate <rss:title> or <dc:creator> elements), the system uses a few conventions to must be respected in the SPARQL query:

- ?uri their URI of the element(s) to be retrieved;
- ?date their creation / modification date;

and some optional ones:

- ?label their label;
- ?author their author¹⁹;

That way, the different elements of the RDF data that is retrieved are converted into elements of the feeds. Notably, the use of a ?date variable is required to order information by date and ensure that information in the feed is ordered as expected, *i.e.* by update date. Moreover, any <item> element of the feed (identified from the ?uri variable) can be a dereferencable URI, so that it can be easily consumed by Linked Data aware clients. That way, we use simple RSS or Atom feeds to transfer RDF information between the original triple-store and the client(s) that request the changes notifications.

Furthermore, these queries (and the complete architecture proposed in this paper) can be used not only to identify new data from a particular type being loaded in a store (sometimes with additional constraints, such as filtering

¹⁸ In addition, as we will describe later, sparqlPuSH provides a form-based interface to directly register queries.

¹⁹ The **?author** variable can bind either to the URI of an author, or to a literal identifying it, while the first way is obviously preferred in a Linked Data scenario.

based on a given topic or author) but also to identify changes corresponding to a particular entity that is being modified. For example, relying on the Talis Changeset vocabulary, one can register a SPARQL query that will — based on the following broadcast system — notify an alert as soon as new statements are edited regarding the resource <http://example.org/FooBar>, as described in Fig. 4.

```
PREFIX cs: <http://purl.org/vocab/changeset/schema#>
SELECT ?uri ?author ?label ?date
WHERE {
?uri a cs:ChangeSet ;
   cs:creatorName ?author ;
   cs:changeReason ?label ;
   cs:createdDate ?date ;
   cs:subjectOfChange <http://example.org/FooBar> .
} ORDER BY ASC(?date)
```

Fig. 4. Registering a SPARQL query to identify changes of a particular resource

4.3 Triggering events and broadcasting updates via PuSH hubs



Fig. 5. Workflow of the sparqlPuSH changes notification, see text for details.

Once feeds have been registered in the system, the process works as follows (Fig. 5):

- 8 Alexandre Passant and Pablo N. Mendes
- 1. RDF data can be loaded in the RDF store through the sparqlPuSH interface, that is compliant with SPARQL Update principles, so that it can be send using HTTP POST. Actually, this interface just acts as proxy that launches triggers when new data is loaded;
- once the data have been loaded, the system runs all the registered SPARQL queries and update — if needed — the according feeds²⁰;
- 3. for each updated feed, a notification is sent to the the PuSH hub;
- 4. immediately, the hub broadcasts the information to all the clients that have registered to this particular feed with it.

In practice, our experiments showed that once the data is loaded in the sparql-PuSH interface, the clients receive it only a few seconds later, using the Google public PuSH hub server²¹.

With regards to the triggering step, since the sparqlPuSH interface lives on the top of any SPARQL endpoint, it is done by (1) sending the update query from the interface to the store via HTTP, then (2) running all the registered queries when it receives a response from this update query. However, this process could be a bit cumbersome, especially when the sparqlPuSH interface and the original RDF store are on the same server, as it still implies running the queries over HTTP. To solve this issue, and as we will now discuss, the interface can be adapted for some particular stores to use their internal API for querying, rather than doing it *via* HTTP.

5 Implementation

We implemented sparqlPuSH in PHP, as a proxy that can be plugged on the top of any RDF store supporting the currently standardised SPARQL Update language, the SPARQL protocol via HTTP, as well as named graphs (in order to register the queries and their mappings in a particular graph)²². The system includes (i) an interface to let users register queries (Figure 6(a)) — while this is generally done remotely, through clients that can then interpret the resulting feed and register to the appropriate PuSH hub —, and (ii) an interface listing the available RSS feeds, and the corresponding SPARQL queries (Figure 6(b)).

The sparqlPuSH implementation is available at http://code.google.com/ p/sparqlpush through SVN, and comes with an example client that can be used to test the approach, in addition to the server part described in this paper. This test client can (1) register SPARQL queries to any sparqlPuSH interface (including the management of the PuSH hub registration when retrieving the response feed) and (2) receive notifications from any PuSH hub to update its interface its real-time (using JavaScript to check every second if a notification has been received) In addition, a video is also available from the project website in order to showcase the interest of approach.

²⁰ In addition, their dct:modified value (as seen in Fig. 3) is updated.

²¹ http://pubsubhubbub.appspot.com/

²² These requirements being common features of many RDF stores.



Fig. 6. sparqlPuSH: (a) registering queries; (b) listing available queries.

Furthermore, as we previously mentioned, in addition to the generic SPARQL connector, sparqlPuSH includes a direct interface to ARC2 using its PHP API. That way, interactions between sparqlPuSH and the RDF store are not done through HTTP but directly using the ARC2 API, as well as the triggering step, making the approach even faster. New wrappers for other APIs could easily be added, in order to avoid this HTTP interactions and making sparqlPuSH part of the RDF store itself.

6 Conclusion

In this paper, we detailed an architecture for triggering data updates in RDF store and broadcasting them in real-time to various clients. Our system, sparqlPuSH, has been made available by combining SPARQL queries, RSS / Atom feeds, and the PubSubHubbub protocol. We believe that this approach can be a first step towards a push-model for the Semantic Web, which becomes mostly needed considering the evolution of the (Semantic) Web towards a continuous stream of data.

In addition, while currently available as a plug-in for any RDF store, we hope that this push approach can become a default model in various RDF store implementations, enabling more capabilities to monitor, in real-time, changes related to RDF data. In the future, we could also imagine to register the queries to be monitored using C-SPARQL [4], instead of the current triggering approach.

Acknowledgements

The work presented in this paper has been funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Líon-2).

10 Alexandre Passant and Pablo N. Mendes

References

- 1. Proceedings of the 1st SemSensWeb2009 Workshop on the Semantic Sensor Web, volume 468. CEUR-WS.org, 2009.
- Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao. Describing Linked Datasets. In Proceedings of the Second Workshop on Linked Data on the Web (LDOW2009) at WWW2009, 2009.
- 3. Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumueller. Triplify: light-weight linked data publication from relational databases. In Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors, Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009, pages 621–630. ACM, 2009.
- Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. C-SPARQL: SPARQL for continuous querying. In WWW, pages 1061–1062, 2009.
- Tim Berners-Lee, James A. Hendler, and Ora Lassila. The Semantic Web. Scientific American, 284(5):34–43, 2001.
- Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data The Story So Far. International Journal on Semantic Web and Information Systems (IJSWIS), 5(3):1-22, 2009.
- Richard Cyganiak, Holger Stenzhorn, Renaud Delbru, Stefan Decker, and Giovanni Tummarello. Semantic Sitemaps: Efficient and Flexible Access to Datasets on the Semantic Web. In Proceedings of the 5th European Semantic Web Conference (ESWC 2008), volume 5021 of Lecture Notes in Computer Science, pages 690–704. Springer, 2008.
- Frank Osterfeld, Malte Kiesel, and Sven Schwarz. Nabu A Semantic Archive for XMPP Instant Messaging. In Proceedings of the 1st Workshop on The Semantic Desktop, 4th International Semantic Web Conference, volume 175 of CEUR Workshop Proceedings. CEUR-WS.org, 2005.
- Alexandre Passant, Uldis Bojars, John G. Breslin, Tuukka Hastrup, Milan Stankovic, and Philippe Laublet. An Overview of SMOB 2: Open, Semantic and Distributed Microblogging. In 4th International Conference on Weblogs and Social Media, ICWSM 2010. AAAI, 2010.
- Peter Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. Request for comments: 3920, Internet Engineering Task Force, 2004. http://www. ietf.org/rfc/rfc3920.txt.
- Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake Shakes Twitter Users: Real-time Event Detection by Social Sensors. In Proceedings of the Nineteenth International WWW Conference (WWW2010). ACM, 2010.
- Joshua Shinavier. Real-time SemanticWeb in <= 140 chars. In Proceedings of the Third Workshop on Linked Data on the Web (LDOW2010) at WWW2010, 2010.
- Jürgen Umbrich, Michael Hausenblas, Aidan Hogan, Axel Polleres, and Stefan Decker. Towards Dataset Dynamics: Change Frequency of Linked Open Data Sources. In Proceedings of the Third Workshop on Linked Data on the Web (LDOW2010) at WWW2010, 2010.
- Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Discovering and maintaining links on the web of data. In *International Semantic Web Confer*ence, volume 5823 of *Lecture Notes in Computer Science*, pages 650–665. Springer, 2009.