Layered Change Log Model: Bridging between Ontology Change Representation and Pattern Mining

Muhammad Javed, Yalemisew M. Abgaz , Claus Pahl

School of Computing
Dublin City University
Dublin, Ireland
E-mail: [mjaved|yabgaz|cpahl]@computing.dcu.ie

Abstract: To date, no ontology change management system exists that records the ontology changes based on the different levels of granularity. Once changes are performed using elementary level change operations, they are recorded in the database at the elementary level accordingly. Such a change representation procedure is not sufficient to represent the intuition behind any applied change and thus, cannot capture the semantic impact of a change. In this paper, we discuss recording of the applied ontology changes in the form of a layered change log. We support the implementation of a layered change operator framework through layered change logs. We utilize the lower level ontology change log in two ways, i.e. *recording* of applied ontology changes (operational) and *mining* of higher level change patterns (analytical). The higher level change logs capture the objective of the ontology changes at a higher level of granularity and support a comprehensive understanding of the ontology evolution. The knowledge-based change log facilitates the detection of similarities within different time series, mining of change patterns and reuse of knowledge.The layered change logs are formalised using a graph-based approach.

Keywords: Layered Change Log Model; Atomic Change Log; Pattern Change Log; Ontology Evolution; Triple-based Ontology Change Representation.

1 Introduction

Ontology-based information models helped researchers to take a step forward from traditional content management systems (CMS) to conceptual knowledge modelling to meet the requirements of the semantically aware information systems. Ontology-based approaches can be used to capture the implicit knowledge, architecture and process patterns [2, 4]. Ontologies can convey the useful semantic information for content managers and domain experts to understand and process. Domain ontologies have become essential for conceptualization and knowledge sharing activities in dynamic information system. Such information systems are always subject to change and ontology change management can pose challenges. The reason for such changes can be the changes in the domain, the specification, the conceptualization or any combination of them [5]. A change in an ontology may originate from a domain expert, a user of the ontology or a change in the application area [6]. Some changes are about the introduction of new concepts, removal of outdated concepts and changes in the structures and the description of concepts. Ontology changes need to be recorded and represented in a suitable format so that they are human and machine processable.

In this paper, we present a novel approach regarding recording of applied ontology changes in the form of a change log. We present a Layered Change Log Model (LCLM) to deal with the customization and abstraction of ontology-based model evolution. The implementation of the change operator framework (discussed in [3]) is supported through a layered change log model. Such layered change logs not only reflect the changes in the domain but also the user requirements, flaws in the initial design and need to incorporate additional information. The central technical contributions of this research work are as follows:

- Explicit representation of the intent of an ontology change at a higher level of granularity using *ontology change patterns*. The model enables dealing with the structural and the semantic changes at two separate levels without losing their interdependence.

- Ontology change logs can provide operational as well as analytical support in the ontology evolution process. We utilize ontology change logs in two perspectives i.e., *recording* of applied ontology changes and *mining* of higher level change patterns.

The paper is structured as follows: In section 2 we present our layered change log model. In section 3, we discuss the RDF framework format that is used to construct and represent the applied changes in RDF triple-form. In section 4, we discuss the layered change log in terms of recording of applied changes. In section 5, we discuss the mining of ontology change patterns from lower-level change log. A short evaluation is given in section 6. We end with related work and a brief discussion.

2 Layered Change Log Model

The representation of ontology changes has a major role in supporting the management of ontology evolution and related activities [7]. The atomic level representation of applied ontology changes can only depict addition or deletion of any axiom from the domain ontology. The representation of intent behind any applied ontology change is missing from such change representation and mostly specified/deduced at a higher level of granularity. It is hard for an ontology engineer to understand why changes were performed, whether it is an atomic level change or a part of composite change and what is the impact of such change.

We used change logs to record the applied ontology changes. The change logs provide detailed information about the ontology changes including who performed the change, when the change was performed, what was the change request and what was the impact of such change request on the neighbourhood entities (i.e. cascaded impact) [1]. Our concern here is not only to determine the ontology differences between the versions, but also how it has changed from an operational perspective and to support an ontology engineer in executing the changes (through identified patterns).

In comparison to the change logs, representing the ontology changes in terms of successive versions of domain ontology provide less details. The successive versions inform only about the previous and the current state of the ontologies. Different versions of domain ontology represent *what* has changed, but the details, about *how* the domain ontology has evolved from one version to the other and how one reaches to the current version of the ontology, are missing. In terms of identification of ontology change patterns, ontology change logs allow to identify the patterns across the sessions. This cannot be achieved using ontology versions.

Based on our layered change operator framework [3], we propose a Layered Change Log Model (LCLM), containing two different levels of granularity, i.e. an Atomic Change Log (ACL) and a Pattern Change Log (PCL)- Figure 1.

- *Atomic change log (ACL)*: Atomic change log contains atomic level change operations. Benefit of storing ontology changes at atomic level is their fine-grained and complete representation. Fine-grained representation of ontology changes help ontology engineer to understand the impact of the ontology changes at the atomic level. However, in most of the cases, the ontology changes are being applied in a group. Thus, the impact of the grouped changes must be identified at a higher level rather than atomic. One can extract such higher level change operations from the atomic change log, using pattern mining mechanisms, that leads to a comprehensive ontology change management approach.
- *Pattern change log (PCL)*: Pattern change log contains the higher level change operations, i.e., composite and domain-specific change patterns [10]. Using pattern change log, one can capture the objective of the ontology changes at a higher level of abstraction that help in comprehensible understanding of ontology evolution. The intent behind any applied change is more visible at the pattern level as compared to the atomic.

The layered change log model has been used to achieve two purposes, i.e.,

- *recording* ontology changes at different levels based on the utilized change operators representing operational change log data (discussed in section 4) and
- *mining* of valuable knowledge such as intent behind any applied change, domain-specific change patterns etc. from analytical change log data (discussed in section 5).

3 RDF Framework Format

We use RDF triple-based representation, i.e., *subject - predicate - object* (spo), to conceptualize the domain ontology changes in the change logs. In order to keep a transparent record of applied ontology changes, we record two types of core metadata, i.e. who performed the change (User) and when the change was performed (Timestamp). To record such metadata, *Provenance Vocabulary Core Ontology*[1] terms can be used. In this regard, an ontology change can be considered as an activity (rdf:type :Activity) that is performed by a certain agent (:Activity :performedBy :Agent) where an agent can be a user (i.e., :HumanActor) or an application (:NonHumanActor). Further, a timestamp can be attached to such activity using the completedAt datatype proprerty (:Acitivity :completedAt xsd:dateTime). Similar to the provenance vocabulary core ontology, we constructed a change metadata model using the OWL language. As we

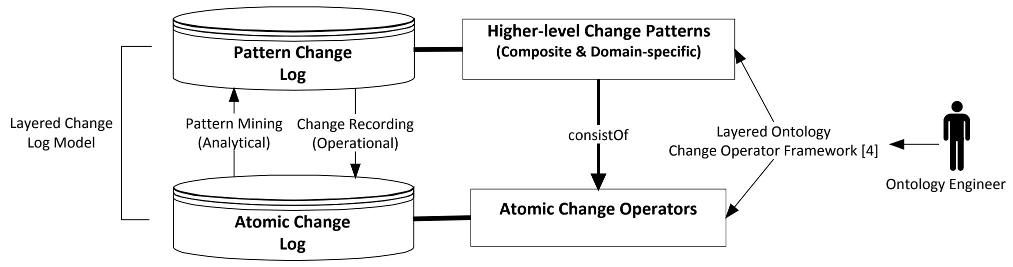
Pattern Change Log
Higher-level Change Patterns (Composite & Domain-specific)
Layered Change Log Model
Pattern Mining (Analytical)
Change Recording (Operational)
consistOf
Layered Ontology Change Operator Framework [4]
Ontology Engineer
Atomic Change Log
Atomic Change Operators

Figure 1 Layered Ontology Change Framework

conceptualize the change metadata model in a form of ontology, we use term "change metadata ontology" to represent the change metadata model. In order to maintain a fine granular ontology change representation, we distinguish five different knowledge gathering aspects (five W's) i.e.

- WHO performed the ontology change (User)
- WHEN the change is actually applied (Timestamp)
- HOW one can find a specific change in the log (Session & Change Id)
- WHAT is the change (Operations & Element)
- WHERE particular change is applied in the ontology - (Parameters)

The classes and properties available in the change metadata ontology assist the ontology engineer to construct the RDF triples, representing an applied ontology change. Similar to the approaches opted for [7] and [8], the idea here is to provide a metadata model that is generic, independent and extendable to represent the changes of the domain ontologies. We used an RDF triple store to record the change logs, domain ontologies and change metadata ontology. Thus, all ontology changes, stored in the ontology change log, are in a form of triples. The main classes and properties of our change metadata model are given in Figure 2.

The central class in the change metadata model is the concept `Change`. The class `Change` is subdivided into `AtomicChange`, `CompositeChange` and `PatternChange`. The metadata details of any applied change, such as, `sessionId`, `changeId`, `hasCreator`, `Timestamp`, etc., are given using object and data properties. The core details of an applied ontology change, i.e., operation, element and parameters, are given using object properties `hasOperation`, `hasElement` (which is further subdivided into `hasEntity` and `hasAxiom`) and `hasParam`, respectively. The object property `hasAxiom` is further categorized into `hasClassAxiom`, `hasObjectPropertyAxiom`, `hasDataPropertyAxiom` and `hasIndividualAxiom`. Similarly, object property `hasParam` is further categorized into `hasTargetParam`, `hasAuxParam1` and `hasAuxParam2`.

In order to express that the domain-specific change patterns are the combination of lower level change operations, the class `PatternChange` is associated to the class `AtomicChange` and the class `CompositeChange` using object properties `hasAtomicChange` and `hasCompositeChange`, respectively. Each stored domain-specific change pattern is an instance of (rdf:type) class `PatternChange`. The descriptive data of a change pattern, such as, label, change id, purpose etc., are given using properties `PatternName`, `changeId`, `PatternPurpose`, respectively. For each composite or pattern change, its constituent atomic or composite changes are recorded; however, a complete decomposition is not intended. A reconstruction of lower levels can be facilitated through a pattern/composite change definition repository to be kept separate from the operational data.

We differentiate between *declaration/remover* axioms, *property restriction* axioms and other axioms. An atomic change can performs three kinds of actions, i.e. i) addition/deletion of an entity (declaration/remover axioms), ii) addition/deletion of a constraint (restriction axiom) or iii) addition/deletion of a other (general) axiom. Therefore, we categorize the class `Ontology_Elements` into three subclasses, i.e. `Entity`, `Axiom` and `Restriction`. The concept `Axiom` is further divided into `ClassAxiom`, `ObjectPropertyAxiom`, `DataPropertyAxiom` and `IndividualAxiom`.

The domain ontology changes can be logged either at the instance level (Abox) of the change metadata model or can be logged separately in a form of change log. In our research, we separate the instance level data from the schema level data. Thus, the changes being applied on the domain ontologies are stored in a form of RDF triples in ontology change logs. (<`Change`> <`hasParam`> <`Entity`>), (<`Change`> <`sessionId`> <`XSD:Id`>) are the examples of such RDF triple types.

4 Recording of Ontology Changes

Based on the utilized change operators, the applied ontology changes are recorded at two different levels of abstraction. If a user employs atomic change operators,

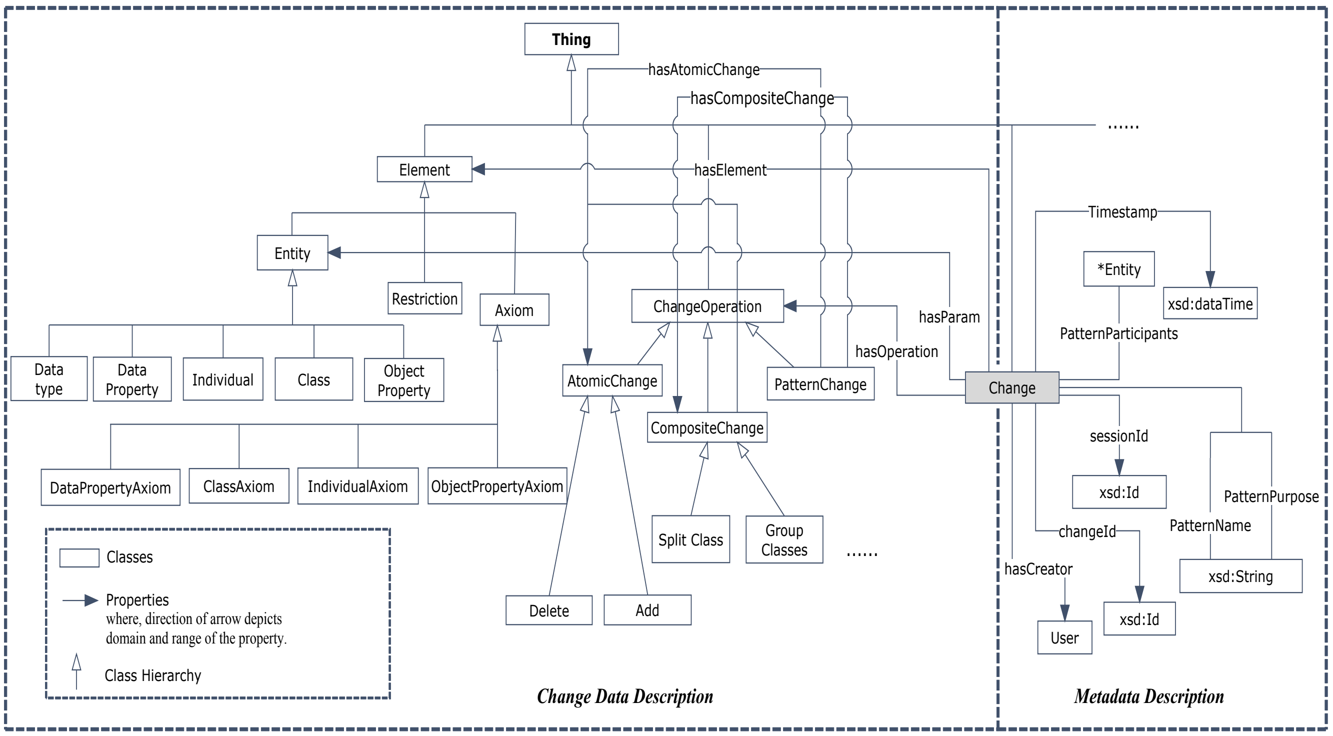

Figure 2 Change metadata ontology

the applied changes are recorded in atomic change log (ACL). ACL is a sequential change log where each ontology change operation is executed one after the other and the concurrent ontology change operations (if any) are sequenced. If a user makes use of higher-level change operator, the applied change is recorded as a change pattern in pattern change log. Further, for a complete representation of applied ontology changes, the applied change patterns are being recorded as a sequence of atomic change operations in atomic change log (Figure 3). The main challenge in recording of ontology changes is to keep the consistency between the two levels. Further, efficient storage and retrieval of change data is also a concern.

4.1 Recording of Atomic Changes in ACL

Atomic change log refers to the atomic level change representation of the applied ontology changes. The five aspects, mentioned in previous section, are combined together to represent a single atomic level ontology change (Figure 4).

4.1.1 Formalization

An atomic change log consists of an ordered list of ontology changes, $ACL = < ac_1, ac_2, ac_3 \cdots ac_n >$ where n refers to the sequence of ontology changes in a change log. Each atomic ontology change is an instance of concept `AtomicChange` of change metadata ontology. The change consists of two types of data, i.e. *metadata* (MD) and the *change data* (CD) - Figure 4. As we can see in change metadata ontology (given in Figure 2), the metadata provides the common details of the change, i.e. who performed the change, when the change was applied and how to identify such change from the change log. Thus, it can be given as $MD = (id_s, id_c, u, t)$ where id_s, id_c, u and t represent `sessionId`, `changeId`, `User` and `Timestamp`, respectively. The change data contain the central information about the change request and can be given as $CD = (o_p, e, p)$ where, o_p, e and p represent the `ChangeOperation`, `Element` and `Parameter Set` of a particular change. In Figure 2, such information is represented using object properties `hasOperation`, `hasElement` and `hasParam`.

4.1.2 Triple-based Representation of an Atomic Change

In order to implement a uniform and efficient storage solution, we used RDF triple-based representation, i.e., *subject-predicate-object* (spo). In this sense, RDF-triple stores can be used for the storage of the domain ontologies, change metadata ontology and the change logs; where, SPARQL format queries can be used to extract the data from the triple store repositories. Based on the atomic level change operation example given in Figure 4, the atomic change log entries for change operation `Add classAssertion (John, PhD_Student)` stored in the triple store, are given in Table 1.

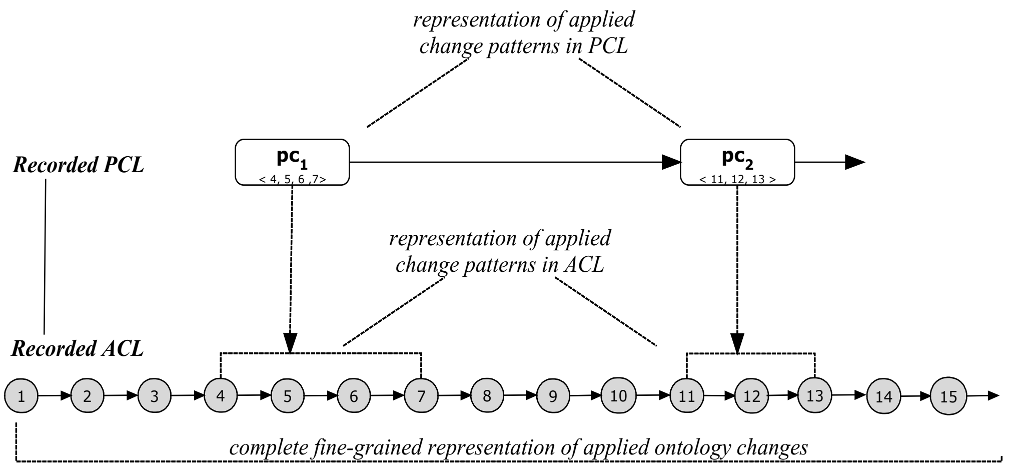
representation of applied change patterns in PCL
Recorded PCL
pc1 < 4, 5, 6 ,7>
pc2 < 11, 12, 13 >
representation of applied change patterns in ACL
Recorded ACL
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
complete fine-grained representation of applied ontology changes

Figure 3 Operational setup of ontology change logging

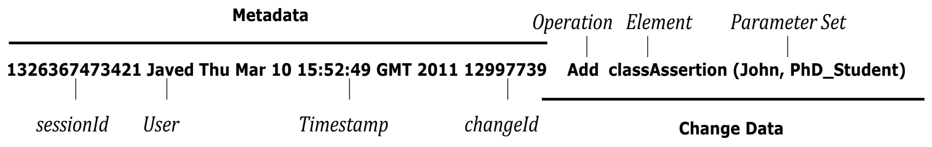
Metadata
Operation
Element
Parameter Set
1326367473421 Javed Thu Mar 10 15:52:49 GMT 2011 12997739 Add classAssertion (John, PhD_Student)
sessionId
User
Timestamp
changeId
Change Data

Figure 4 Representation of an atomic ontology change

Table 1 Triple-based representation of an atomic change

Subject	Predicate	Object
MO:12997739	rdf:type	MO:AtomicChange
MO:12997739	MO:sessionId	"1326367473421"
MO:12997739	MO:hasCreator	MO:Javed
MO:12997739	MO:Timestamp	"Thu Mar 10 15:52:49 GMT 2011"
MO:12997739	MO:changeId	"12997739"
MO:12997739	MO:hasOperation	MO:Add
MO:12997739	MO:hasIndividualAxiom	MO:classAssertion
MO:12997739	MO:hasTargetParam	University:John
MO:12997739	MO:hasAuxParam1	MO:PhD_Student

4.2 Recording of Change Patterns in PCL

Pattern change log refers to the recorded change patterns, being applied using higher-level change operations of the layered change operator framework. Such specification of the applied change patterns help ontology engineer to i) distinguish between the applied similar changes and ii) in understanding the purpose and consequences of the changes.

4.2.1 Formalization

A pattern change log consists of an ordered list of ontology change patterns, $PCL =< pc_1, pc_2, pc_3 \cdots pc_n >$ where n refers to the sequence of ontology change patterns in a pattern change log. These change patterns can either be generic composite change patterns or domain-specific change patterns (c.f. Figure 1). Similar to ACL, each ontology change pattern pc consists of two types of data i.e. *Metadata (D)* and *Pattern data (P)*. The metadata provides meta details about the change pattern and can be given as $D = (id_s, id_c, u, t, p_u)$ where, id_s, id_c, u, t and p_u represent the `sessionId`, `changeId`, `User`, `Timestamp` and `PatternPurpose`, respectively. The pattern data (P) provides description about the involved change operations. Here, P refers to the sequence of the change operations available in a change pattern $P = (c_1, c_2, \dots c_s)$ where, s is the total number of change operations in a change pattern.

4.2.2 Triple-based Representation of Change Patterns

Similar to the atomic change log, RDF triple store can be used for recording of applied higher level change patterns. In this sense, every applied change pattern is being recorded as an instance of either concept *CompositeChange* or *PatternChange*, available in change metadata ontology. Other common details, such as, session id, change id, time of the applied change pattern etc., are recorded using defined object and data properties in change metadata ontology. Table 2 represent a section of a domain-specific change pattern log entries, representing the instantiation of a "`PhD Student Registration`"change pattern in university administration domain. Such change pattern consist of a number of atomic change operations. First, a new individual "`John`" is being added as an instance of concept `PhD_Student`. Next, the details about the student id, assigned supervisor etc. have been added to the specified student. The application of a change pattern is a single transaction on a domain ontology. In such case, either the whole change pattern will be applied on the domain ontology, or will be discarded (rollback) completely.

5 Mining of Ontology Change Patterns

Higher-level change representation (in a form of change patterns) is more concise, intuitive and closer to the intentions of the ontology editors and captures the semantics of the change [9]. These ontology change patterns can either be explicitly defined by a user as a higher-level change operators or implicitly given in atomic change log that can be extracted using data mining techniques. We discussed recoding of explicitly user-defined change patterns in section 4.2. In this section, we discuss the mining of implicit usage-driven change patterns.

One of the benefits of the RDF triple format is its fine-grained level representation and interoperability (i.e. conversion from triple format to others standard formats such as RDF and XML). The fine-grained representation of ontology changes helps the ontology engineer to construct complex queries and extract different types of knowledge from the change log. However, as RDF triples represent the ontology changes at fine-grained level (one ontology change is represented by 8 to 10 triples), visualizing and navigating through the change log alone is time consuming. Graphs can cover this gap. Graph techniques provide the ability to visualize and navigate through large network structures. They enable efficient search and analysis and can also communicate information visually. Moreover, the benefit of a graph-based representation is the availability of well-established algorithms/metrics (for pattern discovery and detection) and its well-known characteristics such as performance (for querying the ontology changes effectively). We used graph-based algorithms for the change patterns discovery and matching from the atomic change log. A data warehouse mechanism can be applied here, where the data warehouse collects the operational domain ontologies and the change log data from different distributed locations and reformulates it into a graph on a periodic basis for analytical processing. The detailed description of the approach and the algorithms can be found in [12] and [13] and are out of scope of this paper.

5.1 Mining of Composite Change Patterns (Pattern Matching)

We mine the composite change patterns from an atomic change log, making it easy for the ontology engineer, (other) users and machines to understand and interpret the ontology modifications. The composite changes can be mined from the atomic change log using a pattern matching approach. Here, the term "pattern matching" refers to the mining of occurrences of a pre-defined change pattern sequences from an atomic change log. One may find the (complete or partial) overlapping among the mined change patterns. This is due to the possibility that a subset of a change pattern satisfy the conditions (to be identified) of another category of change pattern [11]. Apart from being more concise, the ontology changes recorded at a higher-level are more intuitive [9]. Identifying the composite changes from the atomic change log give ontology engineer indication about the intent of the applied changes. Once a user has clear understanding of semantics of a change, he can select appropriate evolution strategy for consistent ontology change management.

We use a graph-based pattern matching approach where the atomic change log (in the form of a graph) and a reference composite change (representing a pre-defined composite change pattern) are given as an input to the algorithm. The algorithm identifies the occurrences of the reference composite change from the atomic change log. The detailed description of the approach and the algorithm is given [13].

5.2 Mining of Domain-specific Change Patterns (Pattern Discovery)

We mine the domain-specific change patterns from the atomic change log. As two identified ontology change subsequences can be different from each other in terms of number or type of involved change operations but may have the same effect on the underlying domain ontology, the main objective here is not only to capture the recurrent change subsequences from ACL that are applied in a same order (i.e., *structurally* identical change patterns) but also the change subsequences that may contain different order of change operations but have the same effect on the domain ontology (i.e., *semantically* identical change patterns). On one hand, similar to the mining of composite change patterns, the mining of domain-specific change patterns from the atomic change log benefited us in terms of higher-level ontology change representation in PCL, but on the other hand, the identified domain-specific change patterns can also be utilized in future as a once-off change pattern specification that can be instantiated whenever a user

Table 2 Triple-based description of (domain-specific) change pattern

Subject	Predicate	Object
MO:13238651	rdf:type	MO:PatternChange
MO:13238651	MO:sessionId	“1326367473421”
MO:13238651	MO:hasCreator	MO:Javed
MO:13238651	MO:Timestamp	“Thu Mar 10 15:52:49 GMT 2011”
MO:13238651	MO:changeId	“13238651”
MO:13238651	MO:PatternName	“PhD Student Registration”
MO:13238651	MO:PatternPurpose	“Purpose is to register a new PhD student in school”
MO:13238651	MO:containAtomicChange	MO:12997738
MO:13238651	MO:containAtomicChange	MO:12997739
MO:13238651	MO:containAtomicChange	MO:12997740
MO:13238651	MO:containAtomicChange	MO:12997741
MO:13238651	MO:containCompositeChange	MO:12997742
MO:12997738	rdf:type	MO:AtomicChange
MO:12997738	MO:sessionId	“1326367473421”
MO:12997738	MO:hasCreator	MO:Javed
MO:12997738	MO:Timestamp	“Thu Mar 10 15:52:49 GMT 2011”
MO:12997738	MO:changeId	“12997738”
MO:12997738	MO:hasOperation	MO:Add
MO:12997739	MO:hasEntity	MO:Individual
MO:12997739	MO:hasTargetParam	“John”
MO:12997739	rdf:type	MO:AtomicChange
MO:12997739	MO:sessionId	“1326367473421”
MO:12997739	MO:hasCreator	MO:Javed
MO:12997739	MO:Timestamp	“Thu Mar 10 15:52:49 GMT 2011”
MO:12997739	MO:changeId	“12997739”
MO:12997739	MO:hasOperation	MO:Add
MO:12997739	MO:hasIndividualAxiom	MO:classAssertion
MO:12997739	MO:hasTargetParam	University:John
MO:12997739	MO:hasAuxParam1	MO:PhD_Student
MO:12997740	rdf:type	MO:AtomicChange
MO:12997740	MO:sessionId	“1326367473421”
MO:12997740	MO:hasCreator	MO:Javed
MO:12997740	MO:Timestamp	“Thu Mar 10 15:52:49 GMT 2011”
MO:12997740	MO:changeId	“12997740”
MO:12997740	MO:hasOperation	MO:Add
MO:12997740	MO:hasIndividualAxiom	MO:dataPropertyAssertionAxiom
MO:12997739	MO:hasTargetParam	University:John
MO:12997739	MO:hasAuxParam1	MO:studentId
MO:12997739	MO:hasAuxParam2	“5810638”

needs to apply the similar changes on the underlying domain ontology.

We used a graph-based pattern discovery approach in order to mine the recurrent domain-specific change patterns from the atomic change log. Here, the term “pattern discovery” refers to the mining of a change pattern from the atomic change log without having any prior knowledge about them. As mentioned above, we selected a flexible notion of a change pattern where the atomic change operations can be in ordered or unordered form. The change patterns are identified based on their support (i.e., number of occurrences of such change pattern sequences in the atomic change log), pattern length (i.e., number of atomic changes available in a change pattern sequence) and the node-distance (i.e., the permissible gap between two adjacent nodes of a change pattern subsequence). The change patterns are divided into two categories i.e., *ordered change patterns* and *unordered change patterns*. Such ordered/unordered subsequences of an identified domain-specific change pattern can be *complete* or *partial* with respect to the candidate change pattern subsequence. The threshold values for minimum pattern support, minimum pattern length and the permissible node-distance are the inputs to the domain-specific change patterns discovery algorithms. The detailed description of the approach and the algorithms are given in [10].

6 Evaluation

The objective here is to evaluate the layered change log model based on its functional suitability. The changes must be represented in such a way that it is useful and understandable by the domain experts and ontology engineers. In this regard, the functional suitability of the layered change log model is, first, to maintain a comprehensive understanding of the evolution of domain ontologies and, second, to explicitly present the intent behind any applied change. We made use of user feedback as a metric in order to evaluate and answer the specified questions. Below, we discuss the experimental setup (in terms of change log construction) and the results.

To validate the change log model, we made use of the existing empirical case study data [3] from the university domain ontology. We utilized a user-based evaluation in order to empirically evaluate the layered change logs. The ontology engineers performed the given change operations using atomic change operators and higher-level change patterns. The applied changes had been logged into atomic and pattern change logs accordingly. We presented the two change logs to the users in order to manually analyze how the changes have been recorded in the layered logs. Do the logs maintain a fine-grained representation of ontology changes? Is representation of ontology change at a higher-level in the form of patterns more intuitive? To answer these questions, we involved five ontology engineers who have expertise in the area of software engineering and large databases. We requested the participants to give a rating to the claims we made about the functional suitability and usability of the layered framework. The claims are rated separately by each ontology engineer from 1 to 5 (where rating 5 represents ”strongly agree”, 4 ”agree”, 3 ”neutral”, 2 ”disagree” and 1 ”strongly disagree”). The claims and the user-based ratings (average) are given in Table 3.

The feedback from the participants confirm that the solution is useful and functionally suitable for the ontology engineers and domain expert. The highest rating was given to claim 1 (i.e. ACL presents a complete fine-grained representation of ontology changes). Participants agree that the representation of the ontology changes at a higher level helped them to understand the intent behind the applied changes - making the solution practically valid. The lowest rating was given to claim 4 (i.e. recording of ontology and change logs in a single RDF repository allows user to concurrently navigate). Participants agree that the framework does allow concurrent navigation. However, a graph-based illustration of associations between change log and domain ontologies would be more intuitive for the users.

7 Related Work

Based on the different perspectives of the researchers, there are different solutions provided to handle ontology evolution [1, 14, 15, 16, 17, 18, 22]. Representation of ontology changes using higher-level change operations was first proposed by Stojanovic [14] and Klein [19]. Recently, some researchers have focused on representation and detection of higher-level ontology changes [9, 20]. Stojanovic choose the *evolution logs* in order to record the applied changes. Evolution log keeps track of applied ontology changes in the exact order in which changes have been applied to the domain ontology. In case of any failure, evolution log makes ontology recovery possible. Plessers selects *version log* [17] in order to represent the evolutionary aspects of domain ontologies. According to Plessers, *a version log stores different versions of every entity (which includes concepts, properties and individuals) ever defined in a domain ontology.* The purpose of version log is to keep record of the different phases the entities pass through, from their creation, modification to deletion.

Where few researchers focused on representation of ontology changes using change logs, others highlight the evolutionary aspects of an ontology by comparing two different versions of it. Noy proposed a fixed point algorithm [22] in order to capture the structural differences between two ontology versions in the absence of ontology change logs. Klein proposed a *transformation set* [19] that provides a list of change operations that if applied to the V_{old} (old version of ontology), the set transforms it to V_{new} (the new version of ontology). Such transformation set can include elementary change operations, complex change operations or combination of them. Transformation sets are different from the basic change logs due to a number of reasons. First, atomic change log contains the record of all the applied changes, however, the transformation sets contains only the necessary set of change operations for achieving the resulting change. Second, basic change logs contain the *ordered* sequence of change operations. Whereas, in transformation set, such ordering is very limited and it is primarily, that all the *additive* operations will take place before any other change operation. Third, a change log is a distinctive representation of the exact applied change operations, whereas, there can be several unique transformation sets that can produce the same resulting change.

8 Discussion

In this paper, we discussed our approach for ontology evolution as a pattern-based compositional framework. We presented a layered change log model that works in line with the given layered change operator framework. While ontology engineers typically deal with generic changes at lower level, other users (such as domain

Table 3 Questionnaire-based evaluation of the layered framework

No.	Claim	User's feedback: 1 - 5 (Avg)
1	Atomic Change Log (ACL) presents a complete and fine-grained representation of applied ontology changes (***Completeness***).	4.67 (93.33%)
2	Pattern Change Log (PCL) supports in understanding the intent behind an applied ontology change (***Validity***).	4.00 (80.00%)
3	The ontology changes recorded in ACL and PCL are easily understandable (***Validity***).	4.33 (86.67%)
4	Recording of domain ontology and change log in a single RDF repository allows the user to concurrently navigate through them (***Functional suitability***).	3.67 (73.33%)
5	The framework is easy to understand, learn and use (***Usability***).	4.33 (86.66%)
6	The customizable evolution strategies allow users to evolve the ontology based on their own needs (***Adequacy***).	4.33 (86.66%)

experts, content managers) can focus on domain-specific changes at patter level. Such a layered change framework enables us to deal with structural and semantic changes at two separate levels without losing their interdependence. Plus, it enables us to define a set of domain-specific changes which can be stored in a pattern catalogue, using a pattern template, as a consistent once-off specification of domain-specific change patterns. The empirical study indicates that the solution is valid and adequate to efficiently handle ontology evolution. We found that a significant portion of ontology change and evolution is represented in our framework.

Mining of higher-level change patterns gives an ontology engineer clues about semantics behind any of the applied change, based on the actual change activity data from change log. We operationalized the identification of higher-level changes using graph-based matching and pattern discovery approaches. We noticed that learning about semantics behind any of the applied change helped us in keeping the ontology consistent in a more appropriate manner.

Acknowledgement

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

References

[1] Abgaz, Y.M., Javed, M., Pahl, C.: Empirical Analysis of Impacts of Instance-Driven Changes in Ontologies. In Proc: On the Move to Meaningful Internet Systems: OTM 2010 Workshops. Volume 6428 of Lecture Notes in Computer Science, Springer-Berlin/Heidelberg, (2010) 368–377.

[2] 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 Publishers. (2009) 36-60.

[3] Javed, M., Abgaz, Y.M., Pahl, C.: A pattern-based framework of change operators for ontology evolution. In: On the Move to Meaningful Internet Systems: OTM Workshops. Volume 5872 of LNCS., Springer (2009) 544-553.

[4] Javed, M., Abgaz, Y., Pahl, C.: Towards implicit knowledge discovery from ontology change log data. In: 5th International Conference on Knowledge Science, Engineering and Management (KSEM). Volume 7091 of Lecture Notes of Artificial Intelligence., Springer-Verlag (2011) 136-147

[5] Noy, N.F., Klein, M.: Ontology evolution: Not the same as schema evolution. In: Journal of Knowledge and Information Systems. Volume 6(4). (2004) 328-440

[6] Liang, Y., Alani, H., Shadbolt, N.: Ontology change management in protégé. In: Proceedings of AKT DTA Colloquium, Milton Keynes, UK. (2005)

[7] Palma, R., Haase, P., Corcho, O., Gomez-Perez, A.: Change representation for owl 2 ontologies. In: Proceedings of the sixth international workshop on OWL: Experiences and Directions (OWLED). (2009)

[8] Pedrinaci, C., Domingue, J.: Towards an ontology for process monitoring and mining. In: Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management. (2007)

[9] Papavassiliou, V., Flouris, G., Fundulaki, I., Kotzinos, D., Christophides, V.: On detecting high-level changes in RDF/S KBs. In: 8th International Semantic Web Conference. Volume 5823 of LNCS., Springer (2009) 473-488

[10] Javed, M., Abgaz, Y.M., Pahl, C.: Graph-based discovery of ontology change patterns. In: Joint Workshop on Knowledge Evolution and Ontology Dynamics (EvoDyn): ISWC Workshops. (2011)

[11] Javed, M., Abgaz, Y., Pahl, C.: Composite ontology change operators and their customizable evolution strategies. In: ISWC Workshops: Joint Workshop on Knowledge Evolution and Ontology Dynamics (EvoDyn), 12th November, 2012, Boston, USA. (2012)

[12] Javed, M., Abgaz, Y., Pahl, C.: Ontology change management and identification of change patterns. In: Special Issue of the Journal On Data Semantics on Evolution and Versioning in Sematic Data Integration Systems. Volume 2(2-3), Springer-Verlag (2013) 119-143

[13] Javed, M.: Operational Change Management and Change Pattern Identification for Ontology Evolution. PhD thesis, Dublin City University, Ireland (2013)

[14] Stojanovic, L., Maedche, A., Motik, B., Stojanovic, N.: User-driven ontology evolution management. In: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web. EKAW 02, Springer-Verlag (2002) 285-300

[15] 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)

[16] Zablith, F.: Dynamic ontology evolution. International Semantic Web Conference (ISWC) Doctoral Consortium, Karlsruhe, Germany (2008)

[17] 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

[18] 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

[19] Klein, M.: Change Management for Distributed Ontologies. PhD thesis, Vrije University Amsterdam (2004)

[20] Groner, G., Staab, S.: Categorization and recognition of ontology refactoring pattern. Technical Report 09/2010, Institut WeST, Univ. Koblenz-Landau (2010)

[21] Plessers, P., De Troyer, O.: Ontology change detection using a version log. In: 4th International Semantic Web Conference, Springer (2005) 578-592

[22] Noy, N.F., Chugh, A., Liu, W., Musen, M.A.: A framework for ontology evolution in collaborative environments. In: In: 5th International Semantic Web Conference, Springer-LNCS (2006) 544-558