A Pattern-based Framework of Change Operators for Ontology Evolution

Muhammad Javed[1], Yalemisew M. Abgaz[2], Claus Pahl[3]

Centre for Next Generation Localization (CNGL),
School of Computing, Dublin City University, Dublin 9, Ireland
{mjaved[1], yabgaz[2], cpahl[3]}@computing.dcu.ie

Abstract. Change operators are the building blocks of ontology evolution. Different layers of change operators have been suggested. In this paper, we present a novel approach to deal with ontology evolution, in particular, change representation as a pattern-based layered operator framework. As a result of an empirical study, we identify four different levels of change operators based on the granularity, domain-specificity and abstraction of changes. The first two layers are based on generic structural change operators, whereas the next two layers are domain-specific change patterns. These layers of change patterns capture the real changes in the selected domains. We discuss identification and integration of the different layers.
Keywords: Ontology evolution, change operators, pattern-based change.

1 Introduction

The dynamic nature [1] of knowledge in every domain requires ontologies to change over time. The reason for change in knowledge can be the change in the domain, the specification, the conceptualization or any combination of them [2]. Some of the changes are about the introduction of new concepts, removal of outdated concepts and change in the structures and the meanings of concepts. A change in an ontology may originate from a domain knowledge expert, a user of the ontology or a change in the application area [3].

Ontology evolution is defined in different ways [4–6]. A comprehensive definition is given as *"the timely adaptation of an ontology to changed business requirements, to trends in ontology instances and patterns of usage of the ontology based application, as well as the consistent management/propagation of these changes to dependent elements"* [4]. Based on the different perspectives of the researchers, there are different solutions provided to handle ontology evolution [4, 7–9]. Different phases of ontology evolution have been identified [4]. Basic changes in the evolving ontology can be captured using operators. However, the identified change operators focus on generic and structural changes lacking domain-specificity and abstraction. Moreover, these solutions lack adequate support for different levels of granularity at different levels of abstraction.

Some central features of our proposed evolution framework that go beyond the current focus on compositionality of change are:

– To make changes in ontologies effective, change needs to be operationalised. The operation can be atomic, composite or complex [2, 10, 11]. This indicates that the effectiveness of a change is significantly dependent on the granularity, how the change operators are combined and the extent of their effect in the ontology. The impact of the change operators can affect the consistency of the ontology. Thus, a coherent treatment of the change operators and their effect on the consistency at each level of granularity becomes vital.
– The changes at a higher level of granularity, which are frequent in a domain, can be represented as domain-specific patterns – which are often neglected by the lower-level compositional change operators addressed in the literature. Thus, the categorization of operators at a domain-specific level enables us to support high-level abstraction. This abstraction enables us to map the domain-specific levels to abstract levels and facilitate the smooth linking of domain ontologies with upper ontologies like SUMO [12].

We present an approach to deal with ontology evolution through a framework of compositional operators and change patterns, based on the empirical evaluation of changes in a couple of different ontologies. We focus on domain-specific perspective-linking structural changes in domain ontologies. We identified four levels of change operators and patterns based on the granularity of changes, the effect the change operators have on the ontology, domain-specificity, i.e., the extent to which the operators are specific to a certain domain and become domain-specific patterns of change, and the degree of abstraction.

The paper is structured as follows: We discuss our empirical study in Section 2. Framework of change operators and patterns is presented in Section 3. A short evaluation is given in Section 4. Related work is discussed in Section 5 and we end with some conclusions.

2 Empirical Study of Evolution of Domain Ontologies

While layered change operators have been suggested, we studied the evolution of ontologies empirically in order to investigate the relationships between generic and domain-specific changes and to determine common patterns of change.

2.1 Domain Selection and Ontology Development

Since our focus is to study, identify and classify the changes that occur in ontologies, a careful selection of the domain is of great importance. As case studies, the domains University Administration and Database Systems were taken into consideration. The former is selected because it represents an organisation involving people, organisational units and processes. The latter is a technical domain that can be looked at from different perspectives – for instance being covered in a course or a textbook on the subject.

The database textbook ontology was derived from the taxonomy arising from the table of content and the index. The technical database domain ontology was developed by domain experts. The university ontology was developed by us.

2.2 Empirical Change and Evolution Analysis

We approached the problem as an empirical study and conceptualization of the two domain areas. The database ontology is constructed by observing the domain area and the patterns are identified by observing the practical changes in the domain. These changes are identified by comparing text books, course outlines, lecture notes and changes in technologies. These practical changes are the changes that we observed following database experts, academics and practitioners, which allowed us to approach the problem from an empirical point of view. Domain experts in both areas have contributed to the study [13]. The abstraction of the changes into a higher level in the hierarchy is done by conceptualization of the results of the empirical study. In the case of the university ontology, we considered Dublin City University (DCU) as our case study and we conceptualized most of the activities and the processes in DCU for the construction of the ontology. Following a similar approach to the database ontology, we investigated abstraction at higher levels during conceptualization.

2.3 Analysis of the Empirical Results

We outline some results here, before discussing details that will lead to the change operator framework in the next section.

Database Ontology. We observed that the changes in the database system can be identified by taking different perspectives into account. In teaching, the course content changes almost every year introducing new concepts, theories and languages. In publishing, new database books in the area appear every couple of years resulting in addition of new chapters, merging or removal of existing chapters and changing of the structure of the topics within and among chapters. In industry, new technologies and languages are emerging. These changes result both in structural and instance level change. For example, in the perspective of teaching databases, an academic may introduce a new technology and make it a precondition for learning some of the chapters. Furthermore, he may merge two chapters into one. These factors make evolution predictable to some extent.

University Ontology. The objective of this ontology is to assist administering the proper execution of the day-to-day activities of a university. We observed that a University ontology changes frequently at instance level due to joining or leaving of Faculty, Student or Staff, introduction of new Courses etc., but less frequently and more irregularly at concept level.

3 A Framework of Change Operators and Patterns

Changes have to be identified and represented in a suitable format to resolve ontological changes [14]. The explicit representation of change allows us to analyse and understand a change request. Changes that occur in the ontology are tracked and identified. Based on our observation of common changes in all versions of the ontologies, we studied the patterns they have in common, resulting in a framework of change operators (Fig. 1):

– level one: elementary changes which are atomic tasks.
– level two: aggregated changes to represent composite, complex tasks.
– level three: domain specific change patterns.
– level four: abstraction of the domain specific change patterns.

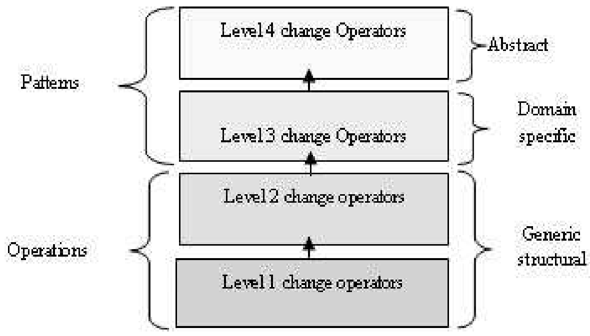

Fig. 1. Different Levels of Change Operators

The operators in level one and level two reflect generic structural changes; however the change patterns in level three and level four need to be customized (Fig. 2). We observed that ontology changes are driven by certain types of common, often frequent changes in the application domain. Therefore, capturing these in the form of common and regularly occurring change patterns creates domain-specific abstractions. A number of basic change patterns may be provided so that users may adapt and generate their own change patterns to meet their own domain demand. This makes the ontology evolution faster and easier. The dots at each level (Fig. 2) depicts that change operators are extensible.

3.1 Generic Structural Levels

Level One Change Operators – Element Changes. These change operators are the elementary operations used to perform a single task by an ontology management tool. These operators add, modify or remove a single entity in the ontology. A single operator performs a single task that can add single concept, a single property or delete a single concept, etc. We can identify these simple operations based on the constituent components of the ontology.

We can create a new concept *"DDL"* as a subconcept of *"Database"*:

– create a concept *DDL* using **CreateConcept**(DDL)
– make *DDL* a subconcept of *Database* using **Subconcept**(DDL, Database)

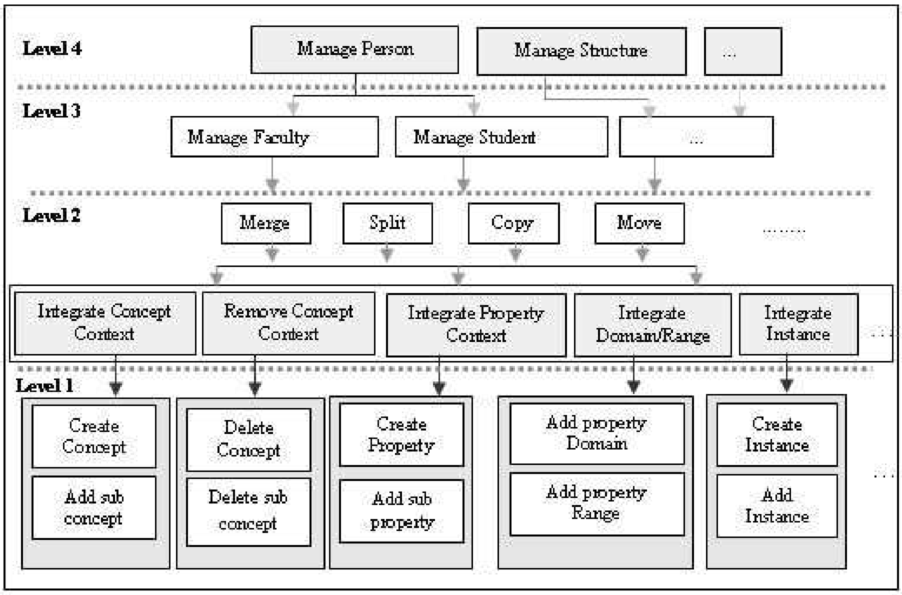

Fig. 2. Architecture of layered change operators

Level Two Change Operators – Element Context Changes. Many evolution tasks cannot be done by a single atomic operation. A set of related operations is required. These change operators are identified by grouping atomic operations to perform a composite task. For example, to delete a single concept *"faculty"* in the university ontology, removing the concept from the concept hierarchy is not sufficient. Before we remove the concept, we have to remove it from the domain and the range of properties like *"hasPublication"* or *"supervises"* that are linked to it. In addition, we need to either remove its subconcepts or link them to the parent concept. Depending on this context of an element, we use different operators from level one resulting in a standardised change pattern. The second level operations affect the structure of the ontology.

If an instructor wants to add a single chapter to his course outline, the operator *"Integrate Concept Context"* that operates on a single targeted concept can be used:

- create concept *SQL* using **CreateConcept**(SQL)
- make *SQL* subconcept of *Database* using **Subconcept**(SQL, Database)

If he wants to merge two or more chapters for an abridged course outline, which involves two or more concepts, the operation requires operators higher than integrate concept context and the corresponding operations, for instance **Merge** (DDL, DML, Database):

- *Integrate Concept Context* by creating concept *Database* using **CreateConcept**(Database)

– *Integrate Property Context* by creating property *isBasedOn* using **CreateProperty**(isBasedOn)
– *Integrate Domain/Range Context* by adding domain *Database* to *isBasedOn* using **AddDomain**(isBasedOn,Database) and adding range *Relational Algebra* to *isBasedOn* using **AddRange**(isBasedOn,Relational Algebra)
– *Remove Concept Context* by deleting concept *DML* using **DeleteConcept**(DML) and deleting concept *DDL* using **DeleteConcept**(DDL)

3.2 Domain-Specific Level

Level Three Change Operators – Domain-specific. This domain-specific perspective links the structural changes to the aspects represented in domain ontologies. In order to execute a single domain-specific change, operations at level two are used. In addition, this level is constructed on the perspectives we identified in the construction stage of the ontology. The change patterns are based on the viewpoints and activities of the users. Two users may have different perspectives to view the ontology which results in the use of a different combination of operations from level two. As the perspectives are different, the number of operations or the sequence of operations might differ. This difference results in patterns of change based on the perspectives of the ontology engineers.

Database Ontology. In the Database domain, the different perspectives we mentioned define their own patterns. From the teaching perspective, *"manage chapter"* has a pattern of calls such as create concept *"chapter X"* for a specific topic, create properties such as *"isRequiredBy"*, *"isAlternateTo"* and *"isBasedOn"* to sequence topics in a course. From the perspectives of authors, a pattern to create concept *"chapter X"* and MoveConceptUp *"chapter X"* is often used. A technology domain expert only needs to include the technology as a new concept and calls to create a concept *"new technology"*.

Level three operators enable us to treat domain-specific operations separately and allow an ontology engineer to define his own change patterns once, which can be executed many times. For example an instructor wants to manage the contents of his database course. He has different ways of managing the chapters by adding new chapters, altering the prerequisites, merging or splitting the chapters or a combination of one or more of the above. Some of the operations that occur frequently are listed in Table 1.

An instructor who wants to adapt a course outline every time he delivers courses can identify patterns of changes that enable him to manage the course outline. From the list in Table 1, options 1, 3, 9 can be chosen whenever a new chapter is added and others are left out. He can determine a pattern of managing chapters by adding a new chapter. When a chapter needs to be deleted, options 5 and 10 can be chosen and when chapters need to be split, then 6, 9, 10 would suit. Specific to the domain and the requirements of the ontology user, with out going to the details of the first and the second levels, he can choose his own patterns and execute change based on the patterns defined before.

Table 1. List of Operations for Course Management

1. Add new chapters (concepts)	5. Delete chapters	9. Set prerequisite
2. Move sub sections to chapters	6. Split chapters	10. Remove prerequisite
3. Add sections	7. Copy chapters	
4. Move chapters to sub chapters	8. Merge chapters	

University Ontology. In case of the University Administration domain, level three may contain change patterns such as *"manage faculty"*, *"open new department"* or *"close department"*. If one user needs to register a new category of faculty using the manage faculty change pattern, say a Lecturer, then he creates a concept *"Lecturer"*, creates a property *"hasPublication"*, *"supervise"* etc. from level two – see Fig. 2. Another ontology engineer may create a new concept *"Lecturer"* without including the *"supervise"* property. Note, that this is a structual change at concept level.

Table 2. List of Operations for Employee Management

1. Add employee	2. Add fields of employee
3. Copy employee	4. Add properties (like Manages, Supervises etc.)

For employees with no management position, the ontology engineer can choose a pattern containing 1, 2 and execute that every time there is a new employee. However, if he has managers with a management function then he can choose 1, 2, 4 and if the employee is assigned in two or more sections he can choose 1, 2, 3 and execute this pattern whenever there is a new employee that fits any of these conditions.

These changes can be aggregated together and patterns of change can be identified. These specific patterns of change are often domain-specific and can be reused at instance level for the addition of new lecturer or manager employees. However, these variation are similar within a domain. A generic employee change pattern with common operations emerges.

3.3 Abstract Level

Level Four Change Operators – Generic Categorisation. These are constructed based on the abstraction of the concepts in level three. The main objective of introducing this level is to provide a facility that allows us to map domain-specific ontologies to available upper level ontologies (i.e. categorising domain concepts in terms of abstract ones) and that helps to generalise and transfer patterns to other domains. For example the University Administration ontology can be mapped and linked to any other organization that has a similar conceptual structure. In the university domain, one can identify concepts such as students, faculties and employees; a production company may have employees, customers, owners or shareholders. Level four provides an abstraction to represent all these concepts using a general concept *"Person"*. In a similar fashion, the University system has research groups, departments, and committees; whereas a company may have research groups, departments and board of directors. We can abstract them as *"Structures"*. Furthermore, we have admission, examination, teaching, auditing in a university system and production, auditing and recruitments in an organization. We can abstract them to *"Processes"*. In the Database Systems ontology we can identify Relational Algebra, Relational Calculus or SQL concepts, whereas an ontology for Java Programming may have concepts such as Control Statements, Class or Thread. Level four provides an abstraction to represent all these concepts using a general concept such as *"Theory"*.

The benefit is the reuse of domain-specific patterns and their re-purposing for other, related domains, i.e. the transfer between domains. Level four provides a common ground to link the ontology with existing higher-level ontologies such as the Suggested Upper Merged Ontology (SUMO) or MIddle Level Ontology (MILO) that provide the bridge between domains. It provides change patterns that can be applied to any subject domain ontology that is composed of a similar conceptual structure. Level four is constructed on top of level three and level two. Fig. 2 represents the architecture of how the four levels are integrated and interconnected to each other.

This can actually be seen as part of the evaluation, where genericity and transferability are important criteria. Level Four is actually a framework aspect that guides transfer of patterns to other domains (rather than being of specific importance for the user of a concrete application ontology).

4 Evaluation

The change operator and pattern framework introduced here has been empirically developed. In our evaluation, we specifically considered validity, adequacy and transferability as criteria.

Validity. The evaluation of the work relating to the validity of the operators in representing real-world problems faced by users and ontology engineers. In this regard, the change operators and patterns we found are based on changes actually carried out by users and ontology engineers and observed by us in both the university administration and database systems ontologies.

Though it is not expected to be exhaustive, we found that a significant portion of ontology change and evolution is represented in our framework.

Adequacy. The adequacy of the solution to be useful and suitable to the users and ontology engineer. For this purpose, we identified different levels of abstractions for those who focus on the generic as well as the domain-specific changes.

The patterns are appropriate to provide domain-specific level change support for users. The lower-level operators are useful to ontology engineers to suitably define their own change operators. Domain experts can use the patterns and alter them to meet their requirements by varying the sequentialisation of the content elements. In such cases change patterns are at the right level of abstraction and consequently more usable for the domain experts. It faciliates a structured evolution process easy and reduces effort in terms of time consumption.

Transferability. The four levels of change operators are evaluated against their transferability and applicability to other domain ontologies. Transferability is a measure of the usefulness of the framework. We looked at the transferability or the applicability of our change operator framework for instance between different academic, but also between universities as academic organisations and industrial organisation.

Basic change pattern are generally transferable and applicable with little customization of the operators to other domain ontologies – a process facilitated by the categorisation of level 3 within level 4.

5 Related Work

We give a brief summary of current practice in the area of ontology evolution, specifically change representation. The author in [10] discusses the complexity of the change management process and presents a six phase ontology evolution process. She discusses the representation of generic changes and categorized them into elementary, composite and complex. In contrast to our work, they do not include aspects such as granularity, domain-specificity and abstraction. In [2], the authors provide a set of possible ontology change operations based on the effect with respect to the protection of the instance-data availability. Their work focuses more to instances than the structural or domain specific operations. In [11], the impact of ontology change to the validity of the instance availability is discussed and changes are subdivided into two categories, i.e. structural and semantic changes. Though their work addresses semantic changes, our work takes the semantic changes further and proposes domain-specific change patterns for semantic changes. In [7], the authors present a declarative approach to represent the semantics of changes, considered as a reconfiguration-design problem. Their work is focused on the realization of the changes, whereas our work is focused on identifying domain-specific change patterns.

6 Conclusion

We discussed our approach for ontology evolution as a pattern-based compositional framework. The approach focuses on four levels of change operators and patterns which are based on granularity, domain-specificity and abstraction. While ontology engineers typically deal with generic changes at level one and level two, other users can focus on domain-specific changes at level three. Using level four, a link to the existing high-level ontologies like SUMO and MILO can be created. Our framework benefits in different ways. First, it enables us to deal with structural and semantic changes at two separate levels without loosing their interdependence. Second, it enables us to define a set of domain-specific changes. Third, domain-specific changes can be shared among other ontologies that have similar conceptualizations and specifications. It facilitates the linking of an ontology with other high-level ontologies.

The empirical study indicates that the solution is valid and adequate to efficiently handle ontology evolution. Currently we are focusing on the formalization of change patterns and the impact of the changes on the consistency of the ontology. The implementation of the approach as an operator and pattern calculus, which includes tools and techniques, and the investigation of pattern customisation and transformation to other domains is our future work.

7 Acknowledgment

This work is supported by the Science Foundation Ireland through its CNGL CSET grant.

References

1. Javed, M., Abgaz, Y.M., Pahl, C.: A pattern-based framework of change operators for ontology evolution. In Meersman, R., Herrero, P., Dillon, T., eds.: On the Move to Meaningful Internet Systems: OTM 2009 Workshops. Volume 5872 of Lecture Notes in Computer Science., Springer (2009) 544–553
2. Noy, N.F., Klein, M.: Ontology evolution: Not the same as schema evolution. Knowledge and Information Systems. **6**(4) (2004) 328–440
3. Liang, Y., Alani, H., Shadbolt, N.: Ontology change management in protégé. In Proceedings AKT DTA Colloquium, Milton Keynes, UK. (2005)
4. Stojanovic, L., Maedche, A., Motik, B., Stojanovic, N.: User-driven ontology evolution management. Lecture Notes in Computer Science. **6**(4) (2002) 285–300
5. Haase, P., Sure, Y.: User-driven ontology evolution management. State-of-the-Art on Ontology Evolution. EU IST Project SEKT Deliverable D3 1.1.b (2004)
6. Flouris, G., Plexousakis, D., Antoniou, G.: A classification of ontology change. Poster Proceedings of the 3rd Italian Semantic Web Workshop, Semantic Web Applications and Perspectives (SWAP-2006) (2006)
7. Stojanovic, L., Maedche, A., Stojanovic, N., Studer, R.: Ontology evolution as reconfiguration-design problem solving. Proceedings of the 2nd international conference on Knowledge capture (2003)
8. Zablith, F.: Dynamic ontology evolution. International Semantic Web Conference (ISWC) Doctoral Consortium, Karlsruhe, Germany (2008)
9. Plessers, P., De Troyer, O., Casteleyn, S.: Understanding ontology evolution: A change detection approach. Web Semantics: Science, Services and Agents on the World Wide Web. **5**(1) (2007) 39–49
10. Stojanovic, L.: Methods and tools for ontology evolution. PhD thesis, University of Karlsruhe (2004)
11. Qin, L., Atluri, V.: Evaluating the validity of data instances against ontology evolution over the semantic web. Information and Software Technology. **51**(1) (2009) 83–97
12. Top level Ontologies, .: http://www.ontologyportal.org.

13. Boyce, S., Pahl, C.: The development of subject domain ontologies for educational technology systems. Journal of Educational Technology and Society (ETS) IEEE **10**(3) (2007) 275–288
14. Oliver, D.E., Shahar, Y., Shortliffe, E.H., Musen, M.A.: Representation of change in controlled medical terminologies. Artificial Intelligence in Medicine. **15**(1) (1999) 53–76