3D_XML: A Three-Dimensional XML-Based Model

Khadija Ali[1] and Jaroslav Pokorný[2]

[1] Czech Technical University, Faculty of Electrical Engineering, Praha, Czech Republic
alik1@fel.cvut.cz
[2] Charles University, Faculty of Mathematics and Physics, Praha, Czech Republic
jaroslav.pokorny@mff.cuni.cz

Abstract. Much research work has recently focused on the problem of representing historical information in XML. In this paper, we describe an ongoing work to represent XML changes. Our model is a three-dimensional XML-based model (3D_XML in short) for representing and querying histories of XML documents. The proposed model incorporates three time dimensions, *valid time*, *transaction time*, and *efficacy time* without extending the syntax of XML. We use XQuery to express complex temporal queries on the evolution of the document contents. We believe that native XML databases (NXDs) present a viable alternative to relational temporal databases when complex time dependent data has to be manipulated and stored. So NXDs will be our choice.

Keywords: XML, 3D_XML model, transaction time, valid time, efficacy time, XQuery, three-dimensional element, native XML databases.

1 Introduction

Recently, the amount of data available in XML [1] has been rapidly increasing. Much research work has recently focused on adding temporal features to XML [2, 3, 5, 6, 7, 11]. Temporal information is supported in XML much better than in relational tables. This property is attributed to the hierarchical structure of XML which is perfectly compatible with the structure of temporal data. Recently, only few works capture the notion of time explicitly in this context. Technically, to develop an XML temporal data model, it is necessary to extend a XML data model by a time dimension. The problem is that there is more XML data models (e.g. Infoset, XPath data model, XQuery data model, etc.) and more times (usually valid and transaction times). The main aim of this paper is to propose a temporal extension of XML. We propose a new scheme to represent XML changes, and show how temporal queries can be supported on this scheme.

An important issue of each data model is its implementation. There are two different ways to store XML documents in a database: *XML-enabled databases* and *native XML databases* (NXDs) [10]. The former map the data to existing (relational) database systems. The latter are XML-database systems whose inner data representation is XML-compliant. (NXDs) preserve data hierarchy and meaning of XML documents. So

(NXDs) will be our choice (particularly *eXist* [12]). In the following points we summarize the motivation of this choice:

1. eXist is open-source, and free to use. It uses the numbering scheme which supports quick identification of relationships between nodes as well as navigation through the document tree.
2. It is schema independent. It is also very user friendly. It has been chosen best XML database for InfoWorld's 2006 Technology of the Year awards. It's a worthwhile open source project for people who are interested in programming, since it's still incomplete.

The paper is organized as follows. After a discussion of related work in the next section, in Section 3 we define formally a new model (3D_XML). In Section 4, we deal with current-time (now), and show how it is supported in 3D_XML. We describe the temporal constructs of 3D_XML in Section 5. In Section 6 we illustrate that XQuery is capable of expressing complex temporal queries, but the expression of these queries can be greatly simplified by a suitable library of built-in temporal functions. Finally, in Section 7, we present our conclusions and future investigations.

2 Related Work

In the following subsections we provide a comparison of some works which have made important contributions in providing expressive and efficient means to model, store, and query XML-based temporal data models [2, 3, 5, 6, 7, 11] according to the following properties: time dimension (valid time, transaction time), support of temporal elements and attributes, querying possibilities, association to XML Schema/DTD, and influence on XML syntax [9].

Time dimension. All the models are capable to represent changes in an XML document by supporting temporal elements, and incorporating time dimensions. Two time dimensions are usually considered: valid time and transaction time. There are several other temporal dimensions that have been also mentioned in the literature in relation to XML. In [7] a publication time and efficiency time in the context of legal documents are proposed.

Temporal elements and attributes. Time dimensions may be applied to elements and attributes. All the models are capable to support temporal elements. In [3] and [11] the temporal attributes are supported. In [3] versions of an element are explicitly associated as being facets of the same (multidimensional) element. Grouping facets together allows the formulation of cross-world queries, which relate facets that hold under different worlds [14].

Influence on XML syntax. Only in [3] the syntax of XML is extended in order to incorporate not only time dimensions but also other dimensions such as language, degree of detail, etc. So the approach in [3] is more general than other approaches as it allows the treatment of multiple dimensions in a uniform manner.

Querying possibilities. The model's power depends also on supporting powerful temporal queries. In [5] and [11] powerful temporal queries expressed in XQuery without extending the language are supported. In [6] a valid time support is added to

XPath. This support results in an extended data model and query language. In [7] querying uses combination of full text retrieval and XQuery extended by some constructs to deal with time dimensions. The other models in [2] and [3] did not discuss the issue of temporal queries; in [2] elements have timestamps if they are different from the parent nodes. This fact complicates the task of writing queries in XPath/XQuery.

Association to XML Schema/DTD. A significant advantage will be added to the model if it is not only representing the history of an XML document but also the history of its corresponding XML schema or DTD as well. In [3], [7], and [11] the temporal XML schema/DTD is supported by extending the existing XML schema/DTD.

3 3D_XML Formalism

We shortly introduce three time dimensions in Section 3.1 as they are usually used in temporal databases. Then in Section 3.2 and Section 3.3, we describe our time and data models.

3.1 Time Dimensions

Three temporal dimensions are considered; valid time, transaction time, and efficacy time.

- *Valid time*: the valid time of the fact is the time when the fact is valid, or true in the modeled reality.
- *Transaction time*: it concerns the time the fact was present in the database as stored data. In other words, the transaction time of the fact identifies the time when the fact is inserted into the database and the time when that fact is removed from the database.
- *Efficacy time*: it usually corresponds to the valid time, but it can be a case that an abrogated data continues to be applicable to a limited number of cases. Until such cases cease to exit, the data continues its efficacy [7].

We extend the above efficacy time definition by assuming that it can be a case that valid data stops to be applicable to a limited number of cases. When such cases cease to exit, the data stops its efficacy.

Example 1: Consider a company database. Suppose the manager `mgr` of `Design` department is `Esra` from `"2002-01-01"` till `"2006-09-25"`. Due to some unexpected circumstances, another person started managing `Design` department from `"2002-09-08"`. `Esra` stopped managing `Design` department from `"2002-09-08"` till `"2006-09-25";` this case represents when valid data stops to be applicable. Suppose the efficacy time start is the same as the valid time start. The element `mgr` is timestamped by `"2002-01-01", "2006-09-25", "2002-01-01",` and `"2002-09-07"` which represent valid time start, valid time end, efficacy time start, efficacy time end, respectively. In this example, the valid time end `"2006-09-25"` is greater than efficacy time end `"2002-09-07"`. Figure 1 represents the valid time interval when `Esra` is a manager of `Design` department, its efficacy time interval, while the last part represents the time interval when `Esra` stopped managing `Design` department. Figure 2 depicts valid and efficacy times relationship. Valid time is represented by a time interval `(vtStart, vtEnd)`. Efficacy time is represented by a time interval `(etStart, etEnd)`. The relationship between valid time and efficacy time falls into three categories:

1. `(vtStart  < etStart)` or `(vtEnd > etEnd)`; it represents a case valid data stops to be applicable.
2. `(vtStart > etStart)`or `(vtEnd < etEnd)`; it represents a case data is applicable although it is not valid.
3. `(vtStart = etStart)`, and `(vtEnd = etEnd)`; it represents the normal case.

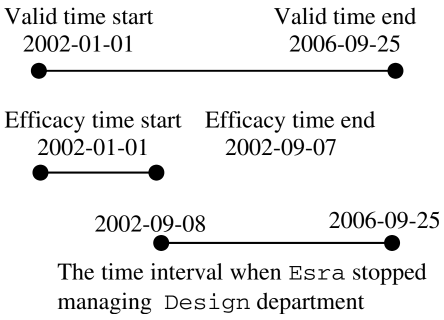

Fig. 1. Valid and efficacy times of `mgr` in Example 1

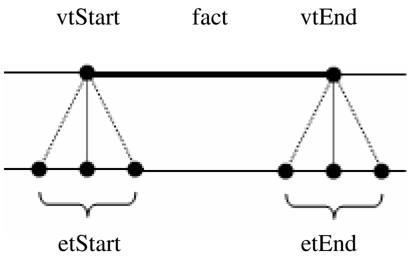

Fig. 2. Valid and efficacy times relationship

3.2 Time Model

In order to represent the changes in an XML document we encode this document as a 3D_XML document in which the syntax of XML is not extended to incorporate the three time dimensions. Instead of retaining multiple instances of the XML document, we retain a single representation of all successive versions of the document. Although time itself is perceived by most to be continuous, the discrete model is generally used. The time can be bounded in the past and in the future. A finite encoding implies bounds from the left (i.e., the existence of time start) and from the right (time end). In any specific application, the granularity of time has some practical magnitude. For instance, the time point of business event, such as a purchase, is associated with a date, so that a `day` is the proper granule for most business transactions.

Our assumptions

- The time domain T is linear and discrete. A time constant $t = [a, b]$, is either a time instant or a time interval. In a time instant constant, $a = b$, whereas in an interval constant $b > a$. It is clear that the time constant is represented with the beginning and ending instants, in a closed representation. In other words a and b are included in the interval. A bounded discrete representation as integer count of the instants since the origin is our option.
- We limit our event measures to dates (granularity = one day).
- *now* is a special symbol, such that $t <$ *now* for every $t \in T$, representing current time. We will highlight supporting *now* in 3D_XML in Section 4.

- The valid time constant *vt* = [*vtStart*, *vtEnd*], *vtStart*, and *vtEnd* represent the valid time start and valid time end, respectively. The efficacy time constant *et* = [*etStart*, *etEnd*], *etStart*, and *etEnd* represent the efficacy time start and efficacy time end, respectively. The transaction time constant *tt* = [*ttStart*, *ttEnd*], *ttStart*, and *ttEnd* represent the transaction time start and transaction time end, respectively.

Definition 1. *Valid time of an element/attribute in 3D_XML document D* is represented as *n* valid time constants $vt_1, vt_2, \ldots, vt_n$, where each vt_i represents a time constant when the element/attribute is valid. Each vt_i is called the i^{th} *version* of *D*. For each pair (vt_i, vt_j), $i, j \in [1, n]$ and $i \neq j$, the following constraint is held:

$$vt_i \cap vt_j = \emptyset \qquad \text{(valid time constants disjunction)}$$

Definition 2. *Efficacy time of an element/attribute in 3D_XML document D* is represented as *n* efficacy time constants $et_1, et_2, \ldots, et_n$, where each et_i represents a time constant when the element/attribute is efficient. For each pair (et_i, et_j), $i, j \in [1, n]$ and $i \neq j$, the following constraint is held:

$$et_i \cap et_j = \emptyset \qquad \text{(efficacy time constants disjunction)}$$

The efficacy time usually corresponds to valid time; in this case, elements/attributes are retrieved by their valid time. Otherwise, if efficacy time is different from valid time, elements/attributes will be retrieved by their efficacy time**.**

Definition 3. (Inheritance constraints). An element *e* with a valid time constant *vt* and an efficacy time constant *et* having *m* children $e_1, e_2, \ldots, e_m$, where child e_i has *k* valid time constants $vt_{i1}, vt_{i2}, \ldots, vt_{ik}$, and *q* efficacy time constants $et_{i1}, et_{i2}, \ldots, et_{iq}$, is consistent if the following conditions hold:

$$\bigcup_{1 \le j \le k} vt_{ij} \subseteq vt \qquad \text{(valid time inheritance constraint)}$$

$$\bigcup_{1 \le j \le q} et_{ij} \subseteq et \qquad \text{(efficacy time inheritance constraint)}$$

Data manipulation system of 3D_XML (left as future work) preserves the above constraints, i.e. inheritance /disjunction constraints, via user-defined functions.

3.3 Data Modeling

A time-varying XML document records a version history, which consists of the information in each version, along with timestamps indicating its lifetime.

Definition 4. A *three-dimensional XML document* (*3D_XML document* in short) is an XML document in which the three time dimensions, valid time, transaction time, and efficacy time are applied to at least one element/attribute.

Definition 5. A *three-dimensional element/attribute* (*3D_XML element/attribute* in short) is an element/attribute whose content depends on all the three time dimensions.

We will show how temporal elements and temporal attributes can be represented in 3D_XML. A temporal element can be specified in DTD notation as one of the following two structures:

```
(1)    <!ELEMENT element_name+>
       <!ATTLIST element_name vtStart CDATA #REQUIRED
                              vtEnd CDATA #REQUIRED
                              ttStart CDATA #REQUIRED
                              ttEnd CDATA #REQUIRED
                              etStart CDATA #REQUIRED
                              etEnd CDATA #REQUIRED>
(2)    <!ELEMENT element_name+>
       <!ATTLIST element_name inherits CDATA #REQUIRED>
```

We can infer from the above two structures the following observations:

1. A temporal element is represented with one or more elements having the same name; each element represents one version.
2. The three time dimensions are added to a temporal element as attributes. For instance, $\texttt{vtStart}_i$, $\texttt{vtEnd}_i$, $\texttt{ttStart}_i$, $\texttt{ttEnd}_i$, $\texttt{etStart}_i$, and $\texttt{etEnd}_i$ represent valid time start, valid time end, transaction time start, transaction time end, efficacy time start, and efficacy time end, respectively, in the i^{th} version, $i \in [1, n]$. The absence of the above three time dimensions implies that the element inherits them from one of its ancestors; the optional special attribute `inherits` = (1, 2, …, n) represents the first ancestor (parent), second ancestor (parent of parent),…, and the root, respectively.

In [2] elements have timestamps if they are different from the parent nodes. This fact complicates the task of writing queries in XPath/XQuery. We preferred to keep track of timestamps of such kind of elements having their timestamps are not different from the parent (or ancestor) nodes by a special attribute `inherits` representing the ancestor's level. The advantage of this approach is obvious; it facilitates the task of writing powerful queries in XQuery, beside supporting a more effective implementation.

To declare a temporal attribute, the following DTD syntax is used:

```
<!ELEMENT temporal_Attribute+>
<!ATTLIST temporal_Attribute name CDATA #REQUIRED
          value CDATA #REQUIRED
          vtStart CDATA #REQUIRED
          vtEnd CDATA #REQUIRED
          ttStart CDATA #REQUIRED
          ttEnd CDATA #REQUIRED
          etStart CDATA #REQUIRED
          etEnd  CDATA #REQUIRED>
```

We infer from the above form:

- A temporal attribute can be supported in our 3D_XML model by representing it by a special empty element `temporal_Attribute.` Representing time dimensions is similar to time dimensions in temporal elements.
- The name and value of the temporal attribute are represented by special attributes `name` and `value`. The transformation from a `temporal_Attribute` element to an attribute is simple and can be implemented in XQuery.

Example 2: Assume that the history of an employee is described in a 3D_XML document called `employee1.xml` as shown in Figure 3, where we shortened `Start` and `End` substrings to `S` and `E`, respectively, due to the space limitations. The element `employee` has five subelements: `emp_no`, `name`, `dept`, `job` and `salary`.

1. `emp_no`, `name`, and `dept` inherit their time dimensions, i.e. valid time, transaction time, and efficacy time from the first ancestor (parent); this fact is represented by assigning 1 to the attribute `inherits` (`inherits` = "1").
2. Notice that `salary` contains a temporal attribute `currency`. Let us assume that the salary is paid in *crown* before `"2015-01-01"`, and in *euro* after that date due to the expected change of currency in Czech Republic. Notice that the used currency *crown* will be valid till 2014-12-31; valid time end of `temporal_Attribute` element (with the value `crown`) is 2014-12-31.
3. `Anas's` job is changed from `Engineer` to `Sr Engineer` on 2005-09-02. Subsequently, his salary is changed from 60000 to 90000, in the same date. In this case the old version of `salary` (`salary=60000`) is definitely no longer applicable, hence efficacy time has been stopped to `"2005-09-01"` like validity.

```
<employee vtS="2000-01-01" vtE="now" ttS="2000-01-01"
ttE="now"   etS="2000-01-01" etE="now">
 <emp_no inherits="1">111</emp_no>
 <name   inherits="1">Anas</name>
 <dept   inherits="1">Design</dept>
 <job vtS="2000-08-31" vtE="2005-09-01" ttS="2000-08-31"
  ttE="2005-09-30" etS="2000-08-31" etE="2005-09-01">
  Engineer</job>
 <job vtS="2005-09-02" vtE="now" ttS="2005-10-01"
  ttE="now" etS="2005-09-02" etE="now">Sr Engineer</job>
 <salary vtS="2000-08-31" vtE="2005-09-01" ttS="2000-08-
  31" ttE="2005-09-30" etS="2000-08-31" etE="2005-09-
  01">60000
     <temporal_Attribute name="currency" value="crown"
       vtS="2000-08-31" vtE="2014-12-31" ttS="2000-09-01"
       ttE="now" etS="2000-08-31" etE="2014-12-31"/>
     <temporal_Attribute name="currency" value="euro"
       vtS= "2015-01-01" vtE="now" ttS="2000-09-01"
       ttE="now" etS="2015-01-01" etE=="now"/>
 </salary>
 <salary vtS="2005-09-02" vtE="now" ttS="2005-10-01"
  ttE="now" etS="2005-09-02" etE="now">90000
     <temporal_Attribute name="currency" value="crown"
       vtS="2000-08-31" vtE="2014-12-31" ttS="2000-09-01"
       ttE="now" etS="2000-08-31" etE="2014-12-31"/>
     <temporal_Attribute name="currency" value="euro"
       vtS="2015-01-01" vtE="now" ttS="2000-09-01"
       ttE="now" etS="2015-01-01" etE=="now"/>
 </salary> </employee>
```

Fig. 3. employee1.xml**:** information about an employee encoded in 3D_XML

4 Supporting for "now"

Now-relative data are temporal data where the end time of their validity follows the current time. Now-relative data are natural and meaningful part of every temporal database as well as being the focus of most queries [13]. Different approaches are used to represent current time in XML temporal databases. A common approach is to represent current time as unrealistic large date most often used "9999-12-31". Due to the nature of XML and native XML databases to store all data as text, it is possible to represent current time by words such as "now" or "UC" or "∞"; "UC" means (untilchanged). We express a right-unlimited time interval as [*t*, *now*]; although "now" is often used in temporal database literature for valid time, we will use it for all the three time dimensions. Usage of the following user-defined function `check-now` ensures that the temporal query yields the correct answer when a right-unlimited time interval [*t*, *now*] is included in the query.

```
declare function check-now ($d )as xs:date
{if ($d = "now") then xs:date(current-date())
                 else xs:date($d)};
```

As "now" can only appear as a time end of an interval, in case of valid and efficacy time intervals it means a fact is valid and efficient until now, respectively, while in the case of transaction time interval it means no changes until now.

5 Temporal Constructs

For simplicity, in all the following temporal constructs, we omitted the above user-defined function `check-now`.

5.1 Get Time Dimensions

The user-defined functions: `get_vtStart`, `get_vtEnd`, `get_etStart`, `get_etEnd`, `get_ttStart`, and `get_ttEnd` retrieve valid time start, valid time end, efficacy time start, efficacy time end, transaction time start, and transaction time end, respectively. The absence of the above time dimensions implies that the element inherits them from one of its ancestors; note that the level of the ancestor is identified by the special attribute `inherits`. Because of space limitation, we define only `get_vtStart`. The other functions can be defined in a similar way.

```
declare function get_vtStart ($s)
{ let $g := string($s/@inherits)
  return if ($g )
         then  xs:date($s/ancestor::node()[$g]/@vtStart)
         else  xs:date($s/@vtStart)};
```

5.2 Fixed Duration

XML and XQuery support an adequate set of built-in temporal types, including date, dayTimeDuration, making the period-based query convenient to express in XQuery. A user-defined function `fixedDuration` is defined as follows:

```
declare function fixedDuration($node, $length as
xdt:dayTimeDuration)
{let $dur := get_vtEnd($node)- get_vtStart($node)
 return (if ( $dur eq $length) then true()
                               else false())};
```

It checks the length of the valid time interval of the element ($node), and returns true if this length equals a given length ($length), and false otherwise.

5.3 Valid/Efficient Times Relationships Constructs

Here we focus on the temporal constructs related to Valid/efficacy times relationship.

```
declare function valid-notEfficient($a)
{if (get_etStart($a) > get_vtStart($a) or get_etEnd($a)<
     get_vtEnd($a)) then true()else false()};
```

`valid-notEfficient` is a user-defined function which checks if the element `($a)` is valid but not efficient (if `$a/@etStart > $a/@vtStart` or `$a/@etEnd < $a/@vtEnd` (see Figure 1 and Figure 2)).

5.4 Snapshot Data

Snapshot data – in the literature of databases – in the simplest sense, is the database state in a specific time point. The time point can be the current date (now), or any time point in the past, it can also be in the future, if it is expected that some facts will be true at a specified time after now. Next, we define the snapshot function `dataShot` which can be used to construct snapshots of 3D-XML documents.

```
declare function dataShot ($e, $v)
{ if (get_vtStart($e)<= xs:date($v) and
                    get_vtEnd($e) >= xs:date($v) )
  then element {name($e)}
    {$e/text(),$e/@* except
     $e/@*[string(name(.))="vtStart" or string(name(.))=
     "vtEnd" or string(name(.))="etStart"  or
     string(name(.))="etEnd" or string(name(.))="ttStart"
     or string(name(.))= "ttEnd"], for $c in $e/*
     return dataShot ($c, $v)} else () };
```

Here `dataShot` is a recursive XQuery function that checks the valid time interval of the element and only returns the element and its descendants if `vtStart <= $v <= vtEnd. (vtStart, vtEnd)` represent valid time interval of the element `($e)` , while `$v` represnts a specific time point. Note that `except` is an XQuery function discarding the attributes: `vtStart, vtEnd, etStart, etEnd, ttStart,` and `ttEnd` from the query's result.

5.5 Interval Comparison Operators

A small library of interval comparison operators is defined to help users with interval-based queries. Due to space limitation we define only three interval comparison operators: *Tcontains*, *Toverlaps*, and *TmeeTs*, respectively.

```
declare function Tcontains ($x, $y)
if ( get_vtStart($x)) <= get_vtStart($y) and
get_vtEnd($x)>= get_vtEnd($y) )then true() else false()};
```

`Tcontains` returns true if one element contains another one and false otherwise; it checks if the valid time interval `$x` contains the valid time interval of `$y`.

```
declare function Toverlaps($x, $y)
{if ( get_vtStart($x) <=  get_vtEnd($y)   and
      get_vtStart($y)  <=  get_vtEnd($x) )
 then true() else false()  };
```

`Toverlaps` checks if the element `($x)` overlaps the element `($y)`.

```
declare function TmeeTs ($x as xs:date, $y as xs:date)
{let $d := $y - $x
 return if(compare($d,"P1D")=0)then true()
                               else false()};
```

`TmeeTs` is a user-defined function checks if the first date `($x)` precedes the second date `($y)` by one day; `"P1D"` is a duration constant of one day in XQuery. Note that `compare` is an XQuery function returning -1, 0, or 1, depending on whether the value of `($d)` is respectively less than, equal to, or greater than one day.

5.6 Break Construct

The valid time constants, efficacy time constants belonging to an element/attribute may appear either with breaks, or without breaks. An occurrence of a *break* implies that there exist at least two versions of the element/attribute, *i* and *i*+1, such that their valid time constants are not adjacent.

Definition 6 (Breaks). An element *e* with a valid time constant *vt* and an efficacy time constant *et*, is said to have *breaks* if there exist at least two versions of *vt*, *i* and *i*+1, such that: $\texttt{vtStart}_{i+1}$ - $\texttt{vtEnd}_{i}$ is greater than one day ("P1D"), ($i \in [1, n\text{-}1]$, n represents the number of versions). If no such versions exist, the element *e* is said to have no *breaks*.

```
declare function Tbreak($g)
{  let $c := count($g) - 1
   let $o := for $i in (1 to $c)
             let $j := $i +1
             return if (TmeeTs(get_vtEnd($g[$i]),
                        get_vtStart($g[$j])))
             then() else "break"
   return count ($o)};
```

The function of `Tbreak` is to check if the temporal element`($g)` has breaks. It calls `TmeeTs` to check every two consecutive versions of the element `($g)`. `Tbreak` returns the number of breaks if exist. `TmeeTs` is defined in Section 5.5.

6 Temporal Queries with XQuery

In all next temporal queries, we omitted the user-defined function `check-now`. Note that, collection C1 consists of XML documents as employee1.xml.

Query 1. Find the employees (their names) who when worked as Engineer, their salaries were 60000 at any time during that period.

```
for $j in collection ("C1")//employee/job[.="Engineer"]
for $s in collection ("C1")//employee/salary[.="60000"]
where $j/../emp_no = $s/../emp_no and Toverlaps($j,$s)
return $j/../name
```

Query 1 checks if the time interval of the `job` element with value `Engineer` overlaps the valid time interval of the `salary` element with value `60000`. `Toverlaps` is defined in Section 5.5

Query 2. Retrieve employees assigned as Sr Engineer but they actually started work in the new position later, return also the inefficient period's length.

```
for $s in collection ("C1")//job[.= "Sr Engineer"]
let $diff := get_etStart($s) - get_vtStart($s)
where (valid-notEfficient ($s))
return if (days-from-duration($diff)>0)
       then <name inefficient_period="{$diff}">
              {data($s/../name)} </name> else ()
```

Query 2 returns the employee name along with the inefficient period length if `valid-notEfficient` returns true. Note that `days-from-duration` is an XQuery function which returns an xs:integer representing the days component in the canonical lexical representation of the value of `$diff`.

Query 3. The next query returns the average of Ebtehal's salaries (paid in crown).

```
for $d in collection ("C1")//employee[name="Ebtehal"]
return <avg>{avg($d//temporal_Attribute[@value="crown"
                and Tcontains(., ..)]/..)} </avg>
```

Query 3 checks if the valid time interval of `temporal_Attribute` element (with the value `crown`) contains the valid time interval of its parent (`salary`); note that `temporal_Attribute` is special empty subelements representing the temporal attribute of an element, so the parent covering constraints is not considered here.

Query 4. Return employee's names who have one break in their employment histories (fired and rehired) and their salaries have been changed for the first time at any time when they are assigned in "Design" department as "Sr Engineer".

```
for $t in collection ("C1")//employee/job[.= "Engineer"]
for $dep in  collection("C1")//employee/dept[.="Design"]
where $t/../emp_no = $dep/../emp_no
return  if  (Tcontains($dep,  $t)  and  Toverlaps($dep/../
salary[1],$t)and  Tbreak($dep/../job)=  1)  then  $dep/../
name else ()
```

Query 4 shows how a complex query can be greatly simplified by using a number of user-defined functions, i.e., `Tcontains`, `Toverlaps`, and `Tbreak`.

7 Conclusions and Future Work

In this paper, we have introduced a new scheme to represent XML changes without extending the syntax of XML. NXDs represent a suitable storage platform when complex time dependent data has to be manipulated and stored, so we chose to implement temporal queries directly in NXDs (particularly DBMS eXist). Although NXDs provide many functionalities to support XML data (particularly temporal XML data), supporting efficiently temporal queries/updates is a challenging issue. XQuery is natively extensible and Turing-complete [8], and thus any extensions needed for temporal queries can be defined in the language itself. This property distinguishes XML temporal querying from that one in relational temporal languages, e.g. TSQL. So, any syntax extension of XQuery towards temporalness, e.g. τXQuery [4], makes only queries easier to write. We conclude that XML provides a flexible mechanism to represent complex temporal data without extending the current standards [9]. The future work is directed to add more temporal constructs in order to support more powerful temporal queries. Many research issues remain open at the physical level, including the support of updates on historical data. Updates will be a real area of future investigation. Also, in order to improve the performance of our system, we plan to evaluate the effectiveness of the temporal queries.

Acknowledgement. This paper was partly supported by the National programme of research (Information society project **1ET100300419).**

References

1. W3C: Extensible Markup Language (XML) 1.1. 3rd edn. W3C Recommendation (February 04, 2004), http://www.w3.org/TR/xml11/
2. Buneman, P., Khanna, S., Tajima, K., Tan, W.: Archiving scientific data. In: Proc. of ACM SIGMOD Int. Conference, pp. 1–12 (2002)
3. Gergatsoulis, M., Stavrakas, Y.: Representing Changes in XML Documents using Dimensions. In: Proc. of 1st Int. XML Database Symposium, pp. 208–221 (2003)
4. Geo, D., Snodgrass, R.: Temporal slicing in the evaluation of XML queries. In: Proc. of VLDB, Berlin, Germany, pp. 632–643 (2003)
5. Wang, F., Zaniolo, C.: XBIT: An XML-based Bitemporal Data Model. In: Proc. of 23rd Int. Conference on Conceptual Modeling, Shanghai, China, pp. 810–824 (2004)

6. Zhang, S., Dyreson, C.: Adding Valid Time to XPath. In: Proc. of 2nd int. Workshop on Database and Network Information Systems, Aizu, Japan, pp. 29–42 (2002)
7. Grandi, G., Mandreoli, F., Tiberio, P.: Temporal Modelling and Management of Normative Documents in XML Format. Data and Knowledge Engineering 54(3), 227–254 (2005)
8. Kepser, S.: A Simple Proof of the Turing-Completeness of XSLT and XQuery. In: Proc. of Extreme Markup Languages, Montréal, Québec (2004)
9. Ali, K., Pokorný, J.: A comparison of XML-based Temporal Models. In: SITIS 2006. Proc. of 2nd int. conference on Signal-Image Technology & Internet–based Systems, Hammamet, Tunisia, December 17-21, pp. 1–12 (2006)
10. Bourret, R.: Going native: making the case for XML Databases, http://www.xml.com/pub/a/2005/03/30/native.html
11. Wang, F., Zaniolo, C.: Temporal Queries in XML Document Archives and Web Warehouses. In: Proc. of 10th Int. Symposium on Temporal Representation and Reasoning, pp. 47–55 (2003)
12. eXist Home page, http://exist.sourceforge.net/
13. Stantic, B., Governatori, G., Sattar, A.: Handling of Current Time in Native XML Databases. In: Proc. of 17th Australian Database Conference, pp. 1–8 (December 2005)
14. Gergatsoulis, M., Stavrakas, Y., Doulkeridis, C., Zafeiris, V.: Representing and querying histories of semistructured databases using multidimensional OEM. Inf. Syst. 29(6), 461–482 (2004)