Collaborative Project

GeoKnow - Making the Web an Exploratory for Geospatial Knowledge

Project Number: 318159 **Start Date of Project:** 2012/12/01 **Duration:** 36 months

Deliverable 4.4.1
Subscription and Notification Service

Dissemination Level	Public
Due Date of Deliverable	Month 20, 31/07/2014
Actual Submission Date	Month 22, 15/09/2014
Work Package	WP4, Spatial-Semantic Exploration, Visualization, Analysis and Authoring
Task	T4.4
Type	Prototype
Approval Status	Approved
Version	0.1
Number of Pages	24
Filename	Subscription and notification service.pdf

Abstract: This deliverable evaluated Rsine System to be reused as a Subscription and Notification Service. In general, Rsine fulfil GeoKnow Supply Chain scenario requirements, however some extensions and contributions were made and a evaluation of this contribution is presented in a Testing System developed within this task.

The information in this document reflects only the author's views and the European Community is not liable for any use that may be made of the information contained therein. The information in this document is provided "as is" without guarantee or warranty of any kind, express or implied, including but not limited to the fitness of the information for a particular purpose. The user thereof uses the information at his/her sole risk and liability.

Project funded by the European Commission within the Seventh Framework Programme (2007 - 2013)

History

Version	Date	Reason	Revised by
0.0	08/06/2014	Initial document Structure	Alejandra Garcia-Rojas
0.1	22/08/2014	Added content to related work, motivation and contribution sections	René Pietzsch
0.2	25/08/2014	Added content to contribution, testing system and test setup sections	Alejandra Garcia-Rojas
0.3	31/08/2014	Completed Supply Chain information	René Pietzsch
0.3	11/09/2014	Internal and Peer review	Andreas Both
0.4	12/09/2014	Address peer review comments	Alejandra Garcia-Rojas
0.5	14/09/2014	Additional Peer review	Hugh Williams
1.0	15/09/2014	Address peer review comments	Alejandra Garcia-Rojas

Author List

Organization	Name	Contact Information
Ontos	Alejandra Garcia-Rojas	alejandra.garciarojas@ontos.com
Brox	René Pietzsch	rpietzsch@brox.de

Executive Summary

The Supply Chain use case of GeoKnow requires a subscription and notification system to allow users to provide and maintain filter criteria for which they would like to receive updates on a RDF store. In the related work an existing system, Rsine, stands out and fulfills our requirements. In this deliverable we tested the usability of Rsine and identified improvement opportunities that are the main contribution in this deliverable, together with a Testing System that is used to evaluate such contributions.

Table of Contents

List of Figures

List of Tables

1 Introduction

A subscription and notification system consists of the possibility for users to be notified of some specific changes in a dataset. For instance new properties of a concept, value changes, etc. Users are required to register queries that represent the reason for the changes of what they want to be notified of. Such a system requires the tracking of the RDF database for inserts, deletes and updates. This tracking may generate a changeset that has to be evaluated to determine if a subscribed query is affected and a notification has to be triggered.

One of the most promising application areas for spatial Linked Data is in the supply chain. Large manufacturers have many thousands of suppliers rendering the information flows accompanying these supply chains a real Big Data challenge. The aim of this use case will be to give enterprises collaborating in supply chains a unified spatial view on the logistics in the supply chain. A goal in supply chain management is full transparency about the flow of material and accompanying information. We aim to serve this information in real-time so that bottlenecks can be identified early and media breaks in the information flows are minimized.

A subscription and notifications system is required in the GeoKnow project mainly for the Supply Chain scenario. Thus, the motivation of such a service is described in section 2. At the beginning of this task (4.4) we started investigating the related work on Subscription and Notification Services. This research directed us to a newly existing implementation that could satisfy the requirements of this task, the Rsine Service [**?**]. This implementation is the state of the art on subscriptions and notifications about specific changes in an RDF Store. Rsine is manifold, but developed with the objective of being able to perform quality assessments of RDF data when using controlled vocabularies (e.g. SKOS). In this deliverable we aim at testing Rsine also for the Supply Chain scenario.

Therefore, the objective for this deliverable was to evaluate the usability of Rsine in our use case. During this evaluation, we identified improvement opportunities, therefore we developed required enhancements and made an evaluation of these contributions. In Section 3, we describe in detail the Rsine Service and cite other related works. We identified opportunities for improvement, that allowed us to develop the components which we could test actual Rsine implementation and improved one which is described in section 4. Contributions are basically the replacement of the Rsine component that will improve the RDF store query response, and a couple of extensions to Rsine and a small bug fix. Finally, to benchmark our contributions we developed a testing system that provides interfaces for a Data Generator simulator of a supply chain, the subscription and notification service and the triple store, which are described in section 5. The actual tests set up and results are provided in section 6.

2 Motivation

Our vision for the Web of Data is that it can become a major cornerstone for the next generation of supply chain networks. As such it needs to scale to arbitrary huge deployments and couple the communication partners as loosely as possible in order to allow for easy setup and reconfiguration.

Certainly communication partners need to have accurate and up to date information. Thus, means for publishing and retrieving updates are required. Pull-based approaches will not scale with the potential size of a global supply chain network. This is of special importance in environments with heavy load, many subscribers and a high frequency of changes, which is not unlikely in supply chain networks.

That is why we build upon a subscription-and-notification-based approach where communication partners could register for exactly the information changes they are investing in and will get notified upon relevant changes only.

As we are sharing major ideas and goals behind Rsine, this aspect is also true for the main use cases. In particular, we are targeting the following ones:

Value Changes. The most basic, hence yet relevant use case is to register for triple additions or changes regarding a specific resource (object) or predicate. This is of special importance in the supply chain dashboard as we will update the live view and metrics calculations upon subscriptions between the supplier and customer. Other than simple value changes subscriptions could directly trigger notifications via eMail, SMS Gateway or other alert transports in case important threshold values are exceeded.

Schema Changes Schema changes are also of interest for us. Subscribers can register for changes in the T-box. Added or removed (sub)classes or properties could trigger a notification. In the supply chain scenario this is of relevance as the schema of the exchanged data is the binding contract between the communicating parties. Thus, changes in the terminology should be revealed, e.g. indicating a pending protocol update or just a non standard conform contact partner. While schema validation will not be part of this prototype the corresponding T-box subscriptions could be used to trigger subsequent schema checks with RDF test suites like RDFUnit[1]. Doing these checks on a change subscription and notification basis would further allow for delta checking of the incoming changes.

Change Prediction The subscription service can act as a supportive tool for predictive analysis. The predictive algorithms can be triggered promptly upon relevant changes. It helps to perform (likely expensive) calculations only if needed. These kinds of analysis could e.g. be used to predict material consumption and outreach incorporating traffic, weather or other incidents that might impact a supply chain network causing delays, out of stock or even potentially dangerous situations.

Geospatial information Since information in the Supply Chain may concern spatial data relevant for the users, we require to be able to integrate spatial queries within the defined subscriptions of the users.

Therefore, besides the Subscription and Notification System requirements specified in [**?**], we have considered additional requirements in order to satisfy our use case, described next.

- The performance of the triple store does not have to be compromised by a tracking changes system ⇒ thus we have to minimize or nullify this problem
- A user needs to be notified of a change within a reasonable period of time (i.e. not more than one hour)

[1] http://aksw.org/Projects/RDFUnit.html

- The system should be able to handle hundreds of different subscription queries
- User can define a change-set size limit
- User can define a change-set complexity and runtime.
- Conditional subscription queries has to accept geospatial queries.

3 Related Work

Some services have being developed to be able to notify users about changes in triple stores. RSS Feed like notifications are proposed by SparqlPuSH[2]. This system allows clients to get informed about data updates in RDF stores via PubSubHubbub[3] protocol. SparqlPuSH does not provide extra information about the update, for instance who has made the change. SDShare[4] is a protocol for the syndication of resource descriptions. It defines how a RESTful service can publish a series of feeds that list snapshots and changes to collections of resources. This protocol also defines how a client should process those feeds and the linked resource descriptions so that a local store can be synchronised [**?**]. Another approach aiming at propagating updates of resources in the cloud is ProLD [**?**]. They explored database propagation approaches and proposed a framework that included a observer, implemented as triggers in the database management system, which sends these changes to a propagator that follows `owl:sameAs` links to propagate changes, where that receiver does integrity checks and triggers the changes at the local dataset.

A more recent implementation that allows for creating personalised subscription based on event-condition-action rules that can be defined by users is the Rsine Service[**?**]. This service has created its own vocabulary to define rules for notifications. These approach has being also explored in [**?**] and [**?**]. The next section will describe Rsine system in detail.

3.1 Rsine

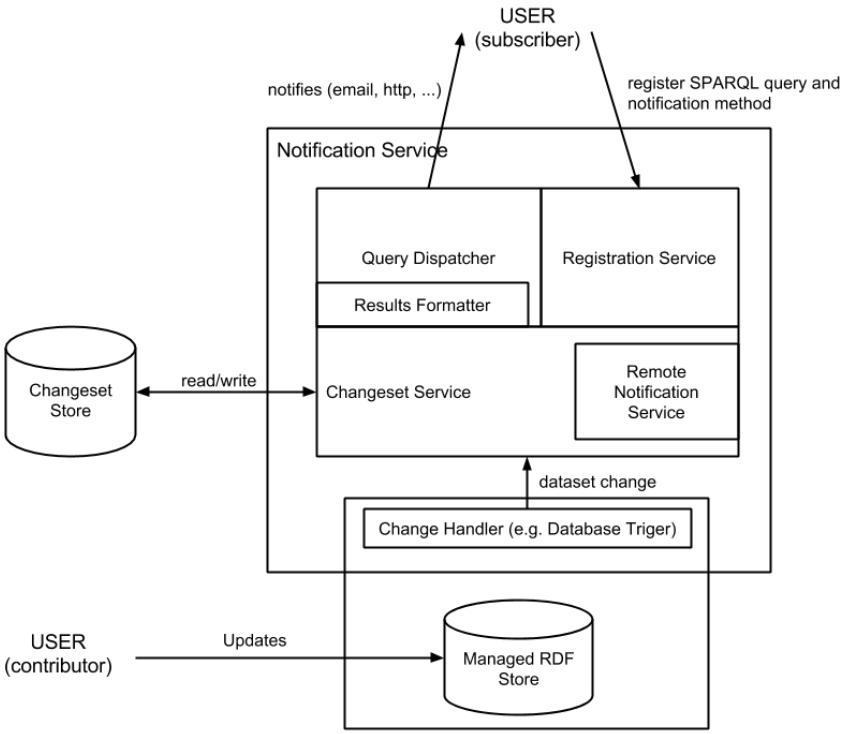


Figure 1: Rsine architecture

[2] http://code.google.com/p/sparqlpush/
[3] http://code.google.com/p/pubsubhubbub/
[4] http://www.sdshare.org/spec/sdshare-current.html

Figure 1 depicts the architecture of Rsine. Changes on the RDF store are detected by a change handler implemented on the top of the RDF store that notifies a Rsine service for every triple that is inserted, deleted or updated. Rsine transforms the received triples to a changeset expressed in RDF using the changeset ontology[5] and persists these changes into a change store. This store is queried to evaluate every registered subscription and generate notifications at some point.

A subscription is defined by using a controlled vocabulary and consists of

- the definition of the type of change (e.g., addition, removal)
- an optional condition that must be fulfilled for receiving the notification
- a pattern that defines the text and data values contained in the notification message
- a notifier that specifies the way the notification is disseminated to the subscriber (e.g, by email)

These subscriptions are expressed as SPARQL queries in terms of the changeset ontology vocabulary. An example of a subscription asking for notifications for circular hierarchical relations is described in listing 1. Specifically, if a concept has a new **skos:borader** relationship (defined query at line 10), Rsine will evaluate for each new insertion of **skos:broader** property where the **rsine:condition** expressing the circular relation (line 27) is true. If the condition is satisfied, Rsine will generate a message using the format provided (line 49) which uses the auxiliary queries (line 36), and finally distribute this messages to the specified **rsine:notification** channels (lines 55 and 59).

```
1  @prefix spin: <http://spinrdf.org/sp/> .
2  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
3  @prefix rsine: <http://lod2.eu/rsine/> .
4  @prefix dcterms: <http://purl.org/dc/terms/>.
5
6  <http://example.org/chr> a rsine:Subscription;
7      rsine:query [
8          dcterms:description "Notification on circular hierarchical
               relations";
9
10         spin:text "PREFIX cs:<http://purl.org/vocab/changeset/schema#>
11             PREFIX spin:<http://spinrdf.org/sp/>
12             PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
13             PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
14
15             SELECT ?concept ?otherConcept WHERE {
16                 ?cs a cs:ChangeSet .
17                 ?cs cs:createdDate ?csdate .
18                 ?cs cs:addition ?addition .
19
20                 ?addition rdf:subject ?concept .
21                 ?addition rdf:predicate skos:broader .
22                 ?addition rdf:object ?otherConcept .
23
24                 FILTER (?csdate > 'QUERY_LAST_ISSUED'^^<http://www.w3.org
                       /2001/XMLSchema#dateTime>)
25             }";
26
27         rsine:condition [
28             spin:text "PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
29                 ASK {
```

[5] http://vocab.org/changeset/schema.html

```
30                      ?concept skos:broader+ ?otherConcept .
31                      ?otherConcept skos:broader+ ?concept
32                  }";
33              rsine:expect true;
34          ];
35
36          rsine:auxiliary [
37              spin:text "PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
38                  SELECT ?conceptLabel WHERE {
39                      ?concept skos:prefLabel ?conceptLabel .
40                      FILTER(langMatches(lang(?conceptLabel), 'en'))
41                  }";
42              spin:text "PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
43                  SELECT ?otherConceptLabel WHERE {
44                      ?otherConcept skos:prefLabel ?otherConceptLabel .
45                      FILTER(langMatches(lang(?otherConceptLabel), 'en'))
46                  }";
47          ];
48
49          rsine:formatter [
50              a rsine:vtlFormatter;
51              rsine:message "The concepts <a href='$bindingSet.getValue('
                    concept')'>$bindingSet.getValue('conceptLabel').getLabel()
                    </a> and <a href='$bindingSet.getValue('otherConcept')'>
                    $bindingSet.getValue('otherConceptLabel').getLabel()</a>
                    form a hierarchical cycle";
52          ];
53      ];
54
55      rsine:notifier [
56          a rsine:loggingNotifier;
57      ];
58
59      rsine:notifier [
60          a rsine:emailNotifier;
61          foaf:mbox <mailto:name@example.com>
62      ].
```

Listing 1: Example Rsine changeset

Rsine supports two triple stores: openRDF and Virtuoso. We are interested in testing Virtuoso's implementation because GeoKnow uses this store that offers geospatial functionalities.

The Virtuoso implementation provided as a VAD package, is available in github[6]. Installing the released VAD package in a Virtuoso instance is quite easy. Once the installation is performed, user can configure Rsine settings in Virtuoso with the Url and port number where Rsine Service is running and the graphs that will be monitored for changes.

While testing the usability of Rsine for our objectives, besides possible improvements mentioned in [**?**], improvement opportunities we observed were:

- RsineVad was only working with Viruoso 6.1.8, we require to use 7.1.
- For reporting changes to Rsine from Virtuoso, the RsineVad installs a trigger that performs a http request to Rsine every time that a triple is added or deleted, update is missing in the current implementation.
- Triggers are according to OpenLink[7] a rather expensive operation in Virtuoso and should be avoided.

[6] https://github.com/Rsine/RsineVad/releases
[7] http://www.openlinksw.com/

- Currently in Rsine persistence of subscriptions and changesets are done in memory.
- Only email notifications can be used.

Next section we describe in detail the contributions performed by GeoKnow in order to be able to use Rsine Service in the Supply Chain Use case.

4 Contributions

We choose Rsine as the foundation of our subscription and notification service. As we start working with Rsine we have seen some potential for improvements that can be made to the Rsine service. The following gives an overview of our modification and extensions to the Rsine service itself:

- Improve RDF store side implementation by reading log file
- Rsine extensions and bug fixes:
  - Extend Rsine with http notifier
  - Added asynchronous change announcement interface
  - Bugfix for blank node handling

4.1 The txr_parser

Part of this deliverable was to contribute an alternative to the RsineVad that utilizes the same Rsine service backend. The implementation consequently respects the Rsine architecture. Which means that there is one component bound to Virtuoso gathering the added or removed triples (currently RsineVad) that communicates via REST http calls to the Rsine service itself. Thus the transaction log parser replaces RsineVad. The source code can be found in the GeoKnow repository on GitHub: https://github.com/GeoKnow/trx_parser, and it has also already a link in the Rsine repository instructions[8].

Our transaction log parser is a python-based implementation that uses the newly[9] introduced Virtuoso function `read_log`. Instead of registering triggers with Virtuoso we choose a less intrusive and thus lower footprint approach. The transaction logs parser needs filesystem level (read) access to the transaction log files written by the Virtuoso instance that should be monitored. A filesystem watchdog listens for any newly created or changed transaction log file and triggers a custom stored procedure which in turn uses `read_log` to parse the added or deleted triples from the log. For log files that are continuously used the transaction log parser incrementally gathers changes from the files using the offset provided from `read_log`.

For each changed triple a http request to the Rsine service is issued. This is where the Rsine service jumps in and uses its internal workflow to persist the published triple and queries the Virtuoso for conditional and auxiliary information if the registered subscription requires to do so.

The transaction log parser command line interface is shown in listing 2 for configuration:

```
1  Usage: python trx_parser.py [-i <virt_isql_path>] [-l <virt_log_path>] [-
       s <virt_server_port>] [-u <virt_dba_user>] [-w <virt_dba_passwd>] [-r
       <rsine_host>] [-p <rsine_port>] [-g <"graph1URI"[,"graph2URI"[,...]]>]
2
3  Options:
4    -h, --help            show this help message and exit
5
6    -i VIRT_ISQL_PATH, --virt_isql_path=VIRT_ISQL_PATH
7                          The path of the isql executable. Defaults to ./
                             bin/isql
8
9    -l VIRT_LOG_PATH, --virt_log_path=VIRT_LOG_PATH
```

[8] https://github.com/rsine/rsine

[9] https://github.com/openlink/Virtuoso-opensource/commit/f0145bd1101bf15b7ccd1cb4c2b5d83d888201f7

```
10                          The path where virtuoso transaction logs are
                               written. Defaults to ./var/lib/virtuoso/db/
11
12    -s VIRT_SERVER_PORT, --virt_server_port=VIRT_SERVER_PORT
13                          The virtuoso server port. Defaults to 1111
14
15    -u VIRT_DBA_USER, --virt_dba_user=VIRT_DBA_USER
16                          The dba user. Defaults to dba
17
18    -w VIRT_DBA_PASSWD, --virt_dba_passwd=VIRT_DBA_PASSWD
19                          The dba password. Defaults to dba
20
21    -g RSINE_INCL_GRAPHS, --rsine_incl_graphs=RSINE_INCL_GRAPHS
22                          A comma separated list of graph URIs to be
                               included. Omitting this parameter defaults to
                               all graphs.
23
24    -r RSINE_HOST, --rsine_host=RSINE_HOST
25                          The hostname of the server where the rsine
                               service is running. Defaults to 127.0.0.1
26
27    -p RSINE_PORT, --rsine_port=RSINE_PORT
28                          The port the rsine services listens on. Defaults
                               to 2221
```

Listing 2: Transaction log parser CLI

4.2 Rsine Extensions and Bug Fixes

One extension for Rsine was to implement a http notifier. This extension was required by our testing system in order to facilitate the registration of notification time. But it can also be used for any other Http-based API where notifications can be sent. Another extension of the system was the asynchronous change announcement from the Notifier component. The objective was to provide a faster response when the change announcement was sent in order that this component doesn't wait for the server to respond and will not block the Virtuoso server thread that issued the REST call longer than minimally needed.

Lastly the current rsineVad component that installed the triggers in virtuoso has only been tested and implemented for Virtuoso 6.x and in our testing we also encountered problem in running it with Virtuoso 7.x. These issues were reported to Rsine developers. The transaction log parser runs with any current and upcoming version of Virtuoso that provides the `read_log` interface.

Finally, last contribution was a bug fix[10] to handle bnodes.

To be able to immediately contribute to Rsine we forked their repository[11] and made corresponding changes. These contributions are available at https://github.com/GeoKnow/Rsine/tree/httpNotification_async to the developer community, and specially to Rsine developers. These changes are in the process of being integrated into the original Rsine as a special branch for GeoKnow.

[10] https://github.com/GeoKnow/rsine/commit/99fc43da2f5f91e5de6baa97f6f9eef6016af70a
[11] https://github.com/Rsine/Rsine

5 Testing System

5.1 Architecture

This testing system consists the following components:

Subscription and Notification System, may be composed of:

Subscription and notification Service to register subscriptions, receive changeset from the RDF store and evaluate subscriptions and send notifications

Change notifier component: specific implementation of the RDF Store to track changes made in the data

RDF Store, the triple store where data is updated

Sparql Client Simulator, will generate activity for the RDF store by sending update queries to the RDF store

Testing System will orchestrate activity between above mentioned components and perform the corresponding measurements.

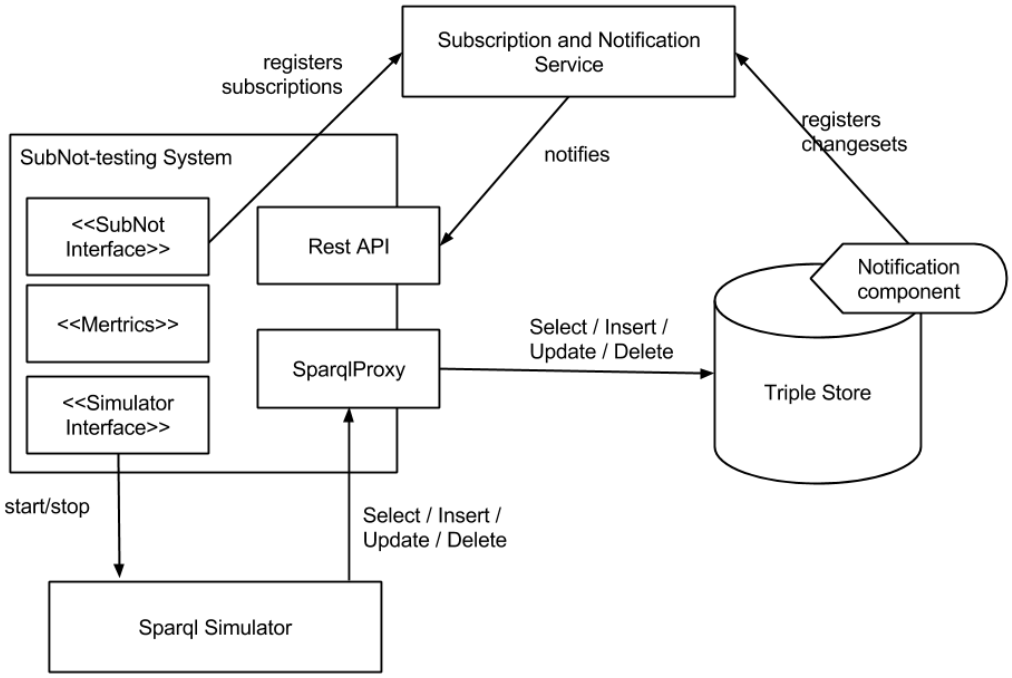


Figure 2: Testing system architecture

The architecture of this testing system is depicted in figure 2. The testing system is controlled through the Rest API. When the test starts, it will register subscriptions to the Subscription and Notification Service and will start the SPARQL simulator which will be sending updates to the RDF store through the SparqlProxy. The SparqlProxy is used to measure queries response time. Changes in the RDF store are tracked by the Notification component that is installed in the same machine as the store and when changes happens it sends

the changeset to the Subscription and Notification Service, thus this service will evaluate subscriptions and perform the corresponding notifications. Finally when the testing is stopped, the SPARQL simulator is stopped, and metrics about query response time and notification time are computed.

5.2 Metrics

The purpose of the testing system is to measure:

The query response time in the triple store. This is with the objective of evaluating the Notification component, which depending on the implementation in the RDF store, can influence the server's query response time.

The user notification time. According to our use case there are notifications that have to be sent the moment the change is registered. Thus, we need to register the moment when the change is made, and the moment the system sends the notification. To be able to better do this measurement a HTTP notification is the most suitable notification method.

5.3 Installation and Configuration

The testing system is available on Github https://github.com/GeoKnow/subnot-testing. Implemented in Java as a web service. To generate a war file the maven command `mvn package` can be used. The system is firstly configured in the web.xml file, where the SparqlProxy has to be configured:

```
  <context-param>
   <description>Destination endpoint</description>
   <param-name>destination-endpoint</param-name>
   <param-value>http://remote.endpoint.url:8899/sparql</param-value>
 </context-param>
  <context-param>
   <description>Proxy endpoint is the absolute URL where this testing
      system is installed, plus the servlet mapping of the proxy</
      description>
   <param-name>proxy-endpoint</param-name>
   <param-value>http://localhost:8080/subnot-testing/sparql</param-value
      >
 </context-param>
```

The specification of which notification component is used in the system has to be installed and configured apart form the Testing System, since it is part of the Notification system. This means that the tester has to be aware of which notification component is installed in the RDF store and make sure it is running with the correct configuration.

Afterwards, the system provides a UI where most of the parameters are passed to run the tests. These parameters are:

- Subscription and Notification Service URL
- Sparql Simulator URL and its parameters. In our test case the Supply Chain can be configured with the Frequency.
- Test duration in minutes, after this time the test will be stopped and metrics will be computed

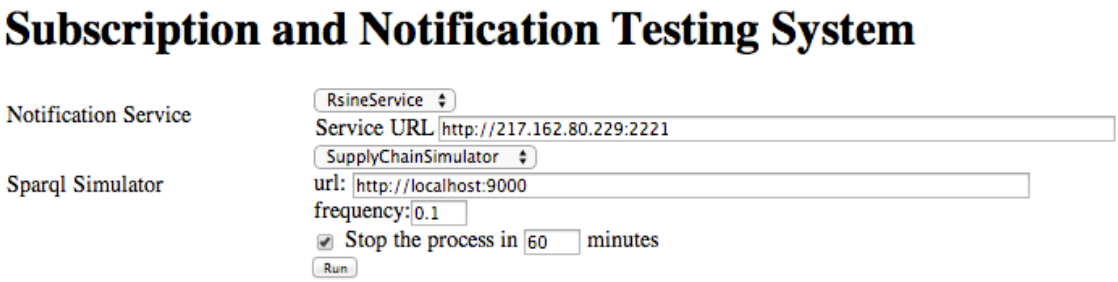


Figure 3: Testing system UI

Figure 3 depicts the user interface of the testing system with the configuration parameters that can be provided at run time.

The testing can be stopped by the user or at the time specified. Then results are displayed in the screen. An example of results is given in the following listing:

```
Simulation started: Thu, 21 Aug 2014 22:38:13 +0200
Simulation ended: Thu, 21 Aug 2014 23:38:14 +0200
Simulation duration: 01:00:01,112
supplyChainSimulatorFrequency: 0.1

# queries: 42926
query average time: 46 ms
query max duration time: 418 ms
query min duration time: 42 ms

Registered subscriptions: 63
# notifications: 3056
# changesets: 3161
changeset notification average time: 1402271 ms
changeset notification max time: 3581354 ms
changeset notification min time: 7636 ms
...
```

6 Test Setup

We implemented the testing system to test Rsine using the Supply Chain Dashboard[12] as a data generator system. Thus, we implemented the corresponding interfaces for both systems. Rsine was already described in 3.1. The dashboard is briefly described next.

6.1 The Supply Chain Dashboard

The Supply Chain Dashboard was developed within the WP5 and described in [**?**]. This is a web application that allows the user to search, browse and to explore supply chain data. The decision of using the Supply Chain Dashboard was driven by the need to evaluate the system under our use case, and that we can have the control over the kind of updates that can done on the data. Figure 4 describes the concepts that are used in the Supply Chain use case. Figure 4 outlines the central classes that are used for representing supply chain data.

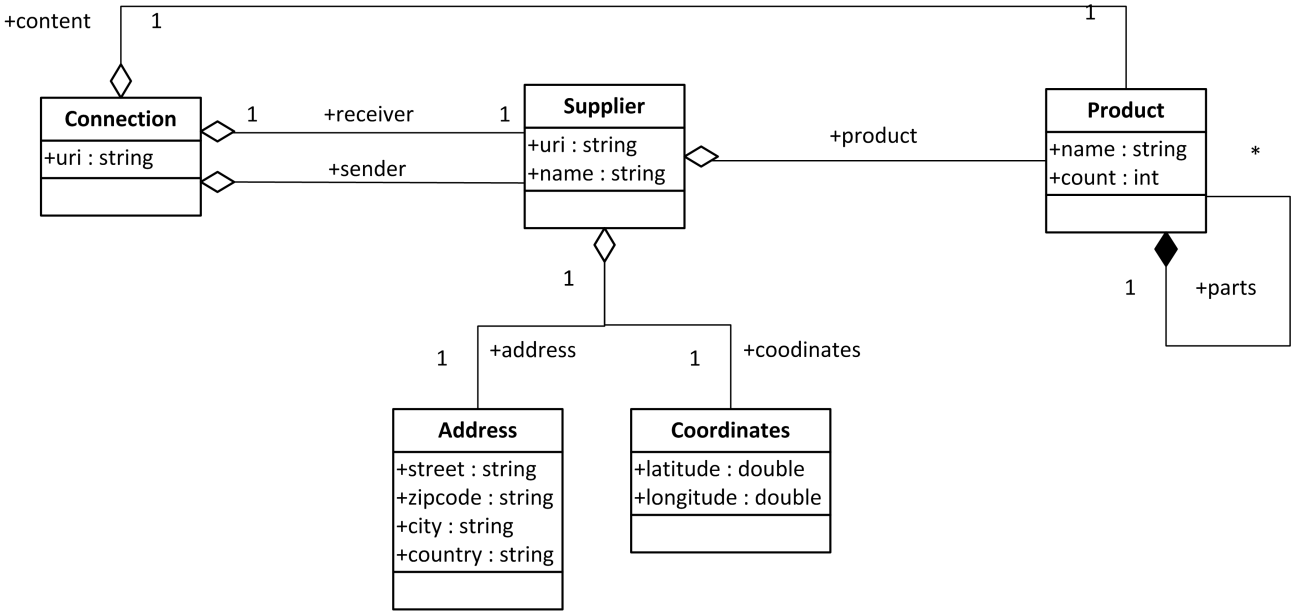


Figure 4: Central classes for representing supply chains.

At the startup of the Supply Chain Dashboard the simulation data is initialized. From a structured parts list the supply chain network is created. Which means that all supplier are constructed beginning at an OEM (root supplier) and following the supply chain down to its subsequent vendors (tiers). Each client vendor relation is established in form of a connection on our class model. After the supply chain has been established the simulation can be controlled from the web or a REST interface. The interface provides means to start, stop and step through the simulation. In addition the update frequency can be adjusted,which basically sets how many ticks the simulator should advance per second (defaults to one simulated day per second).

The simulator then generated orders from the root supplier to its vendor chains. The orders are processed in terms of subsequent orders from tiers 1 to tiers 2 and so on. Production of the pars is simulated as well as

[12] https://github.com/GeoKnow/Supply-Chain-Dashboard

the shipments for order fulfillment.

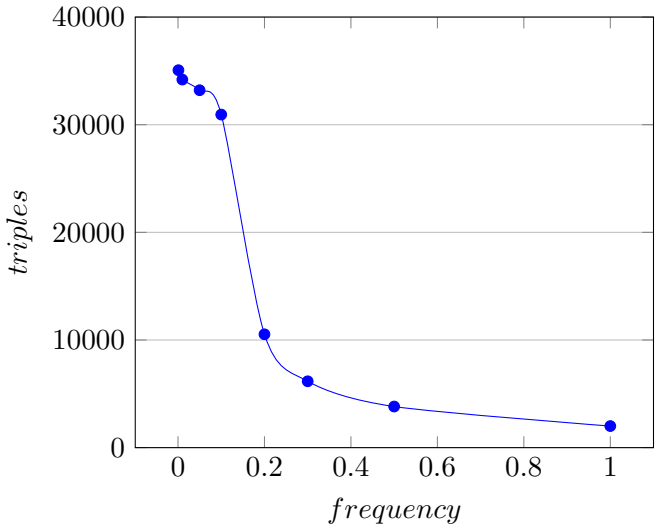
The Dashboard can be accessed through a web-based front-end composed by widgets, or through a rest API.

6.2 Workload

The workload testing system can be controlled in two variables: the Supply Chain Dashboard frequency and the number of subscriptions. Both are described as follow.

6.2.1 Supply Chain Dashboard Frequency

Update queries in the Dashboard are randomly generated in time and in data size. However, the frequency parameter specifies the speed of the simulation in seconds per simulated days. Defaults to 1.0, which means that in each second one day is simulated. For the testing system we required to determine the frequency boundary where the system stops growing in linear way, the graphic in the following figure presents the data growth according to the frequency.



According to these results the more likely frequency value to use is 0.1.

6.2.2 Subscriptions

To test the system we have created four different types of subscriptions that are later specified for each of the suppliers and products present in the datasets. These subscriptions register for the following requests:

1. New orders for X supplier
2. X product arrived
3. X product arrived late
4. New supplier of product X

The query templates fore these subscriptions are presented in Appendix A. These templates are then used to generate subscriptions considering initial data in the dashboard (i.e. suppliers and products). For instance

we get n suppliers, thus we generate n subscriptions for the request "new orders for X supplier", and the same happens for the product related queries. In the testing setup we got 63 different subscriptions using above mentioned requests.

6.3 Results

Tests have to be made at the same time every time. We paid attention to not affect the system during the testing periods (e.g. no OS updates). We provided the following hardware:

- For the RDF store (Virtuoso) and the Notification component (RsineVad and trx_parser) a Dell PowerEdge R415 12 cores (2x AMD Opteron(tm) Processor 4170 HE)
- For Rsine a 1.60GHz Intel Atom(TM), 1Gb RAM with Debian 3.2 i686.
- For the Testing Sytem and the Supply Chain Dashboard a MacBook Pro 2.9GHz Intel Core i7, 8GB RAM with OSX 10.9.4

Since the RsineVad component was not usable in Virtuoso 7.1, but in Virtuoso 6.1 and the trx_parser was only for Virtuoso 7.1 we had to run tests in different versions of Virtuoso. This can of course influence the testing results, thus we firstly compared the query response time (Qrt in milliseconds) isolated in both versions of Virtuoso. We ran the Dashboard for an hour with Frequency=0.1. Results of this test is shown in table 1.

Version	triples	queries	Qrt avg	min Qrt	max Qrt
Virtuoso 6.1	211 839	42 316	47	41	4 130
Virtuoso 7.1	214 714	42 891	46	42	661

Table 1: Virtuoso version comparison

Performance tests of RsineVad and trx_parser are presented in table 2. Tests were run during one hour with Frequency=0.1 and 63 subscriptions.

Test	subscriptions	triples	queries	Qrt avg	notifications	not avg
Virtuoso 6.1, RsineVad	10	494	95	37467	442	1613606
Virtuoso 6.1, RsineVad, async	63	95804	13357	268	2733	1461215
Virtuoso 7.1, trx_parse	10	21444	4237	48	1480	1598334
Virtuoso 7.1, trx_parse, async	63	214884	42926	46	3056	1402271

Table 2: RsineVad vs trx_parse

These results showed that the log parser implementation has better performance in the query response time, which was expected since it does not interfere with the RDF store transactions. Moreover, the asynchronous implementation did help to reduce the time response on the RDF store which improved the query response time and as a consequence better performance of the notification system.

6.4 Geospatial Notifications

An important requirement that was accomplished with the development of a notification component that can work in Virtuoso 7.1 is the possibility of making geospatial queries. We tested this functionality creating subscriptions that require geospatial query information. An example of such subscription is presented in listing 3. This subscription requires notification of shipments that have been made outside of Germany.

```
...
rsine:query [
    spin:text "PREFIX cs:<http://purl.org/vocab/changeset/schema#>
        PREFIX spin:<http://spinrdf.org/sp/>
        PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
        PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
        PREFIX sc: <http://www.xybermotive.com/ontology/>
        SELECT * WHERE {
            ?cs a cs:ChangeSet .
            ?cs cs:createdDate ?csdate .
            ?cs cs:addition ?addition .
            ?addition rdf:subject ?concept .
            ?addition rdf:predicate rdf:type .
            ?addition rdf:object sc:Shipping
        }";
     rsine:condition [
            spin:text "PREFIX sc: <http://www.xybermotive.com/ontology/>
                       PREFIX geo: <http://www.w3.org/2003/01/geo/
                          wgs84_pos#>
                       SELECT ?receiver {
                         ?concept sc:connection ?connection .
                         ?connection sc:receiver ?receiver .
                         ?receiver geo:lat ?a .
                         ?receiver geo:lon ?o .
                         BIND (xsd:decimal(?o) as ?lon) .
                         BIND (xsd:decimal(?a) as ?lat) .
                         <http://example.org/germany> geo:geometry ?geo .
                         FILTER (!bif:st_contains (?geo, bif:st_point (?
                            lon, ?lat)))
                       }";
            rsine:expect "true"^^xsd:boolean;
     ];
];
...
```

Listing 3: Example Rsine changeset

7 Conclusion and Future Work

This objective of this deliverable was to implement a Subscription and Notification system for the Supply Chain use case in GeoKnow. Such a system has to allow users to subscribe to specific changes in the RDF store and be notified by a preferred channel. In the process of analyzing existing solutions, the Rsine system met mostly all our requirements. We decided to reuse Rsine, thus for this deliverable we described the usability of Rsine in our use case and presented some improvement opportunities that are part of our contribution. The main contribution to Rsine was the development of a parser-based change detection in the Virtuoso RDF store based on the transaction log files, which can replace previous RsineVad implementation in Virtuoso 7.1. Other contributions to rsine were to extend some features and to improve the system. In order to evaluate our contributions, we developed a testing system that is closely implemented to adapt Rsine architecture. The results of the tests presented show that the parser approach improved Virtuoso query response time, which was expected because it doesn't interferes with the system transactions.

The integration of this notification system in the GeoKnow generator is ongoing work. As a next step in this task is to continue extending Rsine with the implementation of other notification channels exploring the integration of popular social networks and the integration in the GeoKnow Generator. Also performance of geospatial queries has to be tested.

A Subscription templates

New orders from X supplier

```
@prefix spin: <http://spinrdf.org/sp/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix Rsine: <http://lod2.eu/Rsine/> .
@prefix http: <http://www.w3.org/2011/http#> .
@prefix dcterms: <http://purl.org/dc/terms/>.

<http://example.org/received_order_from_SUPPLIER_NAME> a Rsine:
   Subscription;
dcterms:description "new order to SUPPLIER_NAME Supplier";
Rsine:query [
   spin:text "PREFIX cs:<http://purl.org/vocab/changeset/schema#>
              PREFIX spin:<http://spinrdf.org/sp/>
              PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
              PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
              PREFIX sc: <http://www.xybermotive.com/ontology/>
              SELECT * WHERE {
              ?cs a cs:ChangeSet .
              ?cs cs:createdDate ?csdate .
              ?cs cs:addition ?addition .
              ?addition rdf:subject ?concept .
              ?addition rdf:predicate rdf:type .
              ?addition rdf:object sc:Order }";
   Rsine:condition [
      spin:text "PREFIX sc: <http://www.xybermotive.com/ontology/>
                    ASK {
                    ?concept sc:connection ?connection .
                    ?connection sc:receiver ?receiver .
                    ?receiver sc:name ?name .
                    FILTER (regex(?name, 'SUPPLIER_NAME')) }";
      Rsine:expect "true"^^xsd:boolean;
   ];
];
Rsine:notifier [
   a Rsine:httpNotifier ;
   http:methodName "POST";
   http:absoluteURI <http://testing-system-url/subnot-testing/notify>
].
```

X product arrived

```
@prefix spin: <http://spinrdf.org/sp/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix Rsine: <http://lod2.eu/Rsine/> .
@prefix http: <http://www.w3.org/2011/http#> .
@prefix dcterms: <http://purl.org/dc/terms/>.
<http://example.org/new_supplier_product_PRODUCT_NAME> a Rsine:
   Subscription;
dcterms:description "new supplier of product PRODUCT_NAME";
Rsine:query [
   spin:text "PREFIX cs:<http://purl.org/vocab/changeset/schema#>
                 PREFIX spin:<http://spinrdf.org/sp/>
                 PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns
                    #>
                 PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
                 PREFIX sc: <http://www.xybermotive.com/ontology/>
```

```
                    SELECT * WHERE {
                    ?cs a cs:ChangeSet .
                    ?cs cs:createdDate ?csdate .
                    ?cs cs:addition ?addition .
                    ?addition rdf:subject ?concept .
                    ?addition rdf:predicate rdf:type .
                    ?addition rdf:object sc:Supplier }";
    Rsine:condition [
       spin:text "PREFIX sc: <http://www.xybermotive.com/ontology/>
                          ASK {
                          ?concept sc:product ?product .
                          ?product sc:name ?name .
                          FILTER (regex(?name, 'PRODUCT_NAME')) }";
       Rsine:expect "true"^^xsd:boolean;
    ];
];
Rsine:notifier [
    a Rsine:httpNotifier ;
    http:methodName "POST";
    http:absoluteURI <http://testing-system-url/subnot-testing/notify>
].
```

X product arrived late

```
@prefix spin: <http://spinrdf.org/sp/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix Rsine: <http://lod2.eu/Rsine/> .
@prefix http: <http://www.w3.org/2011/http#> .
@prefix dcterms: <http://purl.org/dc/terms/>.

<http://example.org/PRODUCT_NAME_arrived_late> a Rsine:Subscription;
dcterms:description "PRODUCT_NAME arrived arrived late";
Rsine:query [
    spin:text "PREFIX cs:<http://purl.org/vocab/changeset/schema#>
                    PREFIX spin:<http://spinrdf.org/sp/>
                    PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns
                       #>
                    PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
                    PREFIX sc: <http://www.xybermotive.com/ontology/>
                    SELECT * WHERE {
                    ?cs a cs:ChangeSet .
                    ?cs cs:createdDate ?csdate .
                    ?cs cs:addition ?addition .
                    ?addition rdf:subject ?concept .
                    ?addition rdf:predicate rdf:type .
                    ?addition rdf:object sc:Shipping }";
    Rsine:condition [
       spin:text "PREFIX sc: <http://www.xybermotive.com/ontology/>
                         ASK {
                         ?concept sc:order ?order .
                         ?order sc:dueDate ?dueDate .
                         ?concept sc:date ?actualDate .
                         ?concept sc:product ?product .
                         ?product sc:name ?name .
                         FILTER (regex(?name, 'PRODUCT_NAME')) .
                         FILTER (?actualDate > ?dueDate) }";
       Rsine:expect "true"^^xsd:boolean;
    ];
];
Rsine:notifier [
    a Rsine:httpNotifier ;
```

```
   http:methodName "POST";
   http:absoluteURI <http://testing-system-url/subnot-testing/notify>
].
```

New supplier of product X

```
@prefix spin: <http://spinrdf.org/sp/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix Rsine: <http://lod2.eu/Rsine/> .
@prefix http: <http://www.w3.org/2011/http#> .
@prefix dcterms: <http://purl.org/dc/terms/>.
<http://example.org/new_supplier_product_PRODUCT_NAME>
   a Rsine:Subscription;
   dcterms:description "new supplier of product PRODUCT_NAME";
   Rsine:query [
      spin:text "PREFIX cs:<http://purl.org/vocab/changeset/schema#>
                     PREFIX spin:<http://spinrdf.org/sp/>
                     PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-
                        ns#>
                     PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
                     PREFIX sc: <http://www.xybermotive.com/ontology/>
                     SELECT * WHERE {
                     ?cs a cs:ChangeSet .
                     ?cs cs:createdDate ?csdate .
                     ?cs cs:addition ?addition .
                     ?addition rdf:subject ?concept .
                     ?addition rdf:predicate rdf:type .
                     ?addition rdf:object sc:Supplier }";
      Rsine:condition [
         spin:text "PREFIX sc: <http://www.xybermotive.com/ontology/>
                        ASK {

                        ?product sc:name ?name .
                        FILTER (regex(?name, 'PRODUCT_NAME')) }";
         Rsine:expect "true"^^xsd:boolean;
      ];
   ];

   Rsine:notifier [
      a Rsine:httpNotifier ;
      http:methodName "POST";
      http:absoluteURI <http://testing-system-url/subnot-testing/notify>
   ].
```

References

[1] Erik Behrends, Oliver Fritzen, Wolfgang May, Franz Schenk, and Daniel Schubert. A framework and components for eca rules in the web. In *Proceedings of Second International Conference on Rules and Rule Markup Languages for the Semantic Web, Athens, Georgia, USA (10th–11th November 2006)*, 2006.

[2] Robert Isele and René Pietzsch. Deliverable 5.2.1 initial prototype of supply chain geo data management infrastructure. Working paper, 2014.

[3] Jürgen Jakobitsch and Christian Mader. Deliverable 5.3.1: Subscription and notification service. Working paper, 2013.

[4] Peter Kalchgruber. Prold: propagate linked data. In *Current Trends in Web Engineering*, pages 307–311. Springer, 2012.

[5] Graham Moore and Lars Marius Garshol. SDShare - a protocol for the syndication of resource descriptions. http://www.sdshare.org/spec/sdshare-current.html, 2012.

[6] Alexandra Poulovassilis, George Papamarkos, and Peter T. Wood. Event-condition-action rule languages for the semantic web. In *EDBT Workshops*, pages 855–864, 2006.