Changing Ontology Breaks the Queries

Yaozhong Liang, Harith Alani, David Dupplaw and Nigel Shadbolt

Intelligence, Agents and Multimedia Group, School of Electronics and Computer Science, University of Southampton, Highfield, Southampton, SO17 1BJ U.K.
{y.david.liang, h.alani, dpd, nrs}@ecs.soton.ac.uk

Abstract. Updating an ontology that is in use may result in inconsistencies between the ontology and the knowledge base, dependent ontologies and applications/services. Current research concentrates on the creation of ontologies and how to manage ontology changes in terms of mapping ontology versions and keeping consistent with the instances. Very little work investigated controlling the impact on dependent applications/services; which is the aim of the system presented in this paper. The approach we propose is to make use of ontology change logs to analyse incoming RDQL queries and amend them as necessary. Revised queries can then be used to query the ontology and knowledge base as requested by the applications and services. We describe our prototype system and discuss related problems and future directions.

1 Background

The dynamic nature envisaged for the Semantic Web must have the capability to cope with continuous evolutions of domain models and knowledge repositories. It is therefore important to manage the ontology changes effectively to maintain the relations that specify how the knowledge is related between the different versions of the same ontology to avoid broken communications. There has been much work within the last few years on managing ontology change, so that such updates can be logged and used to provide better maintenance and accessability. Most of them could fall into one or more of three groups as following:

- *Detection and characterisation of change*: Direct comparison is a popular method for identifying changes between different versions of an ontology. OntoView [8] and PromptDiff [6] are two example systems for ontology comparison.
- *Ontology versioning and evolution*: currently, there is no agreed versioning and evolution methodology for ontologies on the Web [4]. Within this research area, most work focused on tracking and storing information about ontology change during the evolution process [7, 5] using changes identified at editing time or by comparing a pair of ontology versions. Other effort includes introducing evolution strategies to allow the developers to control and customise the ontology evolution process [2, 9].

– *Handling inconsistency introduced by ontology change*: Ontology changes will bring unexpected consequences to any dependent applications. However, this realm has received little attention so far in terms of adapting application queries to updated ontologies to ensure a continuous services.

Not much has been done with respect to using change-tracks to eliminate or reduce any impact that ontology change can have on any dependent applications and services. End-users anticipate that the services would be continuously available without too much interruption and the possible "404 Ontology has changed" error. In addition, they also expect to have the knowledge at the right time that the results delivered by the applications and services on the Semantic Web could be updated with the changed dependent ontologies. In this scenario, consistently and efficiently coping with ontology changes will be critical to achieve this requirement. We believe that it would be very beneficial to have a system that could track such changes, relate changes to incoming queries, amend such queries accordingly, and inform the query source of those changes and actions taken.

In this paper we describe a prototype system that targets these problems. The system uses a semantic log of ontology change to amend RDQL queries sent to the ontology as necessary. Such a system could save many hours of application re-development by not only updating queries automatically and maintaining the flow of knowledge to the applications as much as possible, but also to inform the developers of such changes in the ontology that relates to their queries.

2 Approach

In one-year's research work, we developed our understanding by analysing the context of our problem and comparing it with the related works in the area. Based on this, our approach (see. Figure 1) was introduced to explore a number of techniques that is useful to solve some of the problems we identified in the scenarios described above.

The solution to tackle the problems identified in our scenarios is described as a series of steps as follows:

1. **Capture**: The changes made between two versions of the same ontology are captured at this stage. Currently, we identify changes by comparing two versions using PromptDiff in Protégé.
2. **Instantiate**: Based on the changes identified in the first steps, an appropriate representation of ontology change, called *Log Ontology*, is produced and populated with change information.
3. **Analyse**: Queries submitted from the applications are analysed to find out whether any of the entities within the queries could be affected by the changes stored in the Log Ontology.
4. **Update**: If entities within the queries are found to have been changed, they are replaced with their changes to form the new queries with updated entities, and then resubmitted to the queried ontology.

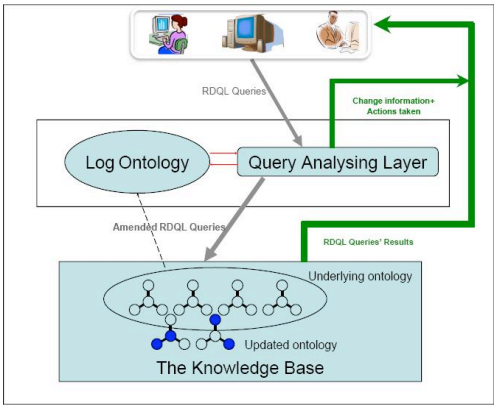

Fig. 1. An overview of the Approach

5. **Response**: After the new-formed queries are submitted to the ontology for processing, the results are returned back to the application. At the same time, a summary of change/update information will also be returned back to the end-users with the query results so as to inform users of the updates.

The working process of Query Analysing Layer in Figure 1 which includes **Analyse**, **Update** and **Response** steps described above is depicted in Figure 2.

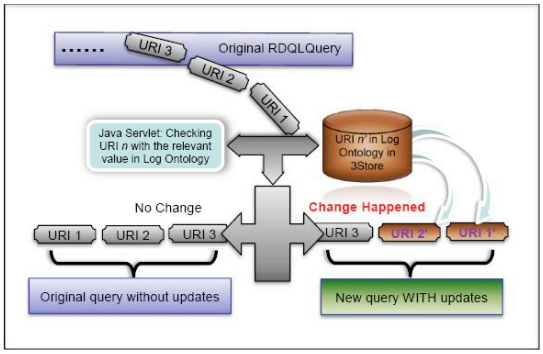

Fig. 2. The working process of the Query Analysing Layer

The ontology we used for our experiments is the CIDOC Conceptual Reference Model [1] (CRM). CRM provides a common language and semantic framework for the experts and developers in the cultural heritage domain. Our system acts as the gateway for accessing CRM to guarantee that a knowledge base organised by CRM queried by the users' RDQL queries would be accessible when the ontology itself has been updated.

3 Future Work

In [1], we present a number of experiments on a selection of typical RDQL queries to show how our system deals with different types of changes in the CRM ontology, including moving a class, modifying a property domain, the relationship between the property and its sub-properties and so on. In the future, we will also work on the following:

- Series of change: currently, the changes we captured are based on two CRM ontology versions only. The representation of change course is therefore relatively simple which is from *Version A* to *Version B*. However, changes between multiple versions of the same ontology can be iterative. For example, changes can be made from *Version 1* to *Version 2* of *Ontology O*. In *Version 3*, some changes might be changed back to their original form in *Version 2*. We need to represent such series of change actions in Log Ontology. Our system must handle series of change actions by coordinating the change information retrieved from Log Ontology. This would allow our system to cope better with more complex ontology changes. It will also assist us to better understand the change evolution process.
- Correlated change: currently the changes we deal with are those explicitly represented in the user's query, however, the knowledge represented by the ontologies can have indirect correlations as well. Our system should have the ability to inform the user of relevant changes appropriately, besides those directly related to the entities explicitly mentioned in the query. It would be one of our questions in the next stage.
- Order of changes: when complex changes take place, the current analysis method may cause unpredictable results. This is because the current method takes each entity in isolation and does not prioritise query replacements when an entity has been subject to a number of changes at once. To handle more complex changes, our system must have mechanism to decide where we should start to cope with the changes within the user's query.
- Ontology: we chose the CRM ontology as the underlying ontology for our experiments due to the number of versions available online. However, the number and type of changes that have been applied to this ontology is rather limited. An ontology which has been subject to bigger and more complex changes is needed to widen our experiments. Currently we are investigating the BioSAIL ontology for this purpose, which has been previously used for ontology change studies [3].

[1] The CIDOC Conceptual Reference Model: http://zeus.ics.forth.gr/cidoc/index.html

4 Conclusion

In this paper, we discussed the ontology changes as an open problem with respect to its potential serious effects on dependent applications and services. We proposed an approach for handling ontology changes by means of using change-tracks to eliminate or reduce any impact that ontology change can have on the application queries. For this purpose, we built the Log Ontology to store and manage change information between ontology versions. As a test example, we populated the Log Ontology with information about changes between two versions of the CRM ontology. We developed a prototype system that analyses the incoming queries, amends the entities within the queries according to the change information stored in the Log Ontology, and informs the end-user of any changes and actions taken.

Acknowledgement This work has been supported under the Advanced Knowledge Technologies Interdisciplinary Research Collaboration (AKT IRC), which is sponsored by the UK Engineering and Physical Science Research Council under grant number GR/N15764/01.

References

1. Liang, Y. Alani, H. Dupplaw. D and Shadbolt, N. Ontologies change and queries break: Towards a solution. Submitted to 15th International Conference on Knowledge Engineering and Knowledge Management, 2006.
2. Stojanovic, L., et al. User-driven ontology evolution management. In *Proceeding of the 13th International Conference on Knowledge Engineering and Knowledge Management, Ontologies and the Semantic Web*, pages 285–300, 2002.
3. Klein, M. *Change Management for Distributed Ontologies.* PhD thesis, Vrjie Universiteit, Amsterdam, 2004.
4. Klein, M. and Fensel, D. Ontology versioning on the semantic web. In *Proceeding of International Semantic Web Working Symposium (SWWS)*, Stanford University, California, U.S.A, 2001.
5. Noy, N.F., Kunnatur, S., Klein, M., and Musen, M.A. Tracking changes during ontology evolution. In *Proceeding of the 3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan, November 2004.
6. Noy, N.F., and Musen, M.A. Promptdiff: A fixed-point algorithm for comparing ontology versions. In *Proceeding of the 18th National Conference of Artificial Intelligence (AAAI)*, pages 744–750, Edmonton, Alberta, Canada, 2002.
7. Ognyanov, D. and Kiryakov, A. Tracking changes in rdf(s) repositories. In *Proceeding of the 13th International Conference on Knowledge Engineering and Management, Ontologies and the Semantic Web*, Spain, 2002.
8. Klein, M., Kiryakov, A., Ognyanov, D., and Fensel, D. Ontology versioning and change detection on the web. In *Proceeding of 13th International Conference on Knowledge Engineering and Management*, Siguenza, Spain, 2002.
9. Plessers, P., and Troyer, O.De. Ontology change detection using a versioning log. In *Proceeding of the 4th International Semantic Web Conference*, Galway, Ireland, 2005.