Applying Model-Transformation Techniques to Ontology Evolution

Longfei Jin, Lei Liu, and Degui Guo

Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education of China, Jilin University, Changchun, China
jinlongfei@vip.163.com

Abstract. Ontology evolution in the Model Driven Semantic Web can be looked as a process of model transformations. A model-transformation based conceptual framework for ontology evolution is presented in the paper. Applications of model transformations in every phase of ontology evolution process are described. The framework combines technologies of ontology evolution, Ontology Definition Metamodel and model transformations, and it can be looked as a method for ontology evolution in the Model Driven Semantic Web.

1 Introduction

The OMG's Model Driven Architecture (MDA) initiative and the W3C's Semantic Web project initially proceeded in parallel with no coordination. The marriage of MDA and Semantic Web can dramatically improve current approaches to ontology and knowledge base development and, at the same time, enable next generation enterprise computing by automating business semantics interchange and execution. In the domain of such marriage named Model Driven Semantic Web (MDSW) [1], ontologies are represented as models derived from Ontology Definition MetaModel (ODM) [2] - a metamodel of MDA, and many problems of ontology engineering can be solved using model technologies.

Ontology evolution is a complex problem of ontology engineering, and is especially important for semantic web. As ontologies are represented as models in MDSW, ontology evolution can be looked as a process of model transformations. In this paper, ontology evolution process was analyzed in a model-transformation viewpoint, and a conceptual framework for ontology evolution is presented. The framework adopts an OMG's Query/View/Transformation (QVT) proposal [3]. It can act as a foundation of ontology management, and benefit to other aspects of ontology engineering.

The remainder of the paper is organized as follows. In section 2, we introduce the OMG's QVT proposal. Section 3 overviews the conceptual framework for ontology evolution, and applications of model transformations in every phase of the ontology evolution process are described respectively. A detailed discussion of related work can be found in section 4. In section 5, we discuss the results and mention some aspects for future work.

2 Introduction of QVT

MOF 2.0 Query/View/Transformation (QVT) [3] is currently undergoing standardization through the OMG. We now present high level definitions of the QVT's subjects, and more details can be seen in the OMG's QVT proposal.

- **Queries** take as input a model, and select specific elements from that model.
- **Views** are models that are derived from other models.
- **Transformations** take as input a model and update it or create a new model.

The kernel of the QVT proposal is a model transformation language named QVT language. Main ideas of definition of QVT language are followed.

1. Transformation is a super-type of Relation and Mapping.
2. Relations are multi-directional transformation specifications.
3. Mappings are unidirectional transformation implementations. Mappings can refine any number relations.
4. QVT language allows model fragments to be matched against meta-model patterns and used in transformations.
5. Domains in QVT language are comprised of patterns and conditions. By utilizing conditions, arbitrarily complicated expressions can be specified to augment patterns.

The general form of a relation when written in textual form is thus:

```
relation R{
   domain{ pattern1 when condition1 }
   ...
   domain{ patternn when conditionn }
   when{ condition } }
```

In QVT, relations and mappings share the same syntax and semantics. The most obvious difference is that whilst a relation simply consists of a number of domains and an overall constraint, mappings also have an 'action body'.

3 The conceptual framework and the ontology evolution process

Our conceptual framework (Fig. 1.) shows the framework with an ontology evolution process comprised of six phases [4]: (1) change capturing, (2) change representation, (3) semantic of change, (4) change implementation, (5) change propagation, (6) change validation. Model transformation technologies based on QVT language are used in every phase of the process.

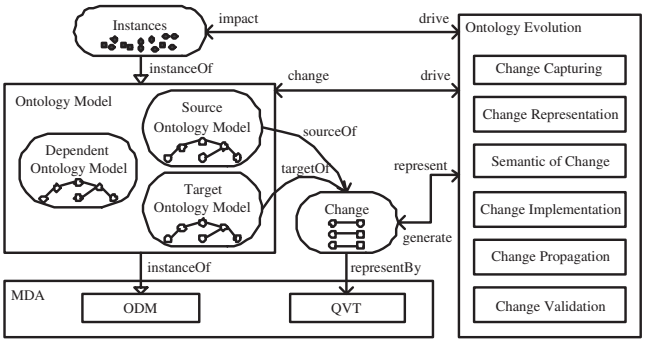

Fig. 1. A conceptual framework for ontology evolution

3.1 Change capturing

The process of ontology evolution starts with capturing changes either from explicit requirements or from the result of change discovery methods, which induce changes from existing data. In the conceptual framework, explicit requirements are represented in model transformation programs. Ontology engineers can transform informal requirement specifications derived from domain analysis to these formalizations. For an example, a informal requirement specification: adding concepts 'Man' and 'Woman' as sub conceptions of 'Person' into an ontology can be represented in a model transformation program as followed.

```
relation R{domain{(Old.RDFSClass)[localName="Person",...]}
    domain{(New.RDFSClass, c1)[localName="Person",...]}
    domain{(New.RDFSClass)[localName="Man",RDFSsubClassOf=c1,...]}
    domain{(New.RDFSClass)[localName="Woman",
        RDFSsubClassOf=c1,...]} }
```

These codes mean that R is a relation of two domains (models) – Old and New. In Old, there is a element typed as RDFSClass whose localName is 'Person', which can be transformed into three model elements of New whose localNames are 'Person', 'Man' and 'Woman'. Furthermore, 'Man' and 'Woman' are subclass of 'Person'.

There are three kinds of implicit requirements resulted from change discovery methods [5]: structure-driven changes, data driven changes and usage-driven changes. Structure-driven changes can be deduced from the ontology structure itself. In the conceptual framework, we can get them through a model analysis process, in which queries and views of QVT language are used. Data-driven changes are generated by modifications to the underlying dataset. We can get such changes represented in model transformation programs through a dataset mining process. Usage-driven changes result from the usage patterns created over a period time. They are counterparts of design patterns of software engineering.

In the conceptual framework, Usage-driven changes are represented as also model transformation programs. An example of such change named "pull concept up" can be represented in a model transformation program as followed (We extend the QVT language in the example).

```
relation PullConceptUp(Old.RDFSClass c /*extended parameter*/){
    domain{(Old.RDFSClass, c)[localName=n3,RDFSsubClassOf=
        (Old.RDFSClass)[localName=n2,RDFSsubClassOf=
            (Old.RDFSClass)[localName=n1,...],...],...]}
    domain{(New.RDFSClass, c1)[localName=n1,...]}
    domain{(New.RDFSClass)[localName=n2,RDFSsubClassOf=c1,...]}
    domain{(New.RDFSClass)[localName=n3,RDFSsubClassOf=c1,...]}}
```

Elements n1, n2 and n3 are grandfather, father and son in domain (model) Old. After applying PullConceptUp, in domain (model) New, n2 and n3 become siblings and they are sons of n1. That is to say, n3 is pulled up.

Besides these requirement driven ontology changes, most popular ontology changes are those caused by edit operations in ontology IDE. In these cases, model transformation programs are generated by the IDE automatically.

3.2 Change representation

Change representations mainly refer to the granularity of changes. There are two granularities representing ontology changes in the conceptual framework: elementary changes and composite changes. Elementary changes cannot be decomposed into simpler changes. A composite change is an ontology change that modifies (creates, removes or changes) one and only one level of neighborhood of entities in the ontology. The QVT language permits recursions of patterns, operations for transformations - NOT, AND, and OR, and calls among transformations. This makes it easier to represent rich ontology changes.

3.3 Semantic of change

The semantics of change phase is the phase in the ontology evolution process that enables the resolution of ontology changes in a systematic manner by ensuring the consistency of the ontology. In the conceptual framework, conditions of model transformation programs are used to keep consistency of ontology models. Checks are invoked before every transformation. For an example, only a RDFS class with URI that does not exist in current ontology model can be added. This can be represented in a model transformation program as followed.

```
relation AddConcept{domain{(Old.RDFSClass, parent)[...]}
    domain{(New.OWLClass)[URI=uri, RDFSsubClassOf=parent, ...]}
    when{allElement->forAll(element, element.URI<>uri)} }
```

There are many evolution strategies encapsulating policies for evolution with respect to user's requirements in the conceptual framework. When ontology models are inconsistent, users can select and execute those strategies. For an example, when a user deletes a concept, he can select whether to delete its entire offspring, or only to pull its sub concept up, or to make its sub concept as a child of the top concept. In the conceptual framework, evolution strategies are represented as different domains with same patterns and conditions. When different domains are satisfied at the same time during a model transformation process, execution is interrupt and will resume after user select a special strategy. For convenient, user can set default evolution strategies for a special transformation process.

3.4 Change implementation

Three phases are required in change implementation: (1) change notification, (2) change application, (3) change logging. In order to avoid performing undesired changes, a list of all implications to the ontology and dependent artifacts should be generated and presented to the ontology engineer, who should then be able to accept or abort these changes. The conceptual framework uses views of QVT to provide change impact analysis views to users. The application of a change should have transactional properties. The conceptual framework realizes this requirement by the strict separation between the request specification and the change implementation. This allows to easily treating the set of change operations as one atomic transaction, as all the changes are applied at once. The conceptual framework uses model transformation programs as logs. Through computing inverse transformations, models can be traced back. As relations are multi-directional transformation specifications, models can be traced back easily through execution of inverse mappings.

3.5 Change propagation

Ontologies often reuse and extend other ontologies. Therefore, an ontology update might also corrupt ontologies depending on the modified ontology (through the inclusion, mapping integration, etc.) and consequently, all the artifacts based on these ontologies. Ontology changes can impact: (1) local ontology in which elements changed exist, (2) dependent ontologies which depend on local ontology, (3) instances of ontologies, (4) applications using ontologies. Change propagations can be computed recursively. As change impact analysis of applications relate to the implements of those applications, we will not discuss them in this paper.

Ontology changes can impact local ontology. There are many relations among elements, such as inherit, equal, different, etc. We can construct a dependency graph of local ontology by computing those relations. When a change occurs in local ontology, we can get elements impacted by the change through the dependency graph. In the conceptual framework, implement of the change impact analysis comprised of five steps: (1) to transform local ontology to a dependency graph, (2) to get impacted elements from the dependency graph through

queries of the QVT language, (3) using those elements as inputs to execute step 2 recursively, and to get change impact views, (4) to check inconsistency and generate model transformation programs, (5) to execute those programs to keep consistency of local ontology.

Ontology changes can impact dependent ontologies. We can get change impact views by querying the dependency graph comprised of local ontology and dependent ontologies. As elements not changed in local ontology will not impact any elements of dependent ontologies, we can optimize the construction of dependency graph through replacing the whole local ontology by a change impact view of it.

Ontology changes can also impact instances. There are two kinds of implementations in the conceptual framework. One is to generate date transformation programs from model transformation programs and to keep consistency of ontology model and instances by data transformations. The other is to keep instances unchanged, and to send model transformation programs as additional inputs to an extended reasoner. Because the reasoner knows changes of ontologies, it can give a transparent reasoning - ability of a reasoner that can give same reasoning results as with old ontologies when instances changed.

3.6 Change validation

Change validation enables justification of performed changes and undoing them at user's request. As we have provided formal requirement specifications and change impact analysis views, many validations have been done in other phases of ontology evolution process.

4 Related work

In [4], the authors identify a possible six-phase evolution process, on which our framework is built. [5] defines three types of change discovery. An implementation of data-driven change discovery is included in the KAON tool suite [6]. [7] describes a set of changes for the OWL ontology language, based on an OWL meta-model. [5] proposes an evolution log based on an evolution ontology for the KAON ontology model. [8] presents an approach for evolution in the context of dependent and distributed ontologies. [9] concentrates on the benefits of modular ontologies with respect to local containment of terminological reasoning. [10] discusses OntoView, a web-based change management system for ontologies. The OMG's QVT proposal introduces a general model transformation language. EMF [11] is a model transformation tool built on the Eclipse [12] platform, and it can transform models to java codes. GMT [13] is general model transformer built on the Eclipse platform, and its basic model transformation language is ATL [14]. YATL is another QVT language, and some examples are discussed in [15]. Ontology Definition MetaModel (ODM) [3] is currently undergoing standardization through the OMG. It is a standard of ontology representation. Our conceptual framework is built on ODM, QVT and ontology evolution.

5 Conclusion and further works

Our work combines research results of ontology evolution, model transformation and Ontology Definition Metamodel. A model-transformation based conceptual framework for ontology evolution is presented in this paper. The framework resolves the problem of ontology evolution in domain of Model Driven Semantic Web, and it can benefit to other researches of ontology engineering. As the QVT proposal and the ODM proposal are not standards, and ontology evolution is far from manual technology, our framework is still conceptual. With researches of related technologies going on, the framework will be developed to a model-transformation based ontology management system.

There are a lot of works need to do, to name just a few. To refine the framework, to cut and extend the model transformation language, to design and implement an interpreter of the model transformation language, to add ontology versioning functions to the framework, to investigate how ontology evolution impact ontology dependencies, to apply other model technologies to ontology evolution, etc.

References

1. Frankel, D., Hayes, P., Kendall, E., McGuinness, D.: The Model Driven Semantic Web. In: 1st International Workshop on the Model-Driven Semantic Web (MSDW 2004) Monterey, California, USA. (2004)
2. Chang, D.T., Kendall, E.: Major Influences on the Design of ODM. In: 1st International Workshop on the Model-Driven Semantic Web (MSDW 2004) Monterey, California, USA. (2004)
3. Object Management Group, Inc.: MOF 2.0 Query / View / Transformation Revised Submission. (2004) OMG Document: ad/04-04-01 http://www.omg.org/cgi-bin/apps/doc?ad/04-04-01.pdf.
4. Stojanovic, L., Maedche, A., Motik, B., Stojanovic, N.: User-driven ontology evolution management. In: EKAW '02: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, London, UK, Springer-Verlag (2002) 285–300
5. Stojanovic, L.: Methods and Tools for Ontology Evolution. PhD thesis, University of Karlsruhe, Universität Karlsruhe (TH), Institut AIFB, D-76128 Karlsruhe (2004) Studer, R.; Weinhardt, Ch.
6. FZI Karlsruhe and AIFB Karlsruhe: KAON the Karlsruhe ontology and semantic web framework-developer's guide for KAON 1.2.7. (2004)
7. Klein, M.: Change Management for Distributed Ontologies. PhD thesis, Vrije Universiteit Amsterdam (2004)
8. Maedche, A., Motik, B., Stojanovic, L.: Managing multiple and distributed ontologies in the semantic web. VLDB Journal **12** (2003) 286–302
9. Stuckenschmidt, H., Klein, M.: Integrity and change in modular ontologies. In: Proceedingso of the 18th International Joint Conference on Artificial Intelligence, Acapulco, Mexico (2003)
10. Klein, M.C.A., Kiryakov, A., Ognyanov, D., Fensel, D.: Finding and characterizing changes in ontologies. In: ER '02: Proceedings of the 21st International Conference on Conceptual Modeling, London, UK, Springer-Verlag (2002) 79–89

11. eclipse.org: EMF - Eclipse Modeling Framework. (2005) http://www.eclipse.org/emf/.
12. Object Technology International, Inc.: Eclipse platform technical overview. (2003) http://www.eclipse.org/whitepapers/eclipse-overview.pdf.
13. eclipse.org: GMT - Generative Model Transformer. (2005) http://www.eclipse.org/gmt/.
14. Bézivin, J., Breton, E., Dupé, G., Valduriez, P.: The ATL Transformation-based Model Management Framework. Technical report, Atlas Group, INRIA and IRIN University of Nantes, Nantes, France (2003) http://www.sciences.univ-nantes.fr/irin/Vie/RR/indexGB.html.
15. Patrascoiu, O.: YATL:Yet Another Transformation Language. In: Proceedings of the 1st European MDA Workshop, MDA-IA, University of Twente, the Nederlands (2004) 83–90