# Temporal aspects of semistructured data

Barbara Oliboni, Elisa Quintarelli, and Letizia Tanca
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci, 32—20133 Milano (Italy)
oliboni,quintare,tanca@elet.polimi.it

## Abstract

*In many applications information about the* history *of data and their* dynamic *aspects are just as important as static information. During the last years the increasing amount of information accessible through the Web has presented new challenges to academic and industrial research on database. In this context, data are either structured, when coming from relational or object-oriented databases, or partially or completely unstructured, when they consist of simple collections of text or image files. In the context of semistructured data, models and query languages must be extended in order to consider dynamic aspects. We present a model based on labeled graphs for representing changes in semistructured data and a SQL-like query language for querying it.*

## 1 Introduction

Classical database models represent *static* aspects of data. Thus, the information in the database consists of data currently true in the world. However, in many applications, information about the *history* of data and their *dynamic* aspects are just as important as static information. Consider, for example, records of various kinds: financial, personnel, medical. When history is taken into account, queries can ask about the evolution of data through time and constraints may restrict the way changes occur [8, 9]. It is clear that this kind of queries is useful whenever data are not static, they change over time and their evolution needs to be taken into account.

During the last years the increasing amount of information accessible through the Web has presented new challenges to academic and industrial research on database. In this context, data are either structured, when coming from relational or object-oriented databases, or partially or completely unstructured, when they consist of simple collections of text or image files. A number of research projects have addressed the problem of accessing in a uniform way this kind of semistructured data [1]. Among these, we can cite LOREL [14], UnQL [3], WebSQL [15], WebOQL [2], StruQL [13], G-Log [17]. It is a common approach to represent semistructured data by using data models based on labeled graphs [4, 17, 11, 5].

As for the classical database field, also in the context of semistructured data it is interesting to take into account the dynamic aspects of data, i.e their evolutions through time and eventually through consecutive updates, in order to query and impose restrictions on how the data changes in time. To query a temporal database, relational languages must be extended to take into account the time coordinate [19]; in the same way , in the context of semistructured data, models and query languages must be extended in order to consider dynamic aspects.

A work which addresses the problem of representing and querying changes in semistructured data is [6, 7]. The authors introduce a model, DOEM, and a language, Chorel, for querying over data and changes. The model uses annotations on both the nodes and edges of OEM graphs (see [16]) to represent changes. Intuitively, annotations are the representation of the history of nodes and edges. To implement DOEM and Chorel they use a method that encodes DOEM databases as OEM databases and translates Chorel queries into equivalent Lorel queries over the OEM encoding. With this implementation they allow annotations on a DOEM graph $D$ to be attached to the nodes and edges of a OEM database $O_D$. In this way, for each object $o$ of $D$ there is a complex object $o'$ in $O_D$ with some sub-objects, which are required in order to represent the history of the node $o$, its old values or relationships with other nodes. This method causes a growth in the number of nodes of the final graph over which it is possible to apply the query.

The main aim of our approach is to overcome this problem: on one hand, we adopt a model which reduces the amount of annotations needed for information representation; on the other hand, we plan to effectively apply model checking techniques as a way to effectively solve queries



0-7695-1107-4/01 $10.00 © 2001 IEEE

which are able to take into account static and dynamic aspects of semistructured data. When the properties belong to some classes of formulae the problem of finding if a model possesses a given property is decidable and algorithms running in *linear time* on both the *sizes* of the *model* and the *formula* can be employed [10]. Consequently it is important to keep the size of semistructured databases as small as possible.

The structure of this paper is as follows. In Sect.2 we present a graphical data model suitable to represent static and dynamic aspects of semistructured data. In Sect. 3 we describe possible changes to a semistructured database and in Sect. 4 we introduce a possible SQL-like query language. Finally, in Sect. 5 we report some conclusions.

## 2 A graphical Temporal Data Model for semistructured data

In this section we introduce a data model based on labeled graphs, useful to represent static and dynamic aspects related to semistructured data. This choice is general enough to apply to several approaches presented in the literature [11, 17, 14].

In order to represent dynamic aspects of semistructured databases and to allow querying about their evolution through time, we add the information about the *time multi-interval* of objects and relations to their graph. In this way, we are able to derive, for example, if a particular information is currently true, or it was true in a previous state, if it has been deleted or updated.

This approach allows us to store the minimal amount of information, in fact we avoid duplicating a lot of nodes and edges.

**Definition 1** *A* semistructured temporal graph *is a directed labeled graph* $\langle N, E, r, \ell \rangle$*, where* $N$ *is a (finite) set of nodes,* $E$ *is a set of labeled edges of the form* $\langle m, label, n \rangle$*, with* $m, n \in N$*, label* $\in (\mathcal{T}_e \times (\mathcal{L}_e \cup \{\perp\}) \times (\mathcal{V} \cup \{\perp\}))$*,* $r \in N$ *is the root of the graph, and* $\ell : N \longrightarrow (\mathcal{T}_n \cup \{\perp\}) \times (\mathcal{L}_n \cup \{\perp\}) \times (\mathcal{S} \cup \{\perp\}) \times (\mathcal{V} \cup \{\perp\})$*.* $\perp$ *means 'undefined', and:*

- $\mathcal{T}_n = \{$ *complex, atomic* $\}$ *is a set of types for nodes;*
- $\mathcal{T}_e = \{$ *relational, temporal* $\}$ *is a set of types for edges;*
- $\mathcal{L}_n$ *and* $\mathcal{L}_e$ *are sets of labels to be used respectively as names for complex or atomic objects (nodes), and relations (edges). We require that* $\mathcal{L}_n \cap \mathcal{L}_e = \emptyset$*;*
- $\mathcal{S}$ *is a set of strings to be used as atomic values;*
- $\mathcal{V}$ *is a set of multi-intervals such as* $[t_1, t_2) \cup [t_3, t_4) \cup \ldots \cup [t_{n-1}, t_n)$*, with* $n > 1$*, to be used as time intervals for nodes and edges. We use a multi-interval to keep trace of different time intervals when an object exists in the database.*

$\ell$ *can be seen as composed by four single-valued functions* $\ell_{\mathcal{T}_n}, \ell_{\mathcal{L}_n}, \ell_{\mathcal{S}}, \ell_{\mathcal{V}}$*. With abuse of notation, when the context is clear, we will use* $\ell$ *for nodes and edges: if* $e = \langle m, \langle t, k, v \rangle, n \rangle$*, then* $\ell_{\mathcal{T}_e}(e) = t$*,* $\ell_{\mathcal{L}_e}(e) = k$*, and* $\ell_{\mathcal{V}}(e) = v$*. It is also required that:*

- $(\forall x \in N)(\ell_{\mathcal{T}_n}(x) = complex \rightarrow \ell_{\mathcal{S}}(x) = \perp)$ *(i.e., values are associated to* atomic objects *only);*
- $(\forall x \in N)(\ell_{\mathcal{T}_n}(x) = atomic \rightarrow \ell_{\mathcal{V}}(x) = \perp)$ *(i.e., time multi-intervals are not associated to atomic objects because in the database the time multi-interval of a value coincides with the time multi-interval of its unique ingoing edge);*
- $(\forall \langle m, label, n \rangle \in E)(\ell_{\mathcal{T}_n}(m) = complex)$ *(i.e., atomic nodes are leaves);*
- $(\forall \langle m, \langle relational, label \rangle, n \rangle \in E)(\ell_{\mathcal{T}_n}(n) = atomic \rightarrow label =$ *"Has Property") (i.e., each atomic node is connected to its parent by an edge labeled "Has Property").*
- $(\forall x \in E)(\ell_{\mathcal{T}_e}(x) = temporal \rightarrow \ell_{\mathcal{V}}(x) = \perp)$ *(i.e., time multi-intervals are associated to* relational edges *only);*
- $(\forall x \in E)(\ell_{\mathcal{T}_e}(x) = temporal \rightarrow \ell_{\mathcal{L}_e}(x) =$ *"Temporal") (i.e., the label "Temporal" is associated to each* temporal edge*).*

Note that two nodes may be connected by more than one edge, provided that edge labels be different.

In this work we assume that each node of a semistructured temporal graph is unequivocally characterized by an *identifier*. Moreover, the label of the (unique) root represents the name of the database.
For the sake of readability, in the examples we usually omit the type label of nodes and edges.

Temporal edges are represented by jagged lines. As we said in Def. 1 we distinguish two kinds of nodes: complex objects, graphically represented by rectangles, and atomic objects, depicted as ovals. Complex objects are related to other complex objects and possibly "possess" a number of attributes (atomic objects), whereas atomic objects represent objects with an atomic value (i.e. a string, an integer, but also a text, an image, a sound) and do not exist independently of their parent complex object. For this reason we choose to replicate atomic objects with the same atomic value but connected to different complex objects, although they represent the same printable information. This is also the reason why we do not associate an independent time multi-interval to atomic objects. For each complex object it

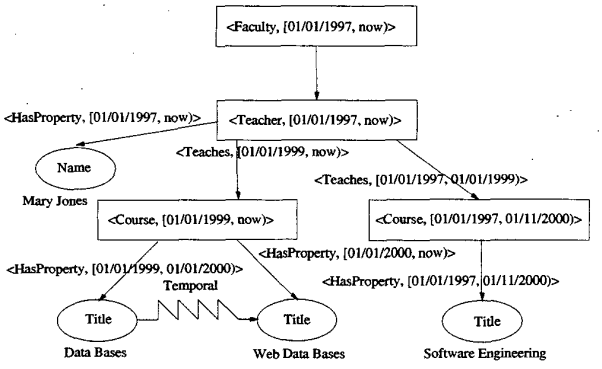


**Figure 1. A semistructured temporal graph reporting Faculty data**

is possible to declare a *key*, that is a subset of its properties (attributes) that identifies in a univocal way the object itself.

For example, in Fig. 1 we show a semistructured temporal graph, called Faculty (note that Faculty is the label of the root node) that represents the information about some teachers and related courses.

In this work we choose to represent *transaction time* in order to keep trace of information changes, and thus we deal with *static rollback databases* (see [19, 18]). By using this approach changes may only be made to the most recent static state, and there is no way to record retroactive/postactive changes, or to correct errors. Moreover, in this way the history of *database activities* is recorded. In this work we will only take care of system or transaction time. Thus ours is an "append-only" database: updates are only performed on time multi-intervals.

As explained above, we associate a multi-interval $I = [t_1, t_2) \cup \ldots \cup [t_{n-1}, t_n)$, with $n > 1$, to each relational edge and complex object; for each time interval $[t_i, t_{i+1})$ in $I$, with $i \in \{1, \ldots, n-1\}$, the instant $t_i$ represents the *start time*, while $t_{i+1}$, called *end time*, represents the time instant when that piece of information (node or edge) ceased to be true in the database. In the following sections, when an entity is currently true, i.e. it is current in the database at the present time, $t_n$ assumes the constant value "*now*". This value can be seen as $\infty$ and it does not cause difficulties when it is used as end time, although the time interval is open to the right.
In the examples we propose later in this work, time granularity depends on the application: it may be one day, or one year, etc. However, our approach will encode instants of time in a discrete fashion.

Conceptually, a temporal database appears as a sequence of states called *snapshots*, indexed by some time domain. Assume we are given a time domain $T = \{t_0, t_1, t_2, \ldots\}$, discrete and totally ordered. It is easy to obtain from a semistructured temporal graph the *snapshot at time t*. In fact:

- the *original snapshot* $S_{t_0}(G)$ of a semistructured temporal graph $G$, contains exactly those nodes and arcs of $G$ that have as start time $t_0$. Note that $t_0$ is the moment of the database creation, i.e. when the first object is recorded.
- The *snapshot at time t* $S_t(G)$ of a semistructured temporal graph $G$, is obtained by considering those arcs and objects of $G$ with a time multi-interval $I$ containing an interval $[t_i, t_{i+1})$ such that $t_i \leq t < t_{i+1}$.
- The *current snapshot* of a semistructured temporal graph $G$ is the snapshot at time *now*, i.e. all the objects and relations of $G$ with a time multi-interval $I$ containing an interval $[t, now)$, for all start times $t$ (note that the constant $now$ is such that $now \geq t$ for each instant of time $t$).

## 2.1 Comparison with the DOEM model

In [6, 7] the authors propose a model to represent changes in semistructured data, based on the Object Exchange Model (OEM) (see [16]), a simple graph-based data model, with objects as nodes and object-subobject relationships represented as labeled arcs. Change operations (i.e. node insertion, update of node values, addition and removal of labeled arcs) are represented in DOEM (for Delta-OEM) by using *annotations* on the nodes and arcs of an OEM graph. For example, in Fig. 2 we show the DOEM graph corresponding to the instance in Fig. 1: note that only edges are labeled with the name of destination objects. Actually, in both OEM and DOEM edges represent containment relations between objects, and do not allow to indicate different semantic relationships between objects, for this reason our model has a higher expressive power. Annotations are represented in DOEM by rectangles containing the name of the change operation and the date when the modification occurred.

The DOEM semistructured databases are not directly implemented by using this simple model, but using OEM databases, thus causing a growth in the number of nodes of the final graphs over which queries are applied. In Fig. 3 we report the OEM database corresponding to DOEM graph in Fig. 2. Note that in this OEM graph annotations have been encoded into complex objects storing the history of change operations by means of connected sub-objects. To highlight the difference between our temporal data model and DOEM we consider an update operation to an atomic object, because it clearly distinguishes the two approaches by considering the number of added nodes. For example, to perform

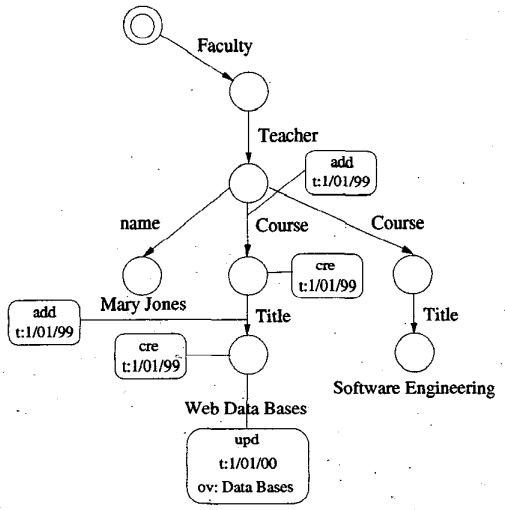


**Figure 2. A DOEM semistructured database**

the update on the Title of the Course previously called Data Bases, into Web Data Bases in the semistructured graph of Fig. 1, we only add an atomic object with the new name, whereas in DOEM the number of added nodes is shown in the dashed region of Fig. 3. Note that we add the minimum piece of information to keep trace of the update.

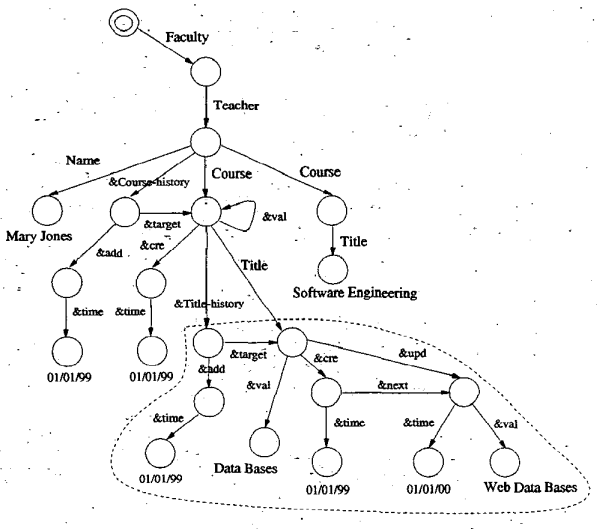


**Figure 3. A OEM semistructured database**

# 3 Operations

In this section we list possible changes to a semistructured database and how its semistructured temporal graph is consequently modified.

Let $G = \langle N, E, \ell \rangle$ be a semistructured temporal graph. We consider the following change operations:

- **Create a complex object**
  *create-complex-node*$(obj, complex, l, P, t_{cre})$ creates at time $t_{cre}$ a node with labels $\ell_{\mathcal{T}_n}(obj) = complex$, $\ell_{\mathcal{L}_n}(obj) = l$, set of attributes $P$. This operation creates also the set of atomic objects $P$ by calling the operation *create-atomic-node* for all $x \in P$, as described below. Note that $obj$ is the identifier of the created object. With a create operation, a subset $K$ of $P$ may also be defined as the object key. When the key is not declared with this operation, then the object is *new*, i.e. $obj \notin N$, and $\ell_{\mathcal{V}}(obj) = [t_{cre}, now)$. Otherwise, if the key $K$ is declared and there is a complex object $o'$ in $G$ that was current in a past instant of time, with the same labels and key, then with this operation we add to the multi-interval of $o'$ another time interval $[t_{cre}, now)$. Otherwise, if $\ell_{\mathcal{V}_n}(o') = [t, now)$ then the creation is rejected. In Figure 4 we show the creation of the complex object "L_Obj1" at time "Create_Date1" and the creation of its atomic object "L_Att1", with the same creation date in the "Has Property" edge between "L_Obj1" and "L_Att1": *create-complex-node*$(obj1, complex,$ L_Obj1$, \{obj2\},$ Create_Date1$)$, where $obj2$ is an atomic object with label L_Att1 and value Value.

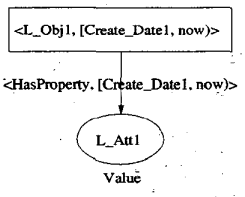


**Figure 4. A complex object has been created**

- **Create an atomic object**
  *create-atomic-node*$(obj, obj_p, atomic, l, s, t_{cre})$ creates at time $t_{cre}$ a node with labels $\ell_{\mathcal{T}_n}(obj) = atomic$, $\ell_{\mathcal{L}_n}(obj) = l$, $\ell_{\mathcal{S}}(obj) = s$ and $\ell_{\mathcal{V}}(obj) = \perp$, and whose parent is $obj_p$. Moreover, we create an edge from the parent complex object $obj_p$ to the atomic object itself, with label "Has Property" and time multi-interval $[t_{cre}, now)$. Note that an atomic object is always connected to a complex object, because it represents a property of the complex object, and thus the

time interval of its ingoing edge must be contained in a interval of its parent.

- **Remove an object**
  *remove-node*$(obj, k, t_{rem})$ removes at time $t_{rem}$ the current node with identifier $obj$ (if it exists). This operation merely consists of modifying the time multi-interval of the node, if it is a complex object, or of the ingoing edge, if it is an atomic object, from $[t_1, t_2) \cup \ldots \cup [t_{n-1}, t_n) \cup [t, now)$ to $[t_1, t_2) \cup \ldots \cup [t_{n-1}, t_n) \cup [t, t_{rem})$. If the object is complex, in order not to leave dangling references, we have to modify the time multi-interval of all its ingoing and outgoing edges. Formally, for each edge $e = \langle m, label, obj \rangle$ or $e = \langle obj, label, n \rangle$ with $\ell_{\mathcal{V}}(e) = [t_1, t_2) \cup \ldots \cup [t_{n-1}, t_n) \cup [t, now)$, we set $\ell_{\mathcal{V}}(e) = [t_1, t_2) \cup \ldots \cup [t_{n-1}, t_n) \cup [t, t_{rem})$. In Figure 5 we show the deletion of the complex object "L_Obj1" in date "Rem_Date1" and the deletion of its atomic object "L_Att1": *remove-node*$(obj1,$ complex, Rem_Date1$)$.

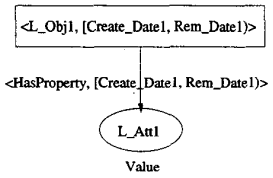


**Figure 5. An object has been removed**

- **Create a relation**
  *create-edge*$(obj_1, label, obj_2, t_{cre})$ adds at time $t_{cre}$ an edge labeled $label$ between $obj_1$ and $obj_2$. Obviously, $obj_1$ and $obj_2$ must be existing and current complex objects. This operation does not make sense between a complex and an atomic object because properties, represented by means of atomic objects, are part of complex objects. With this operation we add to the temporal graph the relational edge $\langle obj_1, \langle relational, label, [t_{cre}, now) \rangle, obj_2 \rangle$. In Figure 6 we show the new relation "Rel1" created in date "Create_Date2" between the complex objects "L_Obj1" and "L_Obj2": *create-edge*$(obj_1, Rel1, obj_2,$ Create_Date2$)$.

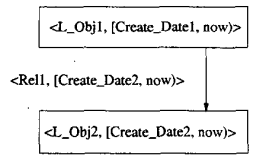


**Figure 6. A relation has been created**

- **Remove a relation** (between complex objects)
  *remove-edge*$(obj_1, label, obj_2, t_{rem})$ removes at time $t_{rem}$ an existing edge. The operation modifies the time multi-interval of the edge from $[t_1, t_2) \cup \ldots \cup [t_{n-1}, t_n) \cup [t, now)$ to $[t_1, t_2) \cup \ldots \cup [t_{n-1}, t_n) \cup [t, t_{rem})$. In Figure 7 we show the deletion of the relation "Rel1" at time "Rem_Date2": *remove-edge*$(obj_1, Rel1, obj_2,$ Rem_Date2$)$.
  Note that if we remove an edge $\langle obj_1, label, obj_2 \rangle$ we do not delete the object $obj_2$, although it is not reachable, because it is a current object (i.e. it is currently true) that could be useful for further change operations.

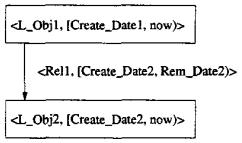


**Figure 7. A relation has been removed**

- **Update a value**
  *update*$(obj_1,$ *old-value*, *new-value*, $t_{upd})$ changes at time $t_{upd}$ the value of an atomic object from *old-value* to *new-value*. In order to compute this operation we remove the atomic object $obj_1$ as described above (note that the deletion date is $t_{upd}$), then we create a new object, i.e. we add an atomic node, with a new identifier $obj_2$ and the same father of $obj_1$, to the set of nodes $N$, and create a temporal edge from the old object $obj_1$ to the new object $obj_2$, labeled with "Temporal". In Figure 8 we show the change of the value of "L_Att1" (we suppose its identifier is $Att_1$) at time "Change_Date2": *update*$(Att_1, OldValue, NewValue, Change\_Date2)$.

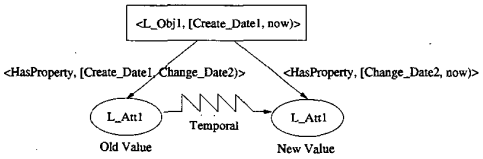


**Figure 8. A value has been updated**

## 3.1 Example

When we create a semistructured database we create a set of objects with their relations. In Figure 9 we show a database containing a teacher and a course: each element has its time multi-interval which contains at the beginning only an interval. Note that if an object is currently true, the

end time is $now$. We now suppose to create the database of Figure 9 at time 01/01/1997, consequently all the objects have as time multi-interval $[01/01/1997, now)$.

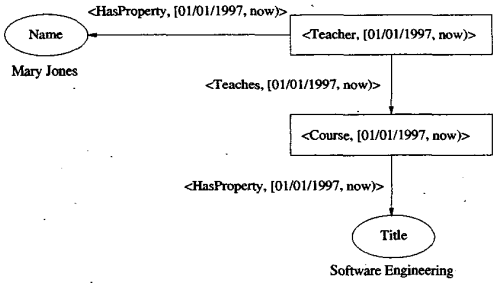


**Figure 9. 01/01/1997: Initial database**

At time 01/01/1998 we suppose to add a new teacher who also teaches "Software Engineering". Accordingly to the previous definitions, the new objects and the new edge have as time multi-interval $[01/01/1998, now)$.

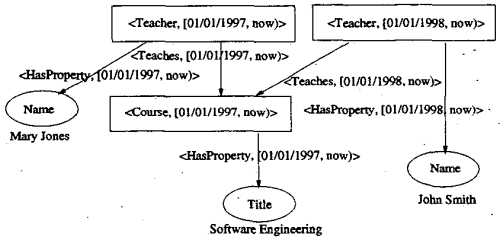


**Figure 10. 01/01/1998: a new teacher has been added**

Now let us suppose that at time 01/01/1999 we add a new course with title "DataBases" (and time multi-interval $[01/01/1999, now)$) (see Figure 11). We also suppose that the teacher Mary Jones stops teaching "Software Engineering" and starts teaching the new course. The time multi-interval of the correspondent relation "Teaches" between "Teacher" (Mary Jones) and "Course" ("Software Engineering") changes from $[01/01/1997, now)$ to $[01/01/1997, 01/01/1999)$. Note that the time multi-interval of the "Software Engineering" course does not change because the course still exists.

In date 01/01/2000 the title of "Data Bases" course changes in "Web Data Bases". In this case, as shown in Figure 12, we add the new atomic object with label "Web Data Bases" and change the time multi-interval of the old object from $[01/01/1999, now)$ into $[01/01/1999, 01/01/2000)$. We also add a temporal edge with label "Temporal" between the two atomic objects.

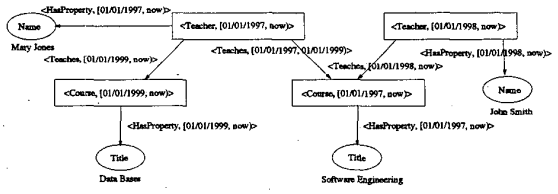


**Figure 11. 01/01/1999: a new course has been added**

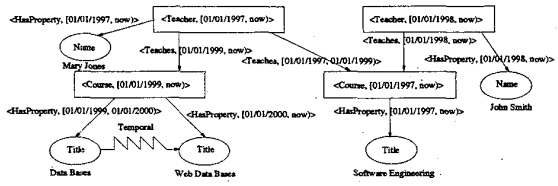


**Figure 12. 01/01/2000: a value has been changed**

In date 01/11/2000 we remove the "Software Engineering" course (Figure 13). To delete one existing object we change the time multi-interval of the object itself and of its "Has Property" edges too. In this case we change the time multi-interval of the "Course" node named "Software Engineering", of its "Title" property (with the ingoing edge "Has Property"), and of the relation "Teaches" (between the teacher John Smith and the course "Software Engineering"), from $[01/01/1997, now)$ to $[01/01/1997, 01/11/2000)$.

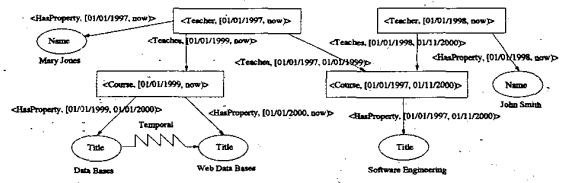


**Figure 13. 01/11/2000: an object has been deleted**

In date 01/01/2001 the previously deleted course "Software Engineering" and its property "Title" become current again (Figure 14). With this operation we add to the time multi-intervals of the objects course and title (actually, of the ingoing edge of the title atomic object) an interval $[01/01/2001, now)$, so the current multi-intervals become $[01/01/1997, 01/11/2000) \cup [01/01/2001, now)$.

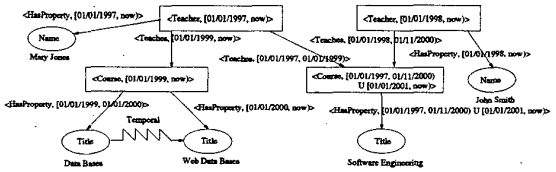


**Figure 14. 01/01/2001: a deleted object has been added**

## 4 TS-QL: Temporal Semistructure Query Language

In this section we give an informal overview of the semantic concepts of the SQL-like query language *TS-QL*, syntactically specified in Appendix 4.1, aimed at querying semistructured data modeled by means of semistructured temporal graphs (see Sect. 2).

We use the concept of path expression, which identifies a path on the information graph. The path expression has to start and finish with a node label and may contain wildcards.

The **SELECT** clause specifies the form of the result of a query by means of a list of path expressions. In particular the result is a set of paths in the instance graph.

A path expression may contain different wildcard characters, whose meaning is the following:

"*" represents a sequence $\langle e_0, o_1, e_1, \ldots, o_n, e_n \rangle$ with $n \geq 0$. Note that if $n = 0$ the sequence is composed by an edge.

"?" represents a sequence $\langle e_0 \rangle$ or $\langle e_0, o_1, e_1 \rangle$.

"#" represents a sequence $\langle e_0, o_1, e_1, \ldots, o_n, e_n \rangle$, with $n \geq 0$ and $\ell_{\mathcal{T}_e}(e_i) = \textit{Temporal}$ for all $1 \leq i \leq n+1$. Note that if $n = 0$ then the sequence is composed by a unique temporal edge.

The **FROM** clause contains a list of database names (actually, a list of root node labels of semistructured temporal graphs) and specifies the name of the databases (i.e. their temporal graphs) to be considered as information sources.

The **WHERE** clause specifies some conditions (possibly linked by connectives AND, OR) on objects or sub-objects matching the **SELECT** clause. It is thus possible to define restrictions on the values of atomic objects or on the time intervals of nodes and edges.

Now we show some examples of queries on a database called Faculty; Fig. 13 reports a part of its semistructured temporal graph. Note that here time granularity is "one day". For each query we report the SQL-like form based on the grammar described in Appendix 4.1, and a possible correspondent graphical representation. In this work we do not give a precise grammar for the graphical version of our language, but it is quite intuitive to associate a meaning to the graphical counterpart of each specific query. A work which addresses the problem of comparing and translating graphs and textual queries is [12].

More in detail, we represent complex objects with thin line rectangles and atomic objects with thin line ovals. Each object has a label composed by its name and the time interval (note that we represent a generic interval with $I$). The part of the graph depicted with thin lines represents the conditions specified in the **WHERE** clause (i.e. the structure of required information). A bold square links the result of the query (i.e. the object specified in the **SELECT** clause).

*Query 1.* Find the names of the teachers who were working (i.e. teaching at least one course) in 1998.

**SELECT** Teacher.HasProperty.Name
**FROM** Faculty
**WHERE** **EXISTS** Teacher.Teaches.Course
**AND** Teacher.Teaches → **INTERVAL**
(begin "01/01/1998"
**OR** begin before "01/01/1998")
**AND** end after "01/01/1998"

This query requires to find in the Faculty database the name of those Teacher objects connected to at least a Course object by means of an edge labeled Teaches, such that its time multi-interval is composed also by an interval which contains the year "1998". The **EXISTS** clause is needed because this is a semistructured database, thus in principle no constraint imposes that the Teaches edge should end with the node of type Course. In the SQL-like query these requirements are specified through paths, which are graphically depicted in Fig. 15. Note that the time intervals of the reported objects are generic, and thus replaced with the symbol $I$, while the interval of the edge Teaches is specified as described in the **WHERE** clause.

The result of this query applied to the database depicted in Fig. 13 are the Teacher objects with names Mary Jones and John Smith.

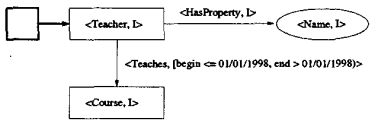


**Figure 15. Example of visual query** *Find the names of teachers who were working in 1998*

*Query 2.* Find the names of teachers who were teaching the "Data Bases" course in 1999.

**SELECT** Teacher.HasProperty.Name
**FROM** Faculty
**WHERE** Teacher.Teaches.Course.HasProperty.Name = "Data Bases"
**AND** Teacher.Teaches → **INTERVAL** (begin "01/01/1999" **OR** begin before "01/01/1999") **AND** end after "01/01/1999"

This query is rather similar to the previous one, but a restriction on the value of the course name is imposed. In this case we do not have the **EXISTS** condition in the **WHERE** clause because we need to find a precise Course object. The graphical version is in Fig. 16, here the name of the Course object is specified under the representation of the atomic object Name.

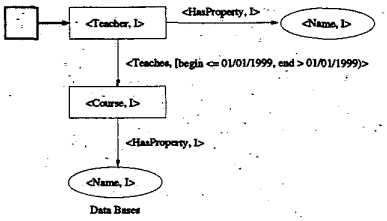


**Figure 16. Example of visual query** *Find the names of teachers who were teaching the "Data Bases" course in 1999*

*Query 3.* Find the current name of the course called "Data Bases".

**SELECT** Course.HasProperty.Title.Temporal.Title
**FROM** Faculty
**WHERE** Course.HasProperty.Title = "Data Bases"

The peculiarity of this query is the use of a temporal edge in order to find the current name of the Course previously called Data Bases and thus, the temporal edge is required to retrieve the value of an atomic object that was updated.

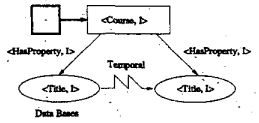


**Figure 17. Example of query** *Find the current name of the course called Data Bases*

Note that in order to find *all the names* of the course called "Data Bases" we have to write the following query:

**SELECT** Course.HasProperty.Title.#.Title
**FROM** Faculty
**WHERE** Course.HasProperty.Title = "Data Bases"

## 4.1 Grammar of TS-QL

In this subsection we introduce a preliminary grammar for the language TS-QL. We plan to add new temporal operators to the language and the possibility to express nested queries and constraints.

<query>::= SELECT [<label> AS] <select list>
[ FROM <from list> ]
[ WHERE [<label> AS] <condition> ]

<select list>::= <general path expression>, <select list>|
<general path expression>

<general path expression>::= <node label>|
<object label>.<edge label>.<general path expression>|
<object label>.<wildcard>.<general path expression>|
<attribute label>.#.<attribute label>

<node label>::= <object label>|
<attribute label>

<wildcard>::= * | ?

<from list>::= <DB label>, <from list>|
<DB label>

<condition>::= <condition> OR <condition>|
<condition> AND <condition>|
<predicate>

<predicate>::= EXISTS <general path expression>|
<complex path expression> <op> <value>|
<complex path expression> <op> <complex path expression>|
<general path expression>.<temporal path expression>
<temporal condition>

<complex path expression>::= <attribute label>|
<node label>.<edge label>.<complex path expression>|
<node label>.<wildcard>.<complex path expression>|

<temporal path expression>::= <object label> → INTERVAL |
<edge label> → INTERVAL

<temporal condition> ::= <temporal op> <value>|
<interval op><general path expression>.<temporal path expression>|
<interval op><interval values>|
<temporal condition> AND<temporal condition>|
<temporal condition> OR<temporal condition>

<op>::= = | >|<|>= | =< | =! =

<temporal op>::= begin | end | begin before | end before |
begin after | end after

<interval op>::= precede | overlap | equal

The intuitive meaning of the remaining nonterminals is the following:

| | |
|---|---|
| <label> | a string |
| <DB label> | a database name |
| <attribute label> | a node name |
| <object label> | a complex node name |
| <attribute label> | a atomic node name |
| <edge label> | an edge label |
| <value> | a constant string |
| <interval values> | a constant interval |

# 5 Conclusion

In this work we have introduced a graphical data model for representing static and dynamic aspects of semistructured data and we have proposed the SQL-like query language *TS-QL*. We plan to give a precise semantics and a graphical counterpart to this language, and to apply model checking techniques and algorithms to efficiently solve the data retrieval problem.

# References


[1] S. Abiteboul. Querying Semi-Structured Data. In *Proc. of the 6th International Conference on Database Theory*, pages 1–18, Lecture Notes in Computer Science 1186, 1997.

[2] G. Arocena and A. Mendelzon. WebOQL: Restructuring documents, databases, and webs. In *Proc. of the 14th International Conference on Data Engineering*, pages 336–350. IEEE Computer Society Press, 1998.

[3] P. Buneman, S. B. Davidson, G. G. Hillebrand, and D. Suciu. A Query Language and Optimization Techniques for Unstructured Data. In *H. V. Jagadish, Inderpal Singh Mumick (Eds.): Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada*, pages 505–516, June 1996.

[4] P. Buneman, S. B. Davidson, G. G. Hillebrand, and D. Suciu. Adding structure to unstructured data. In *Database Theory; Sixth International Conference Proceedings*, pages 336–350, 1997.

[5] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. XML-GL: a graphical language for querying and restructuring xml documents. In *Proceedings of the eight International World Wide Web Conference WWW8*, Toronto, Canada, May 1999.

[6] S. S. Chawathe, S. Abiteboul, and J. Widom. Representing and querying changes in semistructured data. In *Proceedings of the Fourteenth International Conference on Data Engineering, February 23-27, 1998, Orlando, Florida, USA*, pages 4–13. IEEE Computer Society, 1998.

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

[8] J. Chomicki. Temporal integrity constraints in relational databases. *IEEE Data Engineering Bulletin*, pages 33–37, June 1994. Special Issue on Database Constraint Management.

[9] J. Chomicki. Temporal query languages: a survey. In *Proc. International Conference on Temporal Logic*, pages 506–534. Springer–Verlag, July 1994.

[10] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent system using temporal logic specification. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[11] M. Consens and A. Mendelzon. Graphlog: a visual formalism for real life recursion. In *Proc. of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, April 1990.

[12] E. Damiani, B. Oliboni, L. Tanca, and D. Veronesi. Using wg-log schemata to represent semistructured data. In *Proc. of the Eighth Working Conference on Database Semantics (DS-8)*, pages 331–349, January 1999.

[13] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for a web-site management system. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 26(3):4–11, September 1997.

[14] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: a database management system for semistructured data. *SIGMOD Record*, 23(3):54–66, September 1997.

[15] A. Mendelzon, G. Mihaila, and T. Milo. Querying the world wide web. In *Proceedings of the Fourth Conference on Parallel and Distributed Information Systems*, December 1996.

[16] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In P. S. Yu and A. L. P. Chen, editors, *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan*, pages 251–260. IEEE Computer Society, 1995.

[17] J. Paredaens, P. Peelman, and L. Tanca. G–Log: A Declarative Graphical Query Language. *IEEE Trans. on Knowledge and Data Eng.*, 7(3):436–453, 1995.

[18] N. L. Sarda. HSQL: A Historical Query Language. In *Temporal Databases: Theory, Design, and Implementation*, Benjamin/Cummings 1993, pages 110–140.

[19] R. Snodgrass and I. Ahn. A taxonomy of time in databases. In *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 236–246, May 1985.