SmarT INsiGhts (STING) - An Intelligence Application for Querying Heterogeneous Databases

Vikash Kumar, Ganesh Selvaraj, Andy Shin, Paulo Gottgtroy

Inland Revenue, New Zealand
firstname.lastname@ird.govt.nz

Abstract. This paper presents an application implemented at Inland Revenue, New Zealand that enables users to access data seamlessly from different types of databases. It was developed to provide our investigators the ability to get comprehensive information about a business entity or a group of entities. The solution presented so far has been implemented using relational, semantic and document databases. It allows its users to pose fairly complex queries on the mentioned databases and retrieve results without having to know the specific query languages. Up to this point, it supports the presentation of results in graph and tabular formats.

Keywords: Semantic Database, Enterprise Semantics, Heterogeneous Database Access, Semantic Intelligence

1 Introduction

One of the advantages of semantic technologies has been to bridge the silos of data in a typical enterprise architecture. Many enterprises are, for various reasons, still uncomfortable with the idea of completely wrapping their data via a semantic access layer. Some of their concerns arise from having to re-engineer or completely lose out on the host of functionalities already developed upon traditional relational databases. In this paper, we present a real example of how we successfully implemented an application that queries relational, semantic and document databases seamlessly depending on the nature of the query. This implementation does not replace the existing relational database but extends it by pairing it with the semantic and document databases. The resulting application paved the way to make use of the best features of each type of databases enabling efficient key, pattern and text based queries on the available data. The presented application called SmarT INsiGhts (STING)[1] is meant for investigators of Inland Revenue, New Zealand (IR-NZ) for analyzing transactions (events) received from an entity (company) or a group of entities. The main goal of STING is to provide a simple way of accessing a variety of information about these entities. It is achieved by enabling the investigators to answer simple or complex queries through a single interface.

[1] Demonstration video of the Application: http://youtu.be/enVGpQTwusE

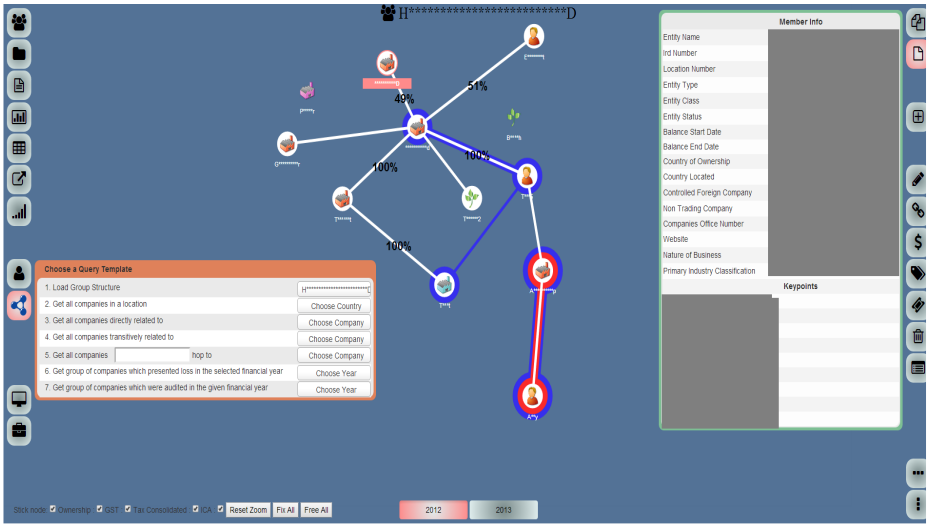

Fig. 1. The STING Application

2 The STING Application

In this section, we describe the STING application and its use in our business along with the details about its architecture and the underlying technologies.

2.1 Motivation

When receiving any transaction from customers, IR-NZ needs to assess the level of risk associated with the transaction. Understanding how companies and group of companies are organized for tax purposes is one of the critical pieces of information that IR-NZ needs to have. The application presented in this paper (Figure 1[2]) helps in visualizing such comprehensive information gathered from multiple sources in interactive graph and tabular representations.

2.2 Architecture

A high level conceptual architecture of STING is presented in Figure 2. The importance of this architecture lies in its model-driven hybrid backend that consists of the following modules:

Common Domain Model This is a java based abstraction layer for all the underlying databases. The main purpose of this layer is to help software developers in building applications based on the data abstraction without worrying about the underlying knowledge representation and implementation database.

[2] For privacy and security considerations, actual data has been masked in this figure and in the demo video

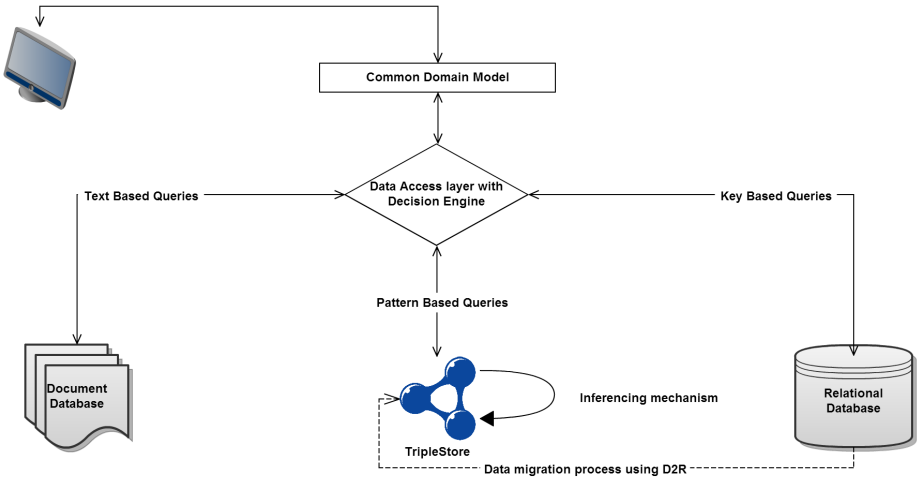

Fig. 2. High Level Conceptual Architecture of the System

The Data Access Layer (DAL) and Decision Engine DAL helps in performing CRUD (Create, Read, Update, Delete) operations on the databases. Since there are several database implementations, this layer is powered by a decision engine that decides on which database a query should be executed. The decision engine is currently working based on several predefined static criteria. The reason for using the static predefined criteria was because of our fixed and well defined business domain. The key features and operations of each database are mentioned in Table 1. The additional advantage of this architecture is its ability to integrate diverse sources of data while maintaining a good performance as seen in this application.

	Relational Database	**Semantic Database**	**Document Database**
Query Type	Key based queries *Example: Get companies with Name = "XX"*	Pattern based queries *Example: Get companies which have risk patterns similar to company "XX"*	Text based queries *Example: Get all documents which has a text containing "XX"*
Schema	Schema constrained data	Uses Ontology but not constrained. Anyone can say Anything on Any topic (AAA)	Flexible Schema
Usage	Existing applications keep working on this.	Data gets migrated in a batch. Reasoning & pattern search is done here.	Data gets migrated in a batch. Text search is done here.

Table 1. Key Features of the Used Databases

STING's Query Approach The application is developed based on a "Single Page" concept where modules are used to allow not only customization of the application but also the usage of docks that allow access to the application's functionalities without impacting on its usability aspect. The query functionality follows this principle by allowing users to pose semi-customized queries in natural language to the system through the selection of a dock icon.

Based on past experiences, we compiled a list of most common queries that an investigator might be interested in and presented them in natural language through the querying interface. In some cases, a user may make slight changes in a query by clicking on its highlighted area, which provides them with alternative variables as options. Some queries are also adapted based on the context of the user, such as user's location, level of authentication, etc. [1]. Behind the natural language presentation, we have a SPARQL serialization of the query that is posed to the semantic triple store. Parametrized SPARQL 1.1 queries are used for binding the variable parameters chosen by the user.

Semantic queries can therefore be posed using a natural language template or predefined "function buttons" on the user interface. The output from these queries is represented as interactive graph structure (single or overlaid graphs) or in tabular format where the users can expand or contract the entities spawning out of a certain node. Examples of few of our pattern based queries in natural language is shown in Table 2.

The whole system uses an adapted version of the Organization ontology[3], which in turn is mapped to the IR-NZ upper ontology. The IR-NZ upper ontology is a lightweight adaptation of Proton[4] and DOLCE+DnS Ultralite (DUL) ontologies[5]. One of the prime considerations in design of ontology was to be able to record provenance information and to query the 'situation' at a certain point of time in past. The reasoning is limited to answering pattern based queries over entity structure using SPARQL 1.1 features. In future, we plan to integrate more detailed information from external open data sources.

Query ID	**Query Text**
Query 1	*Get all entities which are transitively related to a particular entity.*
Query 2	*Get the group structure for a given group of companies.*
Query 3	*Filter companies of a certain group by Location.*
Query 4	*Find relationships between a chosen set of entities.*

Table 2. Some of Our Sample Semantic Queries

[3] `http://www.w3.org/TR/2014/REC-vocab-org-20140116/`
[4] `http://www.ontotext.com/proton-ontology`
[5] `http://www.loa.istc.cnr.it/ontologies/DUL.owl`

Tools and Technologies The User Interface is a single page web application built using Angular.js[6]. D3.js[7] is used to visualize information as graph in Scalar Vector Graphics . The graph is formatted using a customized force layout method based on Barnes-Hut Algorithm [2]. Node.Js[8] is a JavaScript framework that is used as web server enabling asynchronous calls from the I/O to the databases. OWLIM Lite (version 5.4) [3] is used as our triplestore and Sesame (version 2.7.3) API is used to access information from OWLIM Lite. D2RQ[9] (version 0.8.1) is used to generate RDF dump from an existing relational database. We are still evaluating the document database. Our choice would be mongoDb for its powerful features and wide community support. For this application, we used spring data for mongoDb[10] and pymongo[11] libraries to load and query the mongoDb.

3 Conclusions

In this paper, we presented an application intended to help investigators in assessing risk profiles of entities by answering their queries from different databases. SQL, SPARQL and text-based queries are posed asynchronously on the respective databases depending on the nature of a query. The architecture allows for an unhindered user experience bereft of any noticeable latency issues. We showed how a specific application can use best of different worlds in answering queries without compromising its existing infrastructure. During the demo, we will also share our lessons learned in using different technologies along with the semantic technology.

Acknowledgments This work has been supported by Information, Intelligence and Communications unit of Inland Revenue. We would like to thank the team members of Analytics and Insight for their valuable inputs towards finalization of this Application.

References

1. V. Kumar, A. Fensel and P. Fröhlich. Context Based Adaptation of Semantic Rules in Smart Buildings. The 15th ACM International Conference on Information Integration and Web-based Applications & Services (iiWAS2013). Vienna, Austria. December 2013.
2. J. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. Nature 324 (4): 446-449. December 1986.
3. B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev and R. Velkov. OWLIM: A family of scalable semantic repositories. Semantic Web 2, 1, 33-42. January 2011.

[6] http://www.angularjs.org
[7] http://www.d3js.org
[8] http://www.nodejs.org
[9] http://www.d2rq.org
[10] http://projects.spring.io/spring-data-mongodb/
[11] https://pypi.python.org/pypi/pymongo/