Report: Visualizing and Mapping Ontologies in Ultrawrap

Cabrera, Guillermo
University of Texas at Austin
gcabrera@cs.utexas.edu

ABSTRACT

Details on the implementation of the graphical user interface developed for the Ultrawrap project. The Google Web Toolkit was used to develop the web application in conjunction with Jena as the RDF API. The main objective of this work was that of presenting an interface to the user for him/her to be able to visualize the extracted schema from a relational database, as well as the putative ontology generated from this schema. Finally, we explore possibilities in ontology visualization and mapping.

Keywords

Semantic Web, Google Web Toolkit, Jena, Graphical User Interface, ontology visualization, ontology mapping.

1. INTRODUCTION

The vision of the Semantic Web in which people are able to connect and share data, first and foremost requires that the data be easily available and that we have a lot of data. In past years there has been an effort by Semantic Web evangelists including Tim Berners Lee to encourage a wide array of entities (ex. government and industry) to put data on the web in formats adherent to the Semantic Web; this has had a big impact in many fields, and was very evident in aftermath of the Haiti earthquake (2009) where mappers from all over the world with the help of satellite imagery and the OpenStreetMap[9] project were able to map roads, refugee camps, hospitals, etc. thus, being the number one source for aid workers during rescue efforts.

As we put more of this data on the web, efficiently storing it so that it can be easily retrieved later using queries can be a problem. This reason has set forth the ongoing work on triple stores [6] [1]; a framework used for storing and querying RDF data and providing mechanisms for persistent store and access. These triple stores are normally divided into: in memory, native and non native, of which the native approach has gained popularity due to good results on load times.

Unlike native stores which re-implement a databases engine, non-native stores take a layered approach whereby an existing database engines is used and then a query engine (SPARQL) specific to RDF data is connected to the relational database engine. An example of such approach is seen in [11]. Ultrawrap trivializes the process of putting data on the Semantic Web by using the existing SQL infrastructure to allow it to be queried using SPARQL.

In order to have the Relational Database (RDB) ready to accept SPARQL queries a few processes need to take place: First, the schema information for the database is retrieved and used to create a Putative Ontology (PO). Then, this PO is used to create views representing the triples of the data in the RDB. With these views, the RDB is now ready to accept SPARQL queries, by using a SPARQL to SQL translator and finally relying on the existing SQL optimizer to select the most efficient query plan to access data in RDB.

At this point, the PO is available in the Semantic Web, however, one must remember that this PO was generated based on the schema for the RDB which might not make use of a standard vocabulary, thus, limiting its visibility on the semantic web. This where Domain Ontologies (DO) are of importance, in that they define a standard vocabulary for a certain discipline, as such, individuals working in that discipline can model or map their data using the DO. For instance, let us take a RDB schema from a small social networking site and assume it has a table named "individual" where it has attributes fName and lName, this means that somebody running a SPARQL query against the PO generated from this schema must be aware of such names, a difficult task for someone working outside of the social networking site. Nevertheless, if the administrator maps its PO to a DO such as Friend Of A Friend (FOAF) that defines a dictionary of terms for people and the things they make and do [4], more people could query and retrieve information given that they also know of the FOAF vocabulary.

Doing a mapping as in the case above could be done automatically, yet, this raises interesting problems which will briefly be discussed in following sections. Thus, it is important that the administrator take part in this mapping process since it is him/her who best knows the data and whether or not it maps to a certain concept in a DO. To do so, an administrator must be able to visualize the ontologies and then perform the necessary mappings. In addition, an administrator or organization might be constrained in the type of data it can publish and might need to filter out tables or attributes that should not be part of the PO for privacy or security issues.

In this project we focus on a Graphical User Interface (GUI) that serves as an entry point for an administrator to interact with the Ultrawrap project; through this GUI we provide a way to visualize the RDB schema, filter out tables and attributes, visualize ontologies and manually make changes to the PO based on a DO. The GUI takes the form of a web application developed using Google's Web Toolkit (GWT) and the Jena API used to work with RDF data.

In the following sections, I will briefly talk on similar approaches, the process of mapping ontologies, and then expand on the GUI's details including implementation, features, some challenges encountered in the development process. Finally, I give an update on the current status and recommendations for future work on this project

2. RELATED WORK

D2R Map [2] is another project that takes advantage of the high amount of data stored in RDBs and tries to link it to DO. In its Starting in version 0.4 released on November 2007, they make

available a J2EE web application that lets one traverse the contents of the RDB by navigating through the RDF data in a regular HTML atmosphere. As for mapping, the authors state that subsisting names in the generated RDF model from the RDB with names in a DO could be done, yet, it is unclear if this is meant to be an automatic or manual process.

In addition, Protégé [10] offers a rich environment for ontology visualization and management; it is capable of merging ontologies based on different parameters, so that a new ontology is produced that could be the result of merging a PO with a DO.

A recurrent problem with both of these systems goes into ontology mapping which is a field by itself and where different approaches are taken to increase the accuracy of links between different ontologies.

3. ONTOLOGY MAPPING

There are many approaches towards matching two or more different ontologies, [7] alone lists 35 of the most distinctive works. So what exactly does matching mean? What is the core problem in such process? Ontology mapping can take three different forms, thus, take a different meaning for each [3]; for our case, we are concerned with ontology mapping between an integrated global ontology and a local ontology. For this case, mapping means linking concepts found in one ontology into a view or query over other ontologies.

Some strategies for mapping tools include: lexical similarity of the terms to match, heuristics and machine learning. It is also interesting to note that most require user interaction in order to confirm or adjust the output of these tools.

In our project we are currently at the stage where we can start testing with different approaches to map the PO with an existing DO, based on the latest code, a user can visualize the PO in a tree form (listing of classes, object properties and data properties) and next to it visualize a DO, based on this side by side placement a user/administrator could look at the PO resources and modify them according to the vocabulary in the DO.

Visualization can play a key role in the manual mapping process; in the average and worst case scenario an automatic ontology mapper will not produce entirely accurate mapping results and these corrections or remaining links to be added will be done by administrator. If he/she can visualize such ontologies and be able to interact with their resources, the mapping process will take a less daunting task. Some options to be considered are included in [8] and [6].

4. USER INTERFACE DETAILS

As mentioned earlier, in order to create the PO, Ultrawrap takes information from the data dictionary to produce the ontology with all of the data in the schema, thus, the first requirement involved creating a way to have the user select which parts of the schema he/she wanted to export and make visible via the PO. The second thing required was a way to have the user edit the names in the PO since Foreign Keys (FK) between two tables can be exported with a name that concatenates the FK names to show a relationship. As a final requirement, we needed a way to map the two ontologies (PO & DO).

We have decided to take the approach of a web application using GWT, a set of tools that aids in the creation of JavaScript front-end applications using Java. Using this framework reduces the learning curve on new web technologies given that one can develop directly in Java, furthermore, it provides simple RPC mechanisms that are heavily used in our code and full compatibility with a wide array of browsers. And, as for aesthetics of the application, GWT makes it easy to divide such work so that everything can be easily managed through CSS and not affect the Java code for the application. Moreover, the Jena API is used in the modules where we need to read and parse the ontologies (in RDF format), and the JDBC library is used for database connectivity.

Below in Figure 1, a sequence diagram is shown with the use case of someone using the application from start to end based on current implementation, it is worth noting that the UltrawrapCompile component does multiple tasks in order to provide the PO, for further reference the user is encouraged to read [11]. Also, the loop overlay represents the iterative process and administrator can take in mapping his/her PO.

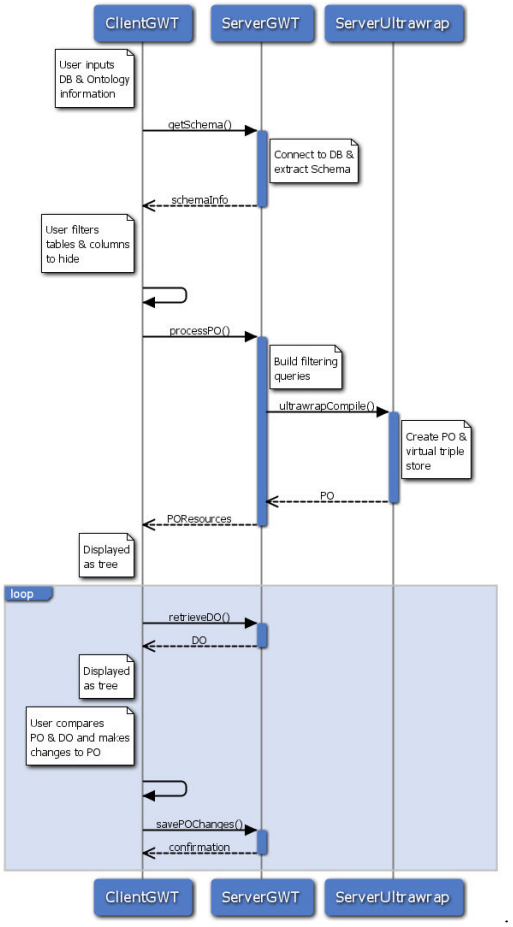

Figure 1. Sequence Diagram for GUI

Conference'10, Month 1–2, 2010, City, State, Country.

5. IMPLEMENTATION

By using GWT we are given access to an extensive library to widgets and functionality that alone would have taken a greater amount of time to code. From this foundation it was possible to extend the application based on the changing requirements for the GUI without much effort. However, there are elements of GWT that have an impact on functionality that are not described very well, such the case of Standard and Quirks mode; two modes an application can take, for which certain widgets only work on a particular mode and can have "weird" behavior if used in alternate mode.

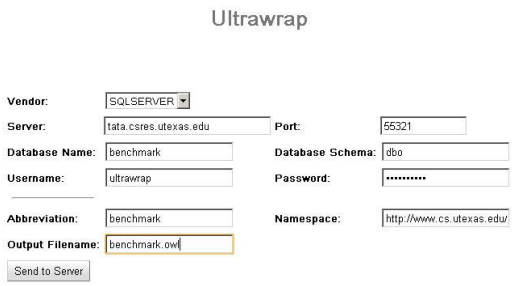

Figure 2. Initial input screen

As part of the workflow, that we have implemented, a user is first required to input information necessary to connect to a RDB as well as information required to build PO; the above figure lists the fields needed to process the PO.

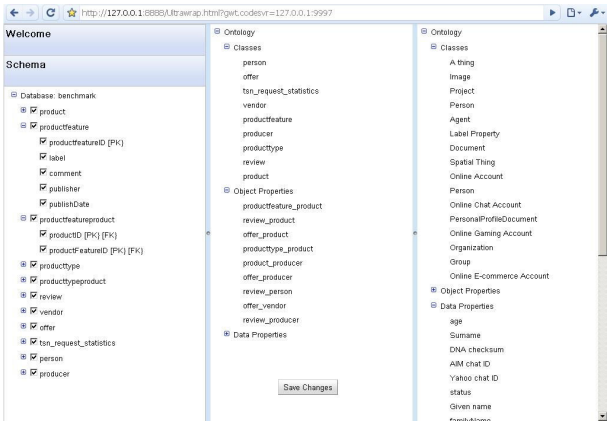

Figure 3. Main view showing the three panels

After going through the steps outlined in Figure 1 (sequence diagram), a user will be presented with a view similar to that of Figure 3, we see three vertical panels, from left to right; the first panel represents the full schema retrieved from the data dictionary, the middle panel represents the PO and the third is showing a DO (FOAF for this example). In the following subsections I will explain the steps and data structures involved in producing each of the panels.

5.1 Schema Panel

Even before this or any of the other panels are displayed, the user is prompted to enter information on the database connection (server, port, database name, etc) and on the ontology to be created (namespace, output file name, etc.). Thereafter, a RPC is made in order to make a connection to the database; after a successful connection using JDBC the schema is extracted by creating a single query that uses subqueries in order to retrieve in a single stream the results of a table, column, PK and FK columns. We use a single query to avoid network latency in the case the database is not in localhost and located elsewhere.

From the result of the query, we iterate over the tuples (number of tuples should match the number of columns in the database) and save them in the data structure shown in Figure 4, a process that takes O(N) where N is the number of columns.

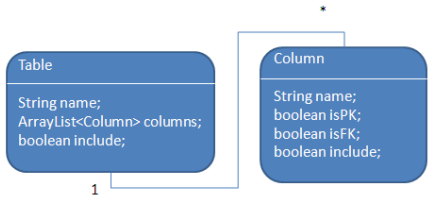

Figure 4. Data structure used to store schema information

This data structure is returned as a result of the initial RPC back to the client from which the GUI will build a tree. An interesting feature of the built tree is its O(1) modification time whenever a user checks or unchecks a box (include or not include attribute in the PO); this is achieved by means of the index access that we have to the elements of the tree and the data structure's "include" field.

5.2 PO Panel

Once a user goes through the filtering process in the previous panel the data structure is sent back to the server with the finalized selection. This data structure is then traversed and used to build the four SQL queries needed for the UltrawrapCompile module that will filter out the tables and attributes not selected by the user. These four queries are then sent to the UltrawrapCompile which uses them in creating the PO and returns that as a result.

The PO is read and we extract the classes, object and data properties and return these three as lists back to the client GUI which will display them in this middle panel.

From this moment, the user can click on any of the nodes in the tree (containing resources from the RDF model) to change names; this allows a 1:1 mapping between a current PO term and a user generated name substitution. These changes will be stored in a HashMap and will also be updated on the tree. Once finished, the user can click a button to save changes, this will send the HashMap to the server so that it can be traversed and serialized to disk under the same filename that was specified from start concatenated with the ".changes: suffix. This file will serve as a reference at runtime whenever a user issues SPARQL queries; he/she will be able to now address the new field names as opposed to the old fields initially generated as part in the PO.

5.3 DO Panel

This panel lets a user introduce a URL that points to a DO in RDF format. An InputStream will be generated from the URL and used in conjunction with the Jena API to navigate and extract the classes, object and data properties. Once the RPC returns, the same procedure is followed from the previous panel in order to produce the tree.

The following Figure shows the structure of the project concerning the GUI development:

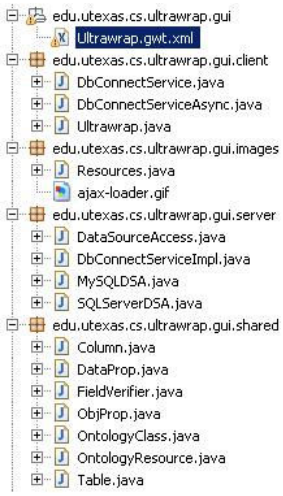

Figure 5. Package structure for GUI module

6. CHALLENGES

One of the first anticipated problems came with developing the wrappers for individual database vendors, we initially started with MySQL and later developed the wrapper for SQL Server which uses a different SQL syntax and when traversing a ResultSet only works with FWD cursors. This last constraint not present in MySQL allowed us to have a cursor that could go forward and if necessary back up to the previous row in the ResultSet; this type of cursor available in MySQL let us group attributes by table when storing into our data structure, and we would need to back up whenever a new table was detected. For future reference, an alternate approach could involve the use of two simultaneous cursors on same ResultSet, one serving as a scout, and the other as the actual cursor pointing to the data to be saved.

Any future expansions should be handled by adding an extra class with the specific connection and SQL syntax in building SQL queries. In regards to the process building SQL queries, we ran into problems when testing the constructed queries and verifying their correctness. The main problem was to efficiently remove the tables or attributes the user had not selected from the schema (ex. those fields he/she did not want to make public). Simply using a WHERE clause and grouping expressions via parenthesis does not evaluate as we had expected, as a proposed solution we are now making use of a UNION to progressively include those items the user did select. We argue that since this is only done once it will not have a big impact on performance, but there is room for query manipulation to increase efficiency in selecting those properties to be included.

In regards to GWT, it takes some time to make all widgets behave as desired, as most are nested inside other widgets, thus, child widgets inherit properties from parent widgets (alignment, dimensions, etc).

7. CONCLUSION & FUTURE WORK

The functionality described in this report is coded and has been checked in to the CVS repository at atol.csres.utexas.edu. The project has successfully accomplished the first two requirements mentioned in the GUI details section and is at a stage where the third and final requirement can be implemented in a short amount of time, depending on how we wish to change the visualization of ontologies and manage the mapping of the PO with DO.

There is further work that needs to be performed before this code goes into production and includes: Validation of fields where user inputs information, inclusion of specific error messages on error by system or user (currently only showing generic error messages). Development of further use cases to check alternate actions a user may take in using the Ultrawrap system interface, thus, checking for potential errors.

Finally, as mentioned earlier, this work has implemented the logic and functionality of the GUI, there has not been little effort on the aesthetics of the web application, however, GWT makes it easy to change the visual aspect of application with the use of CSS. Also, the application has grown very fast, and the client code might need to make use of the UBinder framework, it helps in code reusability and maintenance especially of large projects.

8. ACKNOWLEDGEMENTS

I want to thank Dr. Miranker for suggesting this line of work on the Semantic Web and Juan Sequeda for his guidance on the Semantic Web and Ultrawrap

9. REFERENCES

[1] Abadi, D. Marcus, A. Madden, S. and Hollenbach, K. Scalable Semantic Web Data Management Using Vertical Partitioning. In Proc. VLDB, 2007.

[2] Bizer, C. Cyganiak, R. D2R Server - Publishing Relational Databases on the Semantic Web. In Proceedings of the International Semantic Web Conference (ISWC). 2003.

[3] Choi, N. Song, I.Y. and Han, H. A survey on ontology mapping. SIGMOD Rec., 35(3):34–41, 2006.

[4] FOAF Vocabulary Specification 0.97. 2010. Retrieved May 13, 2010 from the University of Texas at Austin Libraries:http://xmlns.com/foaf/spec/

[5] gwt-graph, Retrieved May 13, 2010 from the University of Texas at Austin Libraries: http://code.google.com/p/gwt-graph/

[6] Harris, S. and Gibbins, N. 3store: Efficient bulk RDF storage. In In Proc. of PSSS'03, pages 1–15, 2003.

[7] Kalfoglou Y. and Schorlemmer M. Ontology mapping: the state of the art. The Knowledge Engineering Review, 18(1):1–31, 2003.

[8] Katifori, A. Halatsis, C. Lepouras, G. Vassilakis, C. and Giannopoulou, E. 2007. Ontology visualization methods a survey. ACM Comput. Surv. 39, 4, 1–43.

[9] OpenStreetMap, Retrieved May 13, 2010 from the University of Texas at Austin Libraries: http://www.openstreetmap.org/

[10] protege, Stanford Center for Biomedical Informatics Research. Retrieved May 13, 2010 from the University of Texas at Austin Libraries: http://protege.stanford.edu

[11] Sequeda, J F. Depena, R. Miranker, D. (2009) Ultrawrap: Using SQL Views for RDB2RDF. Poster in the 8th International Semantic Web Conference