The Knowledge Engineering Review, Vol. 33, e12, 1–22.
doi:10.1017/S0269888918000140

EvoRDF: evolving the exploration of ontology evolution

HARIDIMOS KONDYLAKIS[1] and NIKOS PAPADAKIS[2]

[1]*Computational BioMedicine Laboratory, FORTH-ICS, N. Plastira 100, Heraklion, Crete, Greece;*
e-mail: kondylak@ics.forth.gr;
[2]*Department of Informatics Engineering, Technological Educational Institute of Crete, Estavromenos 71004, Heraklion, Crete, Greece;*
e-mail: npapadak@cs.teicrete.gr

Abstract

Ontologies are constantly evolving as new requirements daily occur and the modeling choices of the past should be updated or adapted. Exploring this evolution will enhance the understanding, augmenting the exploitation potential of the available ontologies. However, recent research focuses mostly on detecting changes between ontology versions, overloading end-users with hundreds or even thousands of changes between ontology versions, making it impossible to explore this evolution. To this direction, in this paper, we present EvoRDF, a novel framework for exploring ontology evolution using provenance queries. Our approach uses a high-level language of changes and effectively answers queries about *when* a specific resource was introduced and *how*—by which change operations. Even more, *why queries* can identify the sequence of changes that led to the creation of a specific resource in the latest ontology version or track the evolution of a specific resource from a past ontology version. The evaluation performed shows the feasibility of our solution and the great advantages gained.

1 Introduction

Ontologies are formal, explicit specifications of a shared conceptualization of a domain of interest (Avgoustaki *et al*., 2016). Ontologies are not static but they are living artifacts and subject to change (Volkel *et al*., 2005; Flouris *et al*., 2008). Due to the rapid development of research, ontologies are frequently changed to depict the new knowledge that is acquired. However, since ontologies are usually managed independently from each other they can be used and extended without the explicit permission of the owner. In several cases, the owner of an ontology is completely unaware of who uses or extends his ontology.

It is therefore vital to be able to support the ontology engineers and the maintainers of the dependent artifacts in this complex process of ontology evolution (Plessers *et al*., 2007). Several approaches so far deal with closely related problems such as consistency maintenance, backward compatibility, ontology manipulation, change propagation, etc. (Volkel *et al*., 2005). In the field of *a posteriori* understanding ontology evolution, most of the approaches use different representation languages to model evolution (Papavasileiou *et al*., 2013), presenting to the users eventually all changes between the various ontology versions. However, although the developed languages of changes have become more concise and compact—by employing high-level change operators (operators that can describe complex updates, e.g. the insertion of an entire subsumption hierarchy) —still ontology understanding relies on just presenting to the users a huge list of changes between ontology versions.

In this paper, we argue that only listing the changes between two versions is insufficient for understanding ontology evolution. Moreover, we provide a solution to this problem by answering *provenance queries* concerning both the data and the schema information of an ontology. More specifically, our contribution is the following:

- We introduce EvoRDF, a framework for exploring ontology evolution. The modules developed get as input the sequence of changes between two or more ontology versions and are able to answer provenance queries requesting fine-grained information about the evolution of the various resources.
- The changes are modeled using an ontology of changes and stored to a triple store using a parser/editor module.
- Then a Protégé Module, the EvoRDF Protégé Explorer, and a Web module, the EvoRDF Web Explorer, allow users to issue fine-grained provenance queries and graphically visualize the results.
- To achieve that, we define the notions of *how* and *when provenance* and we present the corresponding algorithms. Using our framework a user can identify with which change operation a resource was introduced (*how*) and in which ontology version (*when*).
- Moreover, the list of change operations that led to the creation of that specific resource can be computed and presented to the user (*why*) allowing further exploration. This knowledge can be used to drive developer's understanding on ontology evolution for that specific resource.
- Finally, we present our experimental analysis using two well-known ontologies CIDOC-CRM and Gene Ontology (GO). Experiments performed show the feasibility of our approach. In addition, a preliminary usability evaluation shows the potential of our solution, enabling users to understand the evolution of specific resources.

A preliminary version of this work was first presented in a workshop (Kondylakis & Plexousakis, 2014) whereas the implemented platform was demonstrated in the Extended Semantic Web Conference (Kondylakis *et al*., 2017). This paper extends our previous work in many ways, presenting the implementation of a whole framework on top of the developed algorithms and extending our approach to handle multiple language of changes, as we shall see in the sequel. Inversibility is an interesting property of the language of changes used, that we further discuss in this paper. In addition, a ontology of changes is introduced to model the change operations, whereas an additional usability evaluation is presented. By changing the change ontology, our system is ready to use a different language of changes. The simplicity of our approach makes it a valuable tool for ontology engineers and provides a unique vantage point on understanding long and complex evolution histories.

The rest of the paper is organized as follows: Section 2 presents related work and Section 3 introduces the preliminaries and the problem by an example. Then, Section 4 presents in detail the algorithms used. Section 5 demonstrates the high-level architecture of our framework and presents the various modules implemented. Section 6 presents experiments concerning the scalability of our approach and a usability evaluation. Finally, Section 7 concludes this paper and presents directions for future work.

2 Related work

Since our approach is unique in terms of the methods employed for exploring ontology evolution, the aim of this section is to provide an overview on the approaches that focus on closely related areas. We have to note, that the purpose of this section is not to provide an overview on all approaches and methods concerning ontology evolution. The interested reader on this topic is forwarder to our two detailed surveys in the area (Volkel *et al*., 2005; Flouris *et al*., 2008).

2.1 Provenance information

Management of provenance information has been extensively studied in the literature, using different methods and approaches. Different authors define different provenance management techniques (e.g. *Why*-provenance (Buneman *et al*., 2001), *Trio*-Provenance (Benjelloun *et al*., 2008), provenance *semirings* (Green *et al*., 2007)) that either try to provide annotations at the tuple level or to extract provenance

information by analyzing queries. Other works such as (Chiticariu & Tan, 2006) try to describe the relationship between source and target data in a data integration scenario. However, our approach differs in both methods and goals. To the best of our knowledge, there is no other approach answering provenance queries regarding ontology evolution.

2.2 Existing ontologies describing provenance

Several vocabularies exist already allowing the documentation of related provenance information. Among the most well-known ones, are the Open Provenance Model and the PROV ontology. The Open Provenance Model (Moreau *et al*., 2011) is designed to allow provenance information to be exchanged between systems in a precise, technology-agnostic manner, supporting digital representation of provenance for any 'thing', whether produced by computer systems or not. The approach has been evolved to create the PROV ontology (Lebo *et al*., 2013) which is now a standard proposed by W3C for the representation and exchange of domain independent provenance. Both the aforementioned approaches, model provenance data as relations between agents, entities and activities, and cannot really drive ontology engineers understanding on the core evolution of the ontologies. As such, they could be used as complementary to our approach in order to document in more detail, actors, editors and activities related to the process of the evolution.

2.3 Versioning systems

Tools like the Git2PROV (De Nies *et al*., 2013) enable provenance tracking using the PROV ontology for any public repository. However, besides documenting additional metadata described in the PROV ontology, only versioning information related to git is modeled and described. As such, ontology engineers can only identify changes in the RDF document, but they are not able to graphically visualize or query specific information.

Identifying this drawback, Arndt *et al*. (2017), R43ples (Graube *et al*., 2014) and Stardog (Sauro, 2011) focus on versioning on the graph-level. Using those tools versioning happens for each named graph. Revisions are modeled as deltas to the previous versions, used in conjunction with PROV metadata to store activities related to the whole process. Although, these approaches have many similarities to EvoRDF (an ontology is employed to describe the metadata and all changes/deltas are stored in a triple store) still the ontology engineers cannot visualize a sequence of changes for a specific resource since the deltas are modeled at the graph level. As such, coarse-grained information (e.g. a rename) cannot be identified effectively and it is difficult for ontology engineers to trace what happened in the past for a given resource.

2.4 Change detection

Other works that could be employed to understand ontology evolution focus on change detection. Those systems can be classified under two basic dimensions, namely the level of changes they support (low-level or high-level) and the underlying representation language assumed (e.g. Plessers *et al*., 2007, for Description Logic, Papavasileiou *et al*., 2013, for resource description framework/schema (RDF/S), etc.). In its simplest form, a language of changes consists of only two *low-level* operations, *Add(x)* and *Delete(x)*, which determine individual constructs (e.g. triples) that were added or deleted (Zablith *et al*., 2015; Troullinou *et al*., 2016). However, a significant number of more recent works (Plessers & Troyer, 2005; Rogozan & Paquette, 2005; Noy *et al*., 2006; Zablith *et al*., 2015) imply that *high-level* change operations should be employed instead, which describe more complex updates, as for instance the insertion of an entire subsumption hierarchy. A high-level language is preferable than a low-level one (Stefanidis *et al*., 2016), as it is more *intuitive*, *concise*, *closer to the intentions* of the ontology editors and *captures more accurately* the semantics of change. However, there is no agreed-upon list of changes, applicable in any given context. In our case, we do not redefine such a language but we only use one of them. Moreover, our results are not limited to this specific language, as we shall see later in this paper.

2.5 Summaries of ontology evolution

Latest works in the area of exploring ontology evolution, focus on providing an overview of the changes, or the areas of the ontology that were mostly affected by the changes process (Troullinou *et al*., 2017). To this direction, they define multiple assessment dimensions and measures trying to identify the 'intensity' of the evolution as the authors say. However, they focus only on classes and present static overviews and of the evolution without allowing users to explore a resource of interest.

Other approaches like Stefanidis *et al*. (2016) focus on providing statistical summaries about the evolution of the ontology. The specific approach allows focusing on specific categories of changes, getting statistics for the different types of changes in the ontology. However, the system implemented only presents a high-level overview of the implemented changes.

2.6 Tracking evolution

A similar approach to EvoRDF is (Plessers *et al*., 2007) where a change is defined and detected using temporal queries over a version log that contains recordings of the applied changes. However, the version log must be updated whenever a change occurs. This overrules the use of this approach in non-curated or distributed environments. In our approach, on the other hand, the changes can be produced a posteriori and no temporal queries are used.

In Curino *et al*. (2013) the authors provide a mechanism to document schema evolution of relational databases, by automatically presenting the changes (called translational medicine ontology) between those versions. However, those changes are not detected automatically. Moreover, they offer a schema evolution analysis tool, but this tool only provides coarse-grained results.

Finally, in Ruiz *et al*. (2011) the authors present a tool to allow several developers to make changes concurrently and remotely to the same ontology, track changes and manage ontology versions. However, this tool focuses on conflict resolution and cannot provide answers to fine-grained provenance queries.

3 Preliminaries and motivating example

RDF is a language for describing Web resources [RDF Primer, 2004]. Information in RDF is represented using triples of the form (subject, predicate, object) which record that subject is related to object via predicate. RDF data sets have attached semantics through RDFS schemas. RDFS is a vocabulary description language that includes a set of inference rules use to generate new, implicit triples from explicit ones. In this work we focus on RDF as a large portion of the Semantic Web Schemas (85,45% in 2007 according to Theoharis et al. (2007)) are expressed in RDF/S, and RDF/S offers, in our case, an optimum trade-off between expressive power and efficient reasoning support.

Now as an example, consider an ontology shown on top of Figure 1 (ontology version O_0). This ontology is used as a point of common reference, describing people and their contact points. Assume now that at some point in time, the ontology evolves and we get O_1 by adding the class '*Cont.Point*' describing contact points and the property '*has_cont_point*' between the class '*Actor*' and the class '*Cont.Point*'. Moreover, the domain of the literals '*street*' and '*city*' is changed to the class '*Cont.Point*'. Then the ontology designer decides to evolve again the ontology and to produce O_2. Therefore, the domain of the '*has_cont_point*' property is moved from the class '*Actor*' to the class '*Person*', and the property '*gender*' is deleted. Moreover, the '*street*' and the '*city*' properties are merged to the '*address*' property as shown at the bottom of Figure 1. For modeling this evolution, we will use *a language of changes*. Independent of the specific language of changes user, a change operation is defined as follows:

Definition 1 *(Change Operation): A change operation u over an RDF ontology O, is any tuple* (δ_a, δ_d) *where* $\delta_a \bigcap O = \phi$ *and* $\delta_d \subseteq O$. *A change operation u from* O_1 *to* O_2 *is a change operation over* O_1 *such that* $\delta_a \subseteq O_2 \backslash O_1$ *and* $\delta_d \subseteq O_1 \backslash O_2$.

Obviously, δ_a and δ_d are sets of triples. For simplicity we will denote $\delta_a(u)$ the *added* and $\delta_d(u)$ the *deleted* triples of a change u. From the definition, it follows that $\delta_a(u) \cap \delta_d(u) = \phi$ and $\delta_a(u) \cup \delta_d(u) \neq \phi$ if $O_1 \neq O_2$. The application of a change u over an ontology version O, denoted by $u(O)$, is defined as

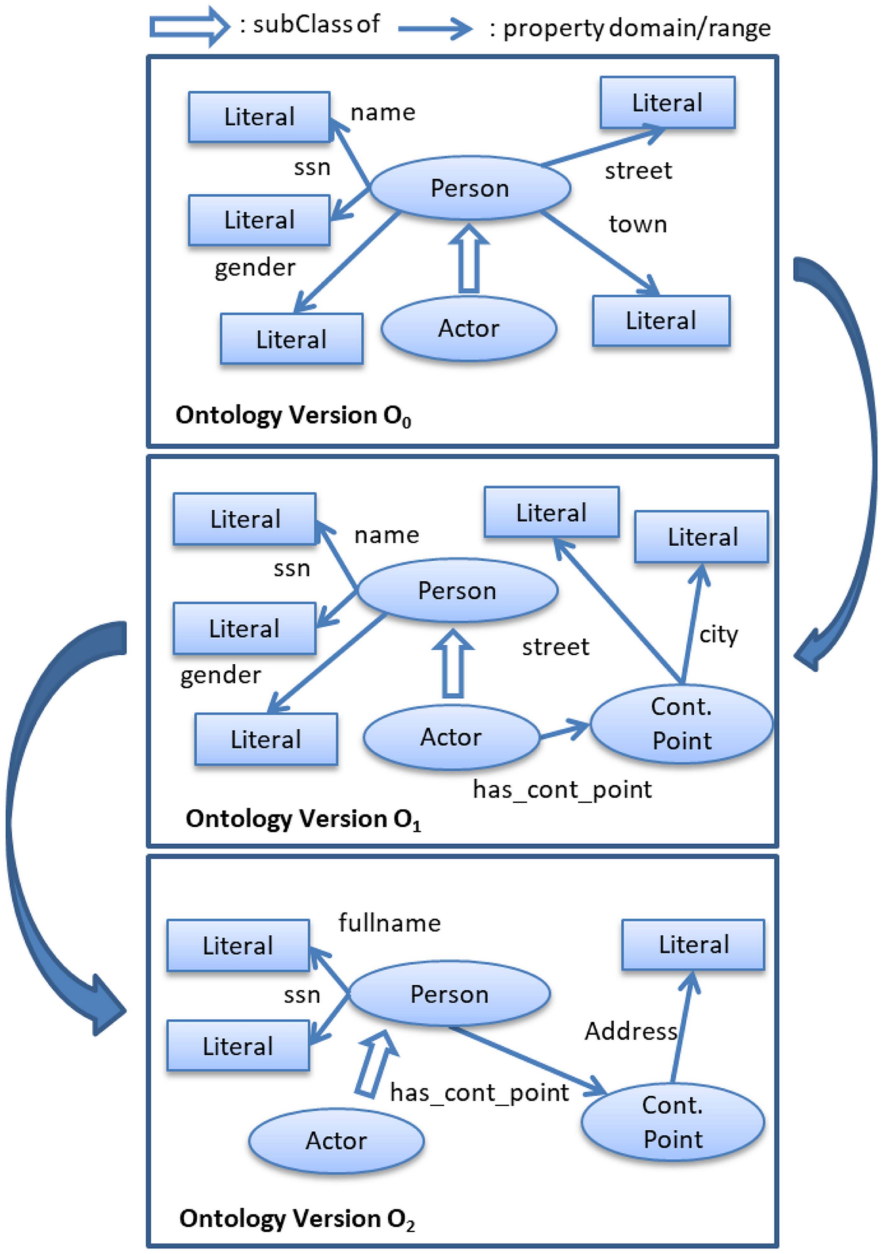


Figure 1 Example ontology evolution from O_0 version to O_1 and then to O_2

$u(O) = (O \cup \delta_a(u)) \backslash \delta_d(u)$). Moreover, the application of a sequence of change operations us to an ontology, that is, $us(O)$, is defined as the sequential application of the change operation in us to O. Our system supports multiple language of changes. However, for any two changes u_1, u_2 in such a sequence we require that $\delta_a(u_1) \cap \delta_a(u_2) = \phi$ and $\delta_d(u_1) \cap \delta_d(u_2) = \phi$. The direct implication of this is that the sequence of changes between two ontology versions should be unique.

An example language of changes that we adopt in the rest of this paper for reasons of simplicity is the one proposed in Papavasileiou *et al.* (2013) and Papavassiliou (2010). The language classifies the high-level changes it uses into basic or composite. Basic changes are fine-grained and describe changes in one node or edge of the RDF/S knowledge base taking into account RDF/S semantics. Composite changes on the other hand, are coarse-grained and closer to the user's intuition as they describe changes affecting several nodes and/or edges. In addition, the language satisfies the properties of *completeness* (capturing any possible change), *non-ambiguity* (each low-level change is associated with one, and only one, high-level change) and supports a *deterministic detection process*. In this specific language of change, the

inverse of each change operation exists as well, whereas non-conflicting changes (changes that the detection of one requires the addition/deletion of a triple whose deletion/addition is required by the other) can be *composed* as well.

Example types of changes in that language are show in Table 1. Using the specific language of changes for our example shown in Figure 1, the change log between O_0 and O_1, that is, the E^{O_0,O_1}, consists of the following change operations:

u1: *Add_Class(Cont.Point, ϕ, ϕ, ϕ, ϕ, ϕ,)*

u2: *Add_Property(has_cont_point, ϕ, ϕ ,ϕ ,ϕ, Actor, Cont.Point, ϕ, ϕ)*

u3: *Change_Domain(town, Person, Cont.Point))*

u4: *Change_Domain(street, Person, Cont.Point))*

u5: *Rename_Property(town, city)*

In the sequel, the change log E^{O_1,O_2} consists of the following change operations:

u6: *Delete_Property(gender, ϕ, ϕ ,ϕ ,ϕ, Person, xsd:String, ϕ, ϕ)*

u7: *Generalize_Domain(has_cont_point, Actor, Person)*

u8: *Merge_Properties({street, city},address)*

u9: *Rename_Property(name, fullname)*

Obviously, $E^{O_0,O_2} = E^{O_0,O_1} \cup E^{O_1,O_2}$. In this paper, we argue that only presenting a similar sequence of changes is not enough for understanding how ontology evolved. Especially in real world scenarios, the large number of change operations makes it impossible for ontology developers to understand ontology evolution based solely on those. In our experiments, for example, just for a few versions, we had 4175 changes for GO and 726 changes for CIDOC-CRM.

4 Provenance queries

As already mentioned, presenting only the list of changes between ontology versions is not adequate for understanding ontology evolution. To this direction, we argue that a system supporting ontology evolution should be able to support the following types of queries:

- When queries: An end-user would like to search for the specific version that a resource was introduced (e.g. When was the '*address*' literal added to the ontology?).
- How queries: In addition, an end-user would like to search for the change operation that introduced a specific resource (how) (e.g. By which change operation was the '*address*' literal added?).
- Why queries: Finally, the change path (i.e. the consecutive list of change operations) that led to the creation of a specific resource can be computed and presented to the user allowing further exploration and understanding of the evolution of the ontology (e.g. What are the change operations that had some influence on the creation of the '*address*' literal? How was the '*street*' literal evolved starting from ontology version O_0?). *Why* queries are possible not only for specific resources but for specific change operations as well.

4.1 How queries

As already mentioned, presenting only the list of changes between ontology versions is not adequate for understanding ontology evolution. To answer queries about the evolution of a specific resource we define the notion of an *affecting change operation*.

Table 1 Example change operations

Change	Generalize_Domain (*a,b,c*)	Rename_Property (*a,b*)	Merge_Properties (*A,b*)
Intuition	Change the domain of property *a* from *b* to superclass *c*	Rename property *a* to *b*	Merge properties contained in *A* into *b*
δ_a	*[(a, domain, c)]*	*[(b, type, property)]*	*(b, type, property)*
δ_d	*[(a, domain, b)]*	*[(a, type, property)]*	$\forall\ a \in A$ *: (a, type, property)*
Inverse	*Specialize_Domain(a,c,b)*	*Rename_Property(b,a)*	*Split_Property(b, A)*
Change	*Add_Class(a,P1,P2,P3,P4,P5,P6)*	*Add_Property(a,P1,P2,P3,P4,p5,p6,P7,P8)*	*Change_Domain(a,b,c)*
Intuition	Add class a with its neighborhood links *P1* = set of new parent classes of *a* *P2* = set of classes that have as parent *a* *P3* = set of new metaclasses of *a* *P4* = set of new individuals that are type of *a* *P5* = set of new comments of *a* *P6* = set of new labels of *a*	Add property a with its neighborhood links *b* *P1* = set of new parent properties of *a* *P2* = set of properties that have as parent *a* *P3* = set of new metaproperties of *a* *P4* = set of new property instances of *a* *P5* = the new domain of *a* *P6* = the new range of *a* *P7* = set of new comments of *a* *P8* = set of new labels of *a*	Change the domain of property *a* *b* = old domain of *a* *c* = new domain of *a*
δ_a	$\forall\ p \in P1$ *: (a, subClassOf , p)* $\forall\ p \in P2$ *: (p, subClassOf , a)* $\forall\ p \in P3$ *: (a, type, p)* $\forall\ p \in P4$ *: (a, type, p)* $\forall\ p \in P5$*: (a, comment, p)* $\forall\ p \in P6$ *: (a, label, p),(a, type, class)* *(a, subClassOf , resource)*	$\forall\ p \in P1$ *: (a, subPropertyOf , p)* $\forall\ p \in P2$ *: (p, subPropertyOf , a)* $\forall\ p \in P3$ *: (a, type, p)* $\forall\ p1, p2 \in P4$ *: (p1, a, p2)* *(a, domain, p5)* *(a, range, p6)* $\forall\ p \in P7$ *: (a, comment, p)* $\forall\ p \in P8$ *: (a, label, p)* *(a, type, property)*	$\forall\ c \in C$ *: (a, domain, c)*
δ_d	Ø	Ø	$\forall\ b \in B$ *: (a, domain, b)*
Inverse	*Delete_Class*(a,*P1,P2,P3,P4,P5,P6)*	*Delete_Property*(a,*P1,P2,P3,P4,P5,P6,P7,P8)*	*Change Domain(a,c,b)*

DEFINITION 2 *(Affecting Change Operation). Let r be a resource of an ontology version O_m and E^{O_k,O_m} $(k<m)$ be the sequence of changes between O_k and O_m. A change operation $u \in E^{O_k,O_m}$ affects the resource r, denoted by aff(r), if $r \in O_m$ and $r \in \delta_a(u)$.*

An affecting change operation captures the way a resource was introduced between two ontology versions. Assuming that we have E^{O_k,O_m} already available, it is quite easy to identify the affecting change operation by just scanning the change log once. We have to note that the affecting change operation if it exists is *unique*. This is due to the fact that for our accepted languages of changes it should hold that for any two changes u_1, u_2, $\delta_a(u_1) \cap \delta_a(u_2) = \emptyset$ and $\delta_d(u_1) \cap \delta_d(u_2) = \emptyset$ as described in Section 2.

In our example the query *how('address')* when applied to E^{O_0,O_2} it will return the change operation *u8: MergeProperties({street, city},address)*. In the case that no affecting change operation is found, no answer will be returned. This means that we have no indication about how a resource was introduced in the ontology and probably the specific resource was introduced in a past ontology version.

4.2 When queries

Now we would like to know in which ontology version the '*address*' resource was introduced. The idea is similar to answering *how* provenance queries. We only have to scan once the change log $E^{O_k,O_m} = E^{O_k,O_{k+1}} \cup \ldots \cup E^{O_{m-1},O_m}$. If $aff(r) \in E^{O_k,O_m}$ and $r \in \delta_a(u)$ then obviously $r \in O_m$ so we can conclude that r was introduced in O_m. In our example the query *when('address')* will return O_2 as an answer.

4.3 Why queries

Presenting only the affecting change operations and the ontology version that a resource has been introduced does not necessarily provide insights on the corresponding ontology evolution. When drastic evolution occurs, those are not enough and we would like to get more information about which parts of the ontology evolved to produce the specific resource. Therefore, instead of providing just the affecting change operation and the ontology version, our idea is to present the history of the evolution of the specific parts of the ontology as an answer to *why* provenance queries.

For example, by checking the change log EE^{O_0,O_2}, presented on Section 3, we can easily identify that the operations shown in Figure 2, describe exactly the evolution of the '*address*' resource.

Presenting such a graph to the ontology engineers, their understanding on the ontology evolution is focused on the specific parts that evolved to produce the aforementioned resource. Such a sequence of change operations that depict the history of the ontology with respect to a specific resource r is called a *change path* for that resource. However, before defining the change path for a given resource we will define the change path for a *change operation* first.

DEFINITION 3 *(Change path for a change operation). A change path for the change operation $u \in E^{O_k,O_m}$, denoted by us^u_{path}, is the minimal sequence of change operations in E^{O_k,O_m} such that $u \in us^u_{path}$ and that $us^u_{path}(O_k) \subseteq O_m$.*

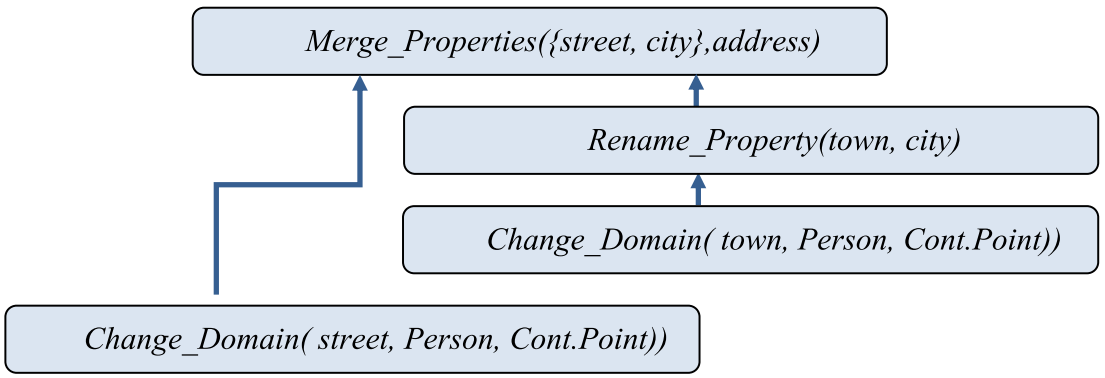


Figure 2 The change path for the 'address' resource visualized as a tree

A change path is *minimal* in the sense that one cannot remove any of the change operations in it and still $us^{u}_{\text{path}}(O_k) \subseteq O_m$. The change path presents the history of the evolution of the specific part of the ontology for a specific change operation. For example, the *change path* for the change *u8: Merge_Properties ({street, city},address)* is $us^{u8}_{\text{path}} = [u_3, u_4, u_5, u_8]$ as shown in Figure 2 and $us^{u8}_{\text{path}}(O_0) \subseteq O_2$.

PROPOSITION 1 (Uniqueness): The change path us^{u}_{path} over $E^{Ok,Om}$ is unique.

PROOF: Assume us^{u}_{path} is not unique. This would mean that we could have two change paths us^{u}_{path1} and us^{u}_{path2}. Since they are both change paths it should hold that $size(us^{u}_{\text{path1}}) = size(us^{u}_{\text{path2}})$ since they both have to be minimal. Now let $us^{u}_{\text{path1}} = [u_{k1}, \ldots, u_{kn}]$ and $us^{u}_{\text{path1}} = [u_{m1}, \ldots, u_{mn}]$. Since they are both change paths $u = u_{kn} = u_{mn}$. For $i < n$, each one of the u_{ki}, u_{mi} deletes a part of the ontology and adds another part. Since the two change paths have the same minimal size and $u = u_{kn} = u_{mn}$ in order to be different there must exist two change operations u_{ki}, u_{mj} such that $u_{ki} \neq u_{mj}$ and $\delta_d(u_{ki}) \cap \delta_d(u_{mi}) \neq \emptyset$ since they should delete a common part of the ontology. However, this is impossible since $\delta_d(u_1) \cap \delta_d(u_2) = \emptyset$ for our change operations.

Now, we will present an algorithm that given a change log produces the change path for a change operation u. The algorithm is shown in Figure 3. The idea is that the algorithm starts from the input change operation and identifies the triples that are added to the ontology, possibly by deleting other triples. Then it searches for the change operations that delete that added information in order to add new information and so on. After the execution of the algorithm, the change path for u will be stored in us'.

THEOREM 1: The ComputeChangePath algorithm computes us^{u}_{path} over $E^{Ok,Om}$.

PROOF: In order to prove that algorithm *Compute Change Path* computes the change path for a given change operation u over a change log $E^{Ok,Om}$ we have to prove that (a) $u \in us'$, (b) $us'(O_1) \subseteq O_2$ and that (c) us' is minimal.

(a) From line 1 of the algorithm indeed $u \in us'$.

(b) Let's assume that $us'(O_k)$ is not a subset of O_m. This would mean that there exists at least one triple, assume t' in $us'(O_k)$ such that it does not exist in O_m. So, to reach O_k, there should be a change operation u' such that $t' \in \delta_d(u')$ such that $t' \in \delta_a(us')$ not identified by our algorithm. However, this is impossible from line 3 of our algorithm.

(c) Now we prove minimality. Let's assume that us' is not minimal. This would mean that there is us_{path} with $size(us_{\text{path}}) < size(us')$. This would mean that there exist $u' \in us'$ such that $u' \notin us_{\text{path}}$. Of course this would mean from lines 3 and 4 that there exist t' such that $t' \in \delta_d(u')$ and $t' \in \delta_a(us')$. However, this would mean that t' does not belong to O_m, and should be deleted by another change operation. However, for our change operations it holds that $\delta_d(u_1) \cap \delta_d(u_2) = \emptyset$ which makes the previous statement impossible. Therefore, us' is minimal as well.

The time complexity of the algorithm is $O(N*M*S)$ where N is the number of change operations, M the maximum size of triples in a change operation u, and S the number of triples in $\delta_a(us')$. Moreover, it is easy

Algorithm 5.1: ComputeChangePath($E^{Ok,Om}$, u)
Input: A sequence $E^{Ok,Om} = [u_1, \ldots, u_n]$ and one change operation u
Output: a sequence of change operations us'
1. $us' := u$
2. For $i=n$ to 1
3. if there exists $t \in \delta_d(u_i)$ such that $t \in \delta_a(us')$
4. $us' := us' \cup u_i$
5. Return us'

Figure 3 Computing the change path for a given change operation

to adapt Algorithm 5.1 in order to retrieve the change path for a given *resource*. This will allow the developers to examine the evolution of the ontology regarding a specific resource:

DEFINITION 4 *(Change path for a resource). The change path* us_{path} *over* E^{O_k,O_m} *for the resource* $r \in O_m$ *is* $us^{r}_{path} = \cup us^{ui}_{path},\ r \in u_i$.

The corresponding algorithm is shown in Figure 4.

The idea is that we would like to retrieve the history of the evolution of resource *r*. However, *r* might appear in several triples so we need to identify all change paths that have to do with it.

THEOREM 2 The algorithm Compute Change Path Resource computes the change path for a given resource *r* over E^{O_k,O_m}.

The algorithm is immediately proved by construction. Algorithm 5.2 needs to scan the change log one time per triple containing the resource *r* in order to identify the change operation that inserts the given resource. So the complexity of the algorithm becomes O(*T***N***M***S*) where *T* is the number of triples containing *r, N* the number of change operations, *M* the maximum size of triples in a change operation *u* and *S* the number of triples in $\delta_a(us')$.

4.4 Inversibility

Our approach is not only applicable for visualizing the change path of a resource from the latest ontology version but can be used to visualize the change path of a resource from a past ontology version as well. To achieve this instead of getting as input to our algorithms the $E^{Ok,Om}$ where $k < m$ we could get the E^{O_k,O_m} and search for a resource belonging to O_k. However, instead of producing again the change log between the available versions we could exploit the inversibility property of the language of changes used (if available).

DEFINITION 4 *(Inverse Change Operation): A change operation invu from* O_2 *to* O_1 *is the inverse of change operation u from* O_1 *to* O_2 *if* $\delta_a(invu) = \delta_d(u)$ *and* $\delta_d(invu) = \delta_a(u)$.

Although inversibility is not a property of all language of changes, if a specific language of changes is inversible, this would allow us to issue queries from a past ontology version and to explore how resources from a past ontology version have been evolved to a latest ontology version without constructing from scratch the change log.

5 Implementation and high-level architecture

To deliver the aforementioned functionality, we designed and implemented the *EvoRDF* framework. The high-level architecture of the framework is shown in Figure 5 and all programmatic modules have been developed using Java.

The whole process starts by getting the change log constructed between two ontology versions output from a change detection algorithm similar to (Papavasileiou *et al*., 2013). An example part of the change

Algorithm 5.2: ComputeChangePathResource ($E^{O1,O2}$, r)
Input: A sequence $E^{O1,O2} = [u_1,\ldots,u_n]$ and one resource *r*
Output: a sequence of change operations *us′*
1. *us′* := ø
2. For *i*=*n* to 1
3. If $t \in \delta_a(u_i)$ such that $r \in t$
4. $us' := us' \cup ComputeChangePath(E^{O1,O2}, u_i)$
5. Return *us′*

Figure 4 Computing the change path for a given resource

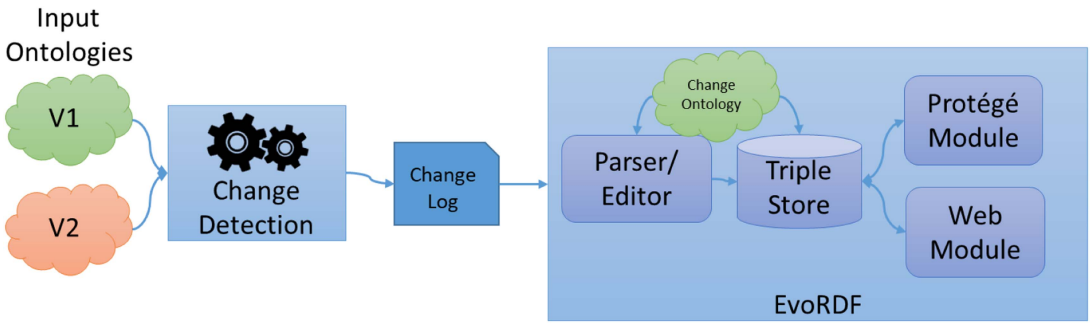


Figure 5 High-level architecture of the EvoRDF framework

```
http://cidoc#E24.Physical_Man-Made_Stuff:
RENAME_CLASS( http://cidoc#E24.Physical_Man-Made_Thing)

http://cidoc#E18.Physical_Stuff:
RENAME_CLASS( http://cidoc#E18.Physical_Thing)

http://cidoc#E65.Creation_Event:
RENAME_CLASS( http://cidoc#E65.Creation)

http://cidoc#E71.Man-Made_Stuff:
RENAME_CLASS( http://cidoc#E71.Man-Made_Thing)

http://cidoc#E16.Measurement_Event:
RENAME_CLASS( http://cidoc#E16.Measurement)

http://cidoc#E70.Stuff:
RENAME_CLASS( http://cidoc#E70.Thing)

http://cidoc#E66.Formation_Event:
RENAME_CLASS( http://cidoc#E66.Formation)

http://cidoc#E12.Production_Event:
RENAME_CLASS( http://cidoc#E12.Production)

http://cidoc#E11.Modification_Event:
RENAME_CLASS( http://cidoc#E11.Modification)

http://cidoc#E8.Acquisition_Event:
RENAME_CLASS( http://cidoc#E8.Acquisition)
```

Figure 6 An example change log

log constructed for the CIDOC-CRM versions v3.2.1 and v4.2 is shown in Figure 6. CIDOC-CRM (Doerr *et al*., 2007) is an ISO standard from the cultural domain, which consists of nearly 80 classes and 250 properties. The detected change log that was automatically produced identified 726 changes in total.

The file containing these change operations is then provided as input to the EvoRDF framework, which is composed of the parser, the triple store, the change ontology, the Protégé module and the Web module that will be described in the following sections.

5.1 The change ontology

In order to model all changes identified by the change detection mechanism a change ontology has been constructed. The ontology models all available types of change operations and their corresponding

arguments. Additional metadata are also modeled, such as the authors, the ontology versions and the corresponding δ_a and δ_d for each change operation. The change ontology has been implemented using Protégé and a part of the top-level class hierarchy is shown in Figure 7. Adapting our system to another language of changes is a matter of only providing a new change ontology with (a) the change operations used and (b) the corresponding δ_a and δ_d for each change operation.

5.2 The parser/editor

This module gets the ontology of the language of changes used and the change logs and initially visualizes in a tabular way all changes. Although we argue that only listing the changes is insufficient, the tool syntactically validates the change log and allows editing and cleaning the individual change operations. A screenshot of this module, visualizing some CIDOC-CRM changes, is shown in Figure 8.

Then, the whole set of changes is transformed to triples using the parser/editor module. More specifically a Virtuoso triple store is published with the change operations included in the change log *as instances* of the change ontology. An example demonstrating some transformed RDF triples is shown in Figure 9.

5.3 The EvoRDF explorer and the Protégé plugin

Having all change operations as instances of the Change Ontology, the EvoRDF Protégé and Web modules and the Protégé plugin can be used to explore ontology evolution. These modules implement all algorithms reported in Section 4. Three types of queries are available using these modules:

- When and how queries: An end-user can search for the specific version that a resource was introduced. In addition, an end-user can search for the change operation that introduced a specific resource (how). As soon as the end-user issues a query, the corresponding SPAQRL query is forwarded to the Virtuoso server in order to retrieve the specific change affecting the requested resource. To retrieve the change operation, that affects the selected resource, multiple SPARQL queries are issued (one per change operation type). An example is shown in Figure 10, requesting the source, the target and the type of a rename change operation affecting a specific resource.

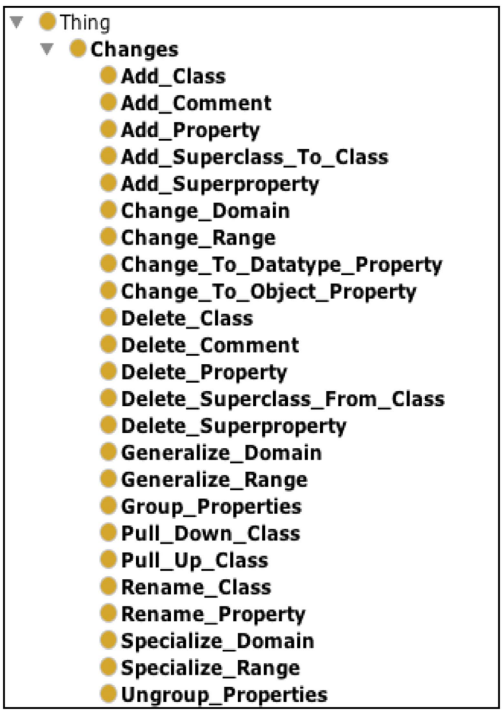
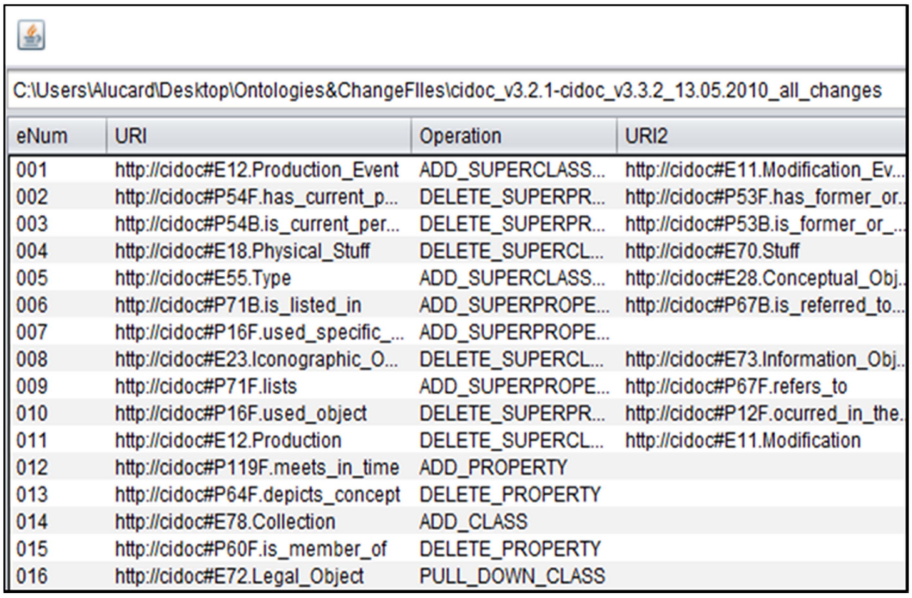


Figure 7 A part of the change ontology



C:\Users\Alucard\Desktop\Ontologies&ChangeFIles\cidoc_v3.2.1-cidoc_v3.3.2_13.05.2010_all_changes

eNum	URI	Operation	URI2
001	http://cidoc#E12.Production_Event	ADD_SUPERCLASS...	http://cidoc#E11.Modification_Ev...
002	http://cidoc#P54F.has_current_p...	DELETE_SUPERPR...	http://cidoc#P53F.has_former_or...
003	http://cidoc#P54B.is_current_per...	DELETE_SUPERPR...	http://cidoc#P53B.is_former_or_...
004	http://cidoc#E18.Physical_Stuff	DELETE_SUPERCL...	http://cidoc#E70.Stuff
005	http://cidoc#E55.Type	ADD_SUPERCLASS...	http://cidoc#E28.Conceptual_Obj...
006	http://cidoc#P71B.is_listed_in	ADD_SUPERPROPE...	http://cidoc#P67B.is_referred_to...
007	http://cidoc#P16F.used_specific_...	ADD_SUPERPROPE...	
008	http://cidoc#E23.Iconographic_O...	DELETE_SUPERCL...	http://cidoc#E73.Information_Obj...
009	http://cidoc#P71F.lists	ADD_SUPERPROPE...	http://cidoc#P67F.refers_to
010	http://cidoc#P16F.used_object	DELETE_SUPERPR...	http://cidoc#P12F.ocurred_in_the...
011	http://cidoc#E12.Production	DELETE_SUPERCL...	http://cidoc#E11.Modification
012	http://cidoc#P119F.meets_in_time	ADD_PROPERTY	
013	http://cidoc#P64F.depicts_concept	DELETE_PROPERTY	
014	http://cidoc#E78.Collection	ADD_CLASS	
015	http://cidoc#P60F.is_member_of	DELETE_PROPERTY	
016	http://cidoc#E72.Legal_Object	PULL_DOWN_CLASS	

Figure 8 The parser/editor tool

```
<owl:NamedIndividual rdf:about="&untitled-ontology-10;019">
    <rdf:type rdf:resource="&untitled-ontology-10;RENAME_CLASS"/>
    <rename_from>
        http://cidoc#E66.Formation_Event
    </rename_from >
    <rename_to>
        http://cidoc#E66.Formation
    </rename_to >
    <from_version>3.4.9</from_version>
    <to_version>4.2</to_version>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="&untitled-ontology-10;020">
    <rdf:type rdf:resource="&untitled-ontology-10;RENAME_CLASS"/>
    < rename_from >
        http://cidoc#E12.Production_Event
    </rename_from >
    < rename_to>
        http://cidoc#E12.Production
    </rename_to >
    <from_version>3.4.9</from_version>
    <to_version>4.2</to_version>
</owl:NamedIndividual>
```

Figure 9 Example instances of the change ontology

- Why queries: In addition, the change path (i.e. the consecutive list of change operations) that led to the creation of a specific resource can be computed and presented to the user allowing further exploration and understanding of the evolution of the ontology (e.g. What are the change operations that had some influence on the creation of the '*address*' literal? How was the '*street*' literal evolved starting from ontology version O_0?). Why queries are possible not only for specific resources but for specific change

```
SELECT *
WHERE{
    ?change_operation a ?change_operation_type.
    ?change_operation change_ontology:from_version ?from_version.
    ?change_operation change_ontology:to_version ?to_version.
    ?change_operation change_ontology:rename_to ? _RESOURCE_.
}
```

Figure 10 The SparQL query for retrieving the rename change operations affecting a resource (_RESOURCE_)

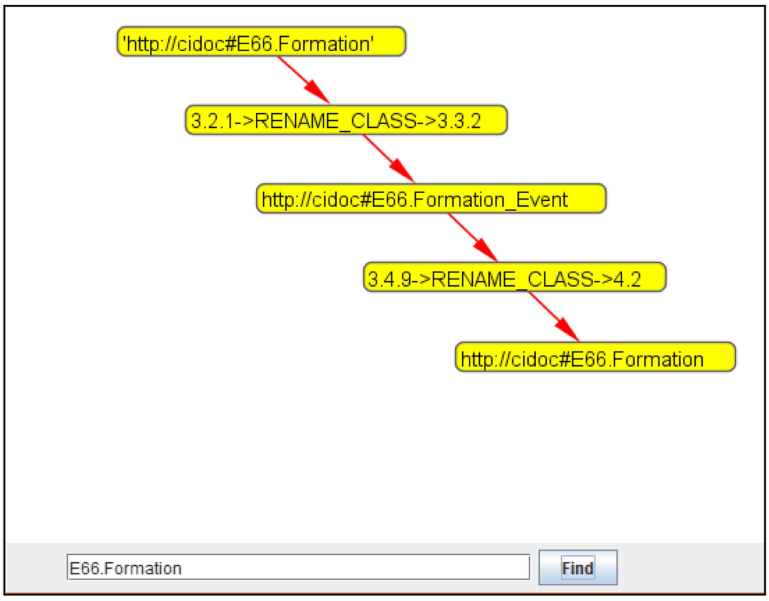


Figure 11 The EvoRDF Protégé module

operations as well. The corresponding SPARQL queries formulated, similar to the SPARQL query shown in Figure 10, are issued to the Virtuoso triple store and the results are returned to the user.

Finally, the graphical user interface retrieves the results and visualizes the returned information. The visualization paradigms adopted by the EvoRDF Protégé and Web modules are shown in Figures 11 and 12, respectively. More specifically the Protégé plugin allows searching a specific resource and automatically the corresponding change path is visualized and presented to the end-user. On the other hand, the online module allows uploading multiple ontology versions and change logs and visually exploring ontology versions and change logs[1]. Besides visualizing the results as a change path, they can also be visualized in a tabular way. The graphical change path is more appealing, however, the tabular presentation includes more detailed information, for example, comments left by the ontology engineers, additional metadata, etc.

6 Evaluation

In order to evaluate the algorithms reported in this paper, we used a workstation with an Intel Core i7 processor running at 3.4 Ghz, and 4-GB memory, using Windows 7 × 64. Moreover, we used two well-known ontologies: one medium-size ontology, CIDOC-CRM from the cultural domain, which is rarely

[1] http://139.91.183.29:8080/exelixis/

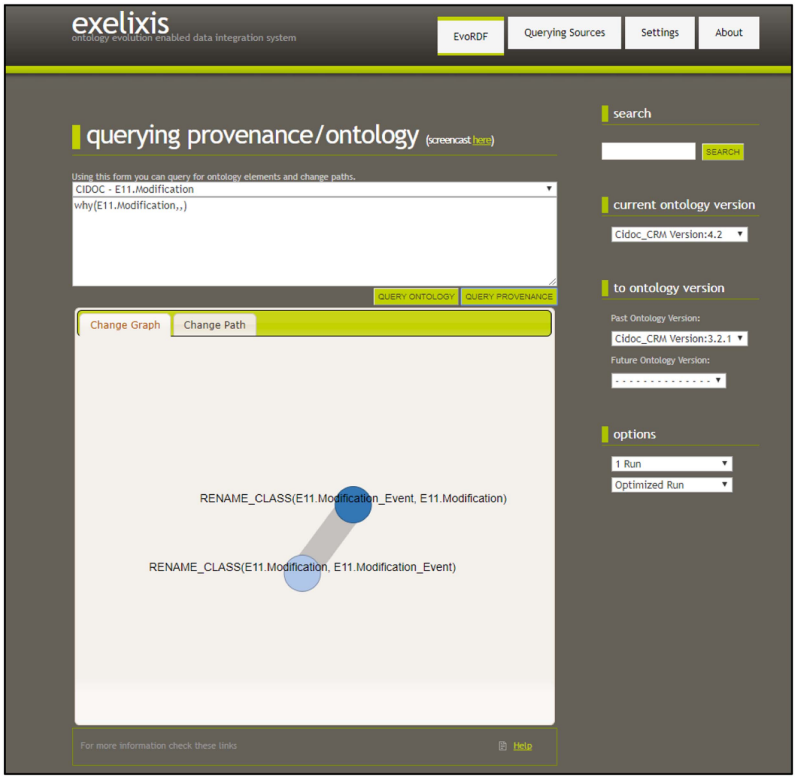


Figure 12 The EvoRDF Web module

changed, and one large-size ontology (GO; Gene Ontology Consortium, 2004) from the bioinformatics domain, which is heavily updated daily.

Besides CIDOC-CRM and the corresponding change log that we already described in previous section, GO is composed of about 28 000 classes and 1350 property instances. GO is updated on a daily basis and for our experiments we used the change log from 25.11.08 to 26.05.09. The change log that was automatically produced contained 4175 changes.

The change logs and the ontologies used in our experiment can be found online[2].

6.1 Answering provenance queries

Next, we present experiments evaluating the scalability of the algorithms for answering provenance queries. We report the average execution time for computing answers to *how*/*when* and *why* provenance queries, exhaustively issuing the corresponding queries for all resources from the latest ontology version. The results are shown in Figure 13.

As shown in the graphs, the average time to produce a change path increases linear to the number of changes we have to search. This is in line with the complexity of our algorithms. Moreover, the time to compute answers to *why* provenance queries is greater than computing answers to *how*/*when* queries. This is reasonable since in the first case more triples are being searched according to the corresponding algorithm. However, we can see that the overhead for searching the added triples in the change path has little impact in the total execution time since the dominant factor is the number of change operations. So, for CIDOC-CRM after 726 change operations we only need 275 millisecond on average to compute *how*/*when* provenance whereas for *why* provenance we need 280 msec. On the other hand, for GO after 4175 changes we need 4611 msec for *how*/*when* queries and 4967 millisecond for *why* queries.

[2] `http://users.ics.forth.gr/~kondylak/Ontologies&ChangeFIles.rar`

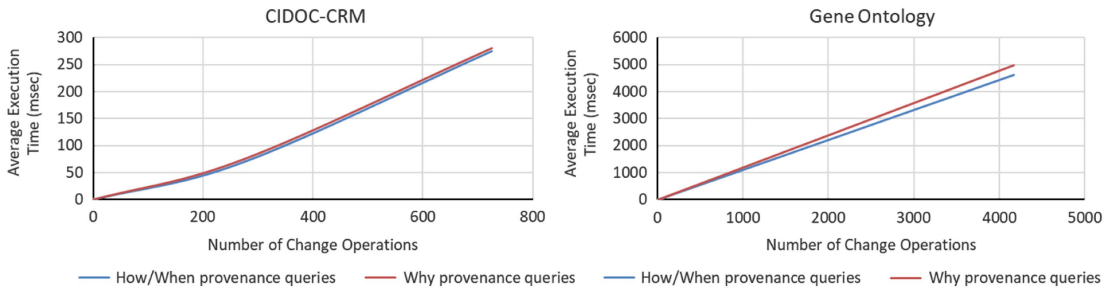


Figure 13 The average execution time compared to the number of changes for CIDOC-CRM and Gene Ontology

Obviously, the time to compute a change path is greater for the GO than for the CIDOC-CRM. This is reasonable, since for GO we have to search 4175 changes, whereas for CIDOC-CRM we only have to search 726 changes.

Moreover, we've identified that the biggest number of changes in a change path for GO was eight, whereas for CIDOC-CRM it was five. So, independent of the number of changes between ontology versions the interested user has to check at most eight change operations to understand how the specific part of the ontology has been evolved. We have to note here that the average number of change operations that a user had to examine was two for CIDOC-CRM and four for GO, which shows the benefit of our approach even for ontologies that change often.

6.2 Usability evaluation

To evaluate the usability of the system we performed two usability evaluations: (a) A preliminary usability evaluation at the Extended Semantic Web Conference, using the System Usability Scale (SUS) and (b) a complete usability evaluation at the Technological Educational Institute of Greece, using both the ISO/IEC 25000 series and SUS.

In the following, we describe in detail how we implemented each one of those.

6.2.1 Preliminary usability evaluation at Extended Semantic Web Conference

The demo paper of the aforementioned work was presented at the Extended Semantic Web Conference (Kondylakis *et al.*, 2017). During the demonstration, ten conference participants were invited to directly interact with the framework and explore CIDOC-CRM ontology evolution. Apart from the interesting discussions made, they were asked to give an overall rating on the usability of the system ranging from 0 to 100 using the SUS (Brooke, 1996), an instrument used for the global assessment of systems usability. The average score of the participants was 87.5 demonstrating the high quality of our implementation. While it is technically correct that a SUS score of 87 out of 100 represents 87% of the possible maximum score, it suggests the score is at the 87th percentile. A score at this level would mean the application tested is above average. Usually, as a gold standard is often used the 68 as reported by Sauro: ''The average SUS score from all 500 studies is a 68. A SUS score above a 68 would be considered above average and anything below 68 is below average' (2009, 2011). In our case, the score received identifies the high usability of our modules.

Additional comments were made from the conference participants. For example, the Protégé module visualizes only the latest ontology version and allows querying over the change log. However, several participants would like to be able to see other ontology versions as well without having to load them in separate Protégé instances. In addition, currently, all changes are stored in a triple store in a remote server and the two modules should have Internet connection in order to be able to issue queries. However, some of the participants would like to be able to explore ontology evolution even without Internet connection. Finally, some other participants would like to be able to explore the changes as open linked data. Currently, we are exploring how these requests can be implemented effectively without overloading users with many ontology versions and other unrelated information.

Table 2 The results for the various evaluation categories

Functionality	Suitability	4.54	4.48
	Accurateness	4.45	
	Compliance	4.45	
Efficiency	Time behaviour	4.63	4.36
	Resource utilization	4.09	
Compatibility	Co-existence	4.00	4.04
	Interoperability	4.09	
Usability	Understandability	3.08	3.88
	Learnability	4.18	
	Operability	4.09	
	Attractiveness	4.18	
Reliability	Fault tolerance	4.00	3.77
	Recoverability	3.54	
Maintainability	Analyzability	3.90	4.22
	Changeability	4.36	
	Stability	4.45	
	Testability	4.18	
Portability	Adaptability	4.09	4.01
	Installability	4.00	
	Conformance	4.09	
	Replaceability	3.90	
Quality of use	Effectiveness	4.27	4.05
	Efficiency	4.45	
	Satisfaction	3.45	
SUS		83/100	

SUS = System Usability Scale.

6.2.2 Usability evaluation at Technological Educational Institute of Crete

For the usability evaluation at Technological Educational Institute of Crete, the quality features from both the product quality model of the ISO/IEC 25000 series (ISO/IEC 42010:2007, 2007; ISO/IEC DIS 25023, 2016), and the SUS were used. At the evaluation stage 20 experts from the Department of Informatics Engineering were selected. All users had a background of ontology development and they were given a short presentation of the tool. Then they were asked to use the tool for 5 minutes completing specific tasks. They had to understand how specific resources over GO and CIDOC-CRM were evolved and to visualize their change path. Then they were left to interact freely with the tool. At the end, they had to complete two questionnaires with simple, accurate, non-time consuming, set of questions and to write some comments (Table 2).

The questionnaire generated for evaluating software quality measures from the ISO/IEC 25000 series included simple questions in natural language (the questions used are shown in the Appendix), that the participants had to answer within a degree of satisfaction using a Likert scale (Likert, 1932). For SUS, we used the standard questionnaire with (Brooke, 1996) 10 questions.

The results of our evaluation are illustrated in Table 1. Values greater than three represent high level of the specific software feature; values between 2.5 and 3 are at low risk, whereas values below 2.5 are considered of high risk. In our case, all software features were graded with a high mark, showing again the high quality of the software. The lowest average score was for the understandability (3.08/5) since the purpose and the functionality of the system was difficult to be understood without our presentation. User's feedback was used to enhance the usability of the modules, so, for example, indicative queries were added to the interface. The highest average score for functionality (4.48/5) shows that the system indeed includes a really interesting set of functionalities. When it comes to efficiency, compatibility, reliability, maintainability, portability and quality of use, our system received high values as well. Finally, a SUS value of 83 received, is in line with our preliminary evaluation, confirming again the high quality of our system.

Additional comments made in the evaluation forms provided additional valuable feedback, suggesting various improvements. For instance, some users proposed to add graphs summarizing the number and

categories of different types of changes similar to (Stefanidis *et al*., 2016). We agree that adding such summaries would enhance user experience and we leave this for future work. Another interesting direction proposed was to focus only on the most interesting changes instead of visualizing them all. To some extent, we have already implemented this, visualizing comments only in the tabular view, but moving beyond such simple selection would require a measure identifying the most interesting ones. Nevertheless, in our experiments, a user had to check at most eight changes which we consider a small number.

6.2.3 Interesting observations

Finally, trying to understand ontology evolution through provenance queries, we made several interesting observations. One of them, for example, was the following: We identified that in the evolution of the CIDOC-CRM ontology from versions v3.2.1 to v3.3.2, one ontology engineer renamed the class '*E11 Modification*' to '*E11 Modification Event*'. A few years later, another ontology engineer was employed to evolve the ontology. In v4.2 we can see that the class '*E11 Modification Event*' was again renamed to '*E11 Modification*'. If the second ontology engineer had an indication of the previous renaming he would avoid cycles, as he would also be able to identify the reasons behind each renaming documented in the corresponding comments. As such, using provenance queries to explore ontology evolution can be a valuable tool reducing greatly the time spent on understanding evolution.

7 Conclusion

In this paper, we present a novel framework for enabling exploration of ontology evolution using provenance queries. Instead of just identifying and presenting the changes from one ontology version to another, our framework can answer fine-grained provenance queries for individual resources. It can identify *when* a resource was created, *how* it was introduced and it can present the change operations that lead to the creation (or deletion) of that resource visualizing its evolution history. This greatly minimizes the total time for understanding ontology evolution. Experiments performed, show the high usability of the two visualization modules and the potential impact of our approach. For example, for CIDOC-CRM provenance queries can be answered at most within 280 msec and for GO at most within 5secs even if there are more than 4000 changes that have to be examined. Moreover, ontology engineers have to examine at most five change operations for CIDOC-CRM and eight change operations for GO to understand how the ontology evolved.

In addition, our approach is independent of the language of changes used. For reasons of simplicity, we employed a rich language containing over 100 types of change operations but other languages could be selected as well, as long as the following requirements are fulfilled:

a. The (δ_a, δ_d) for each change operation type described by the language should be provided.
b. $\delta_a(u) \cap \delta_d(u) = \phi$ and $\delta_a(u) \cup \delta_d(u) \neq \phi$ if $O_1 \neq O_2$.
c. For any two changes u_1, u_2 in a sequence of changes $\delta_a(u_1) \cap \delta_a(u_2) = \phi$ and $\delta_d(u_1) \cap \delta_d(u_2) = \phi$ should hold.

Those properties ensure the uniqueness of the identified change paths. As future work, several challenging issues need to be further explored. A first interesting challenge would be to relax the second and the third requirements for the input language of changes. This would lead to multiple change paths for a given resource that would have to be appropriately visualized. Moreover, cycles might occur, and mechanisms ensuring the termination of the respective algorithms should be provided.

Another interesting direction, as identified by the conference participants, would be to enable ontology exploration in an offline mode. There is no restriction currently on the implemented algorithms, forcing them to use an external triple store. Our algorithms can also work directly on the provided change log files. To this purpose, minor modifications are needed, since both the algorithms and the change log parser exist already. Another dimension highlighted by the end-users is access to the changes as open linked data. This is possible as well, since our triple store includes already a SPARQL endpoint to which anyone can connect and issue queries. However, we believe that the visualization options we provide on top of the triple store, really augment user understanding and enable effective exploration of the ontology evolution, without requiring the formulation of error-prone SPARQL queries. The following months we intend to

release a new version of the Protégé module with the offline functionality and to publish the SPARQL endpoint for exploiting the available changes as open linked data.

An interesting extension would also be to combine our approach with other statistical tools (like the ones in Stefanidis *et al*., 2016) offering the best of the approaches for exploring evolution and identifying statistical information over the available changes. Another interesting direction would be to combine the PROV ontology with the ontology of changes we have created, capturing more information and metadata on the activities related to ontology evolution.

Finally, a possible direction for further exploration is to present summaries (Papas *et al*., 2017) of the evolved change path if they become too big or graphical summaries of the entire evolution of an ontology. Although the latest years, a vast amount of research works has focused independently on summarizing information (Theoharis, 2007; Troullinou *et al*., 2014; Troullinou *et al*., 2015) and on ontology evolution (Roussakis *et al*., 2015; Stardog, 2018; Troullinou *et al*., 2017), currently there is no work combining both worlds in an effective manner and more research is required to this direction. Is obvious, that ontology evolution is becoming more and more important topic and several challenging issues remain to be investigated in the near future.

Acknowledgements

The authors would like to thank Melidoni Despoina, Georgios Glykokokalos, Manos Karapiperakis, Michail-Angelos Lasithiotakis, John Makridis, Panagiotis Moraitis, Aspasia Panteri, Maria Plevraki, Antonios Providakis, Maria Skalidaki, Athanasiadis Stefanos, Manolis Tampouratzis, Eleftherios Trivizakis, Fanis Zervakis, Ekaterini Zervouraki for implementing the framework described in this paper. This work has been supported by the EU project iManageCancer (H2020-643529) and has been partially funded by the European Commission.

Appendix

Table A1 The selected sub-characteristics for the evaluation form and their translation into simple questions

Functionality	Suitability	Can software perform the tasks required?
	Accurateness	Is the result as expected?
	Compliance	Is the system compliant with standards?
Efficiency	Time Behavior	How quickly does the system respond?
	Resource utilization	Does the system utilize resources efficiently?
Compatibility	Co-existence	Can the system share resources without loss of its functionality?
	Interoperability	Can the system share information/data with other components?
Usability	Understandability	Does the user comprehend how to use the system easily?
	Learnability	Can the user learn to use the system easily?
	Operability	Can the user use the system without much effort?
	Attractiveness	Does the interface look good?
Reliability	Fault tolerance	Is the software capable of handling errors?
	Recoverability	Can the software resume working and restore lost data after failure?
Maintainability	Analyzability	Can faults be easily diagnosed?
	Changeability	Can the software be easily modified?
	Stability	Can the software continue functioning if changes are made?
	Testability	Can the software be tested easily?
Portability	Adaptability	Can the software be moved to other environments?
	Installability	Can the software be installed easily?
	Conformance	Does the software comply with portability standards?
	Replaceability	Can the software easily replace other software?
Quality of use	Effectiveness	How accurate and complete is the software for the intended use?
	Efficiency	Does the software improve the time or reduce resources for the intended goal?
	Satisfaction	Does the software satisfy the perceived achievements of pragmatic goals?

References

Arndt, N., Naumann, P. & Marx, E. 2017. Exploring the evolution and provenance of Git versioned RDF data. MEPDaW/LDQ@ESWC, 12–27.

Avgoustaki, A., Flouris, G., Fundulaki, I. & Plexousakis, D. 2016. Provenance management for evolving RDF datasets. In *ESWC*, 575–592.

Benjelloun, O., Sarma, A. D., Halevy, A., Theobald, M. & Widom, J. 2008. Databases with uncertainty and lineage. *The VLDB Journal* **17**, 243–264.

Buneman, P., Khanna, S. & Tan, W. C. 2001. Why and where: a characterization of data provenance. In *ICDT*, 316– 330.

Brooke, J. 1996. SUS - a quick and dirty usability scale. In *Usability Evaluation in Industry*, Jordan, P. W., Thomas, B., McClelland, I. L. & Weerdmeester, B. (eds). CRC Press, 189–194.

Chiticariu, L. & Tan, W. C. 2006. Debugging schema mappings with routes. In *VLDB*, 79–90.

Curino, C., Moon, H., Deutsch, A. & Zaniolo, C. 2013. Automating the database schema evolution process. *The VLDB Journal* **22**(1), 73–98.

De Nies, T., Magliacane, S., Verborgh, R., Coppens, S., Groth, P., Mannens, E. & Van de Walle, R. 2013. Git2prov: exposing version control system content as w3c prov. In *Proceedings of the 2013th International Conference on Posters & Demonstrations Track*, **1035**, 125–128.

Doerr, M., Ore, C. E. & Stead, S. 2007. The CIDOC conceptual reference model: a new standard for knowledge sharing. *Tutorials, Posters, Panels and Industrial Contributions at the ER* **83**, 51–56.

Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D. & Antoniou, G. 2008. Ontology change: classification and survey. *Knowledge Engineering Review* **23**, 117–152.

Gene Ontology Consortium 2004. The Gene Ontology (GO) database and informatics resource. *Nucleic Acids Research* **32**, D258–D261.

Graube, M., Hensel, S. & Urbas, L. 2014. R43ples: revisions for triples. In *Proceedings of the 1st Workshop on Linked Data Quality Co-Located with 10th International Conference on Semantic Systems (SEMANTiCS)*.

Green, T. J., Karvounarakis, G. & Tannen, V. 2007. Provenance semirings. In *ACM SIGMOD-SIGACT-SIGART PODS*. ACM, 31–40.

ISO/IEC 42010:2007, 2007. Systems and software engineering – recommended practice for architectural description of software-intensive systems.

ISO/IEC DIS 25023, 2016. Systems and software engineering – systems and software quality requirements and evaluation (SQuaRE) – measurement of system and software product quality.

Kondylakis, H., Melidoni, D., Glykokokalos, G., Kalykakis, E., Lasithiotakis, M. E., Makridis, J., Moraitis, P., Panteri, A., Plevraki, M., Providakis, A., Skalidaki, M., Stefanos, A., Tampouratzis, M., Trivizakis, E., Zarvakis, F., Zervouraki, E. & Papadakis, N. 2017. EvoRDF: A framework for exploring ontology evolution. In *ESWC (demos)*.

Kondylakis, H. & Plexousakis, D. 2014. Exploring RDF/S evolution using provenance queries. In *EDBT/ICDT Workshops*.

Lebo, T., Sahoo, S., McGuinness, D., Belhajjame, K., Cheney, J., Corsar, D., Garijo, D., Soiland-Reyes, S., Zednik, S. & Zhao, J. 2013. Prov-o: The PROV ontology. W3C recommendation 30.

Likert, R. 1932. A technique for the measurement of attitudes. *Archives of Psychology* **140**, 1–55.

Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plate, B., Simmhan, Y., Stephan, E. & van den Bussche, J. 2011. The open provenance model core specification (v1. 1). *Future Generation Computer Systems* **27**(6), 743–756.

Noy, N. F., Chugh, A., Liu, W. & Musen, M. A. 2006. A framework for ontology evolution in collaborative environments. In *ISWC*, 544–558.

Papas, A., Troullinoy, G., Roussakis, G., Kondylakis, H. & Plexousakis, D. 2017. Exploring importance measures for summarizing RDF/S KBs. In *ESWC*.

Papavasileiou, V., Flouris, G., Fundulaki, I., Kotzinos, D. & Christophides, V. 2013. High-level change detection in RDF(S) KBs. *ACM Transactions on Database Systems* **38**(1), 1:1–1:42.

Papavassiliou, V. 2010. Detecting deterministically high-level changes for RDF/S knowledge bases. Master's Thesis, Computer Science Department, University of Crete.

Plessers, P. & Troyer, O. D. 2005. Ontology change detection using a version log. In *ISWC*, 578–592.

Plessers, P., Troyer, O. D. & Casteleyn, S. 2007. Understanding ontology evolution: a change detection approach. *Web Semantics: Science, Services and Agents on the World Wide Web* **5**, 39–49.

RDF PrimerW3C Recommendation. 2004. `http://www.w3.org/TR/rdf-primer/`

Rogozan, D. & Paquette, G. 2005. Managing ontology changes on the semantic web. In *IEEE/WIC/ACM International Conference on Web Intelligence*, 430–433.

Roussakis, Y., Chrysakis, I., Stefanidis, K. & Flouris, G. 2015. D2V: a tool for defining, detecting and visualizing changes on the data web. In *ISWC (Posters & Demos)*.

Ruiz, J. E., Grau, B. C., Horrocks, I. & Berlanga, R. 2011. Supporting concurrent ontology development: Framework, algorithms and tool. *Data & Knowledge Engineering* **70**(1), 146–164.

Sauro, J. R. L. 2009. Correlations among prototypical usability metrics: evidence for the construct of usability. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1609–1618. ACM.

Sauro, J. R. L. 2011. Measuring usability with the System Usability Scale (SUS). `https://measuringu.com/sus/`(Accessed April 2018).

Stardog. 2018. `https://www.stardog.com/`(Accessed April 2018).

Stefanidis, K., Flouris, G., Chrysakis, G. & Roussakis, Y. 2016. D2V – understanding the dynamics of evolving data: a case study in the life sciences. ERCIM News, 105.

Stojanovic, L. 2004. Methods and tools for ontology evolution. Phd, University of Karlsruhe.

Theoharis, Y. 2007. On graph features of semantic web schemas. *IEEE Transactions on Knowledge and Data Engineering* **20**, 692–702.

Troullinou, T., Kondylakis, H., Daskalaki, E. & Plexousakis, D. 2014. RDF digest: efficient summarization of RDF/S KBs. In *Extended Semantic Web Conference (ESWC)*.

Troullinou, T., Kondylakis, H., Daskalaki, E. & Plexousakis, D. 2015. RDF digest: ontology exploration using summaries. In *International Semantic Web Conference (ISWC)*.

Troullinou, T., Kondylakis, H., Daskalaki, E. & Plexousakis, D. 2017. Ontology understanding without tears: the summarization approach. *Semantic Web Journal* **8**(6), 797–815.

Troullinou, T., Roussakis, G., Kondylakis, H., Stefanidis, K. & Flouris, G. 2016. Understanding ontology evolution beyond deltas. In *EDBT/ICDT Workshops*.

Volkel, M., Winkler, W., Sure, Y., Kruk, S. R. & Synak, M. 2005. Semversion: a versioning system for RDF and ontologies. In *ESWC*.

Zablith, F., Antoniou, G., D'Aquin, M., Flouris, G., Kondylakis, H., Motta, E., Plexousakis, D. & Sabou, M. 2015. Ontology evolution: a process-centric survey. *The Knowledge Engineering Review* **30**(1), 45–75.