Temporal Modelling and Management of Normative Documents in XML Format ⋆

Fabio Grandi [a,*], Federica Mandreoli [b], Paolo Tiberio [b]

[a] *IEIIT.BO-CNR and Dipartimento di Elettronica, Informatica e Sistemistica, Alma Mater Studiorum - Università di Bologna, Viale Risorgimento 2, I-40136, Bologna, Italy*

[b] *Dipartimento di Ingegneria dell'Informazione, Università di Modena e Reggio Emilia, Via Vignolese 905/b, I-41100, Modena, Italy*

Abstract

In this paper, we present the results of a research project concerning the temporal management of normative texts in XML format. In particular, four temporal dimensions (publication, validity, efficacy and transaction times) are used to correctly represent the evolution of norms in time and their resulting versioning. Hence, we introduce a multiversion data model based on XML schema and define basic mechanisms for the maintenance and retrieval of multiversion norm texts. Finally, we describe a prototype management system which has been implemented and evaluated.

Key words: Temporal database, Web information management, Legal information system, XML

1 Introduction

Time is one of the main aspects characterizing several real world facets and phenomena. The ability to model the temporal dimension of the real world and to respond within time constraints to changes in the real world as well as to application-dependent operations is essential to many computer applications. The management of norms represents one of such applications as temporal concerns are ubiquitous

⋆ This work has been partially supported by the MIUR-40% Project: "La dinamica della norma nel tempo: aspetti giuridici ed informatici".

* Corresponding author. Tel.: +39-051-2093555, fax: +39-051-2093540.
Email addresses: fgrandi@deis.unibo.it (Fabio Grandi), mandreoli.federica@unimo.it (Federica Mandreoli), tiberio.paolo@unimo.it (Paolo Tiberio).

in the law domain [32]. With the term “norm” or “normative document” we intend in this paper any kind of Law, Act, Decree, Provision, Regulation etc. with statutory effects as officially produced by lawgiving and government activities. Time in normative systems has become a central topic of the cultural and political debate and is of fundamental concern to legal informatics. The law is under increasing pressure to keep pace with social change: normative texts and amendments follow one another in time and get overlapped. Moreover, the route to e-Government (e.g. [38,15]) pushes administrations for providing access to services for publication and exchange of norms on the Web.

In the context of database research, the management of time has been extensively studied in the last decades [36,14]. In particular, many efforts have been devoted to add time support to database models and system functionalities. Temporal database systems provide special facilities for storing, querying, and updating historical and/or future data. In this context, two time dimensions are usually considered: valid time and transaction time [22]. Valid time is the time of the real world and denotes the time a fact is true in reality. Transaction time is the time of the system and denotes the time during which the fact is present in the database as stored data. In order to make a more complete picture, two other temporal dimensions have been considered useful for advanced applications: event time [25] (also called decision time in [28]), which is the occurrence time of events that initiate and/or terminate the validity of some fact in the real world, and availability time [9], which is the time some fact is available in the information system. By the way, the first relational query language providing for temporal queries (only considering valid time), that is LEGOL 2.0 [23] which appeared in the 1970’s, was developed to support legal applications.

Moreover, in the database research community, there is a much current interest in representing and querying semi-structured data. For example, database-resident data can be published as static or dynamic XML documents, which can then be viewed on Web browsers and processed by various Web-based applications, also executing queries written in languages such as XPath [40] and XQuery [41]. As a consequence, several works took into account change, versioning, evolution and also explicitly temporal aspects, in semi-structured and XML-based data management [19,18], often applying conceptual tools and techniques developed by temporal database research. However, such approaches are not straightforwardly applicable to the legal application domain because of the specificity of the data semantics and operation requirements. In the context of legal computer science, previous approaches already dealt with the reconstruction of *consolidated* norm texts, consisting of their current temporal version [33]. One temporal dimension was usually considered in such approaches.

In this paper, we present the results of the research activity we carried out in the context of the multi-disciplinary project “The dynamics of norms in time: legal and informatics aspects” co-funded by the Italian Ministry of University. Such a

project emphasizes, from a legal point of view, the necessity for a rigorous, effective and efficient management of time-varying norm texts. In this context, the main objective of our work has been the development of a computer system for the temporal management of multiversion norms represented as XML documents and made available on the Web. To this end, we developed a temporal XML data model which uses four time dimensions to correctly represent the evolution of norms in time and their resulting versioning. The model, which will be presented in Section 2, is equipped with basic operators for the modification and retrieval of multiversion norm texts, which will be described in Sections 3 and 4, respectively. For the efficient exploitation of the data model, a system prototype has been implemented and evaluated as described in Section 5. Preliminary results were also presented in [21]. Related works are discussed in Section 6, whereas conclusions can be found in Section 7.

2 The temporal data model

In this section, we present the XML-based temporal data model we propose for the representation and management of versioned normative texts. The model supports multiple temporal dimensions, all involved in the law application lifecycle.

The existence of multiple versions of norms is a consequence of the dynamics of the legislative activity. Assume a norm N1 has been in force since 2000. Further assume that a norm N2, which modifies norm N1 by replacing its Article 1, was passed in 2002 (for this modification, N2 is called the *active norm*, while N1 is called the *passive norm*). After the modification takes effect, N1 has two versions: the former, say v1, corresponding to its initial text, and the latter, say v2, with the new contents of Article 1 as replaced by N2. Both versions are important from an application point of view. Of course, the most important version of N1 is v2, which corresponds to the form in which N1 currently belongs to the regulations and, thus, must be enforced now (such a version, produced by the application of all the modifications the norm underwent so far, is the one called *consolidated version*). However, also v1 is of an utmost application importance *now*: if a Court has to pass judgment today on some fact committed in 2001, the version of N1 which must be applied to the case is the one that was in force then, that is v1. Hence, a legal information system should be able to retrieve or reconstruct on demand any version of a given norm to meet common application requirements.

Moreover, a realistic scenario is far more complex than the one described above, because of the coexistence of multiple interacting time dimensions which rule the actual life of versions. For example, although N2 has been in force and its modification on N1 has taken effect since 2002 (so that actually v2 has *validity* from 2002), it might be the case that the modified Article 1 explicitly contains the sentence: “The present Article becomes effective from 2003”. In such a case, v2 must

not be applied during 2002: though valid, it has no *efficacy* yet. Hence, v1 must still be applied during 2002, whereas v2 starts to be actually applied from 2003 only. Therefore, validity and efficacy of versions are different time notions in the law field.

Notice that similar cases, although they might seem quite odd, are not such unusual indeed. For instance, in order to prepare the national regulations to the introduction of the Euro currency (whose circulation officially has begun on 2002, January 1st), several norms concerning the Euro were issued in Italy and in other European countries in 2000 and 2001. Although such norms came into force before 2002, they all explicitly stated that some of their provisions were effective from 2002 only. By the way, some of such norms were also modified, even several times, before they became applicable (and eventually some of their versions have never been applied at all).

Last but not least, when norms are stored in a legal information system, another specific time dimension comes into action. For example, it might be the case that a public servant takes a wrong decision in settling an affair by following the provisions of a norm he/she retrieves from the system, if the returned consolidated version is actually out-of-date: the decision was taken while a modified version of the norm was already in force, whereas the modification has been recorded in the system only later (retroactively). Hence, *transaction* time is needed to keep track of the update activity in the system and, in our case, to ascertain *a posteriori* that the correct version was stored retroactively and, thus, the public servant acted in good faith.

Therefore, in order to draw a complete picture and meet all application requirements, the time dimensions we consider are the following:

Publication time. It is the time of publication of the norm on the Official Journal. It has the same semantics as event time (and availability time, as the two time dimensions, in such a context, coincide). It is a global and unchangeable property for the whole norm contents and, thus, it will be treated separately.

Validity time. It is the time (some part of) the norm is in force (in the Italian regulations, usually a norm is in force from the publication date plus 15 days on, until its validity is changed by a subsequent act). It has the semantics of valid time, as it represents the time the norm actually belongs to the regulations in the real world.

Efficacy time. It is the time (some part of) the norm can be applied to a concrete case. It usually corresponds to the validity of norms, but it can be the case that an abrogated norm continues to be applicable to a limited number of cases. Until such cases cease to exist, the norm continues its efficacy in the real world though no longer in force.

Transaction time. It is the time (some part of) the norm is stored in a computer system. Obviously, it has the same semantics of transaction time as in temporal

databases.

The first time dimension, publication time, is a global property of the document which cannot be changed after publication and, thus, is not involved in the versioning mechanism. Disregarding publication time, the other three dimensions above are "orthogonal" as far as document versioning is concerned. Notice that validity and efficacy time both have the semantics of valid time but represent different and independent valid time notions. As a matter of fact, validity and efficacy time come out generally independent as a consequence of the legislative practice (although validity and efficacy are *usually* in the future with respect to the time the modifications are applied, retroactive modifications cannot be excluded *a priori*). Moreover, transaction time depends on a completely independent information system maintenance activity (i.e. versions can be stored retro- or even pro-actively with respect to their validity/efficacy).

2.1 Representation of Time and Multiversion Norms

As for many other countries, the textual structure of Italian norms is based on a contents-section-article-paragraph hierarchy. Our data model encodes the hierarchical organization of normative texts into the tree-like inner structure of XML documents conforming to an XML schema [42]. Such an encoding is enriched with timestamping metadata modelling the temporal aspects of normative texts.

Our temporal model supports lossless updates at any level of the hierarchy by means of *temporal versions* representing the results of the changes normative texts undergo. The *timestamp* of a version represents its *temporal pertinence* as a subset of the tridimensional space validity $\times$ efficacy $\times$ transaction. The temporal pertinence of a version can be represented by a *temporal element* [16,22], that is a disjoint union of tridimensional time intervals, each obtained as the Cartesian product of one time interval (open to the right) for each of the supported temporal dimensions. Since any norm version is potentially subject to changes with respect to all the three time dimensions, we will express right-unlimited time intervals as $[t, \mathrm{UC})$, where UC means "Until Changed", though such a symbol is often used in the temporal database literature [22] for transaction time only (whereas, e.g. "forever" or ∞ is used for valid time). Actually, norms in force usually have a continued validity and efficacy until such properties are changed by a subsequent act of the legislator. Publication time, which is a constant and global document property, is not included in the version timestamping mechanism.

The adoption of timestamps made up of temporal elements instead of tridimensional intervals avoids the duplication of version contents in the presence of a temporal pertinence with a complex shape. For example, consider a version v1 with initial pertinence $P_1 = [t_1, \mathrm{UC}) \times [t_1, \mathrm{UC}) \times [t_1, \mathrm{UC})$ and a modification produc-

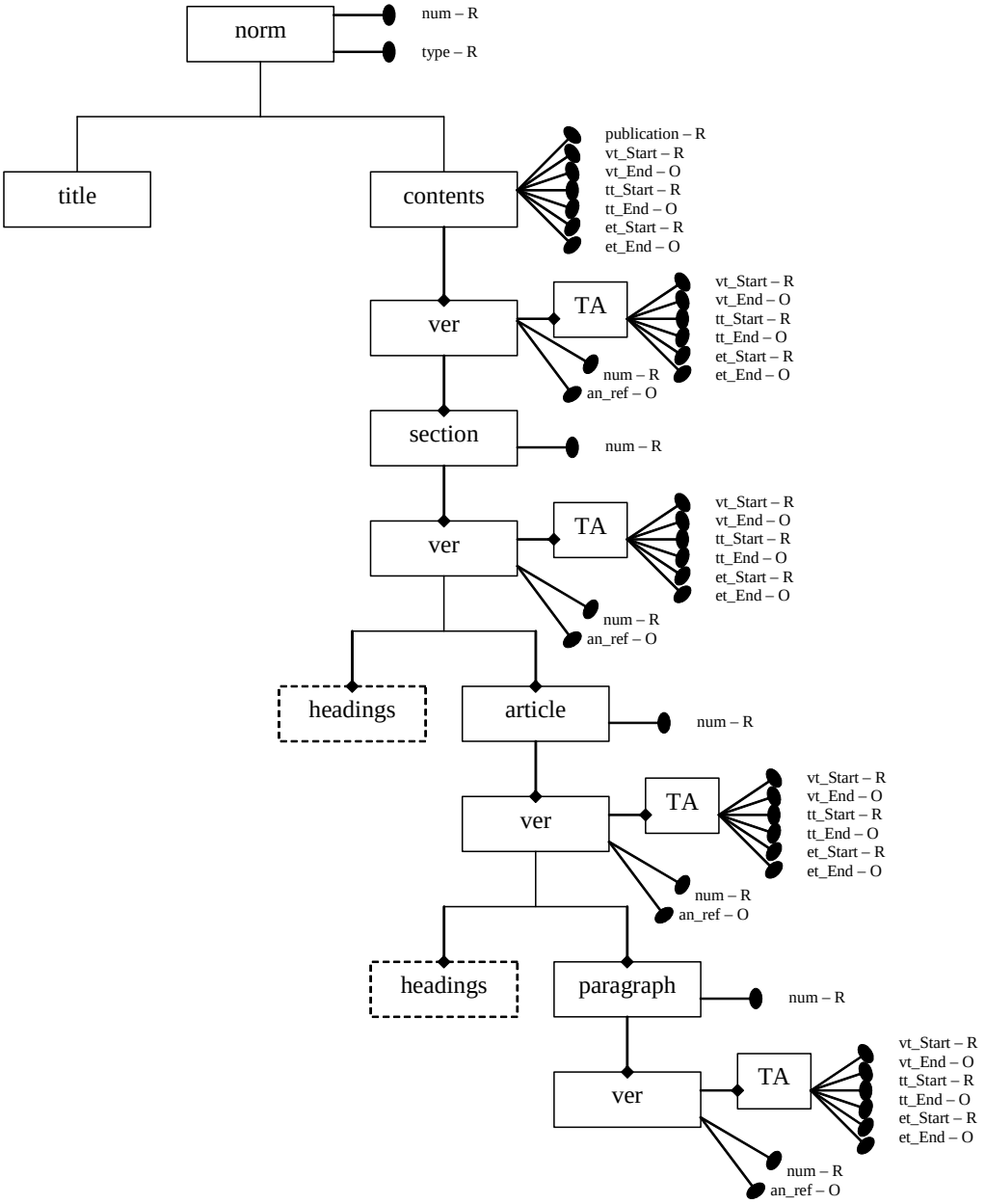


Fig. 1. The XML-schema for the representation of norms in time

ing a new version v2 with pertinence $P_2 = [t_2, \mathrm{UC}) \times [t_2, \mathrm{UC}) \times [t_2, \mathrm{UC})$ (with $t_2 > t_1$). The time pertinence left to v1 after the modification is $P_1 \setminus P_2$, which can be decomposed into a minimal number of three non-overlapping tridimensional intervals (e.g. $[t_1, \mathrm{UC}) \times [t_1, \mathrm{UC}) \times [t_1, t_2)$, $[t_1, \mathrm{UC}) \times [t_1, t_2) \times [t_2, \mathrm{UC})$ and $[t_1, t_2) \times [t_2, \mathrm{UC}) \times [t_2, \mathrm{UC})$). Hence, if we used interval timestamps, at least three copies of v1 would be required to cover the region. Indeed, we make the union of such intervals and produce a single temporal element to timestamp a single copy of v1. In other words, we preferred to store different versions only once with a complex timestamp rather than storing multiple copies of them with a simple times-

tamp. We will show in Section 5 how such a design choice paid off in terms of improved retrieval performance.

As far as the norm hierarchical structure is concerned, timestamps can occur at any level of the hierarchy and obey to an ancestor-descendant inheritance semantics. In particular, the timestamps of any node are inherited by its descendants, unless redefined. Redefinitions can only involve a restriction of the inherited values. In other words, the timestamps owned by each version must be contained in the timestamps of its parent. Finally, the temporal pertinence of sibling versions must be disjoint, that is at each level at most one version is associated to any tridimensional time point (or *chronon* [22]). A typical example of the inheritance and redefinition mechanism, which actually comes out of the modification dynamics, is the following. Let us consider a node with a single version v with time pertinence P, which is initially inherited by all the (single-version) descendant nodes of v, once created. If one child of v is then modified, its single version v′ with pertinence P is "replaced" by two versions, v′ with pertinence P' and v″ with pertinence P'', where v″ and P'' are the outcome and the temporal pertinence of the modification, respectively, and $P' \cup P'' = P$. In general, the inheritance and redefinition mechanism can be formalized as follows:

Definition 1 (inheritance) *If the version v with pertinence P of a node n has m children $n_1, n_2, \ldots, n_m$, where child n_i has k versions v_{i1},$v_{i2}, \ldots$,v_{ik} with pertinence $P_{i1}, P_{i2}, \ldots, P_{ik}$, respectively, we have, for all i, the following constraints:*

$$\bigcap_{1 \le j \le k} P_{ij} = \emptyset \qquad \textit{(pertinence disjunction)}$$

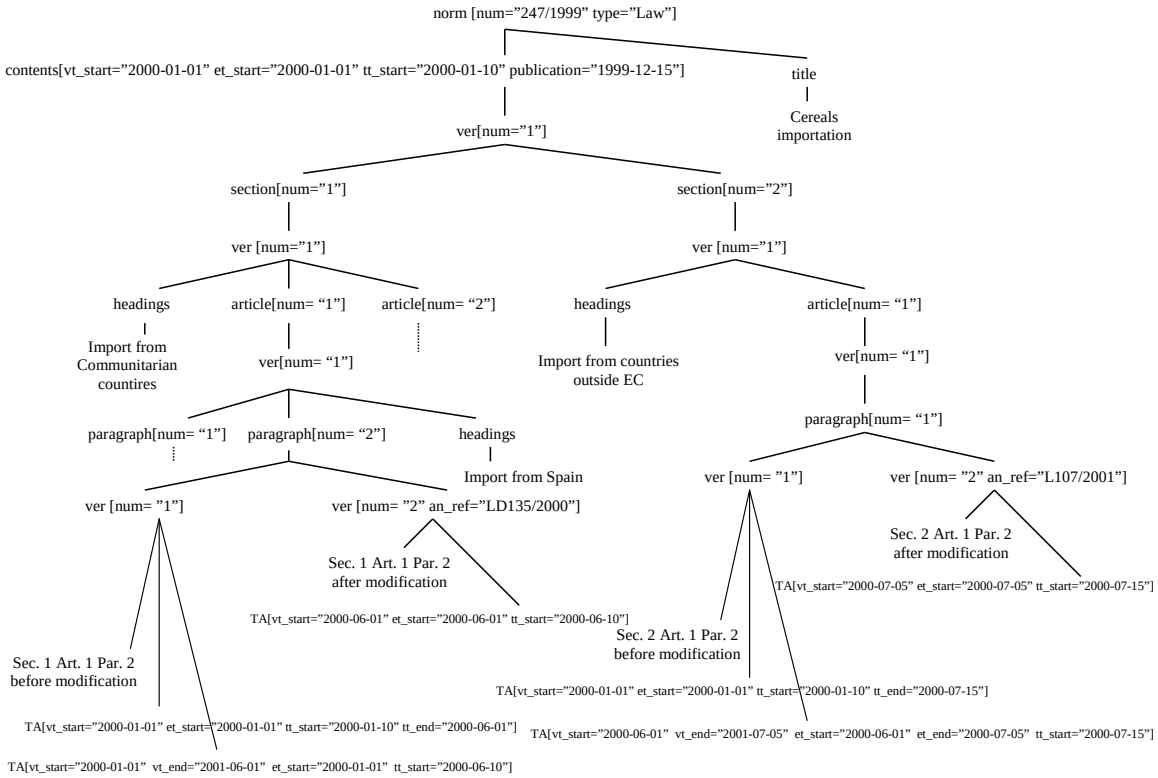
$$\bigcup_{1 \le j \le k} P_{ij} \subseteq P \qquad \textit{(pertinence inclusion)}$$

□

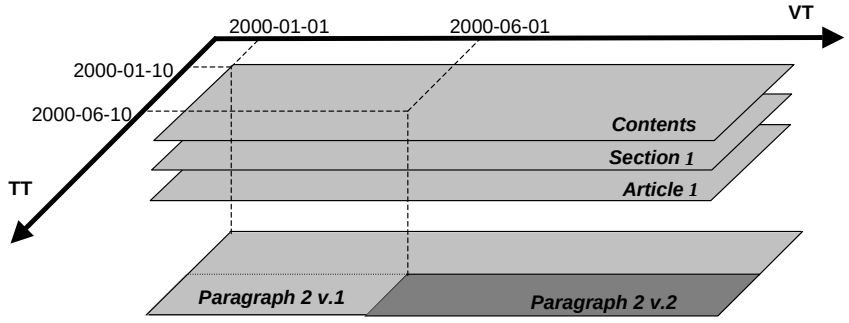
Similar relationships, which extend the hierarchical organization of XML documents with temporal semantics, have also been adopted by other authors (e.g. [1,17]).

The XML Schema [42] we are going to introduce comes out as a quite unfaithful translation and extension of one of the DTDs published by the "Norma in Rete" (Norm on Network) [29] working group. The "Norma in Rete" initiative has been jointly promoted by the Italian agency for the introduction of information technology in public administrations (AIPA) and the Ministry of Justice, and is framed to supply the missing link between information technology and the public administration in the field of legimatics. The project has a major role in trying to overcome the fragmentation affecting online availability of norms by means of the introduction of standards, mainly based on XML-related technologies, for publication and exchange of norms on the network.

The full version of our XML schema is depicted in Figure 1, where "R" and "O"



(a) The tree-like structure of a sample document



(b) Hierarchical representation of temporal pertinence

Fig. 2. An example of multiversion XML document

symbols near attribute names stand for “required” and “optional”, respectively, and one-to-many relationships are denoted with arrows ending with a diamond. The dashed boxes represent optional elements (`headings`). As norms are identified by a (type, number) pair (e.g. type=“Law”, number=“27/2003”) the meta-level of normative texts is rooted at a `norm` top element, characterized by `type` and `num`(ber) attributes, which includes the `title` and `contents` elements. The `contents` element has attributes defining global temporal properties: an attribute `publication` storing the publication date of the norm and also temporal at-

tributes which define a tridimensional bounding box for all the timestamps the document contains and which is used as a summary temporal pertinence of the whole norm for faster query processing. Then, at each level of the contents-section-article-paragraph hierarchy, it is possible to represent one or more temporal versions by means of the `ver` elements, whose attribute `num` represents the version number whereas `an_ref` is the reference to the active norm whose enforcement caused the versioning. Each version is characterized by a temporal element-type timestamp, which is defined by the union of the `TA` XML elements the version contains. Each `TA` element represents a tridimensional time interval whose boundaries are encoded as `TA` attribute values: validity (`vt_start` and `vt_end`), efficacy (`et_start` and `et_end`), and transaction time (`tt_start` and `tt_end`). The "end" attribute associated to each dimension (e.g. `vt_end`) is optional and its absence represents a UC value denoting, thus, a right-unlimited interval. For instance, the XML element: `<TA vt_start="2000-01-01" vt_end= "2003-12-31" et_start= "2000-01-01" tt_start="1999-12-20" />` represents the tridimensional interval: $[2000/01/01, 2003/12/31) \times [2000/01/01, \text{UC}) \times [1999/12/20, \text{UC})$.

Notice that the XML Schema encodes only those aspects of our model which are directly supported by an off-the-shelf XML document validator. Advanced features of the model, like correctness of the inheritance and redefinition mechanism, cannot be embedded in an XML Schema definition and are explicitly enforced by our management application, thanks to a careful definition of the semantics of the modification operators.

Example 1 Figure 2.a shows the tree-like structure of a document conforming to the temporal XML schema. The Law 247/1999 concerns the cereal importation and contains two sections, three articles and four paragraphs. It has been published on 1999/12/15 and is valid from 2000/1/1 (it has been recorded in the system on 2000/1/10). Only (two) paragraphs underwent punctual modifications and thus have more than one version. For this reason, all parts but paragraphs inherit the timestamps from the `contents` tag. For the paragraphs, instead, it is necessary to explicit the temporal attributes since they are redefined by the corresponding versions. Paragraph 2 of Sec. 1, Art. 1 has been modified by the "LD135/2000" Legislative Decree, in force since 2000/6/1 (modification recorded on 2000/6/10). Paragraph 1 of Sec. 2, Art. 1 has been modified by the "L107/2001" Law, in force since 2001/7/5 (modification recorded on 2001/7/15).

Notice that, in the former case the old version continues to be applicable (e.g. to the cases for which it was applicable before the modification), whereas in the latter case the modifying Law has stated that the old version is definitely no longer applicable (hence, efficacy time has been stopped to 2001/7/5 like validity). In both cases, the temporal elements correctly map the temporal pertinence of the text before the modification on a tridimensional space (validity $\times$ efficacy $\times$ transaction). In particular, Fig. 2.b shows the projection on the bidimensional space validity $\times$ transaction of the containment relationship (due to inheritance) between the tem-

poral pertinence of the versions in the path to the second paragraph. In the figure, the different levels of the document hierarchy are represented on the z-axis, v. is used in place of version (e.g. Paragraph 2 v. 1 corresponds to the version number 1 of the Paragraph number 2) and the dotted line on the second version of the second paragraph shows how the temporal pertinence has been represented as union of disjoint intervals, each of which corresponds to a TA element. □

3 Managing the dynamics of norms

During its lifespan, a normative text usually undergoes several modifications. As a consequence, the ability to correctly and efficiently managing the dynamics of norms is essential for many legal information systems. On the other hand, the versioning of norms is a quite complex task. Each modification is enforced by an active norm which contains a reference to the portion of the passive norm to be modified and specifies the kind of modification to be applied. Such relationship is known as normative *nexus* as it connects the active with the passive norm. The modifications of interest in the legistic application field include abrogation, textual substitution or integration, prorogation and suspension [34,32]. Moreover, each modification may affect a different portion of the passive norm, ranging from a single word (or even a single punctuation sign) to the whole text contents. Informally, abrogation, substitution and integration consist of the deletion, update and insertion of some text in a norm, respectively. Prorogation and suspension consist of the enlargement and restriction, respectively, of the validity (or efficacy) of some norm portion without affecting the text.

In order to provide a compact but complete, modular support to the management of norm modifications, instead of introducing one operator for each kind of modifications, we equipped the model with two basic operators on which all the modifications of interest can be mapped: $changeText$ and $changeTime$. The former is devoted to implement an explicit textual modification, that is the replacement of (a part of) the contents of a passive norm with a new text, possibly empty. By means of this operator, it is possible to perform abrogations, suspensions, and textual substitutions and integrations. The latter implements pure temporal modifications as it affects the temporal pertinence of (some part of) the passive norm in order to support prorogations. In both cases, the updates are performed in a lossless way by means of an accurate management of the versions and their temporal pertinence making up the history of the document contents. The granularity of versioning is the single node in the XML document tree. In this way, we are able to support modifications occurring at any level of the hierarchical structure of norm texts.

In the following, we will show the algorithms underlying the implementation of the two operators, $changeText$ and $changeTime$, and describe their properties. Before doing it, we introduce some notation and the basic functions used by the

algorithms.

3.1 Notation

The two operators act on norms complying with the XML schema of Fig. 1 and thus on XML documents similar to the example in Fig. 2. In particular, modifications occurring at any level of the tree-like inner structure of documents are supported by performing punctual updates on the nodes of the tree. XML nodes are univocally identified by paths which can be specified by means of the XPath [40] language. For instance, the path expression:

```
/norm[@num="12/2004" and @type="Law"]/section[@num="1"]
```

identifies the node corresponding to the first section of the Law number 12/2004. In order to work on tree nodes, our algorithms make use of the following basic functions: `parent`: $Node \rightarrow Node$, `children`: $Node \rightarrow \{Node\}$, `addChild`: $Node \times Tree \rightarrow Tree$. The `parent`$(q)$ and `children`(q) functions return the parent and the set of children of the node q, respectively, whereas `addChild`(q, t) returns the tree obtained by adding the argument tree t as child of the node q.

As far as the temporal aspects are concerned, two basic functions are necessary for the pertinence management. The function `restrictTimestamps` deletes from the temporal pertinence of a given version the part that overlaps a given temporal element. Moreover, as the inheritance semantics of our temporal model requires that the pertinence of each version is contained in that of its parent, the function is recursively applied to each version in the subtree rooted at the initial version. For instance, in Fig. 3 the restriction of the temporal pertinence of the version

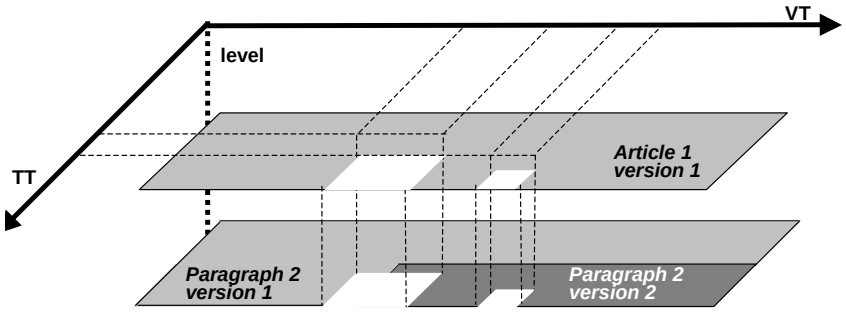


Fig. 3. Effect of the restriction of the temporal pertinence of a version on that of its sub-versions

number 1 of Article 1 caused by the deletion of the part overlapping the temporal element represented by two white rectangles also requires the “propagation” with timestamp restiction to the underlying level(s) corresponding to the versions 1 and 2 of Paragraph 2.

The `restrictTimestamps` code is shown in Fig. 4. Given a version identified

```
function restrictTimestamps(te, $m)
(1)P = ∅;
(2)for each $n in $m/ta
(3)    P = P ∪ [$n/@vt_start,$n/@vt_end)
           ×[$n/@et_start,$n/@et_end) × [$n/@tt_start,$n/@tt_end);
(4)if (P ∩ te ≠ ∅)
(5)  P = coalesce(P \ te);
(6)  set timestamps of $m to P;
(7)  for each $d in children(children($m))
        restrictTimestamps(te,$d);
```

Fig. 4. The `restrictTimestamps` function

by the path `$m` and a temporal element te, in steps 2-3, it determines the temporal pertinence P of the version by incrementally adding the tridimensional intervals embedded in the `ta` tags. If P overlaps te (step 4), step 5 determines the new temporal pertinence obtained by excluding from P the portion overlapped by te and step 6 updates the XML document by updating the timestamps of `$m` with the new temporal pertinence. Notice that the difference between two temporal elements involved in step 5 must be computed carefully in order to obtain another temporal element as a result.

Fact 1 *Let $P_i = \bigcup_{1 \leq j \leq n_i} \mathcal{I}_{ij}$, where all $\mathcal{I}_{ij}$ are multidimensional intervals, be two temporal elements ($i \in \{1, 2\}$). Then the difference $P_1 \setminus P_2$ can be computed as $\bigcup_{1 \leq j \leq n_1} \mathcal{I}_{1j} \setminus P_2$ and it is a temporal element if $\mathcal{I}_{1j} \setminus P_2$ is a temporal element for each j.*

Thus, in the `restrictTimestamps` evaluation, the exclusion of the portion overlapped by te is applied to each tridimensional temporal interval making up P. In this way, each interval of P is simply transformed into a smaller one, or it needs splitting into two or more tridimensional intervals, if the non-overlapped region has a complex shape. If such splitting only produces non-overlapping intervals, the result is a temporal element, according to Fact 1.

For instance, the exclusion of the interval $[5, 7) \times [3, 8) \times [6, \text{UC})$ from $[1, \text{UC}) \times [1, \text{UC}) \times [1, \text{UC})$ can be expressed as $[1, \text{UC}) \times [1, \text{UC}) \times [1, 6) \cup [1, \text{UC}) \times [1, 3) \times [6, \text{UC}) \cup [1, 5) \times [8, \text{UC}) \times [6, \text{UC}) \cup [5, \text{UC}) \times [8, \text{UC}) \times [6, \text{UC})[1, 5) \times [3, 8) \times [6, \text{UC}) \cup [7, \text{UC}) \times [3, 8) \times [6, \text{UC})$, which is a temporal element since the unioned intervals are non-overlapping.

Moreover, in order to minimize the number of intervals after splitting, coalescing [22] can also be applied to merge adjacent temporal intervals. For example,

the decomposition of the difference result above is not optimal, as the intervals $[1, 5) \times [8, \text{UC}) \times [6, \text{UC})$ and $[5, \text{UC}) \times [8, \text{UC}) \times [6, \text{UC})$ can be coalesced into $[1, \text{UC}) \times [8, \text{UC}) \times [6, \text{UC})$ to produce a minimal decomposition [1] . Therefore, coalescing, as effected in step (5) of the algorithm in Fig. 4, minimizes the number of tridimensional intervals for representing a temporal pertinence and, as such, represents a space optimization.

Finally, the function recursively applies itself to the first descending versions (step 7). Notice that they are the grandchildren of `$m` as the XML schema of our model requires a contents/section/article/paragraph node between the given version and each of them. The algorithm avoids accessing the descendants of such versions for which the pertinence does not need updates. More precisely, thanks to the inheritance semantics, if the temporal pertinence of a given version does not overlap te then its first descending versions do not overlap too, as there is a containment relationship, and so on. These observations allows us to state the fact which follows.

Fact 2 *The `restrictTimestamps` function is correct, that is the output is an XML document complying with the temporal model introduced in Sec. 2 and, in particular, with the inheritance semantics.*

Hence the following proposition shows the time complexity of the `restrictTimestamps` function.

Proposition 3.1 *In the worst case, the complexity of the `restrictTimestamps` function is linear in the number of nodes in the XML document in the subtree rooted at `$m`.*

Proof. The `restrictTimestamps` function, given the node `$m` corresponding to the version to be modified, updates the temporal pertinence of `$m` and goes down to the descendants of `$m`, if necessary. Notice that the function works in a depth-first manner and, thus, each node in the subtree rooted at `$m` is visited at most once. Notice that the worst case happens when the restriction has to be propagated up to all the leaves of the subtree rooted at `$m`. □

The other function for the pertinence management is the `extendTimestamps` which extends the temporal pertinence of a given version by adding a given temporal element. Moreover, as the inheritance semantics of our model requires both the pertinence of sibling versions to be disjoint and the pertinence of each version to be contained in that of its parent, it also restrict the pertinence of each version which is sibling of the newly added one and recursively applies itself to the ancestor versions. In particular, as shown in Fig. 5, it works on the version identified by the path `$m` and first it determines the temporal pertinence P of the version (steps

[1] In general, there is no unique minimal representation of temporal elements by multidimensional intervals. However, all our implemented algorithms are designed to always obtain and efficiently maintain one minimal representation.

```
function extendTimestamps(te, $m)
(1)P = ∅;
(2)for each $n in $m/ta
(3)   P = P ∪ [$n/@vt_start,$n/@vt_end)
          ×[$n/@et_start,$n/@et_end) × [$n/@tt_start,$n/@tt_end);
(4)if not (te ⊆ P)
(5)  te = coalesce(P ∪ te);
(6)  set timestamps of $m to P;
(7)  for each $d in children(parent($m))
        restrictTimestamps(te,$d);
(8)  extendTimestamps(te,parent(parent($m));
```

Fig. 5. The `extendTimestamps` function

1-3), then, if P does not already include the temporal element te, it updates P by adding te and then assigns the resulting P to `$m` as timestamp. Notice that step 5 performs the union of two temporal elements and that such an operation not necessarily produces a temporal element, as the pertinence should be. For this reason, an explicit coalescing operation is required (actually, coalescing is applied to the decomposition resulting from the evaluation of $(P \setminus te) \cup te$, which eliminates overlap regions).

Finally, steps 7 and 8 have been included in order to comply with the inheritance semantics. In particular, step 7 restricts the pertinence of the versions which are siblings of the newly added one by calling the `restrictTimestamps` function. The correctness of such operation is ensured by the fact that the function recursively applies itself to the descendant versions, if necessary. Step 8 recursively applies itself to the ancestor version of `$m` which, in the tree representation, is the grandparent of `$m`. The algorithm avoids accessing the ancestor of such versions for which the pertinence does not need updates. More precisely, thanks to the inheritance semantics, if the temporal pertinence of a given version has not been extended then its first ascending version does not need to be extended too, thanks to the containment relationship, and so on. Such observations allow us to state the fact which follows.

Fact 3 *The `extendTimestamps` function is correct, that is the output is an XML document complying with the temporal model introduced in Sec. 2 and, in particular, preserving the inheritance semantics.*

The complexity of the `extendTimestamps` function is shown in the following proposition.

Proposition 3.2 *In the worst case, the complexity of the `extendTimestamps` function is linear in the number of nodes in the XML document.*

Proof. The `extendTimestamps` function, given the node `$m` corresponding to the version to be modified, updates the temporal pertinence of `$m` and that of its

siblings by calling the `restrictTimestamps` function. In the worst case, such an operation is linear in the number of nodes in the subtrees rooted at each of the `$m` siblings. Moreover, the function goes upwards by recursively applying itself to the ancestor versions of `$m`. In any case, each node is visited at most once as the intersection between the siblings of each node and the siblings of its parent is always empty. Finally, notice that the worst case happens when `$m` is a leaf and it is required to go upwards until the root of the document is reached. □

3.2 Modification algorithms

This subsection is devoted to the presentation of the algorithms defining the operational semantics of the two basic operators of our model: $changeText$ and $changeTime$.

The former operator has been included in the temporal XML model in order to support textual modifications occurring at any level of the hierarchical structure of documents. $changeText$ performs a lossless update of the structural element of the normative text to be modified by essentially creating a new temporal version. Moreover, in order to comply with the inheritance constraints, it also accommodates the temporal pertinence of:

- the sibling versions, as their pertinence should be disjoint;
- the ancestor and descendant versions, as the pertinence of each version should be contained in the pertinence of its parent.

As far as the pertinence of the newly added version is involved, the efficacy and validity time are specified by means of input parameters according to the provisions of the active norm, whereas the transaction time semantics forces the interval $[now, \text{UC})$ (now is the current chronon [22]) to be assigned as transaction time pertinence. Indeed, the transparent management of transaction time allows the system to keep track of when the modifications were applied for archival and auditing purposes.

The algorithm implementing $changeText$ is shown in Fig. 6. As input parameters, $changeText$ requires a path `$p` identifying the node to be modified, the validity and efficacy temporal element to be assigned to the new version, specified by the parameter $\bigcup_{1 \leq i \leq n} [vt_{s_i}, vt_{e_i}) \times [et_{s_i}, et_{e_i})$, the new XML text txt, the number of the version $vnum$ and a reference to the active norm an. Starting from the root of the passive norm, the algorithm goes along the tree down to the node rooting the portion undergoing modification and it updates such a node by adding the new version (step 5) with temporal pertinence $\bigcup_{1 \leq i \leq n} [vt_{s_i}, vt_{e_i}) \times [et_{s_i}, et_{e_i}) \times [now, \text{UC})$. Moreover, in order to comply with the inheritance semantics, the algorithm updates the temporal pertinence of the versions which are siblings (steps 1-2) and ancestors (step 4) of the newly added one, if necessary. More precisely, as the temporal per-

```
Algorithm changeText($p, text, vnum, an, ⋃_{1≤i≤n}[vt_{s_i}, vt_{e_i}) × [et_{s_i}, et_{e_i}))
(1) for each node $c in $p/ver
(2)    restrictTimestamps( ⋃_{1≤i≤n}[vt_{s_i}, vt_{e_i}) × [et_{s_i}, et_{e_i}) × [now, uc), $c);
(3) if(text not NULL)
(4)   extendTimestamps( ⋃_{1≤i≤n}[vt_{s_i}, vt_{e_i}) × [et_{s_i}, et_{e_i}) × [now, uc), parent($p));
(5)   addChild($p,
      <ver num=vnum an_rif=an>
          <ta vt_start=vt_{s_1} vt_end=vt_{e_1} et_start=et_{s_1} et_end=et_{e_1}
              tt_start=now />
          ...
          <ta vt_start=vt_{s_n} vt_end=vt_{e_n} et_start=et_{s_n} et_end=et_{e_n}
              tt_start=now />
          text
      </ver>);
```

Fig. 6. The $changeText$ algorithm

tinence of sibling versions must be disjoint, for each version which is sibling of the newly added one, the `restrictTimestamps` function is called in order to exclude from the temporal pertinence of such version that of the newly added one. Similarly, by calling the `extendTimestamps` function, we extend the pertinence of the parent version for which it is required to add the new pertinence.

The content of the new version is a snapshot text in XML format which must comply with the contents-section-article-paragraph hierarchy of our XML schema. In this way, versioning can be performed at any level of the hierarchical structure of the document. Moreover, the addition of the new version and the corresponding extension of the ancestor's pertinence is performed if and only if the XML text is not empty (step 4). Otherwise, when the XML text is empty, it only restricts the pertinence of the already existing versions by excluding the temporal element specified by the parameter $\bigcup_{1\leq i\leq n}[vt_{s_i}, vt_{e_i}) \times [et_{s_i}, et_{e_i})$. In this way, the $changeText$ operator can be used to perform a suspension or an abrogation when the ending values for the validity and efficacy time are set to UC.

The $changeText$ algorithm is correct as it produces an XML document complying with the XML schema and the inheritance semantics of our model. Its complexity is instead shown in the following Proposition.

Proposition 3.3 *In the worst case, the complexity of the $changeText$ algorithm is linear in the number of nodes in the XML document.*

Proof. The $changeText$ algorithm acts on the nodes in the XML document by means of the `extendTimestamps` and `restrictTimestamps` functions. The latter is applied to the versions which are descendant of the node `$p` undergoing the textual modification and its complexity is thus linear in the number of nodes rooted at `$p`. On the other hand, the former is applied to the version which

```
Algorithm changeTime($p, (vt, et, tt), ⋃_{1≤i≤n}[vt_{s_i}, vt_{e_i}) × [et_{s_i}, et_{e_i}))
(1) $v=select($p, vt, et, tt);
(2) extendTimestamps( ⋃_{1≤i≤n}[vt_{s_i}, vt_{e_i}) × [et_{s_i}, et_{e_i}) × [now, uc), $v);
```

Fig. 7. The $changeTime$ algorithm

is $p's parent version (version of $p in the following). Notice that, as already shown in Proposition 3.2, the extendTimestamps function could act on the subtrees rooted at the versions which are siblings of the version of $p and recursively on the version which is ancestor of the version of $p. Thus, the two functions, extendTimestamps and restrictTimestamps, will never act on the same set of nodes and the algorithm visits each node of the XML document at most once. □

The other operator, named $changeTime$, is devoted to the extension of the temporal pertinence of an existing document portion. Such operator too requires a path expression $p denoting the portion of the passive norm requiring temporal pertinence extension. Moreover, it also requires a set of temporal coordinates (i.e. a tridimensional time point (vt, et, tt)) to select the version to be modified, and the new validity and efficacy to be assigned to the selected version, expressed as a bidimensional temporal element $\bigcup_{1\leq i\leq n}[vt_{s_i}, vt_{e_i}) \times [et_{s_i}, et_{e_i})$. The code is shown in Fig. 7. As the previous operator, it works by first identifying the node corresponding to the document portion undergoing modification. Then, in step 1, it selects the node $v corresponding to the version of $p for which the temporal modification is required. If it exists, such a version is the only one whose temporal pertinence contains the time point (vt, et, tt). Indeed, as the temporal pertinence of sibling versions must be disjoint, each time point is contained in the timestamp of at most one version. Then it adds the new tridimensional temporal element $\bigcup_{1\leq i\leq n}[vt_{s_i}, vt_{e_i}) \times [et_{s_i}, et_{e_i}) \times [now, \text{UC})$ to the temporal pertinence of the selected version, by calling the extendTimestamps function, which also ensures that the resulting document complies with the inheritance semantics. As the $changeTime$ operator only makes us of the extendTimestamps function, it can be easily shown that its complexity is in the worst case linear in the number of nodes of the XML document.

As a final remark, notice that we did not include an operator performing the restriction of the temporal pertinence of (a part of) the passive norm. Indeed, temporal restriction can be performed by means of the $changeText$ operator, whenever a suspension or an abrogation is required, or by means of the $changeTime$ operator, whenever the portion of the temporal pertinence to be deleted has to be substituted by the temporal pertinence of another version.

Notice that after modifications effected via the $changeTime$ operator, no trace of the active norm is kept in the modified document. In order to keep a reference to the active norm after a pure temporal modification (e.g. an efficacy suspension),

the $changeText$ operator should indeed be used to create a new version, with the same textual content as the old one, to which the new timestamps and the active norm reference are assigned. However, this solution gives rise to version duplication and, thus, it is not the preferred solution in our system. As a matter of fact, the implemented $changeTime$ operator has an optional parameter an which can be used to add additional active norm references to the modified version (as values of `an_ref2`, `an_ref3`,... attributes in the `ver` element, which have not included in the XML Schema of Fig. 1 for the sake of simplicity).

4 Querying normative documents

Legal text repositories are usually managed by means of traditional information retrieval techniques. In particular, users are allowed to access the repository contents by means of keyword-based queries expressing the subjects they are interested in [8].

We have extended such a framework by offering the possibility of expressing temporal specifications, which are used to select and reconstruct temporally consistent versions of the normative acts of interest. In this way, we allow users to interact with the temporal aspects of the normative acts, which become essential, for instance, when they want to know the text version(s) applicable in a given period, or access the current consolidated version, or retrieve a snapshot in the past or in the future of the normative acts they are interested in. To this purpose, temporal support is required at query level. From a technical point of view, the reconstruction of temporally consistent document version(s), corresponds to a *timeslicing* operation: in particular, the reconstruction of all the consolidated versions qualifying for a given (multidimensional) time period corresponds to the execution of *sequenced queries* in [17], that is queries applied independently at each point in time in order to return a consistent history.

According to the requirements of a legal information system managing time-varying normative texts, our model supports queries involving timeslicing and having the following XQuery [41] format:

```
FOR $a IN path
WHERE constraints on $a
RETURN const-tree(document($a), temporal specs)
```

The `FOR...WHERE` construct follows the XQuery syntax and specifies selection constraints on the variable `$a` iterating over the nodes returned by the XPath expression `path`. Search keywords can be specified by means of the function `contains` [43] in the `WHERE` clause (e.g. `contains($a,'sailing')`). In the `RETURN` clause, the operator "`const-tree()`" is devoted to the reconstruc-

tion of the versions of the XML documents containing the selected nodes and which are temporally consistent with the specifications in the "`temporal specs`" expression. More precisely, the "`temporal specs`" expression defines the temporal conditions the versions of interest must satisfy, as a conjunction of elementary selection predicates on the values of the four supported temporal dimensions. In accordance with a common syntax adopted for temporal query languages (e.g. TSQL2 [35]), each elementary predicate has the form "`dimension [NOT] op VALUE`", where `NOT` is an optional keyword negating the meaning of the comparison operator `op`, and `dimension` can be `PUBLICATION`, `VALIDITY`, `EFFICACY` or `TRANSACTION`, in order to specify a selection condition on the corresponding time pertinence of versions in a normative text. The kind of the operator `op` and the type of `VALUE` which can be used depends on the involved temporal dimension.

Since the publication time of a norm is a single date, it can be compared with another date by means of the `op` operators `=`, `PRECEDES` and `FOLLOWS`. Otherwise, it can be compared with an interval of dates by means of the `op` operator `CONTAINED-IN`. In the former case `VALUE` is a date in the form `'yy-mm-dd'`, whereas in the latter case it is a an interval built from its bounding dates with the function `PERIOD('yy1-mm1-dd1','yy2-mm2-dd2')`. Examples of valid expressions involving publication time are, thus, `PUBLICATION = '2002-01-01'` and `PUBLICATION CONTAINED-IN PERIOD('2002-01-01','2002-05-01')`.

The other three temporal dimensions, validity, efficacy and transaction time, have temporal elements as values and, thus, can be compared both with single dates and intervals. In fact, interval-based `op` operators `PRECEDES`, `FOLLOWS`, `=`, `OVERLAPS`, `MEETS`, `MET-BY`, `CONTAINS` and `CONTAINED-IN` can be used either with intervals or with dates, as dates can be perceived as special cases of intervals. Examples of valid expressions involving these temporal dimensions are, thus, `VALIDITY OVERLAPS '2002-01-01'` and `TRANSACTION MEETS PERIOD('2002-01-01','2002-05-01')`. The semantics of the operators introduced above is the standard one (i.e. the same as defined for the TSQL2 language [35]). Finally, we do not require the "`temporal specs`" expression to contain a temporal condition for each of the four dimensions. Whenever a temporal condition is missing, the query is evaluated by means of default conditions: `PUBLICATION CONTAINED-IN PERIOD('0000-01-01', 'UC')` selecting every version with respect to publication time, `TRANSACTION CONTAINS 'NOW'` selecting the current versions with respect to transaction time, and `VALIDITY CONTAINED-IN PERIOD('0000-01-01','UC')` selecting every version with respect to validity time. If a selection condition involving efficacy time is not specified, the same condition used for valid time is also used for efficacy time selection.

Example 2 The following query asks for the current (w.r.t. transaction time) version(s), whose validity contains the date 1999/1/1, of the normative acts published before 2001/1/1 and containing the word "sailing" in their paragraphs:

```
FOR     $a IN //article/paragraph
WHERE   contains ($a, 'sailing')
RETURN  const-tree(document($a),
        VALID CONTAINS '1999-01-01' and
        PUBLICATION PRECEDES '2001-01-01')
```

In this case default values are used for efficacy and transaction time: `EFFICACY CONTAINS '1999-01-01' and TRANSACTION CONTAINS 'NOW'`. □

The "`const-tree()`" operator takes as input `document($a)`, that is all the XML documents containing a qualifying node, and the temporal specifications. Hence, it reconstructs the documents by selecting –at each level of the hierarchy– the portions whose temporal pertinence satisfies the specifications. The result is in general a time-varying document (i.e. a "thick" timeslice) as it may contain several qualifying versions for the same elements. Notice that, in some cases depending on the temporal selection predicates, since nodes are visited in a depth-first order, the inheritance semantics allows us to prune out portions of the XML trees which surely are "non interesting". For instance, if the temporal specification is `VALIDITY OVERLAPS PERIOD('2000-01-01','2000-12-31')`, when the query engine comes across a node whose temporal pertinence does not satisfy the temporal specifications (e.g. its validity starts from 2002), then the sub-tree rooted at such a node can be pruned out as the temporal pertinence of all its nodes is contained in that of the root and, thus, cannot satisfy the specification.

5 Prototype Implementation and Evaluation

The model described in the previous section has been implemented in a prototype system for the management and maintenance of a collection of time-varying norms. The system is able to store norms encoded as XML documents and efficiently access them by answering queries which can involve both temporal constraints and search keywords. In its current implementation, the prototype manages large collections of XML documents with the aid of the XML document management facilities offered by Oracle 9i [37].

In the following, we will illustrate the details of the prototype system, describe the XML document collections and sample queries used in our experiments and, finally, assess the results of the experiments we conducted.

5.1 The prototype system

Our prototype system has been implemented by means of a stratum approach, in which a stratum accepts:

- time-varying normative texts to be stored, which must be represented as XML documents complying with the XML schema defined in our model;
- modifications on the stored normative texts expressed by means of the operators described in Sec. 3;
- query expressions which can involve both temporal constraints and search keywords as specified in Sec. 4.

The stratum then maps each request to a semantically equivalent expression to be passed to an XQuery engine. Once the XQuery engine performs the required task and delivers its results, the stratum can also perform some additional processing before returning the results to the users. The advantage of this approach is that we can fully exploit the capabilities of an off-the-shelf XQuery engine, including its query optimization features. Moreover, the adoption of a DBMS with XML document management facilities as XQuery engine, allows us to efficiently manage large collections of normative texts and to ensure full compatibility with already existing applications in the legal field which usually employ a relational database. On the other hand, existing XQuery engines do not know anything about the temporal semantics. The stratum is thus devoted to the management of the temporal aspects of our model, in order to provide a complete support to the manipulation of time-varying normative acts.

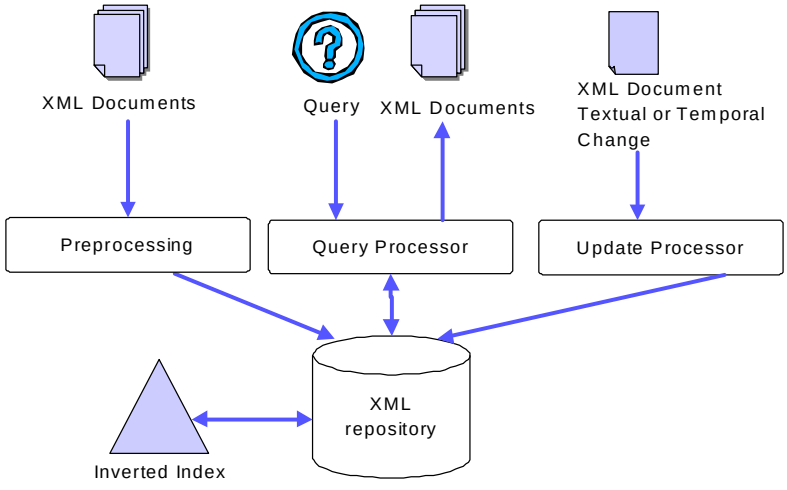


Fig. 8. The overall system architecture

The overall architecture is depicted in Fig. 8. The stratum consists of the three components shown as Preprocessing, Query Processor and Update Processor modules, whereas the temporal XML documents are stored in the XML repository. More precisely, they are maintained as CLOBs into a table having the following schema:

```
tnorms(ID, XML-DOC, TYPE, PUBLICATION, VT-START, VT-END,
       ET-START, ET-END, TT-START, TT-END )
```

Every tuple in this relation represents a temporal XML document whose textual contents are stored in the `XML-DOC` column. The table also contains additional columns storing timestamping metadata: `PUBLICATION`, `VT-START`, `VT-END`, `ET-START`, `ET-END`, `TT-START`, and `TT-END`. The timestamp attributes have the same values as the timestamping tags associated with the `contents` tag. The inheritance semantics of our model guarantees that they represent the summary time values of the whole norm text (i.e. a minimal bounding box for all the timestamps contained in the document). The addition of such metadata is aimed at improving the efficiency of query execution by introducing a preliminary document filtering phase based on the temporal predicates specified in the query. Moreover, all the timestamps which are not present in the input documents but which can be implicitly derived, owing to the semantics of inheritance, are explicited at every level of the document hierarchy before storage. In this way, by fully exploiting the potentialities of the XML query engine provided by Oracle 9i, we further speed up the processing of queries when conditions on temporal values are specified at different levels of the document tree structure. The extraction of document metadata and the explicitation of timestamps are performed once by the Preprocessing module shown in Fig. 8 when new documents are inserted in the system.

Moreover, in order to speed up the retrieval by keywords, an inverted index has been built on the contents of selected XML elements (`heading` and `paragraph`).

When a query having the form shown in Sec. 4 is issued to the system, the Query processor module first maps the request onto a SQL query to be submitted to the Oracle query engine. When it receives back the results, the module eventually performs a temporal slicing of the qualifying XML documents in order to publish the only versions which are consistent with the conditions expressed in the temporal specifications of the query.

In the first phase, the "static" part of the query (i.e. the `FOR...WHERE...` part) is translated into SQL calls in a straightforward way. Moreover, in this phase, we also exploit the fact that the temporal slicing process produces empty results for all the documents containing no version which satisfies the temporal conditions. In fact, the temporal metadata columns of the `tnorms` table saves us from accessing such normative acts, which otherwise should undergo a quite useless postprocessing phase to be eliminated. The temporal conditions are thus translated into SQL calls, which cause a quick elimination of all the tuples of the `tnorms` table representing normative documents which cannot qualify for the temporal selection.

Example 3 In the first phase, the query shown in Ex. 2 is translated into the following XML/SQL query complying with the Oracle query language syntax and where the `$cur_date` variable corresponds to the date the query is issued to the system:

```
SELECT L.XML-DOC.extract('//article').getStringVal()
FROM   tnorms L
WHERE  (VT-START <= '01-JAN-1999')
AND (VT-END is null OR VT-END >= '01-JAN-1999')
AND (ET-START <= '01-JAN-1999')
AND (ET-END is null OR ET-END > '01-JAN-1999')
AND (TT-START <= $cur_date)
AND (TT-END is null OR VT-END >= $cur_date)
AND (PUBLICATION <= '01-JAN-2001')
AND CONTAINS (L.XML-DOC, 'sailing WITHIN paragraph') > 0
```

□

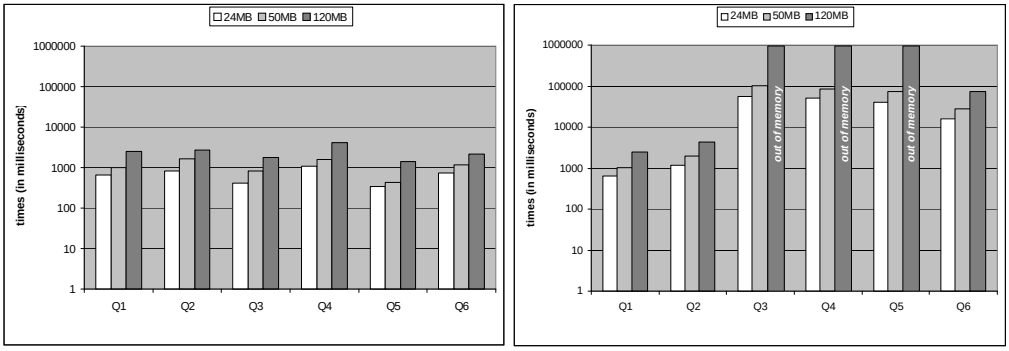
Finally, whenever the query issued to the system contains temporal specifications, the query processor module takes each qualifying tuple and performs a temporal slicing on it by means of the `const-tree` operator.

The last element of the stratum is the Update Processor module which is devoted to the management of changes that normative texts undergo during their life-cycle. Modification requests can be submitted to such a module where the maintenance of the XML repository is performed by programs implementing the `changeText` and `changeTime` algorithms.

For portability purposes, the stratum has been implemented in Java. In particular, we used JDOM for the navigation of the XML documents as required by the implementation of the `const-tree` operator and of the `changeText` and `changeTime` algorithms.

5.2 Document collections and queries

For our performance evaluation experiments, we used different document collections of increasing size: 24MB (1000 documents), 50MB (2000 documents) and 120MB (5000 documents) having an average, minimum and maximum document size of 24KB, 2KB and 125KB, respectively. The document collections are syntectic and have been created with a document generator we developed to deeply test the system performance. In particular, the generator creates one temporal XML document at a time by taking as inputs the width of the document tree, the number of versions, and a list of words. On the basis of such parameters, it generates an XML document consistent with our XML schema and, in particular, conforming to the inheritance semantics, by randomly inserting the required number of versions in the document tree and by randomly choosing words from the submitted list to fill the paragraph contents.



(a) Computing time using the `tnorm` table

(b) Computing time using the `Pnorm` table

Fig. 9. Selection of normative acts (logarithmic scale)

Query processing experiments were conducted by submitting six types of queries:

- query types q1 and q2 represent searches by a variable number of keywords. In particular, in q1 keywords are only specified on the `contents` subtree, whereas in q2 keywords are specified both on the `type` attribute and on the `contents` subtree;
- query type q3 contains temporal conditions on the four dimensions, transaction, validity, efficacy and publication time;
- query types q4, q5, and q6 mix the previous ones and contain both keywords and temporal conditions involving different document parts and temporal dimensions.

5.3 Experiments

In order to evaluate the effectiveness of the system, we conducted a number of exploratory experiments by running the prototype described above on a 600Mhz Intel Pentium III processor with 256MB of main memory and a SCSI disk. In this paper, we report and discuss the most meaningful tests performed on the three XML document sets.

We first tested the system performance in query processing by separately evaluating the following two phases: the retrieval of the qualifying normative texts and the temporal reconstruction of their contents (corresponding to a temporal slicing operation). In the first phase, only those normative texts satisfying the static part and the temporal constraints specified in the submitted query are selected. Fig. 9.a shows (with logarithmic scale) the average computing time in milliseconds required to process four queries for each of the six types on the three document sets. For the smallest document set, the computing time ranges form 343 milliseconds up to

1.096 seconds, whereas, for the biggest document set, the computing time ranges from 1.432 seconds up to 4.064 seconds. From our experiments we can state that the presence of temporal constraints does not disrupt the system performance, even when a large number of documents is selected, as it happens for the queries of type q3 where the average number of selected documents is 1568 (over 5000). Moreover, the computing time grows sub-linearly with the number of documents, thus, showing the scalability of the system. In order to evaluate the improvement given by the temporal metadata columns of the `tnorm` table, we compared our approach with a "naive" approach consisting in the direct access to the XML documents. To this end, we recorded the normative documents into a simpler table without metadata columns `Ptnorm(ID,XML-DOC)` and we fully relied for the selection of the normative texts on the Oracle XML engine, by translating the temporal conditions into XPath conditions on the temporal attributes in the `contents` element. The average computing time in this case is shown in Fig. 9.b. As we expected, the computing time for the first two query types, which do not involve temporal attributes, is the same, which means that the addition of the metadata columns does not affect the system efficiency. On the other hand, efficiency improvement given by the exploitation of the metadata columns is evident for the other four query types, where our approach is even 127 times faster than the naive approach for the two smallest collections, whereas the naive approach fails in processing queries involving temporal constraints on the biggest collection (types q3, q4, and q5). This is due to the fact that, in processing pure temporal queries (type q3) or mixed queries involving keywords non very selective (types q4 and q5), the relational query engine works better than the XML query engine and the inverted indexes built on the XML column are very little useful or even useless. On the other hand, we also evaluated the time taken to insert XML documents in the two tables `tnorm` and `Ptnorm` and to update the related indexes. The evaluation concerns the insertion of 1, 10 and 30 documents (average document size 24KB) into the two tables and the consequent update of the inverted index. The overhead required for the extraction of the information to be inserted in the metadata columns of the table `tnorm` is about 35% in the worst case corresponding to the insertion of one document at a time (from 278 seconds to 376 milliseconds) and it decreases as the number of inserted documents increases up to 12% for the insertion of about 30 documents (from 6376 seconds to 7163 seconds).

As far as the temporal reconstruction of normative texts is concerned, we evaluated the computing time required to process 30 normative texts of different sizes (from 2KB to 140 KB) and containing a different number of versions (from 3 to over 50). For these documents, `cons-tree` takes 1 second on an average. Such an outcome is also due to the fact that the temporal pertinence is represented by temporal elements. If we adopted temporal intervals, any version whose pertinence is the union of intervals would be substituted by different redundant versions with the same content, timestamped with a single interval. In such a case, documents would be larger and the time taken for the temporal reconstruction would obviously increase. We also implemented such an alternative approach and compared it with the temporal

element-based approach. We noticed that any normative text undergoing modifications (about 3) at different levels is three times larger on average when temporal intervals are adopted in place of temporal elements. In such a way, the average size of the documents managed by the system reaches 250KB. The difference between the two alternatives for the temporal reconstruction is not particularly remarkable (passing from 1 second to 1.2 second). The most important aspect to be considered is the main memory space the `cons-tree` function needs during its execution: it is even 3.5-4 times the dimension of the document (we use JDOM to navigate XML documents) and thus processing one 250KB XML document requires about 1MB. It follows that in order to process many requests, the system might recur to virtual memory, with an unavoidable dramatic performance degradation. As far as modifications are concerned, the $changeText$ and $changeTime$ operations take more or less the same computing time as `cons-tree`. This is due to the fact that most of the time is used for the XML tree navigation which is approximately the same in the two cases.

6 Related Work and Discussion

In recent years, a crop of research work addressed temporal and versioning aspects in the Web and, in particular, in the management of XML documents. Actually, about 240 of the papers listed in the bibliography [18] could be considered related work. Therefore, we will just briefly recall some of the aspects which are somehow related with our work (and cite a couple of papers which are chosen as representatives of the field) and discuss more in detail only the few papers which are more strictly related with our approach.

In a broad literature, the main focus of some approaches is on the representation and management of changes, where different versions of data are produced by updates. In these approaches, temporal attributes are often used to timestamp stored versions (e.g. [5,1]). They represent the time the updates were applied and, thus, have the (implicit) semantics of transaction time with respect to the system where the changes are effected. On the other hand, several works also considered management of changes and versioning without taking into account temporal aspects (e.g. [39,6]).

Other approaches considered the classical notion of valid time (e.g. [20,44]). For example, the "Valid Web" approach [20] is an infrastructure designed to represent and manage temporal Web documents (i.e. documents containing historical information, with timestamps explicitly encoded by the document authors to assign validity to information contents). Temporal documents can then be selectively browsed, in accordance with a user-supplied temporal period of interest. Some approaches also considered a bitemporal data model, that is supporting both valid and transaction time (e.g. [13,26]).

Other papers also considered a Web usage specific temporal dimension: navigation time, which concerns the interaction of users during their browsing of Web sites (e.g. [2,12]). Temporal aspects were also considered in the context of Web warehouses (e.g. [4,30]), mainly with reference to the time the source Web sites were navigated in order to build the warehouse and, thus, representing a sort of "validity" for the collected Web pages.

Some authors also considered multidimensional XML documents, where time can be included in the adopted versioning dimensions (e.g. [27,45]), presenting general data models, query languages and implementation solutions. However, the specificity of the four time dimensions and of the manipulation operations involved in the legal domain, prevented us from an adoption or an attempt to a simple extension of such solutions.

In addition to the definition of temporal XML data models and query languages, some works also took into account physical organization and efficient implementation issues (e.g. [45,7,31]). Proposed query languages and implementation solutions can also consider updates, but consistency of the outcomes with respect to a DTD or XML Schema is not usually considered in such approaches.

On the other hand, the paper [24] deals with the problem of consistently updating XML documents with respect to an XML Schema. The proposed solution is based on techniques, implemented as a light-weight middleware system, for rewriting generic XQuery updates into safe ones by embedding constraint-checking subqueries. A somehow "dual" approach is presented in [3], where DTDs are incrementally changed in order to keep pace with the independent evolution of XML documents. Obviously, such an approach could not be used in a context where the available schema corresponds to important application-dependent integrity constraints, as in our case.

Palmirani and Brighi present in [33] a legal document management system, called Norma-System, with limited temporal support. The core of Norma-System is an editor for drafting normative texts, which supports implicit document versioning on the basis of the modifications applied via the editor. The user-defined markup also involves temporal information, including publication, validity and efficacy time. Temporal query support is basically provided though a consolidation module, which helps the user to reconstruct consistent norm versions with respect to validity time. The other time dimensions can be used, at query level, as additional search fields in full-text search.

The paper [17] presents a temporal XML query language, τXQuery, with which the authors add temporal support to XQuery by extending its syntax and semantics. The temporal support concerns the valid time dimension and three kinds of temporal queries are considered: current, sequenced, and representational. The authors also suggest an implementation based on a stratum approach to exploit the

availability of XQuery implementations. With respect to their query classification, in our work we support current and sequenced temporal queries. Furthermore, we not only implemented our temporal query processor by leveraging existing XQuery engines, as suggested in [17], but also by exploiting the potentialities of a relational DBMS query engine. As shown in Sec. 5, such an approach allows us to improve the performance of our system in the query processing phase and to cope with a very large collection of XML documents.

In [11], the authors study the problem of consistently deriving a scheme for managing the temporal counterpart of non-temporal XML documents, starting from the definition of their schema. In particular, they introduce a data model and an architecture, called XSchema, which derives from a non-temporal schema: a temporal schema, a temporal annotation and a physical annotation. The annotations specify which portion(s) of the XML documents can vary over time, how the documents can change and where timestamps should be placed. In our work, we use an XML schema to encode the hierarchical organization of normative texts. Such an encoding is enriched with timestamping metatadata complying with an inheritance semantics. The derivation of our XML schema could also be thought as the outcome of a design process similar to the one described in [11], if we started from a snapshot XML schema corresponding to the base structure of a non-temporal (non-versioned) norm text and, then, we augmented it with versioning and timestamping metadata in order to accommodate time-varying data. As to the involved annotation, using the terminology of [11], our XML Schema supports, at each level of the document hierarchy, elements varying over transaction time and valid time, whose lifetime is described as a continuous state, and whose content can change over time. On the other hand, our data model also support a temporal element which is not included in the options of the temporal annotation in [11], and we have two different temporal dimensions, validity time and efficacy time, which both have the same semantics as valid time in [11] but need independent management.

7 Conclusions

The management of norms and their dynamics requires the adoption of temporally enhanced data models and systems. In this paper, we introduced a temporal XML data model which is able to capture the semantics of norms evolving in time and represent their multiple versions with respect to publication, validity, efficacy and transaction times. The model is based on an XML schema which allows the introduction of timestamping metadata at each level of the hierarchical structure of normative documents which are subject to change, up to the granularity of a single paragraph. A well-defined inheritance semantics rules the interaction between the different levels of the norm structural hierarchy and the temporal pertinence of the versions. Norm text modifications are dealt with by means of two basic operators which implement lossless changes through a careful management of versions and

timestamps. Moreover, the model extends conventional searches by keyword with the possibility of specifying additional temporal constraints the retrieved normative documents must satisfy. Finally, a prototype supporting the model has been implemented and evaluated. The preliminary experimental results on query performance which have been reported in the paper are encouraging.

Out of this context, our work covers a broader interest as we developed a temporal and text-centric application system with IR capabilities, which gave us the opportunity of studying the interaction of (multiple) temporal aspects having a well founded semantics with the structural properties of XML documents. Hence, our approach can provide useful solutions also for other web-based advanced applications with similar requirements (e.g. temporal management of clinical data [9,10]).

In future investigations, in order to improve the performance of our system, we plan to move from the stratum approach described in this paper, that is an implementation on-top of an existing XML-enabled system, to the design and development of a specific XML engine. In particular, we are interested in considering alternative highly-optimized solutions for physical storage of temporal XML documents, indexing and query processing. For instance, solutions based on XML document decomposition and execution of structural joins seem promising starting points for future extensions of our approach. Our future plans also include the possibility to test the effectiveness of our solutions against a real legal text repository, rather than against collections of syntectic documents as it was for the prototype. Gathering the feedback from real users would also be very interesting in such a context. Moreover, we will explore the possibility to adapt our approach to other application domains too.

8 Acknowledgments

The authors want to thank Marco Bergonzini, who helped in implementing the system and run the experiments.

References

[1] T. Amagasa, M. Yoshikawa, S. Uemura, A data model for temporal XML documents, in: Proc. of DEXA 2000, Lecture Notes in Computer Science, vol. 1873 (Springer, Berlin, 2000) 334–344.

[2] J. Andersen, A. Giversen, A. H. Jensen, R. S. Larsen, T. B. Pedersen, J. Skyt, Analyzing clickstreams using subsessions, in: Proc. of DOLAP 2000 (ACM, New York, 2000) 25–32.

[3] E. Bertino, G. Guerrini, M. Mesiti, L. Tosetto, Evolving a Set of DTDs According to a Dynamic Set of XML Documents, in: Proc. of XMLDM'02, Lecture Notes in Computer Science, vol. 2490 (Springer, Berlin, 2002) 45–66.

[4] Y. Cao, E.-P. Lim, W. Ng, On warehousing historical Web information, in: Proc. of ER 2000, Lecture Notes in Computer Science, vol. 1921 (Springer, Berlin, 2000) 253–266.

[5] S.S. Chawathe, S. Abiteboul, J. Widom, Managing historical semistructured data, Theory and Practice of Object Systems 5 (3) (1999) 143–162.

[6] S.-Y. Chien, V. Tsotras, C. Zaniolo, Version management of XML documents, in: The World Wide Web and Databases, Third International Workshop – Selected Papers, Lecture Notes in Computer Science, vol. 1997 (Springer, Berlin, 1997) 184–200.

[7] S.-Y. Chien, V. Tsotras, C. Zaniolo, Efficient management of multiversion XML documents, VLDB Journal 11 (4) (2002) 332–353.

[8] C. Ciampi, R. Nannucci, eds., ITLaw - Information Technology and the Law. An International Bibliography (1958-2001) - 2002 Edition, cd-rom (2002).

[9] C. Combi, A. Montanari, Data models with multiple temporal dimensions: Completing the picture, in: Proc. CAiSE 2001, Lecture Notes in Computer Science, vol. 2068 (Springer, Berlin, 2001) 187–202.

[10] C. Combi, L. Portoni, F. Pinciroli, Visualizing temporal clinical data on the WWW, in: Proc. of the Joint European Conf. on Artificial Intelligence in Medicine and Medical Decision Making (AIMDM'99), Aalborg, Denmark, 1999, pp. 301–314.

[11] F. Currim, S. Currim, C.E. Dyreson, R. Snodgrass, A Tale of Two Schemas: Creating a Temporal Schema from a Snapshot Schema with τXSchema, in: Proc. EDBT 2004, Lecture Notes in Computer Science, vol. 2992 (Springer, Berlin, 2004) 348–365.

[12] E. Damiani, B. Oliboni, E. Quintarelli, L. Tanca, Modeling users' navigation history, in: Proc. of Intl' Workshop on Intelligent Techniques for Web Personalisation (in conj. with IJCAI-01), Seattle, WA, 2001.

[13] C.E. Dyreson, M.H. Böhlen, C.S. Jensen, Capturing and querying multiple aspects of semistructured data, in: Proc. VLDB '99 (Morgan Kaufmann, San Francisco, 1999) 290–301.

[14] O. Etzion, S. Jajodia, S.M. Sripada, eds., Temporal Databases - Research and practice, Lecture Notes in Computer Science, vol. 1399 (Springer, Berlin, 1998).

[15] European Commission e-Government home page, `http://europa.eu.int/information_society/eeurope/2005/all_about/egovernment/index_en.htm`.

[16] S. Gadia, A homogeneous relational model and query languages for temporal databases, ACM Trans. on Database Systems 13 (3) (1988) 418–448.

[17] D. Gao, R.T. Snodgrass, Temporal slicing in the evaluation of XML queries, in: Proc. of VLDB 2003 (Morgan Kaufmann, San Francisco, 1999) 632–643.

[18] F. Grandi, An annotated bibliography on temporal and evolution aspects in the World Wide Web, Tech. Rep. TR-75, TIMECENTER, http://www.cs.auc.dk/TimeCenter/ (2003).

[19] F. Grandi, Introducing an annotated bibliography on temporal and evolution aspects in the World Wide Web, ACM Sigmod Record 33 (2) (2004).

[20] F. Grandi, F. Mandreoli, The Valid Web: an XML/XSL infrastructure for temporal management of Web documents, in: Proc. ADVIS'2000, Lecture Notes in Computer Science, vol. 1909 (Springer, Berlin, 2000) 294–303.

[21] F. Grandi, F. Mandreoli, P. Tiberio, M. Bergonzini, A temporal data model and system architecture for the management of normative texts, in: Proc. of the 11th Natlional Conf. on Advanced Database Systems (SEBD), Cetraro, Italy, 2003, pp. 169–178.

[22] C.S. Jensen, C.E. Dyreson et al., The Consensus Glossary of Temporal Database Concepts - February 1998 Version, in: O. Etzion, S. Jajodia, S. Sripada, eds., Temporal Databases — Research and Practice, Lecture Notes in Computer Science, vol. 1399 (Springer, Berlin, 1998) 367–405.

[23] S. Jones, P. Mason, R. Stamper, Legol 2.0: A relational specification language for complex rules, Information Systems 4 (4) (1979) 293–305.

[24] B. Kane, H. Su, E. Rundensteiner, Consistently updating XML documents using a incremental constraint check queries, in: Proc. ECDM 2002, Lecture Notes in Computer Science, vol. 2784 (Springer, Berlin, 2003) 39–50.

[25] S.-K. Kim, S. Chakravarthy, Modeling time: Adequacy of three distinct time concepts for temporal data, in: Proc. ER'93, Lecture Notes in Computer Science, vol. 823 (Springer, Berlin, 1993) 475–491.

[26] M. Manukyan, L. Kalinichenko, Temporal XML, in: Proc. of 5th East European Conf. on Advances in Databases and Information Systems (ADBIS '01) – Vol. 1, Research Communications, Vilnius, Lithuania, 2001, pp. 143–155.

[27] T. Mitakos, M. Gergatsoulis, Y. Stavrakas, E. Ioannidis, Representing time-dependent information in multidimensional XML, in: Proc. of 23rd Intl Conf. on Information Technology Interfaces (ITI 2001), Pula, Croatia, 2001, pp. 111–116.

[28] M.A. Nascimento, M. Eich, Decision time for temporal databases, in: Proceedings of the 2nd International Workshop on Temporal Representation and Reasoning (TIME'95), Melbourne Beach, FL, 1995, pp. 157–162.

[29] Norma in rete (Norm on network), http://www.normainrete.it.

[30] K. Nørvåg, Temporal XML data warehouses: Challenges and solutions, in: Proc. of Workshop on Knowledge Foraging for Dynamic Networking of Communities and Economies, Shiraz, Iran, 2002.

[31] K. Nørvåg, Supporting temporal text-containment queries in temporal document databases, Data & Knowledge Engineering 49 (1) (2004) 105–125.

[32] F. Ost, M. Van Hoecke, eds., Time and Law. Is the Nature of Law to Last (Bruylant, Bruxelles, 1998).

[33] M. Palmirani, R. Brighi, Norma-system: A legal document system for managing consolidated acts, in: Proc. of DEXA 2002, Lecture Notes in Computer Science, vol. 2453 (Springer, Berlin, 2002) 310–320.

[34] A. Pizzorusso, The maintenance of the Book of Laws and other studies on Legislation (Giappichelli, Torino, 1999) (*in Italian*).

[35] R.T. Snodgrass, ed., I. Ahn, G. Ariav, D. Batory, J. Clifford, C.E. Dyreson, R. Elmasri, F. Grandi, C. Jensen, W. Käfer, N. Kline, K. Kulkarni, T.C. Leung, N. Lorentzos, J.F. Roddick, A. Segev, M. Soo, S.M. Sripada, The TSQL2 Temporal Query Language (Kluwer, New York, 1995).

[36] A.U. Tansel, J. Clifford, V. Gadia, S. Jajodia, A. Segev, R.T. Snodgrass, eds., Temporal Databases: Theory, Design and Implementation (Benjamin/Cummings, Redwood City, 1993).

[37] The Oracle 9i database, `http://otn.oracle.com/products/oracle9i/content.html`.

[38] U.S. president's e-government initiatives, `http://www.whitehouse.gov/omb/egov/`.

[39] F. Vitali, D. Durand, Using versioning to support collaboration on the WWW, World Wide Web Journal 1 (1) (1996) 37–50.

[40] W3C, XML path language (XPath) 2.0, `http://www.w3.org/TR/xpath20/`.

[41] W3C, XQuery 1.0: An XML Query Language, `http://www.w3c.org/TR/xquery/`.

[42] W3C, XML Schema, `http://www.w3.org/XML/Schema`.

[43] W3C, XQuery 1.0 and XPath 2.0 Functions and Operators, `http://www.w3.org/TR/xquery-operators/`.

[44] F. Wang, C. Zaniolo, Preserving and querying histories of xml-published relational databases, in: Proc. ECDM 2002, Lecture Notes in Computer Science, vol. 2784 (Springer, Berlin, 2003) 26–38.

[45] R.K. Wong, F. Lam, M. Orgun, Modelling and manipulating multidimensional data in semistructured databases, World Wide Web 4 (1-2) (2001) 79–99.

Fabio Grandi is currently an Associate Professor in the Faculty of Engineering of the University of Bologna, Italy. Since 1989 he has worked at the CSITE center of the Italian National Research Council (CNR) in Bologna in the field of neural networks and temporal databases, initially supported by a CNR fellowship. In 1993 and 1994 he was an Adjunct Professor at the Universities of Ferrara, Italy, and Bologna. He joined his current department (Dept. of Electronics, Computer Science and Systems) as a Research Associate in 1994. His scientific interests include temporal databases, storage structures, access cost models, WWW extensions. He received a Laurea degree in Electronics Engineering and a PhD in Electronics Engineering and Computer Science from the University of Bologna. Further information can be found at http://www-db.deis.unibo.it/∼fgrandi/.

Federica Mandreoli is currently a Research Associate at the Department of Information Engineering of the University of Modena and Reggio Emilia, Italy. Her research interests include information retrieval, multi-database systems, semantic web, object-oriented databases and schema versioning. She holds a Laurea degree in Computer Science and a PhD in Electronics Engineering and Computer Science from the University of Bologna. She is member of the Association for Computer Machinery (ACM). Further information can be found at http://www.isgroup.unimo.it/federica.asp.

Paolo Tiberio is currently Full Professor of Computer Science in the Enginneering Faculty of the University of Modena and Reggio Emilia, Italy. He was also Research Associate from 1970 and Professor from 1976 to 1998 with the Department of Electronics, Computer Science and Systems of the University of Bologna, visiting scientist at the University of Michigan, Ann Arbor, in 1971 and with ”System R” and related projects of the IBM Research Center, S.Jose, California, in 1978, 1981 and 1984. His present research interests are temporal and multimedia databases and digital libraries. He received his Laurea degree in Electronic Engineering from the University of Pisa, Italy. Further information can be found at http://www.isgroup.unimo.it/tiberio.asp.