Temporal RDF System for Power Utilities

Mohamed Gaha, Arnaud Zinflou, Alexandre Bouffard, Luc Vouligny,
Mathieu Viau, Christian Langheit and Etienne Martin

Institut de Recherche
d'Hydro-Québec
Varennes, QC, Canada
Email: {gaha.mohamed|zinflou.arnaud|bouffard.alexandre|vouligny.luc}@ireq.ca
{viau.mathieu|langheit.christian|martin.etienne}@ireq.ca

***Abstract*—Temporal data is a critical component in many applications. This is especially true in analytical applications for the smart grid. The analytical process often requires uncovering and analysing data and complex relationships from heterogeneous and distributed data sources that change over time. A common approach for this task is the use of relational databases or even data warehouses, which unfortunately do not allow reasoning and inference. Ontologies and semantic technologies are proving useful to leverage the value already embodied in existing systems without replacing the enterprise systems. In this paper, we describe how the usage of a formal representation of knowledge can support elaborated processes such as storing and extracting temporal data. The first example uses a semantic approach to capture and manage time series changes. The second example is a direct application of the first one. It consists on an efficient RapidMiner extension that allows end users to transparently extract temporal data from heterogeneous data sources.**

Keywords–Ontology; Heterogeneous data source; Versioning; Data-Mining tools; Temporal data.

I. INTRODUCTION

As more smart technologies are deployed across the electrical grid, it generates unprecedented data volume. To manage and use this information, utility companies such as Hydro-Québec must be capable of high-volume data management and advanced analytics designed to transform data into actionable insights. In this context, it is often necessary that the analysis process spans across multiple heterogeneous data sources. Ontologies and semantic metadata standards help and facilitate the aggregation and the integration of this content [1]. In addition, standard models for metadata representation on the World Wide Web, such as the Resource Description Framework (RDF), model relationships as first class objects making it very natural to query and analyze entities based on their relationships.

With the advent of semantic technologies and widely shared ontologies, it becomes possible to build an enterprise unified information view over heterogeneous and distributed data sources [2]. In the electric power industry, there exists the Common Information Model (CIM) [3] ontology that has been adopted by the International Electrotechnical Commission (IEC). It is the most complete and widely accepted ontology that offers a common language to exchange information between applications in the electrical field domain. The CIM is defined through a set of IEC international standards, mainly 61970-301 and 61968-11. The first release was standardized in 2003 and now contains more than a thousand concepts covering generation, transmission and distribution of power utilities.

The use of a common language like the CIM presents a significant opportunity to overcome the semantic barriers between existing information islands. The CIM can reap advantages from a formal representation of knowledge in order to support complex processes. So far, ontologies like the CIM and semantic analytics tools have primarily focused on static data, in the sense that entities represented in these ontologies do not change over time. However, in real context, temporal data are often critical to analyze because they refer to evolving phenomena. As a consequence, managing temporal data are necessary for an effective application of ontologies and semantic technologies in the power industry.

In this paper, we investigate the use of ontologies and semantic technologies to support the storage and the extraction of temporal data in industrial context. Two significant issues have been explored: a versioning mechanism for the RDF data and a method to extract the temporal datasets, successfully applied to several sources in a transparent way for the end user. To the best of our knowledge, despite the proposal of different approaches to support RDF versioning [4], and capture and monitor changes [5], a real application on large scale problems was still missing. We describe concrete examples from the power industry.

This paper is divided as follows: Section 2 presents a non exhaustive literature review of recent ontology evolution and versioning techniques. Section 3 describes the context where our research occurs. Section 4 presents the basis for storing RDF versions in triple store according to previous work. In Section 5, we describe our experimental results and we present an application that combines a semantic triple store and a time series database. Finally, we end with a conclusion and future work.

II. BACKGROUND

This section reviews the existing work on ontology evolution and versioning. Most of the studies on ontology versioning focus on the validity, interoperability and management of all versions. OntoView [6] allows ontology engineers to compare versions of an ontology and to specify how the ontology concepts in two versions are related. SemVersion [7] is an RDF-based ontology versioning system that supports query answering across multiple versions and the differences between arbitrary versions. PromptDiff [8] compares different versions

 ISBN: 978-1-61208-420-6

of ontologies using heuristics, and provides the user with their deltas. However, all these versioning systems store all snapshots in a repository so that the deltas between versions must be recomputed on the fly whenever the change information is required. In other words, they do not consider the space overhead in supporting versions. That is, if we redundantly store every version in a separate storage space, the space requirement would be enormous, especially in a large scale ontology system. Furthermore, this approach also has a limitation in that it recalculates the changes between versions whenever the user queries the ontology.

Tzitzikas et al. [9] focus on the storage space in the RDF repositories and propose a storage index, called Partial Order Index (POI), which provides an efficient RDF version insertion algorithm in main memory. Since this storage scheme is based on partial orders of triple sets, it is the most efficient for storage space in which the new version is a subset or superset of the existing versions. However, the new version cannot be a subset or superset when it has both added and removed triples compared to the existing versions. In addition, in order to construct a specific version, it needs to traverse all the ancestor elements of the given element in the POI. Thus, it is not scalable as the data size increases.

Recently, IM et al. [10] proposed a versioning framework for the RDF data model based on relational databases. This scheme stores the original RDF version and the deltas between each two consecutive versions. They store the deltas separately in a *delete* and *insert* tables, and construct a logical version on the fly using SQL statements that join the version from the original version and the relevant delta tables. The proposed framework is promising but needs to be implemented in a relational database and not in a triple store. Therefore, inference and reasoning is not directly allowed on all versions.

III. CONTEXT

In this work, our application context is the Hydro-Québec Distribution network Division (HQD). We used four heterogeneous datasets from the HQD systems. These systems are IRD (French acronym for Inventory of Distribution Network), GSS (French acronym for Underground Structure Management System), GIS (Geographic Information System) and SAP-BW (SAP Business Information Warehouse). The four systems contain data on the distribution network, such as: connectivity, equipment, geographical position, electrical characteristics, etc.

The data of the four systems is not static, but rather changes as a function of time. For each database, a new data dump takes place every weekend. Hence, every week, all the four systems are updated with new datasets. In accordance with the weekly updates, we map the four relational databases to the CIM ontology and *incrementally* export the resulting triples into Oracle 12c RDF Semantic Graph (OSG) triple store. The export process is done by using the *D2R dump-rdf* tool [11]. All the RDF data is bulk loaded into the OSG triple store. The latter was installed on a HP Xeon E7-2830 (2.13 GHz, 8 cores with hyper-threading) processors with 2TB of RAM and 4 ioDrive2 flash block devices of 1.2TB each managed with Oracle ASM. OSG is a secure and scalable platform that supports large RDF graphs of billions of triples and includes capabilities for using forward-chaining inference via RDFS, RDFS++, OWL-SIF, OWL-Prime, OWL2-RL and user-defined rules. It also supports parallel queries.

The weekly stream of new data generates a huge volume (more than 200,000,000 triples) of data and leads to an increase in the complexity of processing. To make valuable business decisions over changing and evolving databases, we decided at the research center of Hydro-Québec (IREQ) to build an architecture capable of dealing efficiently with a vast amount of heterogeneous time series. We developed a set of tools to allow non IT experts to extract and process the time series.

In the next section, we will test three versioning mechanisms, and we will share our experiences on effective means of building a semantic application for heterogeneous time series.

IV. RDF VERSIONING MANAGEMENT IN A TRIPLE STORE

In this section, we do not propose a new RDF versioning system, but rather the basis for storing RDF versions in a triple store according to previous work presented in Section II.

In the context of an RDF triple store, there are a number of ways in which to implement a version control system. A primary choice and probably the simplest is to store the different versions in separate spaces. This approach called the *All Snapshots approach* [12] is very effective for querying but requires excessive storage space. A second choice is to use a *Delta-Based approach* to overcome the excessive space requirements of the All Snapshots approach. Two kinds of delta can be implemented: the *Sequential Delta* [12] and the *Aggregated Delta* [10].

The Sequential Delta approach consists in storing an original version and the delta of each subsequent version separately. Formally, given the original version V_i, let V_{i+1} be the logical version and $\Delta_{i,i+1}$ be the set of change operations between V_i and V_{i+1}. Then, V_{i+1} can be represented as follow:

$$[V_{i+1} = \Delta_{i,i+1}(V_i)] \tag{1}$$

Thus, in order to access a specific logical version, we must construct the logical version on the fly by applying the deltas between the original version and the logical version.

Instead of executing all the in-between deltas in sequence, the aggregated delta can create a logical version directly by storing all of the possible deltas in advance. In other words, given a sequential delta $\Delta_{i,i+1}, \Delta_{i+1,i+2}, ..., \Delta_{j-1,j}$ between V_i and V_{i+1}, an aggregated delta is defined as follow:

$$[\sum_{n=i}^{j-1} \Delta_{n,n+1} = \sum_{n=i}^{j-1} \Delta^{-}_{n,n+1} \cup \sum_{n=i}^{j-1} \Delta^{+}_{n,n+1}, (i < j)] \tag{2}$$

$$[\Delta_{g(i,j)} = \sum_{n=i}^{j-1} \Delta_{n,n+1} - C_t] \tag{3}$$

Where C_t is the set of change operations with overlapped triples in all stored delta.

V. EXPERIMENTS

A. Experimental settings

We implemented all the version schemes in OSG. Table I summarizes the characteristics of our real data sets. The datasets used in the experiments have between 35,000,000 and 125,000,000 triples.

TABLE I. SIZE OF DATASETS

	IRD	GSS	GIS	SAP-BW
#triples	100,000,000	3,560,230	90,155,000	125,236,540

B. Versioning and monitoring temporal data changes

In this section, we compare the performance of three RDF version management methods for our application context: the All Snapshots approach (as used in SemVersion [7]), the Sequential Delta (based on the change detection between consecutive versions) and the Aggregated Delta.

For this part of the experiment we consider only the power transformer equipments from the IRD dataset. This category of equipments represent more than 600,000 equipments in the distribution network, and the delta, the difference between each week, is less than 1%.

Figure 1 shows the number of triples required to store power transformers for each RDF versioning approach. The number of triples of each scheme includes the total number of triples in all the versions and, if any, all the deltas schema. In Sequential Delta and Aggregated Delta, we consider the first version as the original version. With the All Snapshots approach, as shown in Figure 1, the number of triples increases linearly as the number of versions increases, since this scheme stores all the version snapshots in the triple store. In contrast, the Sequential Delta and Aggregated Delta approaches require less triples than the All Snapshots, because they store only the original version and the deltas. When we compare the Sequential Delta and the Aggregated Delta to each other, we notice in Figure 2 that the Aggregated Delta requires more triples than the Sequential Delta. This is because there are duplicated triples stored by the aggregated delta procedure. In terms of number of triples, the Sequential Delta procedure requires less storage space than the Aggregated Delta.

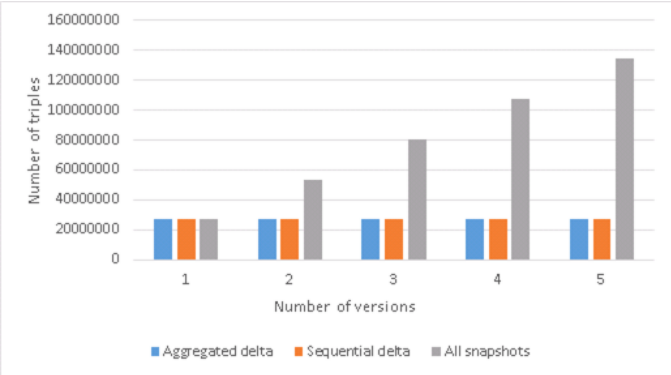

Figure 1. Number of triples for the version management

Figure 3 shows the construction time of versions in the Sequential Delta and the Aggregated Delta. The y-axis represents the construction time of versions in seconds, and the x-axis denotes the specific versions to be constructed. In order to generate versions which are not stored physically, we need to construct logical versions on the fly from the original version. As shown in Figure 3, while the construction time in the Sequential Delta is proportional to the number of versions we need to trace backwards, the Aggregated Delta can compute any specific version at almost a constant time. This is because

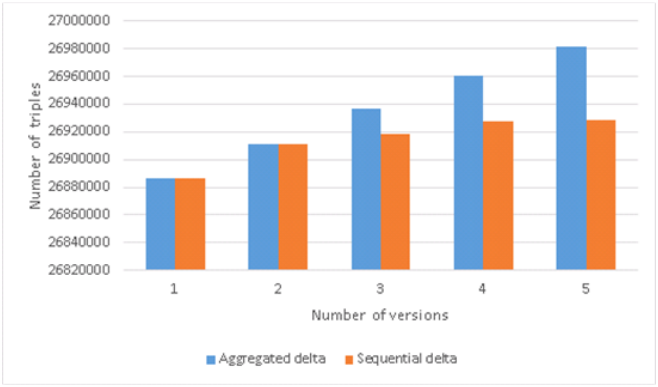

Figure 2. Number of triples for the Sequential Delta vs the Aggregated Delta

the Aggregated Delta recreates any version by applying only a corresponding aggregated delta to the original version. Since the relevant version needs to be constructed on the fly for a given query and it occurs very frequently, the Sequential Delta performance is very critical.

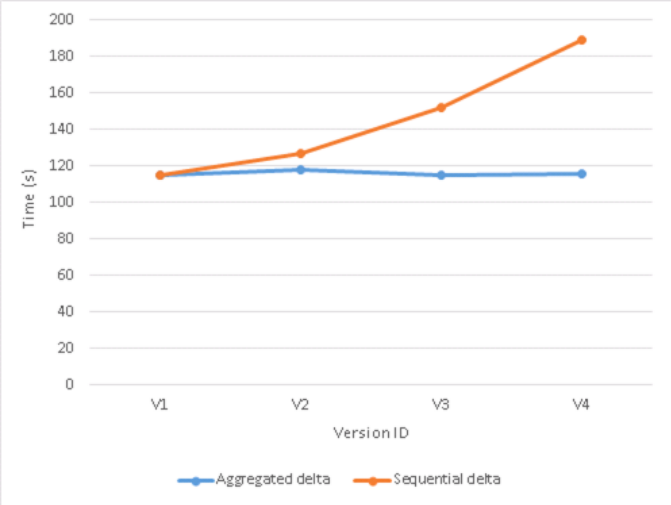

Figure 3. Construction time for the Delta-Based versioning approaches

We also evaluated the query performance of various version storage schemes using the queries in Table II. All the queries are in SPARQL. The SPARQL Query Language is a language for querying the RDF data [13]. Basically, the queries in Table II simply count the number of power transformers for a specific version. Figure 4 shows the average response time of ten executions of the queries of Table II. We notice that the All Snapshots approach is superior to both Delta-Based methods for these queries except for the first version. This can be easily explained by the fact that the sample queries used here require the computation in a specific version and the All Snapshots approach physically stores all the versions. With the Delta-Based approaches, we first need to construct the version on the fly and then query against it.

C. Managing heterogeneous time series

With the advent of the smart meters in the distribution network, power system engineers and utility operators began

TABLE II. SAMPLE QUERY

Method	Query
All Snapshot	SELECT (count(?s) as ?total) where {?s a cim:PowerTransformer}

Seq. delta	SELECT (count(?s) as ?total) where {
	graph <V0>{ ?s a cim:PowerTransformer}
	minus {graph <delete v3_v4>{ ?s a cim:PowerTransformer}}
	union {graph <insert v3_v4 >{ ?s a cim:PowerTransformer }}
	minus {graph <delete v0_v1 >{ ?s a cim:PowerTransformer}}
	union {graph <insert v0_v1>{ ?s a cim:Terminal}}
	}

Aggre. delta	SELECT (count(?s) as ?total) where {
	graph <V0 >{?s a cim:PowerTransformer}
	minus {graph <delete v0_v4 >{ ?s a cim:PowerTransformer}}
	union {graph <insert v0_v4>{ ?s a cim:PowerTransformer}}
	}

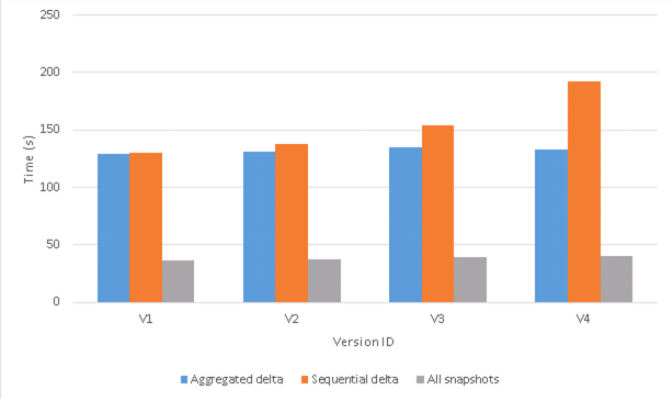

Figure 4. Computation time for the version management.

to extensively study the electrical measures of smart meters. In the province of Québec, there will be a total of 4 million smart meters installed. To make things even more challenging, one smart meter can produce a dozen measurements per customer in a short period of time (approximately every 15 minutes). In fact, it can record dozens of values such as the voltage, the current and the energy consumption. Thus, the huge amount of data newly generated has to be computed in order to make it highly accessible and available for electrical engineers to conduct electrical studies.

One way to deal with the vast volume of data generated by smart meters is to store the time series in a specialized infrastructure. We used the PI historian by OSIsoft for the management of real-time data and events. The PI system is widely used in the power industry.

As stated previously, the data related to the power distribution network, such as the equipment connectivity, the equipment location and their characteristics is stored in the OSG semantic database. On the other hand, the measurement data is stored in the PI historian. The latter is a real-time data infrastructure solution that can capture, store and analyze real-time data. It intends to deal with a massive amount of data while being able to offer a good velocity. At IREQ, we use the PI historian as the main repository for time series such as electrical measurements.

To take advantage of PI scalability and OSG flexibility, it becomes important to bind the two technologies. In fact, OSG is not optimized to efficiently store time series, and PI has not

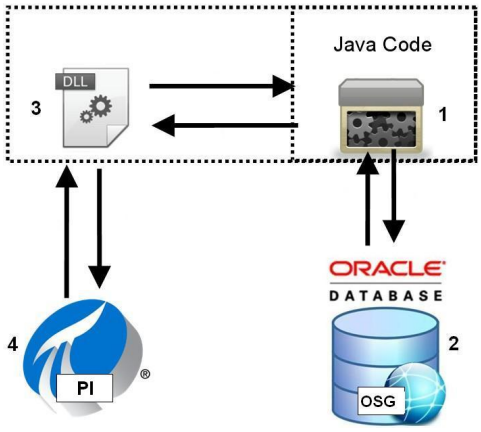

Figure 5. OSG and PI application

an evolved inference engine as the semantic data base. Thus, the data of the distribution network has to remain in OSG and the measurements in PI.

To bridge the gap between the distribution network data and measurement values, we have decided to develop an application that combines the power of both OSG and PI. In Figure 5, we present a high level view of our application composed by four modules, as follows.

The *Java Code* module (1) is responsible for processing SPARQL user queries. It can read and alter the user queries in order to detect which data are located in external data sources. We use the Dublin Core Metadata Initiative (DCMI) [14] and W3C Provenance meta-ontology [15] to inform where the data is located and how to extract it from external data sources. The *Oracle Semantic Graph Triplestore* (2) maintains the ontology snapshot and the meta-ontology of the distribution network. The *DLL module* (3) is a C# *DLL COM service* that behaves as web services. It receives custom queries from the Java code module and extracts values from the *PI module* (4). The latter receives a query from the DLL module with the parameters of the smart meters identifiers and the time duration of the measurement values to extract.

To help the reader understand how the two data sources are binded, we describe step by step the execution trace of our system. We show how the SPARQL user query is shared between the data sources and how the results are merged.
Step 1: The Java code receives an initial query from the user (see Table III).
Step 2: The original user query is altered and optional triples are added to detect if the data is located within external data sources via the property *cim:UsagePoint* (see Table III).
Step 3: The external data sources are described using the DCMI and W3C-Provenance ontology.
Step 4: A subset of the ontology extracted from OSG contains part of the user query results and the related metadata.
Step 5: The Java code reads and analyzes the metadata. The latter describes how to extract the external data and what to extract (i.e., the PI tag names). As a consequence, the Java code sends a compact query to the DLL component with the following parameters: the beginning date, the duration (in days), the measurement type (the voltage in the current example) and the PI tag names (see Table III). The PI tag

names are the unique identifiers for the smart meters and the time duration represents the beginning and the end-time markers of the measurement values.
Step 6: The DLL component queries the PI infrastructure and extracts the related measurement values.
Step 7: The DLL component formats the values and sends them back to the Java code. The data is sructured in order to reduce the amount of data transmitted between the DLL component and the Java code (see Table III).
Final Step 8: The Java code converts the received values into an RDF ontology and merges it with the resulting ontology of Step 2. The original user query is applied to the newly merged ontology and the result is returned to the user.

TABLE III. QUERY EXECUTION TRACES

Step	Query
1	SELECT ?c ?v where { ?c cim:MeterReading.IntervalBlocks ?iB. ?iB cim:IntervalBlock.IntervalReadings ?iR; rdf:label "Volts". ?iR cim:IntervalReading.value ?v.}
2	CONSTRUCT { ?c cim:UsagePoint ?p. ?v cim:UsagePoint ?p. } where { ?c cim:MeterReading.IntervalBlocks ?iB. ?iB cim:IntervalBlock.IntervalReadings ?iR; rdf:label "Volts". ?iR cim:IntervalReading.value ?v.} OPTIONAL { ?c a cim:IntervalReading; cim:UsagePoint ?p. } OPTIONAL { ?v a cim:IntervalReading; cim:UsagePoint ?p. } }
5	query:[2013/01/01 12:00; 1; [V]; [smartMeter1,smartMeter2,...]] {
7	smartMeter1:[V;[2013/01/01 12:00,...,2013/01/02 12:00];[220, 219, ...]] smartMeter2:[V;[2013/01/01 12:00,...,2013/01/02 12:00];[219, 219, ...]]

We embedded our application with RapidMiner [16], a code free modern analytics platform that includes machine learning, data mining, text mining, predictive analytics and business analytics. According to the 15th annual KDnuggets Software Poll [17], released in 2014, RapidMiner remains the most-used free dataming tool. Data mining is the process of analyzing and turning large collections of data into useful knowledge. It can be seen as a natural evolution of information technology, where huge volumes of data accumulated in databases are analyzed, classified and characterized over time.

In Figure 6, we can see the visual interface of the RapidMiner extension. The *begin date* and the *end date* inform about the time duration and the query window allows the user to edit a SPARQL query. We tested our RapidMiner extension by detecting via a K-Nearest Neighbors (K-NN) all the outlier voltage measurements for a subset of customers during one week. The K-NN Global Anomaly Score assigns an anomaly score to each instance prior to the distance between the instance and a K number of neighbors. The higher is the score and the more likely the instance is an outlier. Visually, we can see in Figure 7 that some voltage measurements were detected by the K-NN algorithm as outliers; the bubbles size are proportional to the outlier score.

By combining RapidMiner to a temporal RDF ontology, our goal is to support advanced analytics process and to transform data into actionable insights. In fact, data mining tools offer methods and algorithms that help organizations analyzing large amount of data in order to extract valuable knowledge. For Hydro-Québec, analyzing complex situations and identifying the best solutions for forecasting demand,

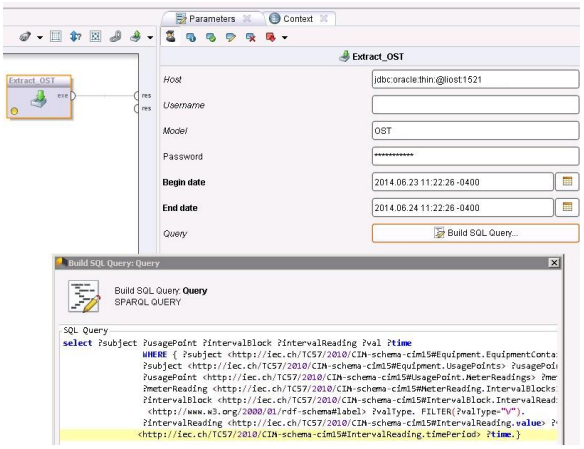

Figure 6. RapidMiner extension

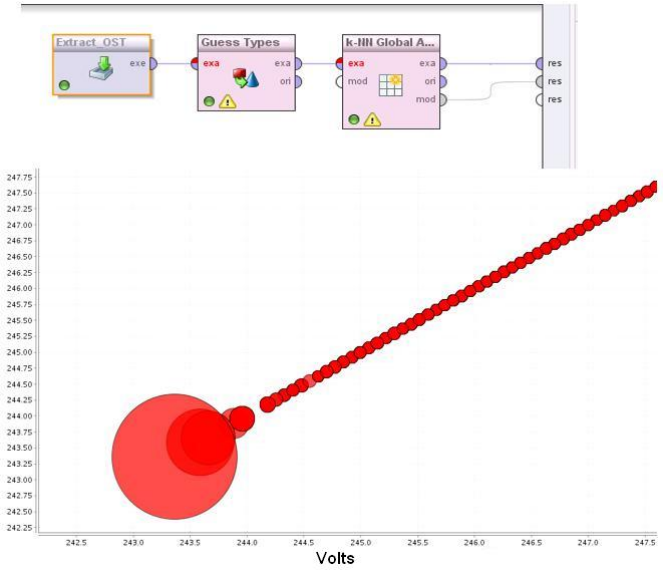

Figure 7. Voltage outliers detection

shaping customer usage patterns, preventing outages, optimizing assets and more is extremely valuable. Transforming a high volume of data into valuable decisions becomes a reality for any business that intends to succeed.

VI. CONCLUSION

The obtained results are promising and highlight the potential of using an ontology-based approach in an industrial context like electric power utilities. In addition to federate heterogeneous data sources across multiple enterprise systems, the semantic architecture proposed by IREQ goes well beyond that. The use of semantic technologies and versioning offers a new way of analyzing information. It gives a better idea on how information changes and evolves over the time. In fact, when heterogeneous data sources are adequately federated and versioned, it becomes possible to monitor the changes between the data sources and to take corrective actions when required.

Finally, the use of a common semantic model enables additional valuable information and knowledge to be inferred

 ISBN: 978-1-61208-420-6

and extracted. The number of databases and information systems in use by utilities reveals the importance of a common language and semantic. This is particularly true for electric power utilities where information is growing fast and will continue to increase because of the introduction of smart grid technologies.

As future work, we are planning to improve the RDF versioning system in order to be efficient both in time and space needed for storage. We also plan to include No-SQL (Hadoop, Cassandra, etc.) data sources in our federated approach.

ACKNOWLEDGMENT

The authors would like to thank all the individuals who participated in the design and development of the architecture.

REFERENCES

[1] A. Zinflou, M. Gaha, A. Bouffard, L. Vouligny, C. Langheit, and M. Viau, "Application of an ontology-based and rule-based model in electric power utilities," in 2013 IEEE Seventh International Conference on Semantic Computing, Irvine, CA, USA, September 16-18, 2013, 2013, pp. 405–411.

[2] M. Gaha, A. Zinflou, C. Langheit, A. Bouffard, M. Viau, and L. Vouligny, "An ontology-based reasoning approach for electric power utilities," in Web Reasoning and Rule Systems - 7th International Conference, RR 2013, Mannheim, Germany, July 27-29, 2013. Proceedings, 2013, pp. 95–108.

[3] W. W. Group. Cim primer for network models. [Online]. Available: http://cimug.ucaiug.org/default.aspx [retrieved: 04, 2015]

[4] N. Popitsch and B. Haslhofer, "Dsnotify - a solution for event detection and link maintenance in dynamic datasets." J. Web Sem., vol. 9, no. 3, 2011, pp. 266–283.

[5] T. Käfer, J. Umbrich, A. Hogan, and A. Polleres, "Dyldo: Towards a dynamic linked data observatory." in LDOW, ser. CEUR Workshop Proceedings, C. Bizer, T. Heath, T. Berners-Lee, and M. Hausenblas, Eds., vol. 937. CEUR-WS.org, 2012.

[6] M. Klein, D. Fensel, A. Kiryakov, and D. Ognyanov, "Ontology versioning and change detection on the web," in Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web, ser. Lecture Notes in Computer Science, A. Gómez-Pérez and V. Benjamins, Eds. Springer Berlin Heidelberg, 2002, vol. 2473, pp. 197–212.

[7] M. Völkel and T. Groza, "Semversion: Rdf-based ontology versioning system," in In Proceedings of the IADIS International Conference WWW/Internet 2006 (ICWI 2006), 2006. [VKZ + 05, 2006.

[8] N. F. Noy and M. A. Musen, "Promptdiff: A fixed-point algorithm for comparing ontology versions," in Eighteenth National Conference on Artificial Intelligence (AAAI-2002), 2002, pp. 744–750.

[9] Y. Tzitzikas, Y. Theoharis, and D. Andreou, "On storage policies for semantic web repositories that support versioning," in The Semantic Web: Research and Applications, ser. Lecture Notes in Computer Science, S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, Eds. Springer Berlin Heidelberg, 2008, vol. 5021, pp. 705–719.

[10] D.-H. IM, S.-W. Lee, and H.-J. Kim, "A version management framework for rdf triple stores," International Journal of Software Engineering and Knowledge Engineering, vol. 22, no. 01, 2012, pp. 85–106.

[11] R. Cyganiak. Accessing relational databases as virtual rdf graphs. [Online]. Available: http://d2rq.org/ [retrieved: 04, 2015]

[12] D. Zeginis, Y. Tzitzikas, and V. Christophides, "On the foundations of computing deltas between rdf models," in The Semantic Web, ser. Lecture Notes in Computer Science, K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, Eds. Springer Berlin Heidelberg, 2007, vol. 4825, pp. 637–651.

[13] A. Seaborne and E. Prud'hommeaux, "SPARQL query language for RDF," W3C, W3C Recommendation, January 2008, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[14] D. Core. Dublin core metadata initiative. [Online]. Available: http://dublincore.org/ [retrieved: 04, 2015]

[15] W. W. Group. An overview of the prov family of documents. [Online]. Available: http://www.w3.org/TR/prov-overview/ [retrieved: 04, 2015]

[16] RapidMnier. Rapidminer - analytics for anyone. [Online]. Available: https://rapidminer.com/ [retrieved: 04, 2015]

[17] G. Piatetsky. Kdnuggets 15th annual analytics, data mining, data science software poll: Rapidminer continues to lead. [Online]. Available: http://www.kdnuggets.com/2014/06/kdnuggets-annual-software-poll-rapidminer-continues-lead.html [retrieved: 04, 2015]

 ISBN: 978-1-61208-420-6