Multi-temporal RDF Ontology Versioning

Fabio Grandi

Alma Mater Studiorum – Università di Bologna, Italy
fabio.grandi@unibo.it

Abstract. In this paper, we present a multi-temporal RDF database model employing triple timestamping with temporal elements, which best preserves in the multi-temporal setting the scalability property enjoyed by triple storage technologies. The data model is equipped with manipulation operations which allow knowledge engineers to maintain a multi-temporal RDF database in order to manage temporal versions of an ontology.

1 Introduction

When an ontology is changed, some applications require the past version to be kept in addition to the new one, giving rise to multi-version ontologies. This is the case of the legal domain where ontologies must evolve as a natural consequence of the dynamics involved in normative systems [7]. Agents in such a domain may often have to deal with a past perspective, like a Court having to judge today on a fact committed several years ago. Moreover, several time dimensions are usually important for computer applications in such domains.

In the design of semantics based information systems, triple store technology [13] based on the RDF data model [12] is supposed to provide scalability for querying and retrieval. Temporal extensions of the RDF data model have already been presented [8, 11, 17], often in conjunction with special index structures which allow for efficient processing of temporal queries (e.g. tGRIN [11] and keyTree [17]).

In order to preserve the scalability property of the triple storage approach as much as possible also in the presence of temporal semantics, one has to limit the proliferation of *value-equivalent* triples which could be generated by a non very careful definition of the operational semantics of modification statements. Actually, in the previous sentence we have extended to RDF triples the notion of value-equivalent tuples defined in temporal database theory [9] (i.e. two triples are value-equivalent iff their non-temporal contents are exactly the same).

The issue has in part been acknowledged also in [11], where the notion of *normalized* temporal RDF databases has been introduced. In particular, in a monodimensional setting employing interval timestamping of RDF triples, the *coalescence* technique is adopted to avoid storage of value-equivalent triples with overlapping or consecutive intervals. As proposed by temporal database researchers [9], coalescence consists of merging overlapping or consecutive intervals into a single cumulative interval to be used as timestamp of a single representative tuple instead of duplicating it.

In this work, we propose to move another step forward and improve the notion of normalized temporal RDF database by employing the whole state-of-the-art temporal database techniques to avoid data redundancy. Therefore, we propose to introduce *temporal elements* [1, 9] as timestamps to definitely avoid the coexistence of value-equivalent triples in the same temporal RDF database.

The paper is organized as follows. In Section 2, we provide some background information on application requirements which lead our design choices. Hence, a multi-temporal RDF database model employing temporal element timestamping will be introduced in Section 3. In Section 4, the operational semantics of modification operations for the maintenance of a multi-temporal RDF database is presented. Conclusions will be finally found in Section 5.

2 A Reference Application Scenario

In this section, we sketch some features of the reference application scenario we consider in the background, which is important for the design choices we have taken.

In particular, we focus on (multi-version) ontologies which are used to support a semantically indexed access to a usually very large repository of (multi-version) data resources. Snapshot queries, that is queries which reconstruct a single ontology version valid a given point in the multidimensional time domain, are of vital importance in this framework [7]. Snapshot queries involve the enforcement of the temporal perspective, current or past, which is of interest for the user.

For the sake of consistency, the same temporal perspective must be used when accessing both the ontology to select a class of interest and then the data resources linked to such a class. In other words, the temporal selection involves the extraction from the multi-version repository of the ontology version and of the linked data resource versions which are valid at the same (multidimensional) time point. Full details on the application scenario can be found in [2, 5].

For instance, in [5] we considered legal documents as data resources and an ontology formalizing the existence of different classes of citizens before the law. References to ontology classes has then be added to the documents stored in a repository in order to support a personalization service, able to retrieve a selection of legal documents tailored to an individual or a given class of citizens, that is only including norms applicable to their case.

In such an application context, at least three temporal dimensions are needed for a correct modelling of the evolution of data resources and of the related ontology:

- **validity time** is the time a norm is in force in the real world
- **efficacy time** is the time a norm can be applied to concrete cases; while such cases exist, the norm continues its efficacy even if no longer in force
- **transaction time** is the time a norm is stored in the information system

Notice that both validity and efficacy time dimensions have the semantics of *valid time* in temporal database terminology [9], but represent different and

independent valid time notions. Transaction time indeed is always needed when data are managed by a computer system in order to support retro- and pro-active changes and to keep track of them for audit or control reasons.

In this vein, we previously considered in [7] ontologies encoded in OWL/XML format and, thus, defined a temporal data model for the storage and management of multi-version ontologies in such a format. In this work, we indeed consider ontologies encoded in RDF format, enjoying the scalability property supported by the triple store technology [13]. Hence, we will introduce in the Section which follows a multidimensional temporal data model for the storage and management of multi-version ontologies in RDF format. With a simple mapping, the approach can be adapted to any kind of RDF data serialization format and triple storage technique (possibly including N3/Notation3, N-triples, TriX, Turtle).

Another application domain we could consider is the medical one: for example, clinical guidelines can play the role of data resources and ontologies can be used to formalize diseases to which guideline prescriptions can be applied to. At least three independent temporal dimensions (including transaction time) are of interest also in such a framework [4, 6].

3 A Multi-temporal RDF Database Model

Let us start from the defintion of an N-dimensional time domain

$$\mathcal{T} = \mathcal{T}_1 \times \mathcal{T}_2 \times \cdots \times \mathcal{T}_N$$

where $\mathcal{T}_i = [0, \text{UC})_i$ is the i-th time domain. Right-unlimited time intervals are expressed as $[t, \text{UC})$, where UC means "Until Changed", though such a symbol is often used in temporal database literature [9] for transaction time only (whereas, e.g. "forever" or ∞ is used for valid time). Such naming choice refers to the modeling of time-varying data, which are potentially subject to change with respect to all the underlying time dimensions. Actually, we consider ontology versions in force having a continued validity with respect to all temporal dimensions until their definition is changed by authorized users.

Hence, we can define a multi-temporal RDF triple as:

$$(s, p, o \,|\, T)$$

where s is a subject, p is a property, o is an object and $T \subseteq \mathcal{T}$ is a *timestamp* assigning a *temporal pertinence* to the RDF triple (s, p, o). We will also call the (non-temporal) triple (s, p, o) the value or the contents of the temporal triple $(s, p, o \,|\, T)$. The temporal pertinence of a triple is a subset of the multidimensional time domain which is represented by a *temporal element* [1, 9], that is a disjoint union of multidimensional temporal intervals, each one obtained as the Cartesian product of one time interval for each of the supported temporal dimensions:

$$T = \bigcup_{1 \le j \le m} \mathcal{I}_j = \bigcup_{1 \le j \le m} [t_j^s, t_j^e)_1 \times [t_j^s, t_j^e)_2 \times \cdots \times [t_j^s, t_j^e)_N$$

where the unioned N-dimensional intervals are all disjoint (i.e. $\mathcal{I}_j \cap \mathcal{I}_k = \varnothing$ for all $1 \leq j < k \leq m$).

This definition of temporal triple is similar to the ones in [8, 11], except that multiple time dimensions and, in particular, temporal element timestamping has been introduced here.

A multi-temporal RDF database can then be defined as a set of timestamped RDF triples:

$$\text{RDF-TDB} = \{\, (s, p, o \,|\, T) \,|\, T \subseteq \mathcal{T} \,\}$$

with the integrity constraint:

$$\forall (s, p, o \,|\, T), (s', p', o' \,|\, T') \in \text{RDF-TDB}\text{: } s = s' \wedge p = p' \wedge o = o' \Longrightarrow T = T'$$

which requires that no value-equivalent distinct triples exist.

The adoption of timestamps made-up of temporal elements instead of (multi-temporal) simple intervals avoids the duplication of triples in the presence of a temporal pertinence with a complex shape. For example, consider a tridimensional time domain and a temporal triple $X_1 = (s_1, p_1, o_1 \,|\, T_1)$, where $T_1 = [t', \text{UC}) \times [t', \text{UC}) \times [t', \text{UC})$, which must be partially replaced by a new triple $X_2 = (s_2, p_2, o_2 \,|\, T_2)$, where $T_2 = [t'', \text{UC}) \times [t'', \text{UC}) \times [t'', \text{UC})$ and $t' < t''$. The time pertinence left to X_1 after the modification is $T_1 \backslash T_2$, which can be decomposed into a minimal number of three non-overlapping tridimensional intervals (e.g. $[t', \text{UC}) \times [t', \text{UC}) \times [t', t'') \cup [t', \text{UC}) \times [t', t'') \times [t'', \text{UC}) \cup [t', t'') \times [t'', \text{UC}) \times [t'', \text{UC})$). Hence, if we used interval timestamps, at least three copies of the triple with contents (s_1, p_1, o_1) would be required to cover the region. With temporal elements indeed, we make the union of such intervals and produce a single temporal element to timestamp a single copy of X_1. In other words, we store different triple versions only once with a complex timestamp rather than storing multiple copies of them with a simple timestamp as in [8, 11, 17].

The memory saving we obtain grows with the dimensionality of the time domain, but it can even be appreciated with a monodimensional time domain, when the temporal pertinence of a triple is not a convex interval. For example, the temporal triples:

$$(s, p, o \,|\, [t_1, t_2))$$
$$(s, p, o \,|\, [t_3, t_4))$$

where $t_2 < t_3$, can be merged with temporal element timestamping into a single triple:

$$(s, p, o \,|\, [t_1, t_2) \cup [t_3, t_4))$$

Whereas the same space is basically required for globally storing the timestamps in both cases (i.e. the space needed by four time points), the space required for storing one occurrence of the triple contents (s, p, o) is saved in the latter case. Moreover, with element timestamping, according to the integrity constraint introduced above, no two temporal triples can have the same non-temporal contents and, thus, checking of uniqueness and of functional properties constraints

can be performed more easily. In particular, in Sec. 4 we will define the semantics of modification operations in such a way that the integrity constraints concerning temporal elements are automatically preserved.

The only retrieval operations we consider in this paper are *snapshot queries* [9], which are used to extract a single temporal version from a multi-version ontology encoded as a multi-temporal RDF database. The result of a snapshot query is a standard (non-temporal) RDF graph, which can be interpreted as the desired ontology version. Given a multi-dimensional time point $\bar{t} = (t_1, t_1, \ldots, t_N) \in \mathcal{T}$, we can define the snapshot valid at $\bar{t}$ as:

$$\text{RDF-TDB}(\bar{t}) = \{\, (s, p, o) \mid (s, p, o \,|\, T) \in \text{RDF-TDB} \wedge \bar{t} \in T \}$$

Using a SPARQL-like syntax [15] with temporal extensions, the snapshot query above could be expressed, for instance, via a statement like the following:

```
CONSTRUCT { ?s ?p ?o }
WHERE { TGRAPH <http://myExample.org/tGraph> { ?s ?p ?o | ?t } .
        FILTER ?t CONTAINS "(t1, t1, ..., tN)" . }
```

where the URI `http://myExample.org/tGraph` denotes a multi-temporal RDF triple store. However, the full definition of a multi-temporal extension of SPARQL is beyond the scope of this paper.

4 Modification Operations

In this section, we introduce modification operations for the maintenance of a multi-temporal RDF database and formalize their operational semantics via a sort of triple calculus analogous to the tuple calculus used for the relational data model. Algorithms for their execution can be derived from such description in a straightforward way, although there were room left for optimization. However, as any modification usually impacts on a few temporal triples at a time, optimization of their execution should not be a real issue. As far as syntax is concerned, also for temporal modification operations we do not provide a complete SPARQL-like syntax, although the examples we provide are inspired by the SPARQL/Update proposal [16]. The contribution is a direct application to temporal RDF of the research work done in designing temporal query languages for relational and XML databases [3, 14].

In an N-dimensional temporal model, users can assign a validity to the effects of a modification by specifying a value for $N-1$ of the temporal dimensions. Like in temporal query languages as TSQL2 [14], we assume this can be done through a `VALID` clause added to the syntax of modification statements. Hence, in the following examples, we assume the validity of the modification tv expressed as an $(N-1)$-dimensional temporal element, in the most general case. The N-th dimension is transaction time, which is automatically managed by the system in a transparent way (most users may actually ignore its existence). The transaction time of any modification is always, by definition, $[\text{NOW}, \text{UC})$, where NOW is the transaction time when the modification is executed.

Notice that the union and difference between two temporal elements involved in timestamp update operations must be computed carefully in order to preserve the temporal element format of the result [3]. If $P_i = \bigcup_{1 \le j \le m_i} \mathcal{I}_{ij}$, where $\mathcal{I}_{ij}$ are multidimensional intervals, are two temporal elements ($i = 1, 2$), then the difference $P_1 \setminus P_2$ can be computed as $\bigcup_{1 \le j \le m_1} \mathcal{I}_{1j} \setminus P_2$ and it is a temporal element if $\mathcal{I}_{1j} \setminus P_2$ is a temporal element for each j. To this end, each interval $\mathcal{I}_{1j}$ must be cut into a smaller one or split into two or more multidimensional intervals if the non-overlapped region has a complex (non-rectangular) shape. If such splitting only produces non-overlapping intervals, the result is a temporal element. The union $P_1 \cup P_2$ can indeed be computed as $P_1 \cup (P_2 \setminus P_1)$ or $(P_1 \setminus P_2) \cup P_2$ in order to ensure that the result is always a temporal element. Eventually, coalescence [9] has to be applied to possibly merge multidimensional adjacent intervals in order to minimize their number in the resulting element. Although coalescence in an N-dimensional space is, in general, quite complex, algorithms for maintaining a minimal decomposition of timestamps after a modification operation can easily be defined with any number of dimensions (intervals which must be considered as candidates for coalescence are a quite limited number in this case).

4.1 Insertion

Assuming tv specifies an $(N-1)$-dimensional temporal element (transaction time is implied), the generic statement for the insertion of a temporal triple (s, p, o) with validity tv into the temporal RDF database can be written as:

```
INSERT DATA
{s, p, o} VALID tv
```

Semantics The effect of the insertion can be specified by the following triple calculus expression, where RDF-TDB and RDF-TDB′ are the temporal RDF database state before and after the modification, respectively:

$$\begin{aligned}\text{RDF-TDB}' = \text{RDF-TDB}\\ &\cup \big\{(s,p,o \,|\, T') \;\big|\; \exists (s,p,o \,|\, T) \in \text{RDF-TDB}\\ &\qquad\qquad \wedge\, T' = \mathit{coalesce}(T \cup tv \times [\textsc{now}, \textsc{uc}))\big\}\\ &\cup \big\{(s,p,o \,|\, tv \times [\textsc{now}, \textsc{uc})) \;\big|\; \neg\exists (s,p,o \,|\, T) \in \text{RDF-TDB}\big\}\end{aligned}$$

In particular, if a temporal triple $(s, p, o \,|\, T)$ with non-temporal contents (s, p, o) is already present in the database, then a new timestamp for it is computed as $T' = \mathit{coalesce}(T \cup tv \times [\textsc{now}, \textsc{uc}))$ and finally the temporal triple is replaced by $(s, p, o \,|\, T')$ in the database; otherwise, the triple $(s, p, o \,|\, tv \times [\textsc{now}, \textsc{uc}))$ is simply added to the database.

4.2 Deletion

Assuming tv specifies an $(N-1)$-dimensional temporal element (transaction time is implied), the generic statement for the deletion with validity tv from the

temporal RDF database of all the temporal triples (s, p, o) satisfying a selection predicate *pred* can be expressed as:

```
DELETE
{s, p, o} VALID tv
WHERE pred(s, p, o)
```

Semantics The effect of the deletion can be formalized in triple calculus as follows:

$$\begin{aligned}\text{RDF-TDB}' = \text{RDF-TDB}&\\ \setminus \big\{(s,p,o \,|\, T) \;\big|\; &\exists (s,p,o \,|\, T) \in \text{RDF-TDB}\\ &\wedge \text{pred}(s,p,o) \wedge T \cap tv \times [\text{NOW}, \text{UC}) \neq \varnothing\big\}\\ \cup \big\{(s,p,o \,|\, T') \;\big|\; &\exists (s,p,o \,|\, T) \in \text{RDF-TDB}\\ &\wedge \text{pred}(s,p,o) \wedge T \cap tv \times [\text{NOW}, \text{UC}) \neq \varnothing\\ &\wedge T' = \mathit{coalesce}(T \setminus tv \times [\text{NOW}, \text{UC}))\big\}\end{aligned}$$

The effect is that for each temporal triple $(s, p, o \,|\, T)$ in the database such that the selection predicate $\text{pred}(s, p, o)$ evaluates to true and $T \cap tv \times [\text{NOW}, \text{UC}) \neq \varnothing$, the triple $(s, p, o \,|\, T)$ is replaced by the triple $(s, p, o \,|\, T')$ with equal contents and timestamp restricted to $T' = \mathit{coalesce}(T \setminus tv \times [\text{NOW}, \text{UC}))$.

4.3 Modification

Assuming tv specifies again an $(N-1)$-dimensional temporal element (transaction time is implied), the generic statement for the substitution with validity tv of all the temporal triples (s, p, o) satisfying a selection predicate *pred* with the triple (s', p', o') can be expressed as:

```
UPDATE {s, p, o}
SET {s', p', o'} VALID tv
WHERE pred(s, p, o)
```

whose effect is equivalent to the execution in the same transaction of a `DELETE` followed by an `INSERT` defined as follows:

```
DELETE
{s, p, o} VALID tv
WHERE pred(s, p, o) ;
INSERT DATA
{s', p', o'} VALID tv
```

While the `INSERT` and `DELETE` statements can be considered the only necessary primitive operations, also more complex modification instructions (e.g. involving multiple or computed insertions, manipulation of RDF graph stores [16], etc.) could be easily defined.

4.4 Ontology changes

High-level ontology change operations (e.g. to add a new property to a class or an inheritance link to a class or to a property) can also be defined in terms of the manipulation primitives presented in this Section.

For instance, considering lightweight RDFs ontologies, a `ADD_PROPERTY`(*Class, Property, Range, Validity*) operator can be introduced to add a new property with a given range to a class. It can be defined as:

```
INSERT DATA
{ Property rdfs:domain Class ;
           rdfs:range  Range . } VALID Validity
```

A `CHANGE_PROPERTY_RANGE`(*Property, NewRange, Validity*) operator, to be used to change the range of a property, can be defined as:

```
UPDATE { Property rdfs:range ?range }
SET { Property rdfs:range NewRange } VALID Validity
```

Finally, a `DEL_PROPERTY`(*Class, Property, Validity*) operator can be introduced to delete a property of a class, defined as:

```
DELETE
{ Property rdfs:domain Class ;
           rdfs:range  ?range . } VALID Validity
```

Example In order to see an example of their application, let us consider a bitemporal domain and assume to (pro-actively) define on 1989/12/1 an ontology valid from 1990 in the current RDF database. Hence, we consider a property P with range $R1$ to be added to a class C in this version. This can be done by means of the command which follows:

```
ADD_PROPERTY(ex:C, ex:P, ex:R1, FROM '1990-01-01')
```

Assuming the statement was executed on 1989/12/1, its execution caused the temporal triples which follow:

$$(\texttt{ex:P rdfs:domain ex:C} \,|\, [1990/1/1, \text{UC}) \times [1989/12/1, \text{UC}))$$
$$(\texttt{ex:P rdfs:range ex:R1} \,|\, [1990/1/1, \text{UC}) \times [1989/12/1, \text{UC}))$$

to be added to the bitemporal RDF database.

Then we consider the creation of a new ontology version valid from 2005, where the range of property P is actually $R2$, for which the following command has been used (on 2005/1/1):

```
CHANGE_PROPERTY_RANGE(ex:P, ex:R2, FROM '2005-01-01')
```

After its execution, the triples above are replaced in the new database state by:

$$(\texttt{ex:P rdfs:domain ex:C} \,|\, [1990/1/1, \text{UC}) \times [1989/12/1, \text{UC}))$$
$$(\texttt{ex:P rdfs:range ex:R1} \,|\, [1990/1/1, \text{UC}) \times [1989/12/1, 2005/1/1) \cup [1990/1/1, 2005/1/1) \times [2005/1/1, \text{UC}))$$
$$(\texttt{ex:P rdfs:range ex:R2} \,|\, [2005/1/1, \text{UC}) \times [2005/1/1, \text{UC}))$$

Finally, we consider the (retro-active) creation of a new ontology version valid from 2009, where property P is deleted from class C. This can be done by executing (now) the statement:

```
DEL_PROPERTY(ex:C, ex:P, FROM '2009-01-01')
```

The database state after its execution contains the triples which follows:

$$(\texttt{ex:P rdfs:domain ex:C} \,|\, [1990/1/1, \text{UC}) \times [1989/12/1, \text{UC}))$$
$$(\texttt{ex:P rdfs:range ex:R1} \,|\, [1990/1/1, \text{UC}) \times [1989/12/1, 2005/1/1) \cup [1990/1/1, 2005/1/1) \times [2005/1/1, \text{UC}))$$
$$(\texttt{ex:P rdfs:range ex:R2} \,|\, [2005/1/1, \text{UC}) \times [2005/1/1, \text{NOW}) \cup [2005/1/1, 2009/1/1) \times [\text{NOW}, \text{UC}))$$

As a result, in the current database state (i.e. as of NOW w.r.t. transaction time), property P has range $R1$ in the ontology version valid from 1990 to 2005 and range $R2$ in the version valid from 2005 to 2009. The domain of P is class C in both such ontology versions. P is no longer a property of class C in the ontology version valid from 2009.

Similar macro commands can also easily be defined to manipulate versions of serialized RDF representations of OWL ontologies.

5 Conclusions

In this paper, we presented a multi-temporal RDF database model employing triple timestamping with temporal elements, which best preserves in the multi-temporal setting the scalability property enjoyed by triple storage technologies.

The data model has been equipped with a snapshot extraction operator and modification operators, whose semantics has been provided, which allow knowledge engineers to maintain a multi-temporal RDF database in order to manage temporal versions of an ontology.

Some of the effected design choices were motivated by the application requirements of an ontology-based personalization service of a multi-version repository of legal documents (or clinical guidelines). It is our intention to explore the applicability of our approach in application fields with more generic requirements in future work.

In future research, we will also consider extensions of the proposed RDF database model, including the development of a complete multi-temporal SPARQL-like query language and the adoption of suitable multidimensional index structures to efficiently support the execution of temporal queries.

References

1. S. Gadia. A homogeneous relational model and query language for temporal databases, *ACM Transactions on Database Systems*, 13(3):418–448, 1998.
2. F. Grandi, F. Mandreoli, M.R. Scalas, and P. Tiberio. Management of the citizen's digital identity and access to multi-version norm texts on the semantic web. In *Proc. of IPSI-Pescara Symposium*, IPSI, 2004.
3. F. Grandi, F. Mandreoli, and P. Tibero. Temporal modelling of normative documents in XML format. *Data & Knowledge Engineering*, 54:327–354, 2005.
4. F. Grandi, F. Mandreoli, R. Martoglia. Issues in personalized access to multi-version XML documents. In E. Pardede, editor, *Open and Novel Issues in XML Database Applications*, pages 199–230. IGI Global, 2009.
5. F. Grandi, F. Mandreoli, R. Martoglia, E. Ronchetti, M.R. Scalas, and P. Tiberio. Ontology-based personalization of E-government services. In C. Mourlas and P. Germanakos, editors, *Intelligent User Interfaces*, pages 167–203. IGI Global, 2009.
6. F. Grandi, F. Mandreoli, and R. Martoglia. Ontology-based personalization of clinical guidelines. *In preparation.*
7. F. Grandi, and M.R. Scalas. The Valid Ontology: A simple OWL temporal versioning framework. In *Proc. of SEMAPRO Conf.*, IEEE Computer Society, 2009 (*in press*).
8. C. Gutierrez, C. Hurtado and A. Vaisman. Introducing time into RDF. *IEEE Transactions on Knowledge and Data Engineering*, 19(2):207–218, 2007.
9. C. S. Jensen, C. E. Dyreson (Eds.) et al. The consensus glossary of temporal database concepts - February 1998 version. In O. Etzion, S. Jajodia, and S. Sripada, editors, *Temporal Databases — Research and Practice*, pages 367–405. Springer-Verlag, 1998. LNCS No. 1399.
10. Web Ontology Language. W3C Consortium, `http://www.w3.org/2004/OWL/`.
11. A. Pugliese, O. Udrea, and V.S. Subrahmanian. Scaling RDF with Time. In *Proc. of WWW Conf.*, pages 605–614, ACM Press, 2008.
12. Resource description framework. W3C Consortium, `http://www.w3.org/RDF/`.
13. K. Rohloff, M. Dean, I. Emmons, D. Ryder and J. Summer. An evaluation of triple-store technologies for large data stores. In *Proc. of OTM Workshops*, pages 1105–1147. Springer-Verlag, 2007. LNCS No. 4806.
14. R.T. Snodgrass (ed.), I. Ahn, G. Ariav, D. Batory, J. Clifford, C.E. Dyreson, R. Elmasri, F. Grandi, C.S. Jensen, W. Käfer, N. Kline, K. Kulkarni, T.Y. Cliff Leung, N. Lorentzos, R. Ramakrishnan, J.F. Roddick, A. Segev, M.D. Soo, S.M. Sripada. *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers, 1995.
15. SPARQL query language for RDF. W3C Consortium, `http://www.w3.org/TR/rdf-sparql-query/`.
16. SPARQL/Update. A language for updating RDF graphs. Jena wiki, `http://jena.hpl.hp.com/~afs/SPARQL-Update.html`.
17. J. Tappolet, and A. Bernstein. Applied temporal RDF: Efficient temporal querying of RDF data with SPARQL. In *Proc. of ESWC Conf.*, pages 302–322. Springer-Verlag, 2009. LNCS No. 5554.