A New Approach to Managing the Evolution of OWL Ontologies

Chuming Chen and Manton M. Matthews
Department of Computer Science and Engineering
University of South Carolina
Columbia, SC 29208, USA

Abstract *The growing demand for large and complex ontologies present new challenges related to their design, maintenance and evolution. In this paper, we propose an approach to managing the evolution of large OWL ontologies. Unlike conventional approaches that focus on the versioning of entire ontology, we use an evolutionary log to track the lifeline of each axiom or annotation that appears in an evolving ontology. We introduce the formal model for our ontology evolution framework, present related algorithms, and discuss the potential implementation scenarios.*

Keywords: Description Logics, OWL, Ontology Evolution, Change Management, Metadata

1 Introduction

There have been great efforts in building ontologies to support annotating, integrating and analyzing diverse sources of high-throughput biomedical data. The NCI Thesaurus [7] and the Gene Ontology [5] are examples of complex and important ontologies in the domains of biology and biomedicine. Such ontologies are usually very large and comprise tens to hundreds of thousands of classes. They constantly evolve to incorporate the latest experimental discoveries and biomedical knowledge.

The dynamic nature of the biomedical domains and the imperfect knowledge acquisition process suggest that ontology developers unavoidably need to revise an ontology. For the large ontologies, more and more often, the changes are only made to a very small portion of the ontology. Simply creating another version of the entire ontology when small changes are made is an inefficient solution. Detecting changes involves comparing two versions of a large ontology, which makes tracking changes of even small fragment of an ontology very expensive.

Large ontologies are usually developed collaboratively by a group of domain experts and ontology engineers with each member contributing to the small fragment of an ontology, which falls into their expertise. In order to guarantee the quality of an ontology, generally, only a small number of people should have the authority to make changes to that part of the ontology. This also facilitates tracking the history of changes.

Modern ontologies are developed with logic-based ontology languages such as OWL [11]. An OWL ontology consists of a set of axioms and annotations. The axioms explicitly make statements that say what is true about the modeled domain. The annotations associate some human friendly information with the elements of an ontology. In this paper, we propose using an evolutionary log to track the lifelines of axioms and annotations in an evolving ontology. We do this by associating each axiom or annotation with a set of metadata (e.g. ownership, access privileges, time stamps of changes etc.). Using an evolutionary log, we can easily reconstruct the evolution of the entire ontology.

The rest of paper is structured as follows: Section 2 introduces the axiomatic ontology model. Section 3 presents the formal model of our ontology evolution framework. Section 4 describes the related algorithms. Section 5 discusses the implementation scenarios. Section 6 reviews the related work. We summarize the paper and outline future work in Section 7.

2 Axiomatic Ontology Model

Description Logics (DL) [2] are a family of concept-based knowledge representation formalisms. They describe the domain of interest in terms of concepts, roles and individuals with precise semantics to enable deducing implicit knowledge from explicitly represented knowledge.

OWL [11] is a family of Description Logics based ontology languages designed to enable automated machine reasoning on the Semantic Web. OWL includes three sublanguages with incremental expressiveness: OWL-Lite, OWL-DL and OWL-Full. OWL-DL is a rich ontology language that supports high expressiveness and decidable reasoning. OWL-DL is a syntactic variant of the Description Logic $\mathcal{SHOIN}(\mathcal{D})$ [10], where $\mathcal{D}$ is a concrete domain of data types. As a logical theory, an OWL-DL ontology consists of a set of axioms that reference a

set of classes, properties and individuals (concepts, roles and individuals in DL). An individual is a single object in the domain. A class is a group of objects in the domain. A property represents the binary relationship between two classes. More expressive Description Logics $\mathcal{SHOIQ}$ [9], $\mathcal{SROIQ}$ [8] and their corresponding automated reasoning algorithms were developed very recently. The logic $\mathcal{SROIQ}$ has been adopted as the logic foundation for the next generation of OWL (OWL 2 [14]).

3 Ontology Evolution

3.1 Motivating Example

Most existing work on ontology evolution are based on either frame-like or object models, which focus on the entities (classes, properties and individuals) in the ontology. Instead, we use axiomatic ontology model that focuses on the changes to the axioms instead of entities.

As a running example, we consider two versions $\mathcal{O}_i$ and $\mathcal{O}_{i+1}$ of a simple evolving ontology modeling an academic department shown as two sets of axioms in Table 1. For reader's convenience, we use OWL 2 functional-style syntax instead of the more verbose XML syntax. We also show their corresponding DL syntax.

Assume ontology engineers Alice and Bob were in charge of maintaining this simple ontology. In ontology $\mathcal{O}_i$, Alice modeled the $Student$ as being disjoint with $Employee$ (Ax2) and commented it with an annotation axiom Ax3. Note Ax3 has no corresponding DL syntax, because it only provides human-friendly information and bears no logical meaning. In ontology $\mathcal{O}_i$, John was a $Student$. Mary was an $Assistant\ Professor$. The assertion $(John, Mary) : hasSupervisor$ is satisfiable with respect to the ontology $\mathcal{O}_i$, because the domain of $hasSupervisor$ role is $Student$, the range of $hasSupervisor$ role is $Faculty$, $Assistant\ Professor$ is a $Faculty$.

Later on in the ontology $\mathcal{O}_{i+1}$, Bob introduced a new class TA ($TA \sqsubseteq Student \sqcap Employee$). To keep the ontology consistent, he removed the axiom $Student \sqsubseteq \neg Employee$ from the ontology. In the ontology $\mathcal{O}_{i+1}$, Mary became an $Associate\ Professor$. John became a TA. Since the domain of $hasSupervisor$ role is $Student$ and the range of $hasSupervisor$ role is $Faculty$, the assertion $(John, Mary) : hasSupervisor$ is still satisfiable with respect to the ontology $\mathcal{O}_{i+1}$.

3.2 Ontology Evolution Management

3.2.1 Change History

Ontology development is in essence an ontology evolution process represented by a sequence of changes to the ontology. Therefore, it is indispensable that we need detailed history of all the changes ever made.

What were the changes? For the axiomatic ontology model used in this paper, we define the primitive change operations from ontology $\mathcal{O}_i$ to ontology $\mathcal{O}_{i+1}$ to be adding or removing axioms or annotations, denoted as $A^+(\mathcal{O}_i \rightarrow \mathcal{O}_{i+1})$ and $A^-(\mathcal{O}_i \rightarrow \mathcal{O}_{i+1})$ respectively. More complex change operations can be defined as a sequence of these primitive change operations. In our running example, we can identify the changes as follows: $A^+(\mathcal{O}_i \rightarrow \mathcal{O}_{i+1}) = \{$Ax8, Ax9, As4, As5$\}$, $A^-(\mathcal{O}_i \rightarrow \mathcal{O}_{i+1}) = \{$Ax2, Ax3, As1, As2$\}$.

Who has made the changes? We certainly want to record who has made the changes. This is particularly important in the collaborative ontology development environment. The expertise and priority of ontology developers are key factors in evaluating the quality of changes.

When did the changes happen? Apparently, we also want to know when the changes were made. In our running example, changes occurred between version $\mathcal{O}_i$ and $\mathcal{O}_{i+1}$. The version numbers of an evolving ontology can be mapped to some real world timestamps.

Were there any more supplementary changes? Since an OWL ontology is a logical theory, the changes we made in one part of the ontology may have logical consequences on the rest of the ontology. In order to maintain the consistency of the ontology and repair any inconsistencies, we may end up introducing more changes. In our running example, Bob was trying to add a new class TA by stating the axiom Ax8 ($TA \sqsubseteq Student \sqcap Employee$), which is contradictory to the axiom Ax2 ($Student \sqsubseteq \neg Employee$). To fix the inconsistency, he removed the axiom Ax2 and associated annotation Ax3 from the ontology $\mathcal{O}_{i+1}$.

What was the rationale behind the changes? Ideally, the reason behind any of the changes we made should be recorded. This may include the intentions to model the changes in the domain or express our knowledge in a better way.

3.2.2 Access Control

In order to support collaborative ontology development and guarantee the quality of an ontology, some access control restrictions to different parts of an ontology should be imposed such that they can be modified by the developers with proper access privileges and expertise. As a result, the ontology evolution management system should also incorporate fine-grained access control model. Furthermore, if an ontology is designed to provide selective sharing or reasoning service, knowledge hiding is essential due to privacy and security concerns.

In our running example, the ontology was originally developed by Alice. If Alice did not want Bob to remove

Table 1: An Evolving Ontology of Academic Department

$\mathcal{O}_i$ OWL Functional-style Syntax	Description Logic Syntax
TBox:	
Ax1: SubClassOf(Faculty Employee)	$Faculty \sqsubseteq Employee$
Ax2: DisjointClasses(Student Employee)	$Student \sqsubseteq \neg Employee$
Ax3: EntityAnnotationAxiom(OWLClass(Student) Comment(Student is not an employee))	
Ax4: SubClassOf(AssistantProfessor Faculty)	$AssistantProfessor \sqsubseteq Faculty$
Ax5: SubClassOf(AssociateProfessor Faculty)	$AssociateProfessor \sqsubseteq Faculty$
Ax6: ObjectPropertyDomain(hasSupervisor Student)	$\top \sqsubseteq \forall hasSupervisor^{-}.Student$
Ax7: ObjectPropertyRange(hasSupervisor Faculty)	$\top \sqsubseteq \forall hasSupervisor.Faculty$
ABox:	
As1: ClassAssertion(Mary AssistantProfessor)	$Mary : AssistantProfessor$
As2: ClassAssertion(John Student)	$John : Student$
As3: ObjectPropertyAssertion(hasSupervisor John Mary)	$(John, Mary) : hasSupervisor$
$\mathcal{O}_{i+1}$ OWL Functional-style Syntax	Description Logic Syntax
TBox:	
Ax1: SubClassOf(Faculty Employee)	$Faculty \sqsubseteq Employee$
Ax4: SubClassOf(AssistantProfessor Faculty)	$AssistantProfessor \sqsubseteq Faculty$
Ax5: SubClassOf(AssociateProfessor Faculty)	$AssociateProfessor \sqsubseteq Faculty$
Ax6: ObjectPropertyDomain(hasSupervisor Student)	$\top \sqsubseteq \forall hasSupervisor^{-}.Student$
Ax7: ObjectPropertyRange(hasSupervisor Faculty)	$\top \sqsubseteq \forall hasSupervisor.Faculty$
Ax8: SubClassOf(TA ObjectIntersectionOf(Student Employee))	$TA \sqsubseteq Student \sqcap Employee$
Ax9: EntityAnnotationAxiom(OWLClass(TA) Comment(TA is both student and employee))	
ABox:	
As3: ObjectPropertyAssertion(hasSupervisor John Mary)	$(John, Mary) : hasSupervisor$
As4: ClassAssertion(Mary AssociateProfessor)	$Mary : AssociateProfessor$
As5: ClassAssertion(John TA)	$John : TA$

any class or property definitional axioms related to the class $Faculty$, i.e. Ax1, Ax4, Ax5, Ax6, Ax7, she could achieve this by setting the permissions on those axioms to be only writable by herself. This kind of fine-grained access control is impossible if the smallest unit an ontology evolution management system can handle is the whole ontology.

3.2.3 Inference

One of the important tasks of ontology management is to support logic-based automated reasoning and querying services. The ultimate goal of maintaining an evolving and consistent ontology is to use the ontology and its automated reasoning services to derive answers to queries asked by the agents on the Semantic Web. To facilitate query evaluation, a set of inferred axioms can be precomputed by a DL reasoner and materialized with the explicitly asserted axioms.

In our running example, Ax8 implies $TA \sqsubseteq Student$ and $TA \sqsubseteq Employee$. We can do the reasoning offline and save these two inferred axioms with the rest of axioms. This eliminates the run-time reasoning during query answering process. To support offline reasoning, the ontology management system then needs to distinguish whether the changes are made to the explicit model or to the inferred model, and have effective methods for handling these changes.

3.3 General Approach

Our approach associates each evolving ontology with an evolutionary log, which keeps track of the lifeline of each axiom or annotation in the ontology. Each version of the evolving ontology consists a set of active axioms and annotations at that time. The schematic representation of the evolutionary log for our running example is shown in Figure 1. Unlike most other existing research, our view of ontology evolution focuses on individual axiom or annotation, not the ontology as a whole.

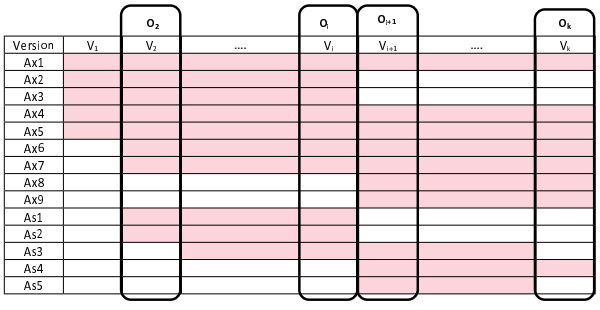

Figure 1: Schematic View of Evolutionary Log

The *Timestamp* model explicitly labels time-varying

information with a time. The *Snapshot* model associates each state of the domain knowledge with a time. In our axiom-centric approach, we break down the evolution of ontology as the evolution of each individual axiom or annotation. We capture its history by labeling it with a pair of timestamps representing its creation and retirement.

In the interval-based temporal domain, possible positions of two intervals on a linear time line can be characterized by Allen's interval relations [1]. In the point-based temporal domain, flow of time is represented as a sequence of discrete and linearly ordered time points. In this paper, the evolution of axioms or annotations fall into the point-based temporal domain, which is a finite sequence of linearly ordered time points correspond to the versions of an evolving ontology.

For an axiom, the *Valid-time* denotes the time when the facts stated by the axiom are true with respect to the modeled domain. The *Transaction-time* denotes the time when an axiom is stated in the ontology. We use *Transaction-time* to record the time when an axiom or annotation is added to or removed from the ontology.

When an axiom is initially created, a data structure containing metadata about it is also created. We call this data structure "anode" as motivated by the notion of "inode" used to maintain the information about files in the Unix/Linux file system. The anodes of an axiom store metadata such as user and group ownership, permissions, timestamps of creation and retirement, who created it, and who made it retired etc.

3.4 Formal Model

We now present a formal model for an evolving ontology based on the notions of axiom log and evolutionary log.

Definition 1 (Ontology) *An OWL 2 ontology $\mathcal{O}$ contains a set of axioms and annotations. The axioms can be declarations, axioms about classes, axioms about object or data properties, and assertions (sometimes also called facts). Annotations are made like axioms in order to simplify the structural specification in OWL 2 [14].*

Definition 2 (Lifeline). *The life line of an evolving ontology $\mathcal{O}$ is represented as a finite sequence of transaction times $T \subseteq \mathcal{N}$, where $\mathcal{N}$ is linearly ordered set of $natural\ numbers$ that correspond to the version numbers of an evolving ontology.*

The history of each axiom in the evolving ontology is tracked by an axiom log that contains the axiom and associated metadata as defined below.

Definition 3 (Axiom Log). *An axiom log AL(α) for an axiom or annotation $\alpha \in \mathcal{O}$ is a pair of $\langle \alpha, anode \rangle$, anode is a tuple of $\langle$**id**, **type**, **uid**, **gid**, **acl**, **ctime**, **mtime**, **luid**, **comment**$\rangle$, where **id** is an unique ID for the anode; **type** indicates whether axiom α is explicitly asserted axiom* (0) *or inferred axiom* (1); ***uid*** *is the id for the owner of α;* ***gid*** *is the id for the group owner of α;* ***acl*** *determines who can read α, and who can write on the anode of α such that eventually retires it;* ***ctime*** *tells when α and its anode was created;* ***mtime*** *tells when the anode of α was last modified;* ***luid*** *tells who was the last one modified the anode;* ***comment*** *is used as a short note.*

Note, α itself can not be modified, only its anode can be modified. If α is to be modified, it will be a different axiom or annotation. The pair of $[ctime, mtime]$, where $ctime, mtime \in T$ are the start and end time of the axiom log. For example, $[i, current]$ indicates that the axiom log was created at time i and hasn't ended yet. "$current$" is a special time point indicates the current time or current version of an evolving ontology. When α was retired at time j, its $mtime$ will be changed from "$current$" to j. We do not reuse the retired axiom log, if the same axiom or annotation is introduced again, it will be assigned a new id and starts a new axiom log. α may have multiple axiom logs during its evolution. We use α and its anode to uniquely identify an axiom log. Similar approach has been used in the Gene Ontology project [5], where ids for the GO terms are not deleted from the ontology, they are only marked as obsolete.

Definition 4 (Access Control List). *For a given axiom or annotation α, the permissions of specific users or groups of users can be administrated by an access control list (ACL). The list specifies who is allowed to access α and its anode, and what operations are allowed.*

We use three distinct categories: $user$, $group$, and $others$ to managing the permissions. The owner of α comprises the $user$ category. Permissions assigned to the $user$ category are only applicable to that user. α can also be assigned to a $group$, which comprises its $group$ category. Permissions assigned to the $group$ category are only applicable to the members of that group except the owner. Users not in both categories comprise the $others$ category. There are two basic permissions that can be applied to α: 1) $read$ permission, which grants user the permission to read α. 2) $write$ permission, which grants user the permission to modify the anode metadata. Note that the $write$ permission implies $read$ permission.

The schema proposed here is just one way to provide fine-grained access control to managing ontological artifacts. It is possible to impose other kinds of access control mechanisms.

Definition 5 (Evolutionary Log). *For a given evolving ontology $\mathcal{O}$, we define the evolutionary log EL as a pair of $\langle \mathcal{O}, \mathcal{L} \rangle$, where $\mathcal{O}$ is associated evolving ontology $\mathcal{O}$, $\mathcal{L} = \bigcup_{\forall \alpha \in \mathcal{O}}$ AL(α).*

An evolutionary log contains the set of axiom logs for each axiom or annotation that ever appeared in the evolv-

Table 2: An Evolutionary Log of Academic Department Ontology

Axiom	anode
SubClassOf(Faculty Employee)	$\{Ax1, 0, 10, 30,$ rwr-r-, $1, current, 10, ''\}$
DisjointClasses(Student Employee)	$\{Ax2, 0, 10, 30,$ rwrwr-, $1, i, 20, ''\}$
EntityAnnotationAxiom(OWLClass(Student) Comment(Student is not an employee))	$\{Ax3, 0, 10, 30,$ rwrwr-, $1, i, 20, ''\}$
SubClassOf(AssistantProfessor Faculty)	$\{Ax4, 0, 10, 30,$ rwr-r-, $1, current, 10, ''\}$
SubClassOf(AssociateProfessor Faculty)	$\{Ax5, 0, 10, 30,$ rwr-r-, $1, current, 10, ''\}$
ObjectPropertyDomain(hasSupervisor Student)	$\{Ax6, 0, 10, 30,$ rwr-r-, $2, current, 10, ''\}$
ObjectPropertyRange(hasSupervisor Faculty)	$\{Ax7, 0, 10, 30,$ rwr-r-, $2, current, 10, ''\}$
SubClassOf(TA ObjectIntersectionOf (Student Employee))	$\{Ax8, 0, 20, 30,$ rwrwr-, $i+1, current, 20, ''\}$
EntityAnnotationAxiom(OWLClass(TA) Comment(TA is both student and employee))	$\{Ax9, 0, 20, 30,$ rwrwr-, $i+1, current, 20, ''\}$
ClassAssertion(Mary AssistantProfessor)	$\{As1, 0, 10, 30,$ rwrwr-, $2, i, 20, ''\}$
ClassAssertion(John Student)	$\{As2, 0, 10, 30,$ rwrwr-, $2, i, 20, ''\}$
ObjectPropertyAssertion(hasSupervisor John Mary)	$\{As3, 0, 10, 30,$ rwrwr-, $3, k-1, 10, ''\}$
ClassAssertion(Mary AssociateProfessor)	$\{As4, 0, 20, 30,$ rwrwr-, $i+1, current, 20, ''\}$
ClassAssertion(John TA)	$\{As5, 0, 20, 30,$ rwrwr-, $i+1, k-1, 20, ''\}$

ing ontology. Let the ***uid*** of "Alice" be 10, the ***uid*** of "Bob" be 20, the ***gid*** of the group "Alice" and "Bob" are in be 30, the evolutionary log of our running example is shown as in Table 2.

Definition 6 (Version of an Ontology). *A version of an evolving ontology O at $t \in T$, $VO(\mathcal{O}, t)$, is defined as a finite set of axioms and annotations A, $\forall \alpha \in A$, AL(α) $\in$ EL, AL(α).anode.ctime $\leq t \leq$ AL(α).anode.mtime.*

4 Evolutionary Log Operations

4.1 Operations on Axioms

Algorithm 1 is used to add a new axiom or annotation to the evolutionary log EL of an evolving ontology $\mathcal{O}$ at time t. Note, if an axiom or annotation α is currently active in the evolutionary log EL, we skip this step. But if it was in the EL and has been retired, we then need to add α as a new axiom log to the EL. This action may make the ontology logically inconsistent, so one of the supplementary actions is checking the consistency of the updated ontology and resolving any inconsistencies. The routine repairConsistency($\mathcal{O}$) may end up adding or removing another set of axioms. We refer the readers to [15] for the techniques and algorithms for repairing ontology inconsistencies. Algorithm 3, retrieveOntology(EL, t), is used to retrieve a version of ontology at a particular time and is defined later.

Once an axiom or annotation is created, it will be valid in the evolving ontology until it is retired. As mentioned before, axiom or annotation itself is not allowed to be modified after its creation, only the anode associated with it can be modified. To modify the anode, users must have the write permission. The $type$, uid and $ctime$ can not be changed once they were created. Thus, the users can only modify the fields of acl, $mtime$, $luid$ and $comment$. If an axiom or annotation is still active in the evolutionary log EL, $mtime$ will be marked as "$current$", so we don't need to update this field every time we change other axioms or annotations in the evolving ontology.

Algorithm 2 is used to retire an axiom or annotation from the evolutionary log EL of an ontology $\mathcal{O}$ at time t. Note, in order to maintain the complete history of all the axioms ever occurred in the ontology, we don't remove retired axiom logs from the evolutionary log. The pair

Algorithm 1 Add an axiom or annotation at time t
Input: an axiom or annotation α, evolutionary log EL, version number t, current editor user id $cuid$, current editor group id $cgid$
Output: updated evolutionary log EL

if $AL(\alpha) \in EL$ and $AL(\alpha).anode.mtime ==$ "current" **then**
  return
otherwise
  $AL(\alpha).\alpha := \alpha$
  $AL(\alpha).anode.id :=$ "new id"
  $AL(\alpha).anode.type := 0$
  $AL(\alpha).anode.uid := cuid$
  $AL(\alpha).anode.gid := cgid$
  $AL(\alpha).anode.acl :=$ "rwrwr-"
  $AL(\alpha).anode.ctime := t$
  $AL(\alpha).anode.mtime :=$ "current"
  $AL(\alpha).anode.luid := cuid$
  $EL := EL \bigcup AL(\alpha)$
  $\mathcal{O} :=$ retrieveOntology(EL, t)
  if isConsistent($\mathcal{O}$) **then**
    return EL
  otherwise
    repeat
      repairConsistency($\mathcal{O}$)
    until isConsistent($\mathcal{O}$)
    return EL
  end if
end if

Algorithm 2 Retire an axiom or annotation at time t
Input: an axiom or annotation α, evolutionary log EL, version number t, current editor user id $cuid$
Output: updated evolutionary log EL

if allowToChangeAnode($AL(\alpha)$, $cuid$) = = false **then**
  return
Otherwise
  if $AL(\alpha) \in EL$ and $AL(\alpha).anode.mtime$!= "current" **then**
    return
  otherwise
    $AL(\alpha).anode.mtime := t$
    $AL(\alpha).anode.luid := cuid$
    return EL
  end if
end if

$[ctime, mtime]$ in the anode of a retired axiom tells us the period it has been active in the evolving ontology.

4.2 Retrieve Version of an Ontology

Algorithm 3 is used to select a specific version of an evolving ontology $\mathcal{O}$ at time $t \in T$. The algorithm is quite straightforward, it basically takes the union of all the axioms and annotations in the evolutionary log that were active at that particular time.

Algorithm 3 Retrieve a specific version of an evolving ontology $\mathcal{O}$ form evolutionary log EL
Input: version number t, evolutionary log EL
Output: a version Ω of an evolving ontology $\mathcal{O}$

$\Omega := \emptyset$
for all $AL(\alpha) \in EL$ **do**
  if $AL(\alpha).anode.ctime \leq t \leq AL(\alpha).anode.mtime$ **then**
    $\Omega := \Omega \bigcup \{\alpha\}$
  end if
end for
return Ω

5 Implementation Scenarios

5.1 Rich Axiom Annotations

The anode information associated with an axiom or annotation α is essentially the annotations for α. Therefore, if we consider our evolutionary log as an ontology and being represented as an OWL document, the annotations then can be embedded with the axioms or annotations in the ontology. In current OWL DL specification (OWL 1.0) [16], annotations can only be used for "entities" (i.e., classes, properties, individuals) via a set of AnnotationProperties. One of the design goals of OWL 2 is to support annotations on axioms. We can not only annotate a class c, but also annotate any axiom references the class c. For the axiom SubClassOf(Faculty Employee) in our evolutionary log, we can add its anode metadata annotations like the follows:

```
SubClassOf(
        Annotation( ( anode:id ) "Ax1" )
        Annotation( ( anode:type ) 0 )
        Annotation( ( anode:uid ) 10 )
        Annotation( ( anode:gid ) 30 )
        Annotation( ( anode:acl ) "rwrwr-" )
        Annotation( ( anode:ctime ) 1 )
        Annotation( ( anode:mtime ) "current" )
        Annotation( ( anode:luid ) 10 )
        Annotation( ( rdfs:comment ) "first axiom" )
        Faculty Employee)
```

Apparently, using annotations on axioms enriches them with the critical metadata that are important for any collaborative ontology development. The drawback of this approach is that as every axiom is described with some extra annotations, the size of our evolutionary log will increase drastically, and become very verbose and cumbersome to handle. One of the persistent storage for an OWL ontology is an OWL document stored as a plain text file in the file system. Whenever we need to access the ontology model, the OWL document is read, parsed and loaded into memory. These procedures seem to work fine when the ontology is in a reasonable size. But when the size of an ontology exceeds the physical capacity of our machine, this approach will doom to fail. It is very inefficient to read, parse the entire large ontology when we are only interested in a small fragment of the ontology. Therefore, we need alternative solutions.

5.2 Temporal Database

Both scalability and efficiency are keys to the success of large-scale OWL ontologies. Two well-known ontology management systems Jena [4] and Sesame [3] have provided persistent storage for ontologies using relational databases as the back-end. But their storage models are based on RDF triple model, which is inefficient for very large ontologies. They do not take into consideration the semantics of OWL ontology and its rich logical constructors and descriptions. Many useful query operations will involve the join of very large triple tables. Furthermore, they do not maintain detailed metadata related to the changes. We propose to implement our ontology evolution approach using relational database management systems with well-designed schema that capture the structural specifications of OWL 2 ontology. The evolutionary log is actually a temporal database managing the changes of an evolving ontology and related metadata. The benefit of this approach is that we can use existing database technology to provide a scalable ontology evolution management framework. On top of that, we can build efficient DL-based reasoning and querying services.

6 Related Work

Eder and Koncilla [6] propose using temporally labeled directed graph to represent changing knowledge. The concepts and relations between them are explicitly labeled with their valid times to enable identifying ontologies that were valid in the past. In this paper, we focus on axiom-centric OWL ontology model, which is more expressive. We use transaction time instead of valid time for the lifeline of an axiom or annotation.

Marwaha and Bedi [13] propose combining the temporal frame and slot versioning to create temporally tagged ontologies with embedded version information to enable applications to work with multiple versions of an ontology. Their approach can only keep track of the evolution of concepts in an ontology, not the axioms or annotations. They do not take into consideration the permissions associated with the axioms or annotations, which is the basic requirement for collaborative ontology engineering.

7 Summary and Future Work

In this paper, we use timestamp temporal model and point based transaction time to describe the evolution of an axiom or annotation through its lifetime. i.e from its creation to ultimate retirement. Instead of maintaining different versions of an evolving ontology, we use an evolutionary log to track the lifelines of axioms and annotations by associating them with a set of metadata (e.g. ownership, access privileges, time stamps of changes etc.).

We present the algorithms for manipulating an evolutionary log. We argue that document-centric and rich axiom annotation based approach is not suitable for implementing our ontology evolution framework. We suggest developing an ontology evolution management system using temporal database for the sake of scalability and efficiency.

Building a temporal database model for our ontology evolution framework is under way. We also plan to do evaluation using OWL ontology benchmark data set [12].

References

[1] James F. Allen. “Reasoning about Plans”. Temporal Reasoning and Planning, p. 2-68. Morgan Kaufmann, 1991.

[2] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi and Peter F. Patel-Schneider. “The Description Logic Handbook: Theory, Implementation and Application”. Cambridge University Press, 2003.

[3] Jeen Broekstra, Arjohn Kampman and Frank van Harmelen. “Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema”. In The Semantic Web - ISWC 2002, LNCS, Vol. 2342, p. 54–68. 2002.

[4] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne and Kevin Wilkinson. “Jena: Implementing the Semantic Web Recommendations”. In Proceedings of the 13th international World Wide Web conference, p. 74-83, New York, NY, USA, 2004.

[5] Gene Ontology Consortium. “The Gene Ontology (GO) Database and Informatics resource”. Nucleic Acids Research, 32(Database-Issue): 258-261, 2004.

[6] Johann Eder and Christian Koncilia. “Modelling Changes in Ontologies”. In OTM Workshops, p. 662-673, 2004.

[7] Jennifer Golbeck, Gilberto Fragoso, Frank W. Hartel, James A. Hendler, Jim Oberthaler and Bijan Parsia. “The National Cancer Institutes Thesaurus and Ontology”. Journal of Web Semantics, 1(1):75-80, 2003.

[8] Ian Horrocks, Oliver Kutz and Ulrike Sattler. “The Even More Irresistible $\mathcal{SROIQ}$”. In KR 2006, p. 57-67. AAAI Press, 2006.

[9] Ian Horrocks and Ulrike Sattler. “A Tableaux Decision Procedure for $\mathcal{SHOIQ}$”. In Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI), p. 448-453, 2005.

[10] Ian Horrocks and Peter F. Patel-Schneider. “Reducing OWL Entailment to Description Logic Satisfiability”. Journal of Web Semantics, 1(4): 345-357, 2004.

[11] Ian Horrocks and Peter F. Patel-Schneider, and F. V. Harmelen. “From $\mathcal{SHIQ}$ and RDF to OWL: The Making of a Web Ontology Language”. Journal of Web Semantics, 1(1): 7-26, 2003.

[12] Li Ma, Yang Yang, Zhaoming Qiu, GuoTong Xie, Yue Pan and Shengping Liu. “Towards a Complete OWL Ontology Benchmark”. ESWC 2006: p. 125-139, 2006.

[13] Sudeep Marwaha and Punam Bedi. “Temporal Extension of OWL Ontologies”. International Journal of Information Technology, 4(1): 53-60, 2007.

[14] Boris Motik, Peter F. Patel-Schneider, Ian Horrocks. “OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax”. W3C Working Draft 11 April 2008, http://www.w3.org/TR/owl2-syntax/.

[15] Stefan Schlobach, Zhisheng Huang, Ronald Cornet and Frank van Harmelen. “Debugging Incoherent Terminologies”. Journal of Automated Reasoning, 39(3): 317-349, 2007.

[16] Michael K. Smith, Chris Welty, Deborah L. McGuinne. “OWL Web Ontology Language Guide”. http://www.w3.org/TR/owl-guide/, 10 February 2004.