International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:12, 2008

Temporal Extension to OWL Ontologies

Sudeep Marwaha, Punam Bedi

***Abstract*—**Ontologies play an important role in semantic web applications and are often developed by different groups and continues to evolve over time. The knowledge in ontologies changes very rapidly that make the applications outdated if they continue to use old versions or unstable if they jump to new versions. Temporal frames using frame versioning and slot versioning are used to take care of dynamic nature of the ontologies. The paper proposes new tags and restructured OWL format enabling the applications to work with the old or new version of ontologies. Gene Ontology, a very dynamic ontology, has been used as a case study to explain the OWL Ontology with Temporal Tags.

***Keywords*—**Frame and slot Versioning, OWL, Ontology Versioning, Semantic Web.

I. Introduction

WITH the increasing popularity of the semantic web among the researchers, the semantic web applications are being conceived in the variety of domains in different streams of the society. For the development of semantic web applications, knowledge representation in the form of ontologies becomes the core issue of research. Researchers worked together to create standards, tools and languages to speed up the development of ontologies. A single ontology is often developed by different groups, may be separated geographically and continue to evolve over time. Due to the distributed and dynamic nature of the web, the knowledge component of these applications changes more rapidly than the other components of the applications. Ontologies being the knowledge representation technique in semantic web applications need to handle this change. Research in ontology engineering nowadays is more focused on ontology management problems rather than ontology creation and ontology formalism. Noy [4] [5] [10] has listed and classified the core issues in the ontology management. These include maintain libraries of ontologies, import and reuse ontologies, translate ontologies from one formalism to another, provide support for ontology versioning, specify transformation rules between different ontologies and versions of the same ontology, merge ontologies, align and map between ontologies, extract semantically independent parts of the ontology, support inference across multiple ontologies, and support query across multiple ontologies.

Manuscript received November 9, 2006.

Sudeep Marwaha, is a research scholar in the Department of Computer Science, University of Delhi and scientist in the Division of Computer Applications, Indian Agricultural Statistics Research Institute, Delhi, India. (e-mail: sudeep_iasri@yahoo.com).

Dr. Punam Bedi is head and reader in the Department of Computer Science, University of Delhi, Delhi. (phone: 91-11-27667591; fax: 91-11-27662553; e-mail: pbedi@ cs.du.ac.in)

In [9] ontology has been defined with the versioning and backward compatibility by keeping each version in different file. In a business world, organizations tend to keep the applications running for years once they reach to stable version as they have invested huge money in building and deploying them. Web is dynamic, huge and a vast resource with millions of applications and people interacting with each other. Keeping multiple versions of a single ontology simultaneously and allowing the dependent applications to choose from the version they depend upon causes some serious problems. First, there will be many of the version files existing at a single moment and they will grow in number very rapidly. Second, it creates confusion to choose from the different versions for the dependent applications. Third, someone has to put resources in terms of money, space, time and efforts to maintain all these versions. Fourth, dependent application requires at least some programming effort each time the application is synchronized with the latest version of ontology. Synchronization with latest version is needed as the knowledgebase applications can not provide good results if they do not update their knowledgebase.

OntoView [8] includes structural comparison of source ontologies. The system identifies different types of changes between versions of the same concept and allows its users to enlist a conceptual description of how the concept has changed. However, if the concept name has changed, OntoView does not attempt to determine that the concept with the new name is just an image of old concept.

This paper focuses on the problems related with ontology versioning, ontology aligning and mapping and specifying transformation rules between different versions of the same ontology. These problems need to address issues such as identifying ontology versions, specifying explicitly logs of changes between versions and determining a set of additional ontology changes that each change specified by the user incurs. Identifying an ontology version is required as one needs to fetch a particular version that suit the requirements of an application or to analyze the differences between a set of versions against the novelty of concepts or usage of different versions. In our approach, we propose that identification of different versions will be carried out very easily as all these versions are stored in a single file and proposed changes in RDF attributes will ensure that at any point of time the particular OWL ontology version will be generated on the fly for the desired user or application. This is possible by deploying a web service that extracts a particular version from the ontology with temporal extension rather than keeping a static OWL file at the server. The service sends the extracted ontology to the requesting semantic web application. Given the decentralized nature of ontology development, logs of changes may not always be available so it becomes all the more difficult to align and map different concepts or determining additional ontology changes that each change incurs in the ontology. Introduction of new tags and the modified structure of the OWL file tackle this problem very effectively. Comparing our approach with the existing systems of managing ontology versioning such as OntoView, we found our approach becomes very easy and efficient as it introduces the changes in the RDF attributes that forces the ontology developers to take care of these issues while developing and updating the ontology rather than the identifying changes after the development and release of each version.

Semantic web applications require a framework [5] that is independent of the ontology versioning and do not need previous ontology versions to exist simultaneously along with the latest version.

We discuss our approach with the very popular Gene Ontology [2] as a case study. In the current research environment, where new genome sequences are being rapidly generated, and where comparative genome analysis requires the integration of data from multiple sources, it is especially germane to provide rigorous ontologies that can be shared by the community. We have chosen the Gene Ontology because it is perhaps the most dynamic ontology with over few thousands of new additions each year. The Gene Ontology consortium releases a new version of the ontology monthly. Daily versions in OWL format can also be downloaded which are generated from the Concurrent Versions System (CVS).

The rest of this paper is organized as follows: section 2 discusses the knowledge representation using frames, temporal frames and slot versioning. Section 3 describes knowledge representation for the web using Ontologies, and versioning of OWL ontologies. Section 4 deals with the problem of ontology versioning and our approach of using temporal tagged ontologies by extending OWL to solve it. Ontology integration is another key issue in solving queries that require merging of knowledge from various sources. Section 5 details the integration of ontologies with temporal extension. A case study of Gene Ontology is given in section 6 in which transformation of existing OWL Gene Ontology, to the temporal tagged OWL Gene Ontology is described. Section 7 concludes the paper with merits of the approach.

II. Knowledge Representation Using Frames

A frame is basically a structure for holding various types of knowledge. Frames are given names, with the presumption that the knowledge contained within a particular frame is in some way interrelated. Many authors and speakers may refer to frame-like structures as units, objects, concepts, schemas or entities. Frames are structured ways of representing descriptive information & they provide a natural mapping for the kind of knowledge that is centred around one concept or object. The organization allows efficient searching because there is immediate access to relevant information. Frame structure provides a natural method for representing hierarchies of information, thus allowing the inheritance of values. Although frames were conceived independently of the object-oriented paradigm they are in fact consistent with it, and provide an excellent demonstration of their power. Indeed frames are capable of representing both specific and general knowledge, and are capable of accommodating both descriptive and prescriptive computations.

Frame Based Systems have many advantages like immediate access to relevant information, easy to include default information and detect missing values etc. Frames can be used easily with Production rules, thereby facilitating partitioning, indexing and organizing production rules of a system. The implicit hierarchy available in frame taxonomies also permits hierarchical segmentation of rules. All these merits of the frame allow it to be the basis for knowledge representation technique called Ontology for the web.

A. Temporal Frame Representation

The inherent capability of Frame Based System for structuring knowledge has been the motivation for enhancing its capabilities by adding various dimensions to it. In some situations, it is not appropriate to discard the old information. The temporal (time) dimension [3] is added keeping these situations in mind. Knowledge in these temporal frame systems has the period of validity, which is attached to either frames or its slots or both. If the period of validity is attached to frames, it is called frame versioning and if attached to its slots, it is called slot versioning. The Tframe system [1] is based on slot versioning in contrast to frame versioning in order to achieve efficiency in terms of time and space. The system based on slot versioning is better because frame versioning approach has high degree of redundancy owing to duplication of the entire frame, especially when the changed portion is relatively small compared with the unchanged portion.

III. Knowledge Representation Using Ontologies

Ontologies are structures or models of known knowledge. Ontology is a partial and explicit account of a conceptualization. The degree of specification of the conceptualization, which underlies the language used by a particular knowledge base, varies in dependence of our purposes. An ontological commitment is thus a partial semantic account of the intended conceptualization. In OWL ontology, concepts are arranged in hierarchical format with each concept is represented by a node in the hierarchy. An OWL class having various properties and relationships with the other classes represents each node. Relating it with the frames and slots, a class in ontology is based on the frame and its properties are slots of the frames. The relationships among different classes or frames are established by referencing related classes or instances of classes in slots or properties.

A. *Resource Description Framework (RDF)*

Resource Description Framework (RDF) [7] is the web metadata language. It is used to describe information about web resources. The semantics associated with this information enables web applications interoperability. An RDF model that describes some web resources is also called an RDF instance. An RDF schema can be used to define application specific vocabularies. This schema can be associated with an RDF instance in order to validate the instance. Both RDF instance and RDF schema are RDF models. RDF is designed to represent information in a minimally constraining, flexible way. It can be used in isolated applications, where individually designed formats might be more direct and easily understood, but RDF's generality offers greater value from sharing. The value of information thus increases as it becomes accessible to more applications across the entire Internet.

B. *Ontology Web Language (OWL)*

The OWL, Web Ontology Language is a language for defining and instantiating Web Ontologies. OWL is evolved from a number of technologies such as its predecessor DAML+OIL, from Description Logics, from the frames paradigm and from RDF. The formal specification of the language is on Description Logics, the surface structure of the language is based on the frames paradigm and has RDF/XML exchange syntax for upwards compatibility with RDF.

C. *Versioning of OWL Ontologies*

Version Support in OWL is very limited and can be described at two levels viz., at ontology level and at class or property level. At ontology level, owl:priorVersion property links to a previous version of the ontology being defined and can be used to track the version history of an ontology. Ontology versions may not be compatible with each other. Within an owl:Ontology element, we use the tags owl:backwardCompatibleWith and owl:incompatibleWith to indicate compatibility or the lack thereof with previous ontology versions. At class or property level, owl:versionInfo is provided. As opposed to the previous three tags, the object of owl:versionInfo is a literal and the tag can be used to annotate classes and properties in addition to ontologies. For many purposes, doing version tracking at the granularity of an entire ontology is not enough. Maintainers may wish to keep version information for classes, properties, and individuals - and even that may not be sufficient. The incremental nature of class expressions in OWL implies that one ontology may add restrictions to a (named) class defined in another ontology, and these additional restrictions themselves may require version information. OWL Full provides the expressive power to make any sort of assertion about a class, i.e. that it is an instance of another class, or that it (and not its instances) has a property and a value for that property. This framework can be used to build ontology of classes and properties for tracking version information. The OWL namespace includes two pre-defined classes that can be used for this purpose: owl:DeprecatedClass and owl:DeprecatedProperty. They are intended to indicate that the class or property will likely be changing in an incompatible manner in a forthcoming release. The versions are mostly maintained by version control systems such as CVS that are developed for keeping track of the code of traditional software.

IV. TEMPORAL EXTENSION TO OWL ONTOLOGIES

OWL class contains rdf:Id or rdf:Resource or rdf:About tag and all the properties of the class has rdf:Resource or rdf:About tag for naming and identification which in turn becomes their globally unique identification when seen in combination with the URI of the XML document. Relating the OWL with frames and slots, we introduce the concept of frame and slot versioning in OWL. The existing standard of OWL contains only an annotation property for capturing the version of the ontology. When the value of a property of a class has changed or name of the property has changed between two versions, the slot versioning is used to capture the change. According to slot versioning, only the version of changed property is created and inserted above the existing latest version in the same OWL file. When the class name or the intrinsic attribute of the class has changed then we use the frame versioning and the whole is inserted above the existing version in the same OWL file. The unique identity of different concepts are maintained by modifying the format of rdf:Id, rdf:About and rdf:Resource value. The values of these attributes are appended to the corresponding version number separated by the delimiter @. We propose to introduce two new tags in the underlying RDF viz. rdf:Validity and rdf:Timestamp. These two attributes will be used along with the rdf:Id and rdf:Resource to identify the changes that occur in subsequent versions of the ontology. The rdf:Validity attribute will have one of the four possible values viz. Always True, Latest, Past or Deleted. The rdf:Timestamp attribute will be used to represent the time when the concept has changed. It will be of the format YYYY-MM-DDThh:mm:ssTZD as specified in ISO 8601. These two attributes will be used to identify the same concepts of the different versions of the ontology. At any point of time the concepts of latest version of the ontology will have the value of rdf:Validity set to Always True or Latest and the value of rdf:Timestamp attribute is set to release date and time. If rdf:Validity is Always True then it means the concept is marked as universal truth and can not be changed in new versions. The Always True value is for the basic concepts, changing which the ontology itself become so much different than the previous version that it will be better to term it as new ontology. If the concept has value Latest then that it will be treated as the concept of the latest version of the ontology. Latest value should be unique among the different versions of the same concept. The value of rdf:Timestamp attribute will be used to extract a particular version on request of the user/application. The changed concept will be added as a new line having all other attribute same as the older one except for the rdf:Id / rdf:Resource, rdf:Timestamp, rdf:Validity and the updated information. The rdf:Timestamp will contain a new date and time while rdf:Validity will have Latest as its value. The rdf:Resource and rdf:Id will be appended by a @ and the number of the version. This is necessary to maintain the uniqueness of the resource or id. The same name before the @ signifies that the new added concept is essentially the next version of the existing concept. The ontology will still keep all the concepts including the latest as well as the past. The new concepts in the ontology can be added by choosing the appropriate location with the Latest as the value of rdf:Validity and current date and time as the value for rdf:Timestamp. The concepts that are obsolete and need to be removed can be deleted from the latest version only by changing the value of the rdf:Validity as Deleted. The concept will remain there in the older version with its existing values of rdf:Validity, rdf:Timestamp and rdf:Resource and rdf:Id tags. The non deletion of the deleted concept of the latest version in the older versions will keep the older versions compatible with the applications that are running on it. The concepts that are added after the latest release will contain the date and time at which they are incorporated in the ontology. This value will be modified to the release date and time when the next version will be released. By that time the concepts will be present in the ontology as valid changes for the next version.

The introduced tags rdf:Validity and rdf:Timestamp and changed format of the ID and Resource tags are required as it becomes very efficient to extract the older versions from the master ontology. The OWL ontologies with temporal tags can be created from the OWL ontologies using an algorithm given below.

```
Algorithm: Creation of OWL Ontologies with temporal tags
Phase-I  : Transforming OWL Ontology to OWL
           Ontology with Temporal Tags
Input    : OWL Ontology
    Read an existing OWL ontology
    Insert the rdf:Validity and rdf:Timestamp tags in every
    rdf:Resource or rdf:Id statement.
    Value of rdf:validity is set to "Latest" and value of
    rdf:Timestamp is set to current date and time.
    Get the version number from the user or search it in
    version annotation property and save it in
    Current_Version.
    Append the Current_Version after the value of rdf:Id
    and rdf:Resource tags with @ as delimiter.
    Output the ontology as a new file.
```

Updating in the ontology concepts mean change in the class names, slot names, slot values, addition of classes, deletion of classes, addition of properties, deletion of classes, deletion of properties, change in the value of properties. Change in classes position in the hierarchy with respect to the other classes will be treated independently by addition of class at new position and deletion of class at the present position. The statement in this algorithm means the block of statements if it is a container statement like class and a single line statement if it is just a property. The rdfs:Comment, rdfs:Label and other tags having neither rdf:Id nor rdf:Resource attribute are taken from the latest version and older values are deleted as they don't affect the semantics of the ontology.

```
Phase-II : Updating the Temporal Tagged OWL Ontologies
Input: Temporal tagged OWL Ontology from phase-I
For any subsequent change in the concept in Ontology do
    Identify the concept and its location that has changed
    Check the rdf:Validity tag value for each version of
    the concept
      If it is "Always True" then
        Exit and print not a valid change
      If it is "Latest" then
        If the task is to update the concept then
          Change the value of rdf:Validity to "Past"
          Insert the new statement describing the updated
          concept before the current statement
          Set rdf:Validity of new statement to "Latest"
          and rdf:Timestamp set to current date and time.
          Append the next version number after value of
          the rdf:Id or rdf:Resource or rdf:About of new
          statement with @ as delimiter.
          Exit
        If the task is to add a new concept then
          Insert the new concept statement/statements at
          the desired location with its rdf:Id or
          rdf:Resource or rdf:About appended with the
          new version number.
          Set the rdf:Validity to "Latest" and
          rdf:Timestamp to current date and time.
          Exit
        If the task is to delete the concept then
          Insert the new statement for deleting the
          concept before the concept to be deleted.
          Set the rdf:Validity to "Deleted" and set the
          rdf:Timestamp value in the new statement.
          Change the rdf:Validity to "Past" in the already
          present statement of the concept.
          Exit
        If it is "Past" then
          Exit
        If it is Deleted" then
          Exit
    End loop
  End Loop
```

The algorithm matches the value of the rdf:Validity tag, rdf:Timestamp and version number appended after the resource with the information given for requested version. The version can be extracted using the given version number as well as the latest version up till the given date and time. Capturing of latest knowledge in ontology with temporal extension will grow its size continuously and resulting in a huge file. We argue that the management of this huge file should not be the problem as it is to be used by the agents and meant for human consumption. Moreover, one can also adopt mechanisms to restrict its size by deleting few older versions of some concepts. The older versions can be deleted after a particular time lapsed or by fixing their maximum number or hybrid of above two approaches or by a user customized approach. The exact mechanism can be dependent upon the particular application and can be built into the web service that is used to access a particular version of the ontology. Putting the restricting mechanism out of the ontology with temporal extension makes it customizable as well as allows the applications to avoid it if the size is manageable.

V. Integration of Ontologies with Temporal Extension

In literature terms such as mapping and integration have been used in various ways by organizations according to their understandings and requirements. In Gruber's ontology definition [12], an ontology O with a specific domain model, T is defined. Thus a conceptualization $\sum$ is a pair of < C,R >, where C represents a set of concepts, and R stands for a set of relations over these concepts. A specification is a pair of < $\sum,\Psi$ > to describe that $\sum$ satisfies the axioms Ψ derived from the domain model. In the following, notation C(O) is used to annotate concepts C of the ontology O. These notations are extended to denote the temporal extension to ontology. A conceptualization, valid during a time interval k, $\sum_k$ is a pair of < C_k, R_k >, where C_k represents a set of concepts valid during time interval k, and R_k stands for a set of relations over these concepts during time interval k. A specification, valid during time interval k, is a pair of <$\sum_k,\Psi_k$> to describe that $\sum_k$ satisfies the axioms Ψ_k during k, derived from the domain model.

Below are three kinds of mutually exclusive semantic relations between existing concept classes from two different ontologies with temporal extension. We assume that O_i and O_j are in the same domain (i, j $\in$ N, where N: natural numbers). c_{ik} where $c_{ik} \in C_{ik}(O_i)$ and c_{jl} where $c_{jl} \in C_{jl}(O_j)$ are two different concepts valid during time intervals k and l respectively.

Definition (Temporal Equivalent): Two concepts are semantically equivalent at time t, if $\exists$ c_{ik}, c_{jl} , s.t. $c_{ik} \sim c_{jl}$. Namely, these two concepts: (1) have the same denotation names (e.g. labels) at time t; (2) are synonyms at time t; or (3) their attributes are same at time t, where $t \in k \cap l$.

Definition (Temporal Inclusive): Two concepts are semantically inclusive at time t ($t \in k \cap l$), if $\exists$ c_{ik}, c_{jl} , s.t. $c_{ik} \leq c_{jl}$ (e.g. c_{ik} is a kind of c_{jl}) or $c_{ik} \geq c_{jl}$ (e.g. c_{jl} is a kind of c_{ik}). Namely, the attributes of one concept are also the attributes of the other.

Definition (Temporal Disjoint): Two concepts are disjoint at time t, ($t \in k \cap l$), if $\exists$ c_{ik}, c_{jl} , s.t. $c_{ik} \cap c_{jl} = \Phi$. Namely, there is no common attribute between them.

For the purpose of ontology integration with temporal extensions, we need to consider the consistency issue of an integrated ontology. The ontology consistency is defined as follows.

Definition (Temporal Consistent): An ontology is consistent at any time t, if no sub-concepts of a particular concept c is a sub-concept of concept disjoint with concept c i.e. if $\forall c^m_{ik}$, c^n_{ik} , c^o_{ik} (c^m_{ik} , c^n_{ik}, $c^o_{ik} \in C_i(O_i)$, and $c^m_{ik} \neq c^n_{ik} \neq c^o_{ik}$ (m,n,o $\in$N), $c^m_{ik} \leq c^n_{ik}$ and $c^n_{ik} \cap c^o_{ik} = \Phi$, s.t. c^m_{ik} is not $\leq$ c^o_{ik}, $t \in k$..

Definition (Temporal Mapping): An mapping $\Re_k$ between two ontologies O_i and O_j exists at any time t, if $\exists c_{ik}$, c_{jl} ,s.t. $\Re(c_{ik}, c_{jl}) \in \{\sim, \leq, \geq\}$, $t \in k \cap l$.

Definition (Integration of Ontology with temporal extension): Reusing available source ontologies with temporal extension within a range to build a new ontology with temporal extension. The new ontology serves at a higher level in the application than that of various ontologies in ontology libraries. It is associated with temporal semantic integration.

Research on integration of ontologies has been the focus of researchers for quite a long time. In [13] integration is treated as a process and its various activities are defined. PROMPT [10] and Chimaera [14] are the tools for merging alignment and testing of large ontologies. In [15] an agent based approach is considered for integration of ontologies. A MAS is presented that integrates the given ontologies and these can be reused within the framework. In this section, we argue that integration of ontologies with temporal extension require small modification within the existing mechanisms. All the existing tools can be enhanced to incorporate the temporal extensions by considering the latest version of the concepts at the merge time. The version history of the matched concepts can be included in the derived ontology in the format described in Section IV. The mechanism adopted in [15] is investigated and the three cases described while mapping the concepts between two ontologies are considered for any possible modification. The two concepts can be semantically equivalent, inclusive or no semantic equivalence for the current two concepts but their corresponding direct ancestors are semantically equivalent. Consider two latest concepts x_{13} and y_{25} in two ontologies O_1 and O_2. The concept x_{13} is the third version of the concept x in ontology O_1 and concept y_{25} is the fifth version of the concept y in ontology O_2. Suppose, x_{13} has two sub concepts and y_{25} has three sub concepts then all these sub concepts have different version history. Consider a situation where the ontology O_2 is to be merged in O_1. The ontology in which concepts are merged i.e. O_1 is treated as primary ontology and ontology whose concepts are merged i.e. O_2 is treated as secondary ontology. After matching the concepts between O_1 and O_2, x_{13} and y_{25} matches, so these concepts are temporally and semantically equivalent at the merger time and it becomes one of the merging points of the ontologies. There can be two possibilities, firstly x_{13}, y_{25} are temporally equivalent but uses different names and secondly they use the same name. If they use different names then in the merged ontology an equivalent relation can be established between them and both concepts can maintain their version history. If x_{13} and y_{25} have a same name then only the concept of the primary ontology (x_{13} in this case) will be able to keep its version history. Version history of all other semantically disjoint concepts (i.e for above mentioned other two cases) of both the ontologies is maintained. Here, it is emphasized that only the latest version at the merge time is used for matching, but version history is maintained for all concepts except the ones that are having same name and are also temporally and semantically equivalent.

VI. Transforming OWL Gene Ontology to OWL Gene Ontology with Temporal Tags – A Case Study

The Gene Ontology (GO) [2] project is a collaborative effort to address the need for consistent descriptions of gene products in different databases. The GO collaborators are developing three structured, ontologies that describe gene products in terms of their associated biological processes, cellular components and molecular functions in a species-independent manner.

A. *Gene Ontology with Temporal Extension*

The concept of temporal tagged ontologies is applied to the Gene Ontology to make it version independent and a particular ontology version can be generated on the fly depending upon the requirement of the user or application. We applied the first phase of creation of OWL ontology using temporal tags algorithm for transforming the Gene Ontology. According to algorithm, the ontology is read, construct by construct from top to bottom and transformation happens for each construct containing either the rdf:Id and rdf:Resource tag. The rdf:Validity and rdf:Timestamp tags are introduced in them and value of rdf:Resource and rdf:Id are modified by appending version number to them. The sample output is shown in table 1 wherein first column contains original Gene Ontology and second contains the transformed one. At the end of the first phase the value of rdf:Validity is set to Latest for all the constructs and a default version number 1.0.0 is appended to the rdf:Id and rdf:Resource values. Current date and time is set for the rdf:Timestamp. The output of phase I was fed to the phase II algorithm for maintaining any subsequent updating of the concepts. Table 2 shows a temporal OWL ontology class that has changed over time. The class has got an another rdf:Resource and owl:Restriction. The addition of these causes a change in the structure of the class, due to the change in the underlying frame structure. These types of updates need frame versioning and are very frequent in the Gene Ontology as different users for the use of new applications enrich the existing classes. The updated and old classes exist simultaneously in the ontology for the seamless integration of temporal tagged OWL ontology with the existing applications.

TABLE I
GENE ONTOLOGY AND TRANSFORMED GENE ONTOLOGY

OWL Gene Ontology	Transformed OWL Gene Ontology with Temporal Tags
<owl:Class rdf:ID="GO_0005952">	<owl:Class rdf:ID="GO_0005952@1.0.0" rdf:Validity="Latest" rdf:Timestamp="01:02:2006 #05:05:00"/>
<rdfs:label>cAMP-dependent protein kinase complex</rdfs:label>	<rdfs:label>cAMP-dependent protein kinase complex</rdfs:label>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">An enzyme complex, composed of regulatory and catalytic subunits,………..</rdfs:comment>	<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">An enzyme complex, composed of regulatory and catalytic subunits,………… </rdfs:comment>
<!-- unlocalized protein complex -->	<!-- unlocalized protein complex -->
<rdfs:subClassOf rdf:resource="#GO_0005941"/>	<rdfs:subClassOf rdf:resource="#GO_0005941@1.0.0" rdf:Validity="Latest" rdf:Timestamp="01:02:2006 #05:05:00"/>
</owl:Class>	</owl:Class>

TABLE II
PHASE II OUTPUT FOR A CLASS WITH FRAME VERSIONING

Phase II OWL Ontology with Temporal Tags
<owl:Class rdf:ID="GO_0005952@1.1.0" rdf:Validity ="Latest" rdf:Timestamp="03:09:2006#10:30:00">
<rdfs:label>cAMP-dependent protein kinase complex </rdfs:label>
<rdfs:comment rdf:datatype= "http://www.w3.org/2001 /XMLSchema#string">An enzyme complex, composed of regulatory and catalytic………..</rdfs:comment>
<!-- protein complex -->
<rdfs:subClassOf rdf:resource="#GO_0043234@1.1.0" rdf:Validity="Latest" rdf:Timestamp="03:09:2006#03:30:00" />
<rdfs:subClassOf><owl:Restriction>
<owl:onProperty>
<owl:ObjectProperty rdf:about="#part_of"/>
</owl:onProperty>
<owl:someValuesFrom rdf:resource= "#GO_0005622@1.1.0" rdf:Validity="Latest" rdf:Timestamp="03:09:2006#03:30:00"/>
</owl:Restriction>
<!-- intracellular -->
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="GO_0005952@1.0.0" rdf:Validity="Past" rdf:Timestamp="09:09:2006#05:30:00">
<rdfs:label>cAMP-dependent protein kinase complex</rdfs:label>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/ XMLSchema#string">An enzyme complex, composed of regulatory and catalytic……..</rdfs:comment>
<!-- unlocalized protein complex -->
<rdfs:subClassOf rdf:resource="#GO_0005941@1.0.0" rdf:Validity="Past" rdf:Timestamp="09:09:2006#05:30:00"/>
</owl:Class>

At the end of the first phase the value of rdf:Validity is set to Latest for all the constructs and a default version number 1.0.0 is appended to the rdf:Id and rdf:Resource values. Current date and time is set for the rdf:Timestamp.

Table 3 shows another class that has become obsolete in the new version. The class just contains rdfs:label and rdfs:comment in the new version but was having a owl:Restriction in the old version. Semantic web applications that are dependent on the old version will have problem due to the change in the OWL class but in case of Temporal tagged OWL class, the application will still continue to work fine. Fig. 1 shows a portion of the class hierarchy in Gene Ontology. The classes of the latest version are depicted in white and classes of the past version are in gray. The class GO_0007275@1.0.0 has a sub class GO_0009835@1.0.0 that has changed to GO_0009835@1.1.0 between the two versions taken in the case study. Due to the change, the sub classes of GO_0009835@1.0.0 should become the subclasses of the latest version GO_0009835@1.1.0. Since, the sub classes of GO_0009835@1.0.0 contains only a slot or resource that points to it base class, slot versioning is applied here and the rdfs:subClassOf tag is replicated with its resource pointing to the new version. The new version of resource is depicted in white and old version in gray, similar to the classes. The new version of the class GO_0009835@1.1.0 is appeared under another class GO_0048608@1.0.0. The dotted line in the figure1 depicts that there are many other classes present in between that are not shown in the figure. Other classes shown in the Fig. 1 are not expanded to show their resources, as they have not changed between these versions. One can judge from the code of temporal tagged ontology presented in Table 1, Table 2 and Table 3 that the temporal tagged ontologies will be having larger size due to the presence of extra tags and due to the repetition of constructs for each version.

TABLE III
PHASE II OUTPUT FOR AN OBSOLETE CLASS WITH FRAME VERSIONING

Phase II - OWL Ontology with Temporal Tags
<owl:Class rdf:ID="GO_0000067@1.1.0" rdf:Validity="Latest" rdf:Timestamp="03:03:2006#05:05:00"> <rdfs:label>DNA replication and chromosome cycle</rdfs:label> <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string" > OBSOLETE (was not defined before being made obsolete).</rdfs:comment> </owl:Class> <owl:Class rdf:ID="GO_0000067@1.0.0" rdf:Validity="Past" rdf:Timestamp="03:09:2005#05:05:00"> <rdfs:label>DNA replication and chromosome cycle</rdfs:label> <rdfs:subClassOf> <owl:Restriction> <owl:onProperty> <owl:ObjectProperty rdf:about="#part_of"/> </owl:onProperty> <owl:someValuesFrom rdf:resource="#GO_0007049@1.0.0" rdf:Validity="Past" rdf:Timestamp="03:09:2005#05:05:00"/> </owl:Restriction> <!-- cell cycle --> </rdfs:subClassOf> </owl:Class>

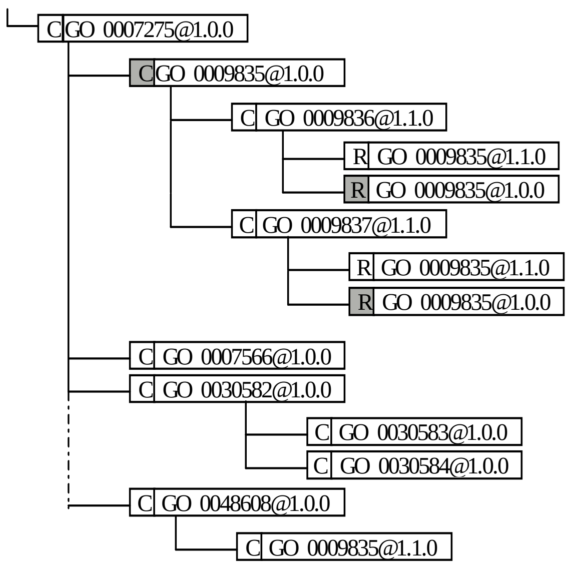

Fig. 1 Portion of class hierarchy of OWL Gene Ontology with Temporal Tags

The code has also become less readable because of the replication. We argue that both the size and readability of OWL files are having very low effect on the performance of OWL ontologies because these ontologies are primarily designed for the semantic web applications usage and particular OWL ontology version can always be generated back from them.

VII. Conclusion

In this paper an approach providing consistent and dynamic ontology support to knowledgebase applications is presented. The approach combines the concepts of temporal frame and slot versioning with the ontology to create temporal tagged ontologies with embedded versioning. We also propose to enhance the existing OWL to enable the creation of temporal tagged OWL ontologies. The two new tags i.e. rdf:Validity and rdf:Timestamp are introduced and a scheme is presented for the value of the rdf:Id, rdf:About and rdf:Resource tags for making the temporal tagged ontologies consistent with the non-temporal ontologies. The existing mechanisms of ontology integration can easily be extended to incorporate temporal extensions. OWL temporal tagged ontologies are designed for the machine consumption and not for human readability as it includes repetition of concepts with even minor changes. These ontologies will be of great use for the semantic web applications and will make them independent of the ontology versions. This means that the semantic web applications developer can devote more attention to the application logic and agent behaviors development without worrying the ongoing changes in the domain knowledge. The dynamic behavior of the temporal tagged ontologies will allow the application to use the most recent concepts of the ontology without sacrificing the stability of the application.

References

[1] P. Bedi, K. D. Sharma, S. Kaushik, "Time Dimension to Frame Systems", Journal of Information Science and Technology, Vol.2, No.3, pp.212-228, April 1993.

[2] Gene Ontology Consortium, "The Gene Ontology (GO) Database and Informatics Resource" Nucleic Acids Research, Vol. 32, Database Issue, pp. 258-261, 2004.

[3] R. Maiocchi and B. Pernici, "Temporal Data Management Systems: A Comparative View", IEEE Trans. on Knowledge & Data Engg., Vol.3, No.4, Dec.1991.

[4] N. F. Noy, and M. Klein, , "Ontology Evolution: Not the Same as Schema Evolution" Knowledge and Information Systems, Vol. 6, pp. 428–440, 2004.

[5] N. F. Noy, and M. A. Musen, "Ontology Versioning in an Ontology Management Framework" Intelligent Systems, IEEE, Vol. 19, No. 4., pp. 6-13, 2004.

[6] P. Bedi, and S. Marwaha, "Framework for Ontology Based Expert Systems: Disease & Pests Identification in Crops - A Case Study", in: Proceedings of the International Conference of Artificial Intelligence (IC-AI) 2005: 256-259, 2005.

[7] D. Brickley, and R. V. Guha, "RDF Vocabulary Description Language 1.0: RDF Schema", W3C Note, World Wide Web Consortium, URL http://www.w3.org/TR/rdf-schema/ , February 2004.

[8] M. C. A. Klein, D. Fensel, A. Kiryakov, and D. Ognyanov, "Ontology Versioning and Change Detection on the Web", in: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02), Spain, LNCS vol. 2473, pp 197-212, October 1-4, 2002.

[9] J. Heflin, and Z. Pan, "A Model Theoretic Semantics for Ontology Versioning" 3rd International Semantic Web Conference 2004, Lecture Notes in Computer Science vol. 3298, Springer-Verlag, pp. 62–76, 2004.

[10] N. F. Noy, and M. A. Musen, "PromptDiff: A Fixed-Point Algorithm for Comparing Ontology Versions," in Proc. 18th Nat'l Conf. Artificial Intelligence (AAAI 2002), AAAI Press, pp. 744–750, 2002.

[11] M. K. Smith, C. Welty, and D. L. McGuinness, "OWL Web Ontology Language Guide". W3C Note, World Wide Web Consortium, February 2004. URL http://www.w3.org/TR/owl-guide/ .

[12] Gruber, T. R., Toward principles for the design of ontologies used for knowledge sharing, KSL-93-04, Knowledge Systems Laboratory, Stanford University, 2003. http://ksl-web.stanford.edu/ .

[13] Pinto, H. S., and Martins, P. J., A methodology for ontology integration, in proc. of the International Conference on Knowledge Capture, pp 131-138, 2001, Victoria, British Columbia, Canada.

[14] McGuinness, D., Fikes, R., Rice, J., and Wilder, S., An environment for merging and testing large ontologies, in Proc. of the 7th Internationational Conference on Principles of Knowledge Representation and Reasoning (KR2000), pp483-493, 2003, Breckenridge, Colorado, USA.

[15] Li, L., Wu, B., and Yang, Y., Agent-based Ontology Integration for Ontology-based Applications, in Conferences in Research and Practice in Information Technology (CRPIT), Vol. 58, 2005.

Mr. Sudeep Marwaha has got his masters degree in Computer Science from the IARI, New Delhi in 1998. His research interests include Intelligent Information Systems, Knowledge Representation, Expert Systems, Ontologies, Semantic Web, and Multi-Agent Systems.

He is a research scholar in the Department of Computer Science, University of Delhi and is a scientist at Division of Computer Applications, Indian Agricultural Statistics Research Institute, Delhi, India. He is working as a scientist in the Institute since 1999. He has got about 10 publications in reputed international conferences and journals.

Mr. Marwaha is a life member of the Indian Society of Agricultural Statistics.

Dr. Punam Bedi has got her Ph.D. in Computer Science from the Department of Computer Science, University of Delhi, Delhi, India in 1999. She has received her M.Tech from IIT Delhi, Delhi, India. Her research interests include Artificial Intelligence, Web Intelligence, Semantic Web, Intelligent Information Systems, Intelligent Software Engineering, Intelligent Agents, Intelligent User Interfaces, Fuzzy Logic, Genetic Algorithms, Knowledge Representation, Ontologies AI approaches to Software Engineering, Software Engineering and Data Mining, Software Reliability, Software Metrics, Requirement Engineering, Human Computer Interaction (HCI), Trust, Information Retrieval, Personalization.

She is head and reader in the Department of Computer Science, University of Delhi. She has 18 years of experience in research and teaching in different capacities with University of Delhi. She has about 30 publications in reputed journals and conferences.

Dr. Bedi is a member of IEEE and life member of Computer Society of India.