DATA ACCESS THROUGH A DYNAMIC DATA MODEL
A Concept for Accessing Heterogenic Data Structures in RDF Databases

Alexander Wendt, Benjamin Dönz and Dietmar Bruckner
Institute of Computer Technology, Vienna University of Technology, Gusshausstrasse 27-29, A-1040 Vienna, Austria

Keywords: Data model, Ontology, OWL, RDF, Graph, Triple-store, Query, Concept, SPARQL, Protégé.

Abstract: This paper introduces an alternative method for using ontologies to create a dynamic data model for RDF databases or other schema-less databases. The main challenge is how to continuously adapt the data model and its queries to new data, which may be imported with any given structure.

1 INTRODUCTION

When using schema-less NoSQL databases (Levitt, 2010), (Stonebraker, 2010), the possibility to import data in its original structure is opened. Often, this data is imported from flat tables without further normalization. The advantage of such an import is that any type of data can be imported and that the effort of pre-processing data is reduced. Instead, this work is transferred from the import phase to the data access. The challenge in building a relational data model is to create a model, which is robust and general enough to be able to accommodate new data. The challenge in building systems with schema-less databases is to find a way to give the user access to all the data in a heterogenic structure within a single request to the system.

The dynamic data model introduced in this paper provides a method for implementing ontologies as an additional layer between the user interface and a database with heterogenic structure. The challenge and the motivation for developing such a model can be summarized by five requirements:

- For a schema-less database, where it is possible to import data in its original structure.
- It shall be possible to access all data by the user with a simple query.
- The data model shall be exchangeable and expandable at any time.
- It shall be possible for the user to adapt and create new queries if new sources are added or new demands emerge.
- It shall be investigated how ontologies can be used to classify and sort the data in a schema-less database.

2 BACKGROUND

Evaluation of the concept will be done by implementing the dynamic data model on a system, where the mentioned requirements are present.

2.1 System Environment and Data Sources

The evaluation system can be categorized as a decision support system (Shim et al., 2002). Its goal is to provide the user with an overview of the situation of a geographical region in the hypothetical case of a disaster. This will make it possible to estimate consequences of a disaster as well as providing insights for strategic planning.

The system is a web-based solution, where a common data store is accessible by several users via a web interface. The user interface provides possibilities to execute queries and display the results on a map (Google Maps, (n. d.)) and in tabular form. New sources and data structures can be added during operation. For this system, the RDF (Resource Description Framework) database Allegrograph (AllegroGraph, (n. d.)) was used. It supports geospatial queries and several programming interfaces, e.g. Java Jena, Java Sesame, Python and Lisp.

Typical questions in case of an epidemic, which should be answered by combining different data sources in this system are: “In a certain region, how many hospitals are there per inhabitant?” or “What could be the potential economic loss for companies in the grocery branch in a given region, when all fowl in another region would be lost?”. Such

Wendt A., Dönz B. and Bruckner D..
DATA ACCESS THROUGH A DYNAMIC DATA MODEL - A Concept for Accessing Heterogenic Data Structures in RDF Databases.
DOI: 10.5220/0003873704390444
In *Proceedings of the 4th International Conference on Agents and Artificial Intelligence* (ICAART-2012), pages 439-444
ISBN: 978-989-8425-95-9

questions are answered by combining several data sources and by performing calculations within the queries.

The available data sources can be categorized into the following four domains: Political regions, demography, company data and branch data for categorizing companies. The original data sources are collected in six tables. Additionally, political maps are available as shape-files (ESRI, 1998). They are imported from csv files, where each row defines the subject URI, each column the predicate URI and the cell value the object literal. The cell value can also be converted to an own URI.

In this paper, the dynamic data model will be explained with an example of the domain "Company data". Companies have provided financial data like profit, asset and revenue. The example is based on the querying of some attributes of that company data. In the original input table, revenue is available as pairs of "year1" and "revenue1", "year2" and "revenue2" and so on. "Asset" and "Profit" refer to the latest year of revenue, which is "year1".

2.2 Potential Usage of Ontologies

The source data structure uses several predicates and subjects for the same thing, in this case pairs of attributes like "year1" and "revenue1". One of the advantages of ontology-based models is the potential use of reasoners for classifying data. In projects like HarmonISA (HarmonISA, (n. d.)) the task is to classify land types (grassland, forest, sea). The main task for the reasoner is to classify new data from different sources into a skeleton ontology, see (Peedell et al., 2005) based on its attributes. This is used to merge data sources and models and to query the whole system, which contains several models with a single query. In the query, everything that fulfills certain criteria is queried independent of the reference model.

Another application of reasoners is presented in (Fallahi et al., 2008, p. 354). In this service oriented architecture, the reasoner is used for matchmaking of requests to services. Each available service is modeled in an ontology. The requests, which are also modeled in the ontology, are more specialized than the services and are classified into classes by the inference engine. From the potential services, which could fulfill the request, the best match is used for the task.

In our system and in the example with company data, the reasoner could be used to classify companies into e. g. small, middle and large sized companies depending on certain defined criteria. A company could be defined as anything that has some values from the classes "address", "employees" and "revenue". However, in order to do that, the "address" of a company must not be a literal of the class "company", which happens at the import of flat tables (unprocessed in its original form), but it has to be assigned its own class "address". A new layer of hierarchy has to be inserted between the company URI and the actual address values. The flat data structure in the database would have to be normalized like in relational databases; i.e. literals would have to be transformed to URIs. Otherwise, an ontology model would be highly populated with only a few subjects and several predicates connecting them to objects or in this case to literals. As long as the data is not processed, this type of classification does not make much sense here. In order to still be able to combine the heterogenic data sources in flat tables, the dynamic data model was developed.

3 CONCEPT

This data model is based on two main concepts: The creation of artificial classes and the creation of database queries by combining elements of subject-predicate relations. If the content of the model is queried, a new query is generated with the concepts of the model and used for the actual database. The main function of the data model is to provide a flexible way to automatically generate queries for the database by considering the requests of the user. This is done by defining a meta-ontology, which consists of the following classes: *Class*, *SubjectClass*, *QueryConcept*, *SubQueryConcept*, *Group*, *AtomQuery*. The classes and instances in the meta-model are completely separated from the classes and instances in the actual database. The only thing they share is the RDF-database as a storage medium. In Figure 1, the database is shown to the left with a class "Class1", an instance "Instance1", two literals "Literal1" and "Literal2", which are connected with "Instance1" via the predicates "Predicate1" and "Predicate2".

Within the meta-ontology, on the right side of Figure 1, instances of *Class* (Meta-Ontology classes are written in italic) are created, which represent subdomains in the database and are usually defined from the predicates in the database, i.e. the predicates in the actual databases are transferred into instances of *Class* in the model. The instances of *Class* are independent of the real classes in the database (to the left in Figure 1). In Figure 1, the real

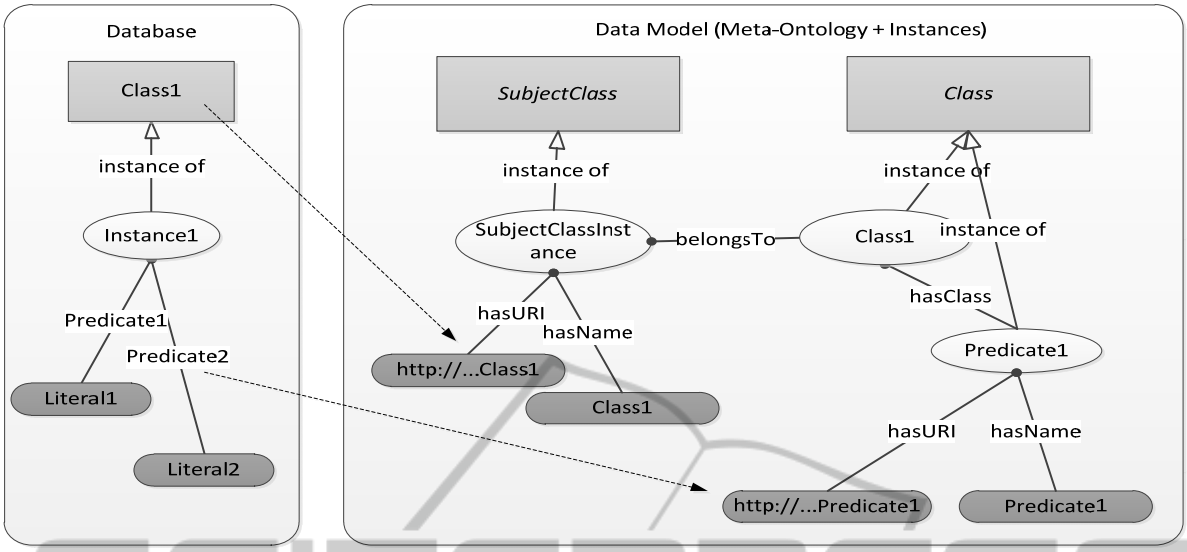

Figure 1: Creation of artificial classes for the data model.

class “Class1” is created as “Class1”, which is an instance of *Class*. *SubjectClass* instances are created, which are mapped to the real existing classes in the database. “Class1” in the database is represented here by the *SubjectClass* instance “SubjectClassInstance”. This is done by adding the URL and name of “Class1” to “SubjectClassInstance”. *SubjectClass* instances are then mapped to the instances of *Class* with the predicate “belongsTo”. In that way, the data model is independent of the existing classes. Further, the predicates in the database e. g. “Predicate1” can also be created as classes. This is useful when bundling data. The instances of *Class* are directly accessible on the user interface. By selecting an instance from *Class* the possibility to execute queries assigned to these instances, is given. The instances of *Class* can be seen as categorizer for queries.

In the company data example, the following instances are created in *Class*: “Organization”, “Finance”, “Revenue”, “Profit” and “Asset”. In the database, the class “Organization” can be found, but not the class “Finance”. “Finance” is created as a new hierarchical layer between the “Organization” and its attributes “Revenue”, “Profit” and “Asset”. “Revenue” is made an instance of *Class*, as it contains pairs of “revenue1” and “year1” and so on. Although the database is flat, the user interface gets a hierarchical structure.

In order to query the data, *QueryConcepts* are defined and assigned to the corresponding instances of *Class*. A *QueryConcept* is a template of the structure of a query for the database. It works like a function with filter parameters as inputs and a SPARQL-query string as output.

The idea of using *QueryConcepts* origins from (Lutz, 2007), (Klien et al., 2004) and (Bügel et al., 2007), where a concept with the same name is applied for service discovery. Services in systems and requirements stated by the user can be described using ontologies. It is possible to compare service descriptions with the requirements’ descriptions of a certain operation. According to (Lutz, 2007, p. 14), domain ontologies contain the primitives of a domain and provide a shared vocabulary for services in the system. Application ontologies, which are derived from a domain ontology, contain necessary constraints for a certain operation of the system. They allow the creation of semantic queries (Lutz, 2007, p. 21), which contain information about requirements and conditions from a certain user request. Semantic advertisements are equivalent to the semantic queries, but are created for each provided service. The semantic queries and semantic advertisements are matched against each other, in order to select the best fitting service for a selected operation. It is possible for a user to select a domain, choose an operation and set constraints for that operation via a *QueryConcept*. In the dynamic data model, only the idea to define a domain and then to select an operation is used.

A *QueryConcept* can be explained bottom-up, starting with the smallest element, the *AtomQuery*. An *AtomQuery* is a representation equal to an SPARQL statement of one subject and predicate (see below). The subject is related to one instance of *SubjectClass*. This is a subject-predicate relation. The constraints for the object are set by filters. An

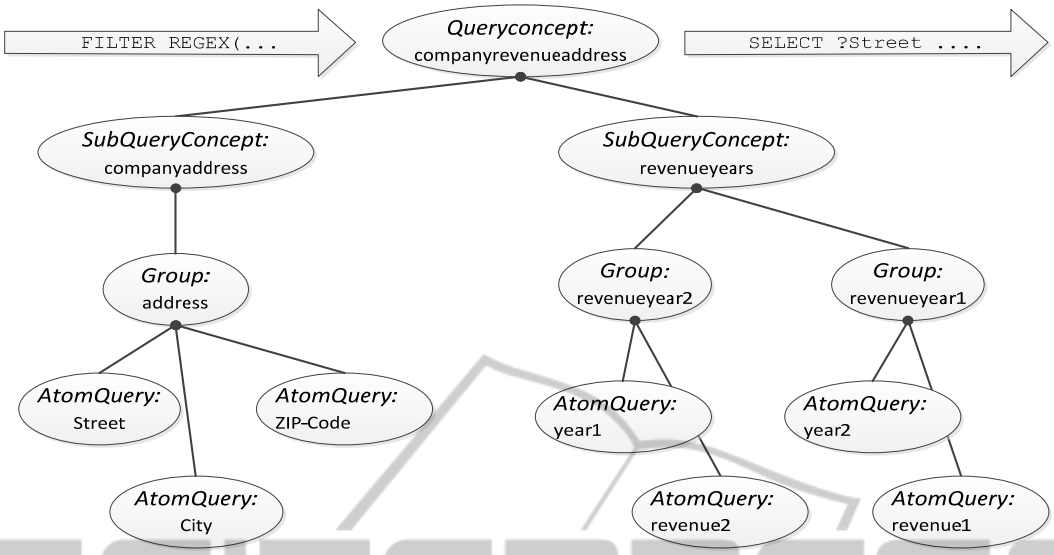

Figure 2: The structure of a *QueryConcept*.

example of the structure of a *QueryConcept* is shown in Figure 2.

In the company data example, the predicate "revenue1" and its domain instance "Company" is represented as an instance of *SubjectClass*, forming such an *AtomQuery*. The *AtomQuery* additionally contains information about the data type of the predicate for use in filters. As seen in the result section, the order of the SPARQL-Statements plays a major role for the performance of the system. Therefore, it is also possible to define a certain order for a statement, which can be adapted for each *QueryConcept*.

One or more *AtomQueries* are assigned to a *Group*. Within a *Group*, each *AtomQuery* is connected by a logical conjunction (AND, in SPARQL ".").

The purpose of the *Group* is to allow parallel querying of the same thing, which is addressed by different predicates.

One or more *Groups* define a *SubQueryConcept*. Within the *SubQueryConcept*, the *Groups* are connected with a logical disjunction (OR, in SPARQL "UNION"). In this way, several *Groups* are queried in parallel but with common result columns. This process simulates the use of a higher normal form in the database. The result format and column names of the executed queries for each group are the same for all groups.

Finally, the *SubQueryConcepts* define the *QueryConcept*, where the *SubQueryConcepts* are connected with a logical conjunction (AND, in SPARQL "."). It allows adding common predicates of parallel groups to the query without having to include them in each group.

The construction of a *QueryConcept* can be demonstrated with the company data example. An *AtomQuery* contains the predicates "year1" and "revenue1". The subject of the statement is an instance of the real class "Company" (in the database). The *AtomQueries* are assigned the *Group* "revenueyear1". The next *Group* "revenueyear2" is made up of "year2" and "revenue2". The *SubQueryConcept* "revenueyears" is a union of the *Group*s "revenueyear1" and "revenueyear2". Another *SubQueryConcept* is the company address "companyaddress", which contains one *Group*. This *Group* contains the *AtomQueries* for "Street", "ZIP-Code" and "City". The *SubQueryConcept*s "companyaddress" and "revenueyears" now define the *QueryConcept* "companyrevenueaddress". The result of the executed *QueryConcept* contains the following columns: "Street", "ZIP-Code", "City", "year" and "revenue". The code parts below, show how a standalone *Atomquery*, *Group*, *SubQueryConcept* and *QueryConcept* could look like:

Atomquery:
```
SELECT ?year WHERE {?company x:year1 ?year}
```

Group:
```
SELECT ?year ?revenue WHERE {?company x:year1 ?year; x:revenue1 ?revenue}
```

SubQueryConcept:
```
SELECT ?year ?revenue WHERE {{?company x:year1 ?year; x:revenue1 ?revenue} UNION {?company x:year2 ?year; x:revenue2 ?revenue}}
```

QueryConcept:

```
SELECT ?name ?revenue ?year WHERE {?company x:name1 ?name. {?company x:year1 ?year; x:revenue1 ?revenue} UNION {?company x:year2 ?year; x:revenue2 ?revenue}}
```

As a consequence of the independence from the actual database structure, it enables the use of several different data models at the same time. The data model is customizable for each user. It allows data to be imported in its original structure. After the import of the original data, where flat tables are imported, the data model can be generated in an OWL-editor like Stanford Protégé (Stanford Protégé, (n. d.)). There, the data model has to be manually adapted for the new data sources. Afterwards, it can be imported into its own namespace and context in the database.

4 IMPLEMENTATION AND RESULTS

The dynamic data model was implemented with an Allegrograph RDF database on a set of about 300 000 Austrian companies. A Java Sesame API was used to access the database. In the software, each class of the meta-ontology was equally modeled in Java. From these classes, the user interface was automatically generated based on the data model. In order to use the model, inputs are given. In the user interface (see Figure 3), the user selects the class (here equivalent to a domain), which contains the requested query. After getting the class of interest, all available *QueryConcept*s are shown. The user selects a *QueryConcept*, fills out the constraints and generates the query. The query is executed in the database returning a result table and/or geographic structures on the map.

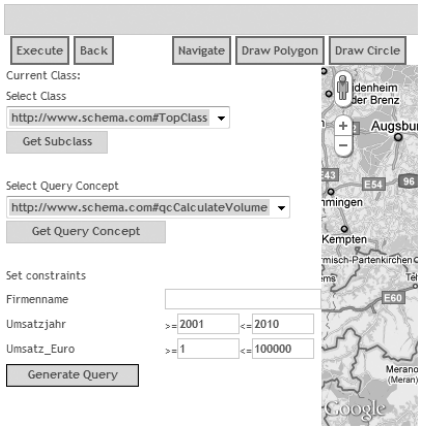

Figure 3: A screenshot of the user interface.

Tests on an Intel® Xeon® 3.0 GHz processor with 4 GB RAM did show that it was possible to construct several types of SPARQL queries, by combining the "building blocks". Also JOINs can be created efficiently. The main issue when using the dynamic data model was performance, but that is an issue for SPARQL and triple stores in general. Often, due to the long query times, the execution time was not satisfactory and the query did not complete. Here, the query complexity was too high due to the use of several joins as well as keywords like ORDER BY. In order to be able to complete a query, the LIMIT keyword had to be used. Furthermore, one of the most important performance factors, which is can be optimized, is the order of the single statements within the query (Stocker et al., 2008), (Vidal et al., 2010). Therefore, statement ordering was considered within the data model. For instance, if the order of the statements was considered, a certain query did complete within 2 s, else it did not complete before execution time-out (4 min). This shows the impact of wrong statement ordering.

The usage of Stanford Protégé as a data model editor has both advantages and drawbacks. Users, which are supposed to modify the model, complain about the large effort, in order to understand how to use the editor and to create queries. On the other side, for a person with basic knowledge about ontologies, Stanford Protégé provides a very fast method for constructing database queries. It is done by populating the data model with instances without the need of knowing SPARQL.

5 CONCLUSIONS AND OUTLOOK

The purpose of the dynamic data model was to fulfil the requirements stated in Section 1. Results showed that it is possible to adapt the data model on a running system when new data is imported into the schema-less database. The user just needs to add the new structure to the model and creates new *SubQueryConcepts* for the corresponding *QueryConcepts* replacing the model in the database. This way, the model is exchangeable and expandable at any time. A user who knows Stanford Protégé or any other OWL editor can add and modify each query by manipulating the instances of the meta-model. Finally, it could be shown that it is not useful to implement the reasoner for this type of data but this approach provides an alternative for accessing data. The main issue is the performance

and the high effort to understand how to use Stanford Protégé. It is possible to optimize the queries by changing the order of the statements.

Further development of the dynamic data model would be to involve functions from other programs or databases, i.e. to let the result table of a *QueryConcept* be the input of another *QueryConcept* containing functions. An *AtomQuery* could contain the link to an operation instead of the link to a predicate in the database. That way, the dynamic data model would be more powerful in calculation and simulation tasks. Another use could be to transport the dynamic data model to a relational database instead of the RDF-Triple store and use it as a query editor. The queries would not be SPARQL, but SQL. This would solve the performance problem and the RDF-Triple store would only be used as storage of the data model.

ACKNOWLEDGEMENTS

This work was partly supported by the FFG (Austrian Research Funding Organization) [819065].

REFERENCES

AllegroGraph (n. d.), *AllegroGraph® RDFStore 4.2.1*. Retrieved June 14, 2011, from http://www.franz.com /agraph/allegrograph

Bügel, U., Hilbring, D., Denzer, R. (2007). *Application of Semantic Services in ORCHESTRA*, at the *International Symposium on Environmental Software Systems (ISESS)*, Prague, May 22-25, 2007, Czech Republic

ESRI, Environmental Systems Research Institute, Inc. (1998). *ESRI Shapefile Technical Description*, an ESRI White Paper—July 1998. Retrieved June 21, 2011, from http://www.esri.com/library/whitepapers/ pdfs/shapefile.pdf, USA

Fallahi, G. R., Frank, A. U., Mesgari, M. S., Rajabifard, A. (2008). An ontological structure for semantic interoperability of GIS and environmental modeling, in *International Journal of Applied Earth Observation and Geoinformation 10 (2008) 342–357*, doi:10.1016/ j.jag.2008.01.001

Goolge Maps (n. d.), *Google Maps*. Retrieved June 21, 2011, from http://maps.google.com

HarmonISA (n. d.), HarmonISA. Retrieved June 21, 2011, from http://www.isamap.info/html/harmonisa.html

Klien, E., Lutz, M., Kuhn, W. (2004), Ontology-based discovery of geographic information services—An application in disaster management, in *Computers, Environment and Urban Systems 30 (2006) 102–123*, doi:10.1016/j.compenvurbsys.2005.04.002

Levitt, N. (2010). Will NoSQL Databases Live Up to Their Promise? in *Computer 43 Issue:2*, doi:10.1109/ MC.2010.58

Lutz, M. (2007). Ontology-Based Descriptions for Semantic Discovery and Composition of Geoprocessing Services, in *Geoinformatica*, 11:1-36, doi:10.1007/s10707-006-7635-9

Peedell, S., Friis-Christensen, A., Schade, S. (2005). *Approaches to Solve Schema Heterogeneity at the European Level*, 11th EC GI & GIS

Shim, J. P., Warkentin, M., Courtney, J. F., Power, D. J., Sharda, R., Carlsson, C. (2002). Past, present, and future of decision support technology, in *Decision Support Systems 33 page 111 –126*, doi:10.1016/S0 167-9236(01)00139-7

Stanford Protégé. (n. d.). welcome to protégé. Retrieved June 21, 2011, from http://protege.stanford.edu

Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., Reynolds, D. (2008). SPARQL Basic Graph Pattern Optimization Using Selectivity Estimation, in *Proceeding of the 17th international conference on World Wide Web WWW 08 (2008)*, page 595-604, ACM Press, ISBN: 9781605580852, doi: 10.1145/136 7497.1367578

Stonebraker, M. (2010). SQL databases v. NoSQL databases, in *Communications of the ACM Volume 53 Issue 4*, New York, NY, USA

Vidal, M., Ruckhaus, E., Lampo, T., Martínez, A., Sierra, J., Polleres, A. (2010). On the efficiency of joining group patterns in SPARQL queries, in *Proceedings of the 7th European Semantic Web Conference (ESWC2010)*, Heraklion, Greece. Springer