Eder J. & Koncilia C. (2004). *Modelling Changes in Ontologies*. Proceedings of the On The Move - Federated Conferences (OTM 2004), October 25-29, Agia Napa, Cyprus, LNCS 3292, pp. 662-673.

Modelling Changes in Ontologies

Johann Eder, Christian Koncilia

University of Klagenfurt
Dep. of Informatics-Systems
{eder, koncilia}@isys.uni-klu.ac.at

Abstract. Ontologies are shared conceptualizations of certain domains. Especially in legal and regulatory ontologies modifications like the passing of a new law, decisions by high courts, new insights by scholars, etc. have to be considered. Otherwise, we would not be able to identify which knowledge (which ontology) was valid at an arbitrary timepoint in the past. And without this knowledge we would for instance not be able to identify why a user came to a specific decision.
In this paper we will show how a simple ontology description formalism, namely a directed graph, has to be extended to represent changing knowledge. Furthermore, we will present the operations that are necessary to manipulate such an ontology. Finally, we will discuss different implementation approaches.

1 Introduction

Ontologies are generally seen as a promising approach for adding semantics to data processing. Semantic Web, semantic web services, enterprise integration are but a few research avenues where ontologies are researched with high expectations.

An ontology is a shared conceptualization of a certain domain [Gru03]. It describes the concepts of a domain, their properties and their relationships. Much work has already been done to analyse multiple heterogeneous ontologies, their integration and their coexistence.

Surprisingly little attention was drawn to the fact that the reality an ontology describes and/or the view of the observers sharing the conceptualization on the reality may change. In legal and regulatory ontologies the passing of a new law, decisions by high courts, new insights by scholars, new entities in the real world which require a new or changing interpretation of legal concepts are changes which have to be considered.

There are three different basic approaches of how to deal with such changing knowledge. First of all, we could simply ignore modifications, and describe the world in a completely static way. Obviously, this approach is of limited suitability for real world applications. The second approach is a little bit more sophisticated: by adopting our knowledge description we always represent and store the most recent version of knowledge. This is the most frequent approach nowadays. It has the disadvantage that we lose knowledge about the past. If for example we

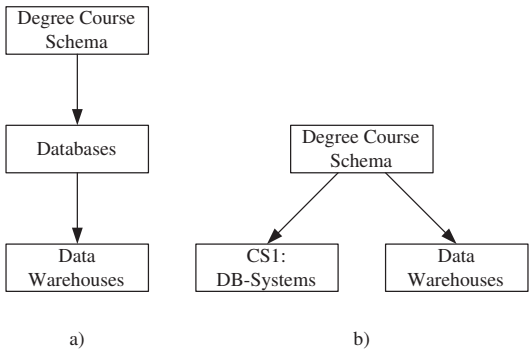

Fig. 1. Our running example represented as ontology graph

study an old court decision and we do not know the state of the law at *decision time* and at *case time*, we probably will not be able to understand the verdict. So only working with the most recent ontology is definitively not satisfactory. The third approach takes into account that the knowledge about modifications is again knowledge that may be important. In this approach we would have to describe different versions of knowledge and the relations between these versions. The last two approaches are well known in the temporal database community. The first one is called (Schema) Evolution, the latter one (Schema) Versioning [JD98].

During the last years several specification languages for ontologies have been developed. These includes DAML+OIL which is now going to become the Web Ontology Language OWL (both languages stem from the area of description logics), Loom, Open Knowledge Base Connectivity (OKBC), or Common Logics (CL).

In this paper we will discuss how a simple ontology description formalism, namely a directed graph, has to be extended to represent changing knowledge, and which extensions to such a "specification language" would be meaningful.

In order to achieve this goal the following extensions to an ontology description are necessary:

- **Temporal extension**: ontology data (classes and relations between classes) has to be time stamped in order to represent their valid time (the time in which a fact is true in the real world [JD98]).
- **Ontology versions**: by providing time stamps for ontology data we have to cope with different versions.
- **Ontology selection**: Our system has to support functions to select a specific version of an ontology.

The remaining parts of the paper are organized as follows: In section 2 we introduce a running example. In section 3 we discuss related work. After that, we will introduce our model of a versioning graph in section 4. Three different

implementation approaches will be briefly discussed in section 5. Finally, we will conclude in section 6.

2 Running Example

Through the rest of this paper we will use the following example to show that ontology versioning is vital for the correctness and expressiveness of ontologies and ontology queries: consider that we built an ontology that covers different degree course schemes at a university.

In 1990 the first version of a degree course schema for the Computer science study has been adopted. In this version, the database course was called "Databases". Furthermore, it was necessary to pass the exam of this course in order to take the "Data Warehouses" lecture. This first version is depicted in Fig. 1a).

Now consider that in 2000 a new version of this regulation became effective. In this new version the "Databases" has been renamed to "CS1: DB-Systems". Moreover, it is no longer required to pass this lecture in order to take the "Data Warehouses" lecture. This recent version is depicted in Fig. 1b).

If we would use a simple ontology evolution approach, i.e., if we would represent and store only the most recent version of our ontology, we could get problems with applications or other ontologies that use our ontology. In the best case, these ontologies / applications would recognize that the ontology changed and report an error. In the worst case, the new version doesn't produce syntax errors in the application / ontology but leads to wrong results.

Moreover, there are many cases in which the knowledge that an ontology changed and how it changed is not enough. In fact, we often need to know during which time interval a specific ontology version was valid! Just imagine that a student took "Databases" after he took "Data Warehouses". In order to answer questions like "was this correct according to the regulations valid in 1995?" we would have to timestamp each ontology version.

3 Related Work

Our approach builds on the techniques developed for temporal databases, schema evolution and schema versioning of databases [FGM00, JD98]. However, as shown and discussed in [NK03b] these approaches designed for database systems cannot be directly applied to ontologies. The authors of [NK03b] argue, that the main reasons for this are: (1) ontologies are both at the same time, schema and instances. (2) In contrast to (relational) database systems ontologies themselves incorporate semantics. (3) Ontologies are more often reused. (4) Ontologies lack of centralized control mechanisms as they are de-centralized by nature. (5) The data model of an ontology is usually richer. (6) Finally, they argue that in ontologies there is no clear distinction between classes and instances.

In [NK03a] the same authors present their framework for ontology evolution. In this paper the authors focus mainly on ontology evolution and not on ontology

versioning. As a consequence they do not provide any information about the valid time of a specific ontology. They discuss different change mechanisms and present an ontology of change operations.

In [KOF02] an ontology versioning methodology is presented. The authors present a tool (called OntoView) that helps the user to compare different versions of an ontology. Therefore, the systems analyzes changes between two version on a structural level. After that, the user may specify relations between the ontology versions, i.e., he may specify whether two concepts are "identical" or if a "conceptual change[1]" took place. The main focus of this paper is how to detect what changed from one version of an ontology to another one. Again, information about the valid time has not been considered.

4 Ontology Versioning Graph

A good definition of what an ontology is has been given in [Gru03]. In this paper, the author defines an ontology as an explicit specification of a conceptualization of a domain. This explicit specification may be specified by using one of the languages mentioned above, e.g., DAML+OIL, OWL or CL. Another possibility to specify such a conceptualization is to use a graph where nodes represent concepts, and edges represent the relations between two concepts [MWK00]. Figures 1 a) and 1 b) are examples for ontology graphs.

Until now we described intuitively how an ontology may be represented as a graph. We will now extend this description to define a temporal extension that supports valid time. Therefore, we have to introduce:

- **Time model**: Our model uses a linear and discrete model of time. This means that each point in time (each instant[2]) can be mapped to an integer, i.e., the model is isomorphic to the natural numbers, and each time point has a single successor. Furthermore, A chronon Q is defined as "a non-decomposable time interval of some fixed, minimal duration" [JD98]. In our model, a chronon is the smallest unit of time. Or, in other words, the time axis is a series of chronons.
- **Valid Time**: The term *Valid Time* is well-known in the temporal database community. It defines the time, in which a fact (in our model a fact may be both, a class or the relation between two classes respectively) is true in the real world. A fact may have more than one time points or time intervals during which it is true in the modelled world. [JD98]
- **Time Intervals**: In order to introduce valid time in an ontology all classes and all relations between these classes may exist in several versions. Each version must have a time interval $[T_s, T_e[$ representing the valid time where

[1] According to the authors a conceptual change is "a change in the way a domain is interpreted..."

[2] We use the term Instant as defined in [JD98]: "An instant is a time point on an underlying time axis."

T_s is the beginning of the valid time, T_e is the end of the valid time. We represent that a fact is valid until now by $T_e = \infty$.
Please note that we use the syntax $[A, B[$ to represent a half-closed interval. In this half-closed interval the instant A is included, whereas the instant B is excluded.

Until now we intuitively defined our model of an ontology versioning graph. We will now give a formal definition of such an ontology versioning graph.

Definition 1 (Ontology Versioning Graph Definition): *A ontology versioning graph G is defined as a tupple $G = (\mathbb{N}, \mathbb{E})$ where $\mathbb{N}$ is a set of classes C and $\mathbb{E}$ is a set of relations R between these classes.*

Definition 2 (Class Definition): *A class C is defined as a triple $C = (label, VT, S)$ where label is the label of the class (usually a noun representing a concept), VT is a tuple $[T_s, T_e[$ representing the valid time of the class, and S is a set of slots (attributes and properties assigned to a class).*

Definition 3 (Relation Definition): *A relation R is defined as a n-tuple $R = (C_f, C_t, type, VT)$ with $C_f \in \mathbb{N} \wedge C_t \in \mathbb{N}$ where C_f represents the class from which the relation leads to the class C_t, type represents the the type of the relation (i.e., $InstanceOf$, $SubclassOf$, ...) and VT is a tuple $[T_s, T_e[$ representing the valid time interval of the relation.*

Figure 2 a) shows an example of such a ontology versioning graph, where nodes represent the classes *Databases*, *Data Warehouses*, *Degree Course Schema* and *CS1: DB-Systems* and edges represent relations between these classes. As can be seen, each node and each edge has a time stamp representing its valid time. For sake of readability, we did not depict different types of relations and slots in this example.

4.1 Versioning Operations

Some basic operations have to be defined in order to manipulate a ontology versioning graph G. These operations enable us to insert, update and delete classes, or nodes respectively and to insert, update and delete relations, or edges respectively. They are defined as follows:

1. *Insert Class*: inserts a new class C' which is valid at a given time interval $I = [T_s, T_e[$ into a given graph $G = (\mathbb{N}, \mathbb{E})$. This operation results in a graph $G' = (\mathbb{N}', \mathbb{E})$ where $\mathbb{N}' = \mathbb{N} \cup C$.
2. *Update Class*: updates a class C which is an element of $\mathbb{N}$ of a given graph $G = (\mathbb{N}, \mathbb{E})$ to C' at an instant T. This operation results in a graph $G' = (\mathbb{N}', \mathbb{E})$ where $\mathbb{N}' = \mathbb{N} \cup C'$. The valid time of C' is $[T, \infty[$ and the valid time of C is $[T_s^C, T[$ where T_s^C is the start of the valid time of C as it was before the update operation.
   This operation should not be confused with the `UPDATE` operation known from relational database systems. In fact updating a class in a temporal sense means to create a new version of the class concerned, e.g., because an attribute of that class changed, or an attribute of that class was deleted, etc.

3. *Delete Class*: "deletes" a class C (whose ending valid time equals ∞, i.e., has not been set yet) which is an element of $\mathbb{N}$ of a given graph $G = (\mathbb{N}, \mathbb{E})$ and sets its ending valid time to T. This operation results in a graph $G' = (\mathbb{N}', \mathbb{E}')$ where $\mathbb{N}' = \mathbb{N} \setminus C \cup C'$ and $C' = C$ except that the end of the valid time of C' is set to T. Formally $C' = (C.label, [C.T_s, T[, C.S)$.
   Furthermore, we have to set the end of the valid time of all relations that lead to or from class C. Hence, if $\mathbb{E}_c$ is the set of all relations that lead to or from class C ($\mathbb{E}_c = \{x | x \in \mathbb{E} \bullet x.C_f = C \vee x.C_t = C\}$), and $\mathbb{E'}_c$ is the set where the end of the valid time of all relations from or to C has been set to T then $\mathbb{E}' = \mathbb{E}|\mathbb{E}_c \cup \mathbb{E'}_c$.
   Again, this operation should not be confused with the `DELETE` operation known from traditional relational database systems. In fact, this operation does not delete anything - it sets the end time of the valid time of a class.
4. *Insert Relation*: inserts a new relation R which is valid at a given time interval $I = [T_s, T_e[$ into a given graph $G = (\mathbb{N}, \mathbb{E})$. This operation results in a graph $G' = (\mathbb{N}, \mathbb{E}')$ where $\mathbb{E}' = \mathbb{E} \cup R$.
5. *Update Relation*: updates a relation R which is an element of $\mathbb{E}$ of a given graph $G = (\mathbb{N}, \mathbb{E})$ to R' at an instant T. This operation results in a graph $G' = (\mathbb{N}, \mathbb{E}')$ where $\mathbb{E}' = \mathbb{E} \cup R'$. The valid time of R' is $[T, \infty[$ and the valid time of R is $[T_s^C, T[$ where T_s^C is the start of the valid time of R as it was before the update operation.
   This update operation can be used to change the type of a relation, e.g., to change a relation of type $PartOf$ to a $SubclassOf$ relation.
6. *Delete Relation*: "deletes" a relation R which is an element of a given graph $G = (\mathbb{N}, \mathbb{E})$ and sets its ending valid time to T. The ending valid time of R has to be equal to ∞. This operation results in a graph $G' = (\mathbb{N}, \mathbb{E}')$ where $\mathbb{E}' = \mathbb{E} \setminus R \cup R'$ and $R' = R$ except that the end of the valid time of R' is set to T. Formally $R' = (R.label, [R.T_s, T[, R.S)$.

4.2 Integrity Constraints for a Ontology Versioning Graph

To bring in the concept of temporality into an ontology has several consequences - one is, that some constraints have to be fulfilled in order to guaranty the integrity of our model. In this paper, we will not fucus on a detailed description and definition of all integrity constraints. However, we have to define some constraints that we will need later on in this paper.

A basic constraint is that for all time stamps $[T_s, T_e[$ in our model $(T_s \leq T_e) \vee (T_e = \infty)$ has to be true, i.e., a class may not end before it starts.

Furthermore, the valid time of a relation $r = (C_f, C_t, type, VT)$ of a graph $G = (\mathbb{N}, \mathbb{E})$ between two classes C_f and C_t has to be within (to exist in) the valid time of both classes.

$$\forall r \in \mathbb{E} : exists\_in(r, C_f) \wedge exists\_in(r, C_t) \tag{1}$$

where $exists\_in(X_i, X_j)$ describes that the time interval representing the valid time of temporal component X_i is a subset of the time interval representing

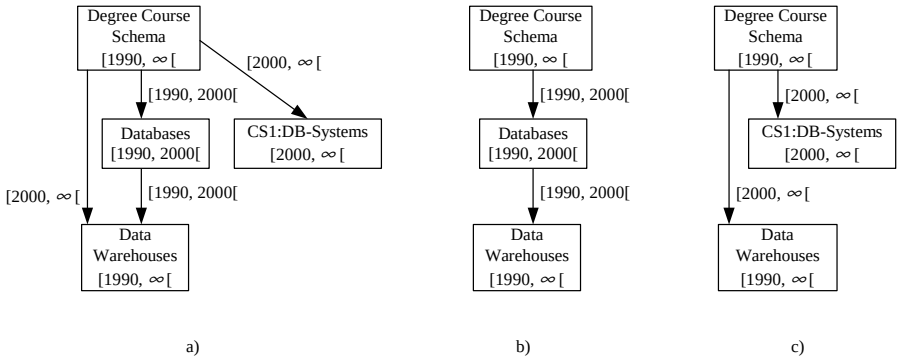

Fig. 2. a) A ontology versioning graph and its deduced versions b) and c)

the valid time of temporal component X_j. A *temporal component* is a node or an edge in a ontology versioning graph $G = (\mathbb{N}, \mathbb{E})$.

Formally, we can define $exists\_in(X_i, X_j)$ as follows:

$$exists\_in(X_i, X_j) = \begin{cases} True & \text{if } \forall t \in [T_s^{X_i}, T_e^{X_i}[\bullet t \in [T_s^{X_j}, T_e^{X_j}[\\ False & \text{otherwise} \end{cases} \tag{2}$$

4.3 Selecting a Specific Ontology Version

The graph defined in section 4 consists of all possible ontology versions. Or, in other words, we do not define several ontologies where each ontology represents a version of an ontology. In fact, we define a single ontology which is composed of all ontology versions.

Figure 2 a) shows the ontology versioning graph for our running example. As can be seen, this ontology versioning graph consists of two versions b) and c). In this example, version b) is valid during the interval $[1990, 2000[$ and version c) with a valid time $[2000, \infty[$. In this example the chronon (as defined in section 4) is a year. Hence, $[1990, 2000[$ represents that this version is valid from 1990 until 1999 (2000 is not included as we use half-closed intervals).

Intuitively we can say that if we represent all timestamps $[T_s, T_e[$ of all temporal components within our ontology versioning graph on a linear time axis, the interval between two consecutive timestamps on this axis represents the valid time of an ontology version.

Formally, we can define such an ontology version $G(T)$ (an ontology versioning graph valid at instant T) over a graph $G = (\mathbb{N}, \mathbb{E})$ as follows:

$$G(T) = (\mathbb{N}_\mathbb{T}, \mathbb{E}_\mathbb{T}) \tag{3}$$

where

$$\mathbb{N}_\mathbb{T} = \{x | x \in \mathbb{N} \wedge x.T_s \leq T \leq x.T_e\} \tag{4}$$

and

$$\mathbb{E}_\mathbb{T} = \{x | x \in \mathbb{E} \wedge x.T_s \leq T \leq x.T_e\} \tag{5}$$

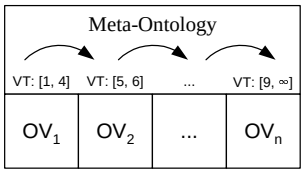

Fig. 3. Using a Meta-Ontology over ontology versions $OV_1, ..., OV_n$

Intrinsically, we should define that all nodes (classes) referenced by edges (relations) have to be valid in the selected ontology version $G(T)$. However, there is no need to check whether both classes C_f and C_t of all edges $R = (C_f, C_t, type, VT)$ are valid within the selected ontology version. In fact, this constraint can be deduced from the constraint (1) as defined in section 4.2. We will give a proof of this in appendix A.

This leads us to the definition of a stable interval. Intuitively, we can say that such a stable interval is a view defined on an ontology versioning graph that is valid for a given time interval $[T_s, T_e[$. All classes and relations within this ontology versioning graph are also valid for the given time interval. In other words, within such a stable interval there cannot exist different versions of classes or relations. In the example shown in Fig. 2 we have two stable intervals: the first is valid during the interval $[1990, 2000[$, the second one during the interval $[2000, \infty[$.

Formally, we define a stable interval as follows: Let $\mathbb{T}$ be the set of all instants of changes, i.e., $\mathbb{T} = \{x.T_s | x \in \mathbb{N} \vee x \in \mathbb{R}\} \cup \{x.T_e | x \in \mathbb{N} \vee x \in \mathbb{R}\}$. For the example shown in Fig. 2 a) $\mathbb{T}$ would be $\mathbb{T} = \{1990, 2000, \infty\}$. Then a stable interval is a interval $[T_s, T_e[$ between to contiguous instants T_s and T_e of the set $\mathbb{T}$. Two instants T_1 and T_2 (with $T_1 \leq T_2$) are contiguous if there exists no instant T such that $T_1 < T < T_2$.

5 Implementation Approaches

In general, our approach for ontology versioning can be implemented in three different ways:

- **Meta-Ontology:** The basic idea of the first approach is depicted in Fig. 3. In this approach a meta-ontology is used to store the valid time of different ontology versions and to deal with inter-structure relationships between these ontology versions. The main advantage of this approach is that any ontology description language (for instance, OWL) can be used to define the different ontology versions.
- **Standard Extension:** In this approach a temporal extension for a standard representation language for ontologies has to be developed. The main advantage of this approach is that a uniform standard produces uniform solutions. Hence, such a standard could be defined in a way in which for instance temporal reasoning is supported.

- **Using Standard:** The last approach uses the techniques that a standard like OWL, or DAML+OIL provides. In contrast to an extension of the standard this approach will result in different solutions. Hence, it cannot be guaranteed that for instance reasoning algorithms take into account the temporal aspects of an ontology in a correct way.

We will now briefly discuss how OWL may be used to implement an ontology as presented in section 4. We would like to emphasize that the implementation discussed here is only one possible (straight forward) implementation. Other approaches are of course feasible. Moreover, we will see that using OWL to represent valid time has its drawbacks.

OWL is a ontology representation language defined by the W3C Web Ontology Working Group. In contrast to other languages like for instance RDF (Resource Description Framework) OWL provides more expressiveness [AvH04].

However, the support for ontology versioning is very limited in OWL. A tag `versionInfo` may be used to describe the current version of the ontology, for instance by using RCS/CVS keywords. Nevertheless, this tag doesn't contribute to the logical meaning of the ontology at all. The following example how this `versionInfo` tag may be used has been taken from [Wor02]:

```
<Ontology rdf:about="">
  <versionInfo>$Id: Overview.html,v 1.4 2002/07/31
               19:44:09 henri Exp $</versionInfo>
  <rdfs:comment>An example ontology</rdfs:comment>
  <imports rdf:resource="http://www.w3.org/2002/07/owl"/>
</Ontology>
```

Furthermore, two tags may be used to indicate whether or not two ontologies are compatible: `owl:backwardCompatibleWith` and `owl:incompatibleWith`.

The tags `owl:DeprecatedClass` and `owl:DeprecatedProperty` may be used to indicate that a class or property is likely to change.

We will now discuss how OWL may be used to incorporate time stamps, i.e., versioning information, into an ontology. In order to achieve this we have to time stamp the nodes and edges of our graph model, i.e., the `Class` and `ObjectProperty` (a binary relation between two classes) elements in OWL.

First we have to define a `DatatypeProperty` (a binary relation between classes and datatypes) that represents the beginning of the valid time interval, and one to represent the end of the valid time interval. The following statement defines such a `DatatypeProperty` called `vtStart` whose range is a date:

```
<owl:DatatypeProperty rdf:ID="vtStart">
 ...
 <rdfs:rangerdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
</owl:DatatypeProperty>
```

Please note that "..." indicates that there is additional text. The full example can be downloaded at `http://www.ifi.uni-klu.ac.at/Publications`.

Now, we can use this property to define the valid time interval of classes. This can be done by using `owl:hasValue` to "restrict" the class to a specific value that the property `vtStart` must have. For instance, we can use the following to specify the beginning of the valid time interval (January, 1st 1999) for the class `DegreeCourseSchema`:

```
<owl:Class rdf:ID="DegreeCourseSchema">
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#vtStart"/>
         <owl:hasValue rdf:datatype="&xsd;date">
         1990-01-01
         </owl:hasValue>
      </owl:Restriction>
      ...
```

Something similar can be used to describe the valid time of a relation between two classes, i.e., an `ObjectProperty` in OWL.

According to the integrity constraint defined in section 4.2 the valid time of a relation between two classes has to be within the valid time of both classes. Usually, the starting time of the valid time interval of the relation will be equal to the starting time of one of the classes. Accordingly the ending time of the relation is usually equal to the ending time of one of the classes. If so, we could use the `isDefinedBy` tag to define this relation between valid times:

```
  ...
<rdfs:isDefinedBy>
   <Databases rdf:ID="620.202">
     <ValidTime rdf:datatype="&xsd;date"
     >2000-01-01</ValidTime>
   </Databases>
 </rdfs:isDefinedBy>
```

One drawback of the example given above is that the valid time of the relation is in fact bound to the valid time of an instance of the class and not the class itself.

Another approach would be to use the `versionInfo` tag mentioned before:

```
<owl:ObjectProperty rdf:ID="attendanceOrder">
  <rdfs:domain rdf:resource="Databases"/>
  <rdfs:range rdf:resource="DataWarehouses"/>
  <owl:versionInfo rdf:datatype="&xsd;date"
  >vtStart: 1990-01-01</owl:versionInfo>
  ...
```

All these techniques to incorporate valid time into OWL ontologies have their drawbacks. The first drawback is that such an ontology would not support

reasoning as long as the time dimension is not considered in a correct way. For instance, such a temporal reasoning algorithm should consider that if $A \rightarrow B$ (where $A \rightarrow B$ means "A semantically implies B") and $B \rightarrow C$ then $A \rightarrow C$ is only true if there exists at least one timepoint which is within the valid time of both, A and C.

Another drawback is, that OWL would not guarantee that all integrity constraints are considered in a correct way. For example, the valid time of a relation between two classes has to be within the valid time of both classes.

Taking all together we conclude that all these techniques are "extralogical". This means that they don't incorporate any additional (temporal) semantics that maybe used by a reasoning algorithm or any other application.

6 Conclusions

We have shown a novel approach to represent changes in ontologies. By introducing information about the valid time of concepts represented in ontologies, we are able to identify which knowledge (which ontology) was valid at an arbitrary timepoint in the past. This enables a user of an ontology to understand a verdict if, for instance, he studies an old court decision and doesn't know the state of the law at *decision time* and at *case time*.

Incorporation of other time dimensions will be one of the main research avenues that we will investigate in the near future. This includes for instance *transaction time* (the time when a fact is current in the database [JD98], i.e., in the system). For instance, this time dimension is important when we would like to deal with regulations and laws that become effective retroactively.

References

[AvH04] G. Antoniou and F. van Harmelen. Web Ontology Language: OWL. In *Handbook on Ontologies*, pages 67–92. Springer-Verlag, 2004. In [SS04].

[EJS98] O. Etzion, S. Jajodia, and S. Sripada, editors. *Temporal Databases: Research and Practise*. Springer-Verlag (LNCS 1399), 1998.

[FGM00] E. Franconi, F. Grandi, and F. Mandreoli. Schema Evolution and Versioning: a Logical and Computational Characterisation. In *Workshop on Foundations of Models and Languages for Data and Objects*, 2000.

[Gru03] T. Gruber. A Translation Approach to Portable Ontology Specification. In *Knowledge Acquisition 5(2):199-220*. World Wide Web Consortium (W3C), 2003.

[JD98] C. S. Jensen and C. E. Dyreson, editors. *A consensus Glossary of Temporal Database Concepts - Feb. 1998 Version*, pages 367–405. Springer-Verlag, 1998. In [EJS98].

[KOF02] M. Klein, A. Kiryakov D. Ognyanov, and D. Fensel. Ontology Versioning and Change Detection on the Web. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)*, 2002.

[MWK00] P. Mitra, G. Wiederhold, and M. Kersten. A Graph-Oriented Model for Articulation of Ontology Interdependencies. In *Proceedings Conference on Extending Database Technology 2000 (EDBT'2000), Konstanz, Germany, 2000*, volume LNCS: 1777, pages 86+, 2000.

[NK03a] N. Noy and M. Klein. A component-based framework for ontology evolution. In *Proceedings of the Workshop on Ontologies and Distributed Systems (IJCAI'03)*, 2003.

[NK03b] N. Noy and M. Klein. Ontology Evolution: Not the Same as Schema Evolution. In *Knowledge and Information Systems*, volume 5, 2003.

[SS04] S. Staab and R. Studer, editors. *Handbook on Ontologies*. Springer-Verlag (ISBN 3-540-40834-7), 2004.

[Wor02] World Wide Web Consortium. *OWL Web Ontology Language 1.0 Reference, W3C Working Draft 29 July 2002*. World Wide Web Consortium, 2002. URL: http://www.w3.org/TR/2002/WD-owl-ref-20020729/.

A Proof of Theorem

Let $G(T) = (\mathbb{N}_\mathbb{T}, \mathbb{E}_\mathbb{T})$ be an ontology version (an ontology versioning graph valid at time point T) over a graph $G = (\mathbb{N}, \mathbb{E})$ such that $\mathbb{N}_\mathbb{T} = \{x | x \in \mathbb{N} \wedge x.T_s \leq T \leq x.T_e\}$ and $\mathbb{E}_\mathbb{T} = \{x | x \in \mathbb{E} \wedge x.T_s \leq T \leq x.T_e\}$. Q is the defined chronon.

If $\forall r \in \mathbb{E} : exists\_in(r, C_f) \wedge exists\_in(r, C_t)$ (as defined in section 4.2) is true, than from this it follows that $\forall T \in \{r.T_s, \ldots, r.T_e - Q\} \bullet T \in \{C_f.T_s, \ldots, C_f.T_e - Q\} \wedge T \in \{C_t.T_s, ldots, C_t.T_e - Q\}$ □