Towards Implicit Knowledge Discovery from Ontology Change Log Data

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. Ontology change log data is a valuable source of information which reflects the changes in the domain, the user requirements, flaws in the initial design or the need to incorporate additional information. Ontology change logs can provide operational as well as analytical support in the ontology evolution process. In this paper, we present a novel approach to deal with change representation and knowledge discovery from ontology change logs. We look into different knowledge gathering aspects to capture every single facet of ontology change. The ontology changes are formalised using a graph-based approach. The knowledge-based change log facilitates detection of similarities within different time series, discovering implicit dependencies between ontological entities and reuse of knowledge. We analyse an ontology change log graph in order to identify frequent changes that occur in ontologies over time. We identify different types of change sequences based on their order and completeness. Analysis of change logs also assists in extracting new change patterns and rules which cannot be found by simply querying or processing ontology change logs.

Keywords: Ontology Change Representation, Change Log, Pattern Discovery, Implicit Knowledge Discovery, Ontology Evolution.

1 Introduction

Ontology change log data is a valuable source of information, based on which domain ontologies can evolve in order to reflect the changes in the domain, the user requirements, flaws in the initial design or the need to incorporate additional information [6]. Change log data can also be used to capture sequential change patterns and implicit dependencies between ontological entities. Ontology change logs can play a significant role and can provide operational as well as analytical support in the ontology evolution process. 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 the changes ever applied to any entity of the ontology.

In this paper, we present an approach to deal with ontology change representation and discovery of implicit knowledge which cannot be captured by simple queries on ontology change logs. We look into different knowledge gathering aspects to represent a single ontology change and formalise it using a graph-based approach. We analyse the change log graph in order to identify frequent changes that occur in the ontologies over time. We identify co-occurrences and implicit dependencies between different ontological entities. Some central features of our approach are:

- A fine granular ontology change representation (in form of rdf triples) which helps in sharing the semantics of the data and representing the intent and the scope of the change explicitly.
- Discovery of implicit knowledge. It consists of identifying frequent change sequences, discovery of association rules, correlations between entities etc.
- Change pattern discovery by capturing frequent change subsequences (as change patterns) from ontology change log data. We utilize the discovered sequential change patterns to support pattern-based ontology evolution.

The paper is structured as follows: We discuss our study of change log data and the data preparation steps towards implicit knowledge discovery in Section 2. In Section 3, our observations and analysis on the preprocessed change log data is given. In Section 4, the discovery of implicit knowledge from change log data is discussed. We end with some related work and discussion.

2 Ontology Change Log Data

As a case study, the domain *University Administration* was used. The domain is selected as it represents an organisation involving people, organisational units and processes. The objective of university ontology was to assist in the proper execution of the day-to-day activities. We conceptualized most of the activities and the processes of the university. All the changes applied to the university ontology were captured in an ontology change log.

An ontology change log (CL) consists of an ordered list of ontology changes, $CL =< C_1, C_2, C_3 \cdots C_N >$ where N refers to the sequence of ontology changes in a change log. Each ontology change contains two types of data, i.e. *Metadata* (MD) and the *Change data* (CD) - Figure 1. 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. Metadata can be given as $MD = (id, u, t)$ where id, u and t represent ID, user and timestamp. The change data contain the central information about the change request and can be given as $CD = (Op, E, P)$ where, Op, E and P represent the change operation, element and parameter set of a particular change.

In order to conceptualise the changes, we construct 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 [8], types of ontology elements (such

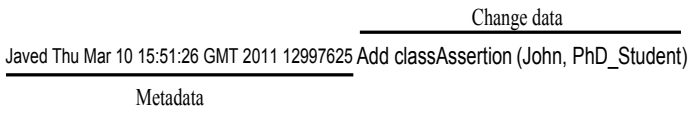

Fig. 1. Representation of a Single Ontology Change

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. We used an RDF triple-store to record the change log, domain ontologies and static metadata ontology. SPARQL queries are used to run against the change log to capture the required knowledge. Based on the different aspects, one can easily filter out the required knowledge, enhancing the query performance significantly.

We adopt as the first step the main data preparation steps from knowledge discovery for web log data [2, 3]. We assume for our discussion here that data is cleaned and filtered, i.e. unnecessary and incorrect entries are removed.

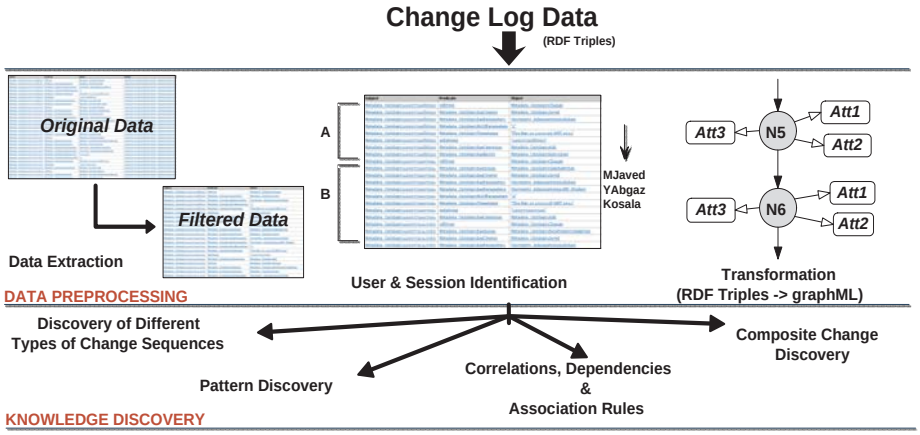

Fig. 2. Knowledge Discovery Process from Ontology Change Log Data

The second step is *session identification*. We perform identification in two steps. First, we identify the users. Secondly, we divide the sequential changes applied by a single user into sessions, based on the ID and change time request. Different threshold values can be found in literature from 5 min to 2 hr [13][14][15]. In our case, based on our empirical study and observations on the ontology changes, the threshold value was set to 15 min. Thus, all those change operations which are i) applied by the same user and ii) have less than 15 min change time request difference between two consecutive change operations constitute a single ontology change session.

The last step is *data transformation*. We formalise the change log data using a graph-based approach. We follow the idea of attributed graphs which are typed over an Attribute Type Graph (ATG) with node and edge attri-

bution. Based on the ATG idea, an ontology change log graph (G) can be given as $G = (N_G, N_A, E_G, E_{NA}, E_{EA})$ where $N_G = \{n(g_i)|i = 1, 2, \cdots p\}$ refers to the set of graph nodes. Each node represents a single ontology change log entry. $N_A = \{n(a_i)|i = 1, 2, \cdots q\}$ refers to the set of attribute nodes. $E_G = \{e(g_i)|i = 1, 2, \cdots p - 1\}$ refers to the set of graph edges which connect two graph nodes $n(g)$. $E_{NA} = \{e(na_i)|i = 1, 2, \cdots r\}$ is the set of node attribute edges which joins a graph node $n(g)$ to an attribute node $n(a)$ and $E_{EA} = \{e(ea_i)|i = 1, 2, \cdots s\}$ is the set of edge attributes which join a node attribute edge $e(na)$ to an attribute node $n(a)$. The benefit of a graph-based representation is the availability of well established algorithms and its well-known characteristics such as performance, which can be used for querying the ontology changes effectively. We used SPARQL queries in order to capture more than five hundred ontology changes from university change log (>5000 log triples) and converted them into a linear graph format using a graph API. As an example, a small portion of the linear graph (in a form of listing) is given in Table 1. Each line is a single node of the graph, representing a single change request. The id in each line represents the order of the graph nodes in the change log graph. Note, that the example here was chosen because it can fit on a small scale and the metadata attributes attached to each node have not been mentioned.

3 Analysis of Preprocessed Change Log Data

We analyze the change log graph empirically in order to understand how ontologies evolve over time, learn about the intent of changes and identify recurring change sequences. We discuss our findings here in the form of suitable metrics and patterns.

3.1 Metrics and Patterns

1) Sequential/Subsequence Patterns: In Table 1, we can identify a number of occurrences of the process of student enrollment. We found that users performed the same change in differnt order of change operations at different times. For example in Table 2, sequences s_1, s_2 and s_3 have the same order of change operations. First, the user adds the individual x to the ontology. In the next step, he adds individual x as an instance of concept $PhD\_Student$ and in the last step, he adds an individual x as a member of a university department y. The order of the graph nodes in any sequence is of vital importance as it represents the consistent or inconsistent state of the ontology at any particular instance of time.

2) Node–Distance between Change Operations: As different users may adopt different orders of change operations, there could be gap between two adjacent change operations of a sequence. We call it node-distance (or in short N-Distance), as it refers to the distance between two adjacent nodes of a sequence in a change log graph. For a sequence s in a change log graph G, the node gap between two adjacent nodes can be represented by a series of wild-cards

Table 1. A sequential list of change operations

Id	Change Operations (extracted from ontology change log graph)
1	Add concept ("PhD_Student")
2	Add subclassOf (PhD_Student, Student)
3	Add Instance ("Javed")
4	Add classAssertion (Javed, PhD_Student)
5	Delete subclassOf (PostGraduate, Student)
6	Delete concept (PostGraduate)
7	Add objectPropertyAssertion (Javed, isMemberOf, Computing)
8	Add dataPropertyAssertion (Javed, studentId, "58120348")
9	Add Instance ("Yalemisew")
10	Add classAssertion (Yalemisew, PhD_Student)
11	Add objectPropertyAssertion (Yalemisew, isMemberOf, Electrical_Engineering)
12	Add dataPropertyAssertion (Yalemisew, studentId, "58123857")
13	Add Instance ("Kosala")
14	Add classAssertion (Kosala, PhD_Student)
15	Add individual ("ECOWS2009")
16	Add objectPropertyAssertion (Kosala, isMemberOf, Electronic_Engineering)
17	Add Instance ("Aakash")
18	Add objectPropertyAssertion (Aakash, isMemberOf, Mechanical_Engineering)
19	Add classAssertion(Aakash, PhD_Student)
20	Delete inverseObjectProperty (hasSupervisor, isSupervisorOf)
21	Add dataPropertyAssertion (Aakash, studentId, "58121143")
22	Add domainOf (courseCode, Course)
23	Add rangeOf (hasCourseCode, CourseCode)
24	Add Instance ("Wong")
25	Add dataPropertyAssertion (Wong, studentId, "58129070")
26	Add classAssertion (Wong, PhD_Student)
27	Add Instance ("Pooyan")
28	Add classAssertion (Pooyan, PhD_Student)
29	Add objectPropertyAssertion (Pooyan, isMemberOf, Electronic_Engineering)
30	Add objectPropertyAssertion (Pooyan, isMemberOf, Electrical_Engineering)

(denoted by symbol x), where a wild-card x is a special symbol that matches any change operation in the change log. For example, sequence s_3 in Table 1 can be written as $s_3 = \{n_{13}, n_{14}, x, n_{16}\}$. The overall N-Distance of the sequence is denoted by upper case alpha A. It represents the number of wild-cards present in the whole sequence whereas, lower case alpha α refers to the gap between two distinct adjacent graph nodes of a sequence. A sequence s with N-Distance $A_s = \sum_b^a \alpha$ (where a and b refer to the graph node ids) can be denoted as $s = (n_1, \alpha_2^1, n_2, \alpha_3^2, \cdots, n_m)$. Thus, sequence s_3 in table 1 can also be written as $s_3 = \{n_{13}, +0, n_{14}, +1, n_{16}\}$.

3) Type Categorisation of Change Operations: We found a difference between fully distinct and type-equivalent change operations. Two ontology change operations can be type-equivalent based on the type of their operation, element and parameters. For example, in the list below, operations 1 and 2 are type-equivalent as both of them have the same operation type (Add), the same element type (ObjectPropertyAssertion) and the same type of instances (i.e. Javed and Kosala are instances of type PhD_Student and Computing & Elec-

Table 2. Node Sequences of PhD Student Enrollment Process

Sequence	Node Set of Ontology Change Log Graph
s_1	3 - 4 - 7 - 8
s_2	9 - 10 - 11 - 12
s_3	13 - 14 - 16
s_4	17 - 18 - 19 - 21
s_5	24 - 25 - 26
s_6	27 - 28 - 29 - 30

tronic_Engineering are types of university department). Operation 3 is distinct in comparison with operations 1 and 2 as parameter IEEE is not of type university department (instead of type Research Society).

1- Add ObjectPropertyAssertion(Javed, isMemberOf, Computing);
2- Add ObjectPropertyAssertion(Kosala, isMemberOf, Electronic_Engineering);
3- Add ObjectPropertyAssertion(Abgaz, isMemberOf, IEEE);

4) Variations between Change Sequences: Two change sequences can be different from each other in a number of ways such as in terms of their length, order and type of operations involved etc. We can identify three types of variation which can occur between two sequences (Table 3), i.e. Len-variation, ST-variation and DT-variation.

- *Len-Variation:* Len-variation captures the variation between two sequences based on the number of graph nodes present in them. For example in Table 1, Len-variation between sequences s_1 and s_6 is 0 as the number of graph nodes available in both sequences is 4.
- *ST-Variation:* ST-variation is a measure to capture the variation between two sequences based on the number of distinct operations, but in different quantity. For example, in Table 1, ST-variation between sequences s_6 and s_1 is 1 as the former sequence contains one extra change operation (i.e. operation 30), which is not present in the other; however, a similar type of change operation is there (operation 07).
- *DT-Variation:* DT-variation captures the variation between two sequences based on the number of distinct operations which are present in the first sequence but missing in the other, e.g. DT-variation between sequences s_1 and s_6 is 1 as the former sequence contains one extra change operation (operation 08), which is of a type not present in the other.

Table 3. Variations b/w sequences s_1 and s_6

-	Len-Variation	ST-Variation	DT-Variation
s_1 vs. s_6	00	00	01
s_6 vs. s_1	00	01	00

4 Discovery of Sequential Abstractions from Ontology Change Log

Mining of sequential abstractions is the key focus here. Based on our analysis of evolutionary aspects of domain ontologies and the identified sequen-

tial/subsequence change patterns from ontology change log graph (Sect. 3), we categorized different type of change sequences into two basic subdivisions and used them as the basis for change pattern discovery algorithms. We exploited the identified change sequences to discover further implicit knowledge which includes composite changes and causal dependencies across the ontology taxonomy hierarchy. Below, we discuss each section in detail.

4.1 Discovery of Different Types of Change Sequences

To perform some change operations in the exact same order over time is unlikely. In a real world scenario, users perform changes by using different orders of change operations. However, the end result of the change sequences may be the same. Based on our sequential/subsequence pattern observations (discussed above), we identified four different types of the sequences (in comparison to referenced candidate sequence) based on their ordering and completeness (i.e. *Len-Variation*). We merged these different types of sequences into two basic divisions:

- ***Ordered Change Sequences (OS)***
  - Type 1 - Ordered Complete Change Sequence (OCS)
  - Type 2 - Ordered Partial Change Sequence (OPS)
- ***Unordered Change Sequences (US)***
  - Type 3 - Unordered Complete Change Sequence (UCS)
  - Type 4 - Unordered Partial Change Sequence (UPS)

Ordered Change Sequences comprise ordered change operations from a change log. That means, such sequences (complete or partial) may have only positive node distance between two adjacent graph nodes (w.r.t. referenced change sequences). In Table 1, w.r.t. sequence s_1, the sequence s_2 is of type 1 as the sequence s_2 is *complete* (i.e. $Len - variation = 0$) and change operations are in the *same order* (i.e. $\alpha = +ve$). However, the sequence s_3 is of type 2 as the change operations in it are in the *same order*, but the sequence is *partial* (i.e. $Len - variation > 0$). Similarly, w.r.t. sequence s_1, sequence s_6 is also of type 2 due to the presence of $DT - variation$ (i.e. operation 8 is not *type-equivalent* to operation 30).

Unordered Change Sequences comprise unordered change operation from a change log. That means, such sequences (complete or partial) may have positive or negative node distances between two adjacent graph nodes (w.r.t. referenced change sequences). In Table 1, sequence s_4 is of type 3 (w.r.t. sequence s_1) as the sequence is *complete*. However, the change operations are *unordered* (i.e. $\alpha = +ve/-ve$). Furthermore, the sequence s_5 is of type 4 as the change operations are in *different order* as well as the sequence is *partial*.

The discovery of such change sequences has a number of benefits. First, it helps for documenting evolving ontologies, i.e. representing how entities evolve over time (entity evolution). Second, these change sequences are used to discover the correlations & the causal dependencies between different ontological entities which evolve together (Sect. 4.4). Third, and most importantly, the identified

change sequences are used in discovering usage-driven change patterns (Sect. 4.2).

4.2 Discovery of Sequential Change Pattern

Ontology change logs can be used to discover the usage-driven change patterns. Such sequential change patterns provide guidelines to content change management systems and support in the evolution process [11]. Identifying recurring sequenced change patterns from a change log is a problem of recognising frequent pattern in a graph. A set of candidate pattern sequences *CP*, to be considered as a domain-specific change pattern *P*, must meet the following criteria:

- The length of the each candidate pattern sequence *cp* must be equal to or greater than the threshold value set by the minimum pattern length.

$$len(cp) \geq min\_len(p) : cp \in CP \tag{1}$$

- Each candidate pattern sequence *cp* must reflect a consistent state of the ontology. There may be graph nodes available in *cp* which transforms a consistent ontology to a non consistent (or vice versa). However, at the end of the sequence, the ontology must be back in a consistent state.

$$endOf(cp) \rightarrow consistent(O) : cp \in CP \tag{2}$$

- The support for a candidate pattern *cp* in a change log graph *G* must be above the threshold value of minimum pattern support of a pattern.

$$sup(cp) \geq min\_sup(p) : cp \rightarrow P_{domain\_specific} \tag{3}$$

The basic idea of the change pattern discovery algorithms is to i) start an iteration process on each graph node, ii) generate the candidate sequence starting from that particular graph node and iii) search the similar sequences within the graph G. Details of the change pattern discovery algorithms can be found in [12]. The change sequence types 1 & 3 (Sect. 4.1), where the change sequences are of the same length, i.e. $Len - variation = 0$, were applied as a reference in order to discover the change patterns. The performance-based comparison of two algorithms, Ordered Complete Change Pattern (OCP) and Unordered Complete Change Pattern (UCP), is given in Table 4. The OCP algorithm is efficient in terms of time consumption. The reason is the permissibility of only positive node distance ($+\alpha$) in a change sequence. However, the UCP algorithm is more efficient in terms of number of discovered patterns (9:5). Similarly, in terms of size of maximal patterns, the UCP algorithm could discover patterns of greater size as compared to OCP.

The discovered change patterns are based on the operations that have been utilized frequently by the user and guarantee the consistent state of the ontologies. Once the sequential patterns are associated with the user category, patterns will be more effective since the classified patterns are often more useful [7].

Table 4. Comparison between OCP and UCP Algorithm

	a - **OCP Algorithm**			b - **UCP Algorithm**		
Max. Node Distance	Patterns Found	Time (ms)	Max. Pattern size	Patterns Found	Time (ms)	Max. Pattern size
0	0	469	0	4	1359	6
1	3	609	5	7	2282	7
2	5	875	6	6	3906	7
3	5	985	6	8	4968	7
4	5	1110	6	8	6078	7
5	5	1203	6	9	7141	7

4.3 Discovery of Composite Changes

We made use of the correlations between different identified change sequences to capture the composite changes from a change log. We use an example to illustrate this. In Table 4, three separate identified change sequences are given. The first two change sequences represent an addition of new concepts *PhD_Student* and *MSCByResearch_Student* and provide a description of the concepts by adding them as domain/range for object properties and linking individuals to them. The last change sequence represents the deletion of a concept *Research_Student* and deleting all descriptions about such concept. If we look the three changes as a sequence of change operations, they look like an addition and deletion of entities. However, there are implicit correlations, which exist among these sequences. First, if we look at the node ids, it is clear that these three change operations are applied one after another. Secondly, these three change sequences have the parent of the target entity concept (*Student*) in common. Third, most of the parameters of the change operations have been shared among each sequence. If we look these three change sequence together, one can understand that these are not three separate change sequences, but are linked to each other and there exists a correlation between them. The intent of change is not to add two new concept and delete one, but to *split* a concept into two and the three change sequences must be applied as a single transaction. We identify such composite changes with minimal user input. Once such composite change operations are identified, they are represented as a single change request in a higher level (composite) change log [9]. Later, such composite operations can be applied as a single transaction whenever similar changes have to be performed. Such discovery of composite changes supports the represention of the intent of change at a higher level and improves the ontology evolution process in terms of consistency and time consumption.

4.4 Discovery of Causal Dependencies

A causal dependency is related to the identification of ontological entities which frequently (if not always) evolve together even if they are not directly connected. That means, change in one part of the domain ontology has a direct impact on another section of the ontology. We are interested in detecting such causal dependencies across the ontology taxonomy. For example, in the change sequences of

Table 5. Three separate ontology change sequences

Node Id	Change Operations
114	Add concept (PhD_Student)
115	Add subClassOf (PhD_Student, Student)
116	Add domainOf (hasSupervisor, PhD_Student)
117	Add rangeOf(isSupervisorOf, PhD_Student)
118	Add classAssertion (Javed, PhD_Student)
119	Add concept (MSCByResearch_Student)
120	Add subClassOf (MSCByResearch_Student, Student)
121	Add domainOf (hasSupervisor, MSCByResearch_Student)
122	Add rangeOf(isSupervisorOf, MSCByResearch_Student)
123	Add classAssertion (Zubair, MSCByResearch_Student)
124	Delete domainOf (hasSupervisor, Research_Student)
125	Delete rangeOf(isSupervisorOf, Research_Student)
126	Delete classAssertion (Javed, Research_Student)
127	Delete classAssertion (Zubair, Research_Student)
128	Delete subclassOf (Research_Student, Student)
129	Delete concept (Research_Student)

the university ontology (given in Table 1), whenever a user adds a new instance of *PhD Student*, he/she also attaches a *student id* with the instance. Furthermore, user adds additional information such as *university department*, *email id* etc. of that particular student. These types of changes demonstrate that the existence of the properties such as *student id* or *isMemberOf* are causally dependent on the concept *Phd Student*. Thus, the existence of any PhD student without such linked knowledge does not have any semantic use. Another example of captured causal dependency is the *introduction of new course*. Whenever a new course is introduced in a university department, new subjects (and subject codes) have to be introduced, course-related books have to be purchased and added to the library, new vacancies have been advertised in order to provide the expertise etc.

We analyzed different types of change sequences (Sect. 4.1) and the identified composite changes (Sect. 4.3) in order to capture the ontological entities which evolve together. Based on our analysis, we generate the association rules which are implemented as sequences and contains higher change support and confidence within the change log. These association rules were used to capture causally dependent ontological entities. We employed such causal dependencies in existing ontology change management systems in order to discover new trends within the domain, change request recommendations etc. This, in turn, can also serve as basis for process improvement actions, e.g., it may trigger patterns redesign or better control mechanisms [4].

5 Related Work

An early work that relied on the activity logs for producing formal process models corresponding to actual process execution is given in [16]. The author

describes different methods based on an analysis of the activity traces. Metrics such as event frequency and regularity were taken into consideration to discover process models. Results show that the proposed technique is promising and useful in activities such as process model discovery, reengineering, software process improvement etc. In [17], the focus is on detection of invisible tasks from event logs. Their definition of invisible tasks is *tasks that exist in a process model, but not in its event log (such as initialise, skip, switch, redo, etc.)*. Author proposed an algorithm which extends the mining capability by detecting of invisible tasks. In contrast to their work, we are interested in the detection of composite changes such as *split*, *move*, *merge* etc. In terms of mining of sequential patterns, it was first proposed by Agrawal and Srikant [18] and later, these authors proposed an algorithm (GSP Algorithm) based on an apriori property [19]. Since, many authors have proposed a number of sequential pattern mining algorithms based on their specific domains and suitability [2, 4, 5, 7, 10]. We have neglected early process stages such as log parsing here [2, 3] in order to focus on the semantic analysis, but an analysis of the log processing could bring further insights.

6 Conclusions

Activity log mining is not restricted to creating new formal process models but can be extended to discover implicit knowledge. Such knowledge may give an ontology engineer clues as to how and in what direction the defined ontology should evolve, based on the actual current data of change activities. The presented work continues our previous research [8][9] by adding a comprehensive ontology change representation and approach towards discovery of implicit knowledge from the ontology change logs. In this paper, we proposed a fine-granular ontology change representation by using different knowledge gathering aspects. We studied the ontology change log data empirically in order to capture the implicit knowledge. We utilised ontology change logs to identify different types of change sequences, entity correlations, association rules and discovery of sequential change patterns. Such discovered knowledge is used for the purpose of patterns redesign, documentation of changes at higher level, classification of ontology users etc. Our future work includes the definition of a pattern language based on the implicit knowledge discovered from a change log.

Acknowledgment

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. Yu, L.: Mining Change Logs and Release Notes to Understand Software Maintenance and Evolution.In CLEI Electron Journal, Vol. 12, No. 2, pp. 1–10, 2009.

2. Ivancsy, R., Vajk, I.: Frequent Pattern Mining in Web Log Data. Acta Polytechnica Hungarica. Journal of Applied Sciences, Vol. 3, No. 1, pp. 77-90, 2006.
3. Pabarskaite, Z., Raudys, A.: A process of knowledge discovery from web log data: Systematization and critical review. In Journal of Intelligent Information Systems, Vol. 28(1), pp. 79–104, 2007.
4. Guenther, C., Rinderle, S., Reichert, M., van der Aalst, W.: Change Mining in Adaptive Process Management Systems. In: Proc. 14th International Conference on Cooperative Information Systems(CoopIs), LNCS 4275, pp. 309-326, 2006.
5. Peng W., Li, T., Ma, S.: Mining logs files for data-driven system management. In Journal of SIGKDD Explorations, volume 7, No. 1, pp. 44-51, 2005.
6. Haase, P., Sure, Y.: Usage Tracking for Ontology Evolution. EU IST Project SEKT Deliverable D3.2.1 (WP3.2) 2003.
7. Pinto, H., Han, J., Pei, J., Wang, K., Chen, Q., Dayal, U.: Multi-Dimensional Sequential Pattern Mining. In ACM International Conferenece on Information and Knowledge Management (CIKM '01), pp. 81-88, 2001.
8. 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.
9. Javed, M., Abgaz, Y., Pahl, C.: A Layered Framework for Pattern-Based Ontology Evolution. In 3rd International Workshop on Ontology-Driven Information System Engineering (ODISE), London, UK, 2011.
10. Kosala, R., Blockeel, H.: Web mining research: A survey: Newsletter of the Special Interest Group on Knowledge Discovery and Data Mining, ACM, Vol. 2, No. 1, pp. 1–15, 2000.
11. Gruhn, V., Pahl, C., Wever, M.: Data Model Evolution as Basis of Business Process Management. In 14th International Conference on Object-Oriented and Entity Relationship Modelling O-O ER95. Springer-Verlag (LNCS Series), 1995.
12. Gacitua-Decar, V., Pahl, C.: Automatic Business Process Pattern Matching for Enterprise Services Design. 4th International Workshop on Service- and Process-Oriented Software Engineering (SOPOSE-09). IEEE Press. 2009.
13. He, D., Goker, A.: Detecting session boundaries from Web user logs. In Proceedings of the 22nd Annual Colloquium on Information Retrieval Research (pp. 57-66). Cambridge, UK: British Computer Society, 2000.
14. Pitkow, J., Margaret, R.: Integrating bottom-up and top-down analysis for intelligent hypertext. In Conference on Intelligent Knowledge Management, Intelligent Hypertext Workshop, Dec. 12, 1994, National Institute of Standard Technology.
15. Montgomery, A.L., Faloutsos, C.: Identifying web browsing trends and patterns. In Proceeding of IEEE Journal Computer, Vol. 34, issue 7, pp. 94-95, 2001.
16. Cook, J.E., Wolf, A.L.: Discovering models of software prosses from event-based data. ACM Transactions on Software Engineering and Methodology. Vol. 5, no. 3, pp. 215–249, 1998.
17. Wen, L., Wang, J., van der Aalst, W.M.P., Huang, B., Sun, J.: Mining process models with prime invisible tasks. In journal of Data Knowledge Engineering, Vol.69, no. 10, pp. 999–1021, 2010.
18. Agrawal, R., Srikant, R.: Mining sequential patterns. In Proceedings of the Eleventh Int. Conf. on Data Engg., Philip S. Yu and Arbee L. P. Chen (Eds.). IEEE Computer Society, Washington, DC, USA, pp. 3–14, 1995.
19. Srikant, R., Agrawal, R.: Mining sequential patterns: generalizations and performance improvements. In: Proc. of the Int. Conf. on Extending Data Base Technology. Lecture Notes in Computer Science, vol. 1057. Springer Verlag, pp. 3–17, 1996.