

# A Layered Framework for Pattern-based Ontology Evolution

Muhammad Javed , Yalemisew M. Abgaz , Claus Pahl

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

**Abstract.** The challenge of ontology-driven modelling of information components is well known in both academia and industry. In this paper, we present a novel approach to deal with customisation and abstraction of ontology-based model evolution. As a result of an empirical study, we identify a layered change operator framework based on the granularity, domain-specificity and abstraction of changes. The first two layers are based on generic and structural ontology change operators, whereas layer three is based on user-defined domain-specific change patterns. The implementation of the operator framework is supported through layered change logs. Layered change logs capture the objective of ontology changes at a higher level of granularity and support a comprehensive understanding of ontology evolution. The layered change logs are formalised using a graph-based approach. We identify the recurrent ontology change patterns from an ontology change log for their reuse. The identified patterns facilitate us in optimizing and improving the definition of domain-specific change patterns.

**Key words:** pattern-based ontology evolution, ontology change operators, layered change logs

## 1 Introduction

Ontology-driven modelling is beneficial for a wide range of information systems aspects. Ontology-based approaches can capture the architecture and process patterns [1]. Ontology-based software models have helped the researchers to take a step forward from traditional content management systems (CMS) to conceptual knowledge modelling to meet the requirements of the semantically aware software systems. Research as presented in [3] and [9] stress the contribution of ontologies to conceptual knowledge modelling. Domain ontologies have become essential for conceptualisation and knowledge sharing activities in dynamic enterprise software system. Ontologies can convey the useful semantic information for software developers to understand and process.

Ontologies can play a critical role in conceptualisation of domain models. However, ontology change management is a challenging area. The dynamic nature of knowledge in every conceptual domain requires ontologies to change over time. The operationalisation of changes to ontologies is one of the vital parts

of ontology evolution. The reason for change in ontologies can be the change in the domain, the specification, the conceptualization or any combination of them [4]. Some of the changes are about the introduction of new concepts, removal of outdated concepts, change in the structures and the meanings of concepts. A change in ontology may be initiated by a domain expert, an ontology engineer or by a change in the application domain area. We have noticed a significant frequency in changes, which makes a support framework highly beneficial.

We present an approach to deal with ontology evolution through a framework of compositional operators and change patterns, determined through an empirical evaluation of changes in selected domain ontologies. We introduce a notion of layered change logs for explicit operational representation of ontology changes. Some central features of our approach that go beyond the current focus on compositionality of ontology changes are:

- The impact of the change operators can affect the consistency of the ontology. Thus, a consistency-preserving definition of the change operators and thus maintaining overall integrity 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 enable us to support abstraction of change operations.
- Most of the time, the intent of an ontology change is not explicitly defined in lower-level change operators, thus require representation of changes at higher levels. A layered change log framework fills this gap and helps an ontology engineer in explicit understanding of ontology changes.
- Discovering the recurring content change pattern from layered change log provides an opportunity to define reusable domain-specific change patterns that can be implemented encapsulating existing information systems.

We discuss our empirical study and framework of change operators and patterns in Section 2. Layered change logs for ontology change representation are introduced in Section 3. We discuss metadata and storage aspects for ontology change logs in Section 4. A short evaluation is given in Section 5. Related work is discussed in Section 6 and we end with some discussion.

## 2 A Framework of Ontology Change Operators and Patterns

Different solutions are provided to handle ontology evolution [2][4][10]. In [2], the authors identified different phases of ontology evolution and captured the changes of an evolving ontology using operators. However, the identified change operators focuses on generic and structural changes, lacking domain-specificity and abstraction that we aim to address through domain-specific perspective-linking structural changes in domain ontologies.

## 2.1 Empirical Study of Evolution of Domain Ontologies

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. Here we provide a small description of our empirical study – more details can be found in [6].

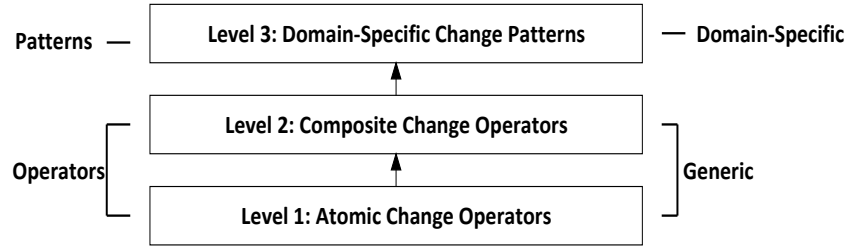
As case studies, the domains *University Administration* and *Database Systems* were taken into consideration. The former is selected because it represents an organisation involving people, organisational units and processes. The latter is a technical domain that can be looked at from different perspectives for instance, being covered in a course or a textbook on the subject. 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. In the university ontology, changes are frequent at instance level due to people joining or leaving, the introduction of new courses etc., but do also occur, albeit more irregular at concept level. Domain experts in both areas have contributed to the study [5]. Based on our observation of common changes in all ontologies, we studied the patterns they have in common, resulting in a layered framework of change operators (Figure 1): *level one* captures elementary changes which are atomic tasks, *level two* captures aggregated changes to represent composite, complex tasks, and *level three* captures domain-specific change patterns.

## 2.2 Layered Framework

*Level One Change Operators - Element Changes:* These change operators are the elementary operations used to perform a single task on a single targeted entity by an ontology management tool. These operators add 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.

*Level Two Change Operators - Composite Changes:* Many evolution tasks cannot be done by a single atomic operation. Change operators need to be composed to perform a composite task. For example if an instructor wants to split a chapter into two individual chapters, the operator “*Split Concept*” can be used. If s/he wants to combine two or more chapters for an abridged course outline, the operator “*Merge Concept*” can be used.

*Level Three Change Operators - Domain-specific:* This domain-specific perspective links the structural changes to the aspects represented in domain ontologies.



**Fig. 1.** Layered Framework of Change Operators and Patterns

In order to execute a single domain-specific change, operations at level one and two are used. 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 in terms of their numbers and sequence.

Level three operators enable us to treat domain-specific operations separately and allow an ontology engineer to define customised, reusable ontology change patterns. For example, 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, patterns 1, 3, 9 can be chosen whenever a new chapter is to be added. 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 user, without going to the details of the lower levels, customised patterns can be chosen and change executed based on the complex patterns defined.

**Table 1.** List of Change Patterns 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	

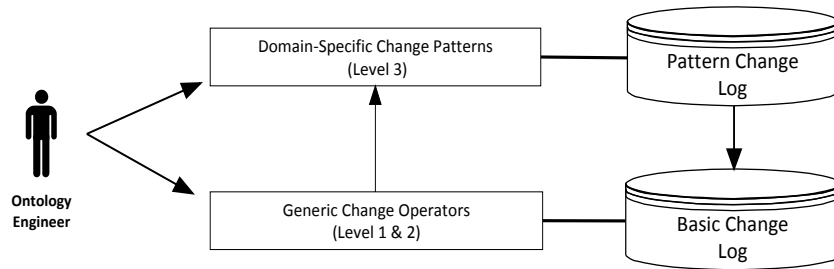
In the university ontology, if a user needs to register a new category of faculty using the manage faculty change pattern, say a Lecturer, then a concept “*Lecturer*” and properties such as “*hasPublication*” and “*supervise*” are created. Another user may create a new concept “*Lecturer*” without including the “*supervise*” property. Note, that this is a structural change at concept level.

### 3 Layered Logs for Ontology Change Representation

Ontology change logs can play a significant role in ontology evolution if there is a need to reverse a change, we use the change log to undo/redo the changes applied in the past. This is a common function in e.g. software versioning support.

In collaborative environments, change logs are also used to keep the evolution process transparent and centrally manageable. It captures all changes ever applied to any entity of ontology using elementary changes. However, we propose (and focus) here on a mechanism of representing ontology changes expressively at different levels of granularity (i.e. fine-grained changes such as the creation of a single class and also coarse-grained changes such as the merging two sibling classes) [11].

Representing the change log at the elementary level does not suffice. As the intent of the ontology change is missing from such change logs (and mostly specified at higher, domain-specific level of granularity), the ontology engineer is unable to understand why changes were performed, whether it is an elementary level change or a part of composite change and what is the impact of such changes is. We attempt to mine valuable information from a change log, making it easy for the ontology engineer, (other) users and machines to understand and interpret the ontology modifications. We propose a layered change log model, containing two different levels of granularity, i.e. a Basic Change Log (*BCL*) and a Pattern Change Log (*PCL*), shown in Figure 2.



**Fig. 2.** Layered Change Log

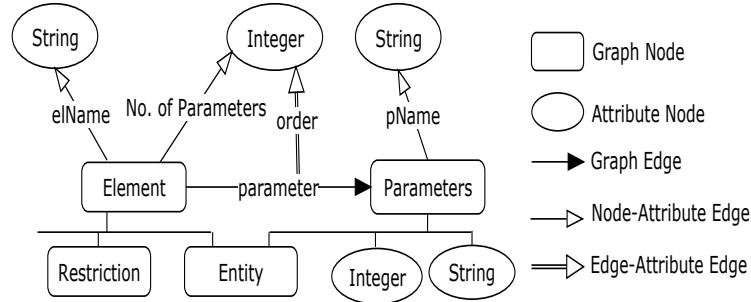
Layered change log work with the layered change operator framework presented in Section 2. The basic change log contains generic level changes and the pattern change log contains the information about the domain-specific change patterns. Using pattern change logs, one can capture the objective of the ontology changes at a higher level of abstraction and will help in comprehensible understanding of ontology evolution. Storing ontology changes at two different levels of abstraction can also help us in identifying recurring domain-specific change patterns from low level logs. We discuss this *pattern discovery* and *pattern matching* in Section 3.2 after investigating log representation and storage.

### 3.1 Graph-based Representation for Layered Change Logs

A graph-based representation is an operational representation for the layered change logs. Graphs enable efficient search and analysis and can also communicate information visually. The benefit of a graph-based representation structure

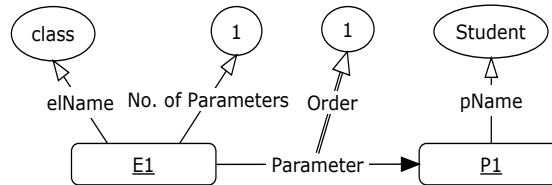
is the availability of well established algorithms and its well known characteristics such as performance, which can be used for querying the change logs effectively.

Our graph is linear sequential, i.e. there are no concurrent change operations reflected in the graph. We followed [8], where the idea is to use attributed graphs which are typed over an attributed type graph (*ATG*) with node and edge attribution. Attributed type graphs ensure that all edges and nodes of a graph are typed over the *ATG* and each node is either a source or target connected by an edge. *BCL* and *PCL* are typed by a generic *ATG* where attributes carry labels, types of nodes refer to the respective ontology elements and types of edges refer to the change operation applied. However, several instances of a *PCL* can occur in a *BCL*, that is a *PCL* could be a subgraph of *BCL* expressing that potentially several patterns can capture the same sequence of elementary changes. Figure 3



**Fig. 3.** Attributed Type Graph (*ATG*) Representation of Change Log

shows a portion of an Attributed Type Graph for a *BCL*. Each attribute node is named after the data type an instance can have and each graph node represents a conceptual representation of a change log entity. An attributed graph typed is given in Figure 4. The types defined on the nodes and edges can be represented as  $t(E1) = \text{Entity}$ ,  $t(\text{Class}) = \text{String}$ ,  $t(1) = \text{Integer}$  and  $t(P1) = \text{String}$ .



**Fig. 4.** Typed Attributed Graph of Change Log

### 3.2 Identification of Recurrent Patterns in Change Log

During our empirical case study, we observed that a number of sequentially ordered change operations are exercised by the users repeatedly during the evo-

lution of domain ontologies. Such a sequenced bundle of change operations are presented multiple times as a chain of single atomic changes in a basic change log. We are interested in identifying such frequent recurring change patterns automatically. The motivation behind it is the reusability of domain-specific change patterns, in line with the idea of managing change and maintaining consistency through pattern-based ontology evolution.

We considered identifying recurring sequenced change operations from change log as a problem of recognition of frequent pattern in a graph i.e. graph-based pattern discovery and pattern matching.

- *Pattern Discovery*: As discussed in Section 2.1, domain-specific change patterns can provide guidelines to content change management and support for evolution of information systems. In a number of organisations, content change management is already supported by domain-specific ontologies. Discovering the recurring (but not yet explicitly defined) change patterns can provide an opportunity to define reusable domain-specific change patterns that can be implemented encapsulating existing information systems.
- *Pattern Matching*: A user can also search in layered change logs for already defined domain-specific change patterns for better understanding that how ontology evolves through time and in which segments such domain-specific change patterns had been used.

Details of these algorithms can be found in [1]. The result of the change pattern identification is a set of subgraphs (change patterns). Based on the resulting subgraphs, a user can select the potential change pattern candidates and store them for further reuse. User may also customise the candidate change patterns by adding/deleting or editing the change operations.

## 4 Metadata and Storage System for Ontology Change Logs

In order to conceptualise the ontology changes, we constructed a *metadata ontology* by looking into concrete structure of OWL-DL syntax-based domain ontologies. The metadata ontology represents different categories of ontology changes based on our layered change operator framework (i.e. atomic, composite, pattern-based), types of ontology elements (such as concept, axioms, restriction etc.) and other concepts such as change, users, timestamp etc. Each instance of the change log is of type *Change*, available in the metadata ontology.

In order to implement a uniform and efficient storage solution for both ontologies and change logs, we have used RDF triple stores. The Sesame native triple store has been used for storage of the domain ontologies, static metadata ontology and change logs. The change log and ontologies are recorded in the form of RDF triples. Sesame provides an open source API for fine-granular access to the repository. It offers methods to infer the knowledge which is not explicitly given in the ontology. SPARQL format queries are used to extract the data from the triple store repository.

We have identified two types of information, which are essential to be stored into any change log instance i.e., static properties and change properties. Table 2 shows an example of single ontology change log instance *Add subclassOf* (“*PhDStudent*”, “*Student*”), stored in the triple store.

Our ontology editing framework (OnE) offers an graph API which can be used for generating graphs (of type GraphML) from change log triples, extracted from sesame repository and also for reading graphs.

**Table 2.** An Example of Change Log Triple stored in a Triple Store

Subject	Predicate	Object
<b>Static Properties</b>		
Change:142845	rdf:type	Metadata:Change
Change:142845	xsd:ID	“142845”
Change:142845	Metadata:hasCreator	Metadata:Javed
Change:142845	Metadata:Timestamp	xsd:Jan 18 16:28:14 GMT 2011
<b>Change Properties</b>		
Change:142845	Metadata:hasOperation	Metadata:Add
Change:142845	Metadata:hasAxiom	Metadata:subclassOf
Change:142845	Metadata:hasParameter1	University:PhdStudent
Change:142845	Metadata:hasParameter2	University:Student

## 5 Evaluation

We have looked at practical validity and adequacy of the pattern framework as evaluation criteria. 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. Pattern change log support higher level of abstraction of ontology changes which are not visible at lower levels. Though it is not expected to be exhaustive, we found that a significant portion of ontology change and evolution is represented in our layered framework, making the supported operators and patterns valid from a practical perspective. Our empirical study results confirm that the lower-level change operators are useful to ontology engineers to suitably define their own change operations, i.e. provide an adequate customisation solution. Domain experts can use the patterns and alter them to meet their requirements by varying the sequentialisation of the content elements.

We conducted experiments on a number of change log case scenarios empirically in order to identify the frequent change patterns. We found that the identified patterns capture the core segments of the ontology evolution and can be reused to construct new domain-specific change patterns. The identified patterns facilitate us in discovering co-occurrences of the change operations and building association rules. Such rules become very helpful in optimizing and improving the layout of the user-defined domain-specific change patterns. The results acknowledged that the proposed pattern identification framework facilitates a structured evolution process and reduces effort in terms of time consumption.

## 6 Related Work

A number of researchers have addressed layered change operator framework. The most closely related approach in this regard is given in [2]. The author focuses on generic and structural changes, but lacking the domain-specificity and abstraction. Moreover, this solution lacks adequate support for different levels of granularity at different levels of abstraction. In [4], the author provides a set of possible ontology change operations based on the effect with respect to the protection of the instance-data availability. This work focuses more on instances than on structural or domain-specific operations. In [10], 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 and goes beyond our presented solution in terms of impact analysis, our work takes the semantic changes further and proposes domain-specific change patterns for semantic changes.

Regarding ontology change representation, researchers have adopted different representations if ontology changes. In [11], the author chooses the version log approach to represent the evolution of ontology entities and as a source for ontology change detection. In [12], structural differences between two different versions of ontologies are captured. This method compares current and previous versions of the ontology structurally; however, it does not illustrate how the current version of the ontology is reached. In contrast, in [13], the author uses the representation of conceptual relations of frames between two versions of ontology.

## 7 Discussion

The presented work continues our previous research [6][7] by adding operational, pattern-based change representation and analysis mechanisms. We discussed our approach as a pattern-based compositional framework. The approach focuses on three levels of change operators which are based on granularity and domain-specificity. Multilayered change logs have been proposed for ontology change representation and a graph-based representation has been suggested as the formalism to capture and analyse ontology change logs.

Our framework enables us to deal with structural and semantic changes at separate levels without losing their interdependence. Furthermore, it enables us to define a set of domain-specific changes. The empirical study indicates that the solution is valid and adequate to efficiently handle ontology evolution. We have investigated application scenarios where ontologies are used to annotate content. Our observation here is that both ontology changes affecting content, but also content changes affecting the ontology used for annotation occur. In these larger system, change is very frequent and combined with the volume of information affected, makes automated tool support to manage change at the right (i.e. higher, application domain) level of abstraction highly beneficial.

In terms of future work, the pattern-based identification algorithms we currently use have been developed for directed, dynamic graph-based patterns. An

optimisation of these in terms of better capturing structural ontology aspects is planned. We will also supplement our technique with improved impact determination and consistency management.

## Acknowledgment

This research is supported by the Science Foundation Ireland (Grant 07/CE/I1142) as part of the Centre for Next Generation Localisation ([www.cngl.ie](http://www.cngl.ie)) at Dublin City University.

## References

1. Gacitua-Decar, V., Pahl, C. Ontology-based Patterns for the Integration of Business Processes and Enterprise Application Architectures. In G. Mentzas et al. (Eds). *Semantic Enterprise Application Integration for Business Processes: Service-Oriented Frameworks*. IGI Pub. 2009.
2. Stojanovic, L.: Methods and tools for ontology evolution. PhD thesis, University of Karlsruhe (2004)
3. Filipowska, A., Kaczmarek, M., Markovic, I.: Organisational ontology framework for semantic business process management In Proc. 12th Intl Business Information Systems Conference (BIS 2009). Springer LNBIP, Vol. 21, pp. 1-12.
4. Noy, N.F., Klein, M.: Ontology evolution: Not the same as schema evolution. SMI technical report SMI-2002-0926, 2002
5. Boyce, S., Pahl, C.: The development of subject domain ontologies for educational tech. systems. In *Journal of Educational Tech. and Society* **10**(3) (2007), pp. 275–288.
6. Javed, M., Abgaz, Y., Pahl, C.: A pattern-based framework of change operators for ontology evolution. In 4th International Workshop on Ontology Content. Volume 5872 of *Lecture Notes in Computer Science.*, Springer (2009), pp. 544–553.
7. Abgaz, Y., Javed, M., Pahl, C.: Empirical analysis of impacts of instance-driven changes in ontologies. In: 6th International Workshop on Ontology Content. *Lecture Notes in Computer Science*, Springer (2010)
8. Ehrig, H., Prange, U., Taentzer, G.: Fundamental theory for typed attributed graph transformation. In *Proc. of 2nd Int. Conf. on Graph Transformation* (2004), pp. 161–177.
9. Hesse, W. In: *Engineers Discovering the Real World From Model-Driven to Ontology-Based Software Engineering*. Volume 5 of *Lecture Notes in Business Information Processing*. Springer (2008), pp. 136–147.
10. Qin, L., Atluri, V.: Evaluating the validity of data instances against ontology evolution over the semantic web. *Information and Software Technology*. **51**(1) (2009), pp. 83–97
11. Plessers, P., De Troyer, O.: Ontology change detection using a version log. In: *Proc. of the 4th International Semantic Web Conference*, Springer (2005), pp. 578–592.
12. Noy, N.F., Musen, M.A.: A fixed-point algorithm for comparing ontology versions. In *Eighteenth National Conference on Artificial Intelligence* (2002)
13. Klein, M., Kiryakov, A., Ognyanov, D., Fensel, D.: Ontology versioning and change detection on the web. In: *13th Int. Conf. on Knowledge Engineering and Knowledge Management*. (2002), pp. 197–212.