Historical objects for software engineering environments

R. Casallas
Dpto. Ingeniería de Sistemas y Computación
Universidad de los Andes
AA 4976
Bogotá, Colombia

rcasalla@uniandes.edu.co

Abstract

Process centered Software Engineering environments (PSEs) introduce new requirements on the repository capabilities. Modeling and managing capabilities must be offered to address product and process evolution problems.

We present in this paper a temporal data model designed to represent software artifacts, associations between them and their evolution over time. This model is called HOAM which stands for *Historical Object Association Model*. A high-level declarative language for querying a HOAM database is also presented.

This work has been achieved and it is currently experimented on the Adèle configuration management [EC94].

This work intends to benefit from temporal database domain by integrating these concepts into a framework for building Software Engineering Environments (SEE).

Key Words

Temporal databases, versions, software engineering environments, software process.

1. Introduction

It has been well understood that product quality control can be achieved only if the process by which software is produced is itself controlled. But for measuring, evaluating, controlling, and improving the software process, extended traceability services are needed. This traceability needs the recording of intermediate product and process states all along the software production process (i.e. during a very long time).

In Software engineering, *versioning* has been the natural answer for recording intermediate product states. But still there are several difficulties. Versioning is considered a *mechanism*, not a *concept*; and thus is used for many unrelated purposes: cooperative works, copies, transient or work version, variants, histories, etc. There is a confusion about the concept involved [Sci91]. Applications must define themselves their own concept of version, they must define the semantics for creating and retrieving versions, and they must also assume the same understanding from other applications which share the same objects.

We proposed in [EC95] a clear separation between three version concepts: historical, logical and cooperative. We showed that these versioning dimensions are orthogonal:

- *Historical*: it contains the evolution of the object according to the time dimension. It is to be used for recording intermediate object values and for extended *traceability*.
- *Logical*: objects may exist simultaneously in multiple *variants* for logical reasons.
- *Cooperative*: multiple and concurrent *activities* are taking place in an SEE. At a given moment in time, concurrent activities may have a cooperative version of the same object.

In this paper, we present our proposal to manage *the historical dimension*. It is based on a temporal object model called HOAM which stands for *Historical Object Association Model* (cf. section §2). For the other two

dimensions see [EC95] [Est96].

A question that arises when dealing with temporal databases in the context of object-oriented systems is whether to associate time with attributes or with objects.

Attribute timestamp, as a first solution, has the advantage that information is not duplicated between states. This approach has been chosen in [Gad88] [EW90]. The major shortcoming is time overhead in processing data to manipulate non-normalized structures. *Object timestamp*, as a second solution, gives to the notion of *state* an essential role. Its advantage is the possibility of considering states as objects (first class objects) and then, referenced them, established associations with them, etc. It has been chosen in [Sno93] [JS92] [SC91] and also in HOAM where the problem of duplicated information is solved with a delta mechanism [Tic85] [SC91].

In software engineering it is necessary to represent the history of associations between objects during a period of time (e.g. "last month user Smith was responsible of code activity") but also, it is necessary to represent associations between objects belonging to different periods of time (e.g. "create a software configuration, namely release4.1, composed by the program objects tested last month"). Our model allows to manage with both kinds of associations, called *temporal* and *non temporal* respectively.

Furthermore, a high-level declarative temporal query language has been defined for querying historical data. This language is based on path expressions and filter notions *(cf. section §3)*.

2. The data model

2.1 Core model

The core model is object-oriented. This paradigm is well suited for representing software components and associations between them. Despite the high grain variability of software components, all entities are represented in a uniform way. Objects can represent files, activities, functions as well as simple values like strings or dates. Associations are independent entities (external to objects) because they model relationships with a different semantics such as derivation, dependency or composition. An association is established from an *origin* object to a *destination* object, i.e. associations are directed.

Objects and associations are typed. A type describes the common structure (attribute definitions) and behavior (methods) of its instances. Association types describe also the association domain, i.e. between which object types the association can be established.

2.2 Historical Objects

Based on the core model we have defined a temporal data model called HOAM.

In our model, we consider only one dimension of time. We assume that time consists of discrete equidistant instants, T= {0,1,2,..., now} together with the chronological ordering <.

A *historical object* is a sequence of states. Each state is an object (as in the core model) and its value corresponds to the historical object value during a time interval. This interval defines the lifespan of the state. An instance of *historical-of T* type is a historical object defines by:

$$OH = <oid, \lceil state_0\ state_1, \ldots, state_n \rceil>$$

oid is the historical object identifier and $\lceil state_0\ state_1, \ldots, state_n \rceil$ is the value of OH, i.e. a sequence of states of type *T (cf. fig. 1)*.

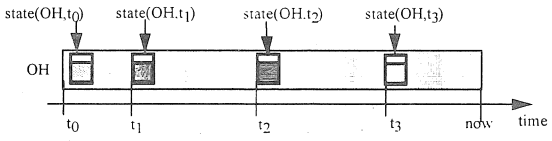

Figure 1 A historical object and its sequence of states.

Continuity assumption

Continuity assumption is taken in this model, i.e., attributes values are valid from the beginning of one state until just before the next one [CW81]. So, the value of *OH* at time *t* is defined by:

if t_k and t_{k-1} are two instants at which two consecutive states of OH were created, then for all t such as,

$t_k \leq t < t_{k-1}$.

value (OH.t) = value (state (OH, t_k))

Lifespan

Users can stop the evolution of a historical object. In this case, we say that this historical object is *dead.*

Lifespan function on a historical object returns the interval during which the object has been considered alive: $[t_0, t_{dead}]$, if it is dead $[t_0, now[$ otherwise.

When a historical object *OH* is alive, the last state created is called the *current state*. If this state was created at instant t_n, its value is valid during $[t_n, now[$ as a consequence of the continuity assumption. This interval corresponds to its lifespan.

The *lifespan* function returns the interval during which a state object was the *current state*. If

$\lceil state_0, state_1, ..., state_n \rceil$ Then,

- $lifespan(state_k) = [t_k, t_{k+1}[$ *where* t_k, t_{k+1} *are the instants of creation of the successive states:* $states_k$ *and* $state_{k+1}$
- *If OH is alive,* $lifespan(state_n) = [t_n, now[$, else,
  $lifespan(state_n) = [t_n, t_{dead}[$, where t_{dead} is the dead instant of OH.
- $\forall i,j\ i \neq j\ lifespan(state_i) \cap lifespan(state_j) = \varnothing$

Example

Our example describes a database of software development projects. Entities managed here are projects, agents, activities and programs. A project has a manager (an agent). Agents are responsible for activities. For the sake of simplicity, only the coding activities are here taken into account. The properties of an activity (coding activity) are respectively its input and its output configurations. A configuration is a set of programs and a program is an object which represents a source code file or a binary file.

In our example, the project, called TDBP (Temporal DataBase Project), was created in 1990. Between 1990 and 1992 its *budget* was $10M and, in 1993, it was raised to $20M. The *duration* of the project has been estimated to 24 months but in 1992 it has been augmented to 32 months and in 1995 to 42 months *(cf. fig. 2)*.

Some attributes are not characteristics of any specific state but of the historical object as a whole. For instance, while *budget* and *duration* are properties of each state, *creation date* and *name* project are two properties of the historical object.

Our model provides *common* and *immutable* attributes concepts. The former are used to represent properties shared by all states. The latter are used to specify properties of each state. Between two successive states, there is at least one immutable attribute which has changed. Updating an *immutable* attribute creates automatically a new state of the object. The simultaneous update of more than one

immutable attribute needs to be done in the same transaction. It is the classic Check-in Check-out paradigm [Fei91], i.e., for changing an attribute of an object, the object is treated out of the control of the system (Check-out) and then, a new state is created with the new attribute values which are given (Check-in).

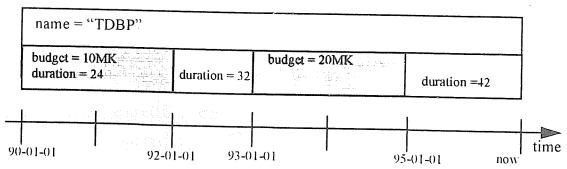

Figure 2 A TDBP historical object.

2.3 Associations

A historical database must manage not only the evolution of objects, but also the history of the relationships among objects. For instance, in a historical database of employees, it is essential to maintain information such as "during the summer 1995 employee *Smith* worked for the *Toys* department" or "*Smith was the manager of the toys department between 1994 and 1995*". These examples show relationships (*work-for, manage-by*) which exist in the real world.

In the example of the figure 3, the *managed-by* association is a temporal association. It allows to indicate the relationship between a project and its boss. For instance, the manager of the TDBP between 1990-1993 and since 1995 is the agent named Peter. Between 1993 and 1995, agent Peter was the manager of the project SCMP (Software Configuration Management Project).

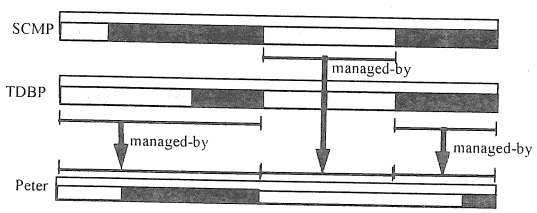

Figure 3 *managed-by* is a temporal association.

As it is presented in [WD93], a relationship between two historical objects can exist only if both participating objects are alive. That is a temporal constraint which extends the classical referential integrity constraint.

In HOAM, a *temporal* association can be established between two historical objects. It is defined by a *validity span* which determines the periods of time during which the association is valid. This period of time is a *temporal element* [JEG+94].

The associations *managed-by(TDBP, Peter)* and *managed-by(SCMP, Peter)* were valid during

{[90-01-01,93-01-01[, [95-01-01,now[} and *{[93-01-01,95-01-01[}*

respectively, since during these periods the objects (TDBP and Peter or SCMP and Peter) were alive.

In the context of software engineering, we must also establish associations between objects without temporal constraint. We called these associations *non temporal associations*. The intersection of participating objects lifespan can be empty.

In the example of the figure 4, the *component* association is a non-temporal association. An instance of a component association can be established between a state of a *configuration* object and a state of a *program* object. In the example, there are three instances of the component association. All have for origin *state(rel-4.1,t_k)* and the three destinations are: *state(P_1,t_l)*, *state(P_2,t_j)* and *state(P_3,t_j)*.

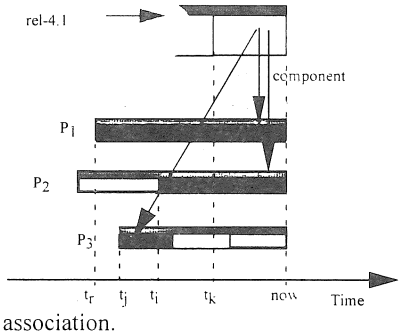

Figure 4 *component* is a non-temporal association.

Note that the intersection between *state(rel-4.1,t_k).lifespan* and *state(P_3,t_j).lifespan* is empty.

A non temporal associations can be established between a historical object/state object and historical object/state object *(cf. fig. 5)*.

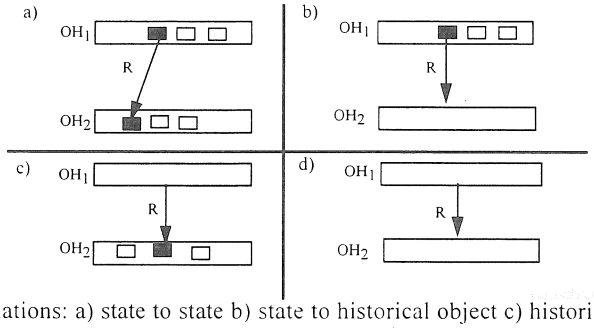

Figure 5 non-temporal associations: a) state to state b) state to historical object c) historical object to state d) historical object to historical object.

For the sake of simplicity and when the confusion does not arise, we will call *object* for historical object and *state* for state object.

3. The query language

HOAL is the query language proposed for querying a database modeled in HOAM. Objects, states and associations can be seen as graphs. The language uses this fact to reach the instances by navigation through graphs. Navigation can be achieved through both temporal and non-temporal associations in a similar syntactic way.

HOAL also offers a filtering mechanism for selecting the instances to which operations will apply.

In this section we first present basic operators to access historical objects and states object. Then, we describe the filtering mechanism and finally, we show the navigation through associations.

3.1 Objects

3.1.1 Historical references

An object is managed via *historical references*. A historical reference is defined by:

HistRef = < *oid-OH, VS*>

where *oid-OH* is the historical object identifier and *VS* is the visibility span on this object. The *VS* is defined by a temporal element and determines the periods of time during which the object is observed. For an object there can be several *historical references*.

3.1.2 Historical references and temporal associations

Temporal associations can be established only between (historical) objects. Each object participating in the association is designed by a historical reference. The temporal constraint signifies that the visibility span of both historical references is the same. In the example of the *managed-by* association *(cf. fig. 3)*, the corresponding historical references are:

<TDBP, {[90-01-01, 93-01-01[, [95-01-01, now[}>

<Peter, {[90-01-01, 93-01-01[, [95-01-01, now[}>

The validity span of the association corresponds to the visibility span of the historical references.

3.1.3 Access to objects

The language offers methods to access the visibility span, the object and the states of an object designed by a historical reference. Suppose a historical reference called *Peter-TDBP* which is a reference to the object *Peter* during the time when he was in relationship with *TDBP* object:

Peter-TDBP= <Peter, {[90-01-01, 93-01-01[, [95-01-01, now[}>

one can apply the method *VisibilitySpan:*

Peter-TDBP.VisibilitySpan

which returns the temporal element: *{[90-01-01, 93-01-01[, [95-01-01, now[}.* Also, one can apply the method *HistObj:*

Peter-TDBP.HistObj

which returns a historical reference:

< oid-OH, oid-OH.lifespan>

To access states, HOAL offers the *First* and *Last* methods. For instance,

Peter-TDBP.First

returns the first visible state through the historical reference, while,

Peter-TDBP.Last

returns the last one.

The *AllStates* method allows to transform a sequence of states into a set of states. For instance,

TDBP.AllStates

{TDBP@90-01-01,TDBP@92-01-01, TDBP@93-01-01, TDBP@95-01-01}

The "@" operator can apply to a historical reference to access a particular state. Its argument is a time expression *t*. For instance,

Peter-TDBP@*90-01-01*

an exception is produced if the *t* expression is not into the visibility span of the historical reference.

3.2 States

In a state object, one can access the interval which defines the lifespan of the state, the successor or the predecessor state and also, we can access attribute values. If state_ref is a state reference, state_ref.TimeStart and state_ref.TimeEnd return the instants which define its lifespan *(cf. section §2.2)*. One can get the successor and predecessor of one state using state_ref.successor and state_ref.predecessor.

In order to access attribute values, the *point* operator has been defined: state_ref.attr. A query example is,

Q1. What was the first budget of the TDBP project?

```
TDBP.First.budget          return $ 10M
```

Q2. What was the role of agent *Peter* when the TDBP project began?

```
Peter@(TDBP.First.TimeStart).role
```

3.3 Types conversion

State to historical reference

A type conversion from a state to a historical reference occurs when in a HOAL expression a value of type historical reference is expected and a value of state is found. In this case, state is converted to a historical reference consisting of the object, to which the state belongs, and of the state lifespan as visibility span.

Historical reference to state

A historical reference is converted to state giving the reference to the last visible state through the historical reference. With this conversion rule, the expression:

```
OH.attr
```

is equivalent to:

```
OH.attr = OH.Last.attr
```

Q3. What is the budget of the TDBP project?

```
TDBP.Budget
```

The conversion of TDBP to a state object is equivalent to:

Q4. What is the *current* budget of the TDBP project?

```
TDBP.Last.Budget                return 20
```

3.4 Filters, sets and historical references

The HOAL filter expression:

```
S [Filter]
```

is interpreted according to the type of S. A filter is used for different purposes depending on whether it applies to historical references, to a set of historical references or to a set of states.

The role of a filter on a historical reference, called *temporal filter*, is to restrict the visibility span of the original reference.

A Filter on a set of states selects elements among the set which satisfy it, we can say that the filter restricts the cardinality of the original set.

A Filter on a set of historical references applies to each element of the set as a temporal filter.

Before presenting each case, some definitions are given. These definitions are important to understand the interpretation of S [Filter] HOAL expression according S type.

On sets, two special operators have been defined: filters and image. Let | to be the filter operator so that:

$$S \mid P = \{ x \mid x \in S \text{ and } P(x) \}$$

returns the subset of the elements of S which satisfy the predicate P. For instance, if we want to select the elements minor to "4" in the set of integers {2,3,4}:

$\{2,3,4\} \mid (this<4)$ *where this represents an element of the set*

the result is the set {2,3}.

Let map be the image operator such that S map F signifies compute the image of the set S by the function F:

$$S \text{ map } F = \{x' / x' = F(x) \text{ and } t\, x \in S\}$$

For instance, if we want to add 2 to each element into the set of integers {2,3,4}, we must compute the image set:

$\{2,3,4\}$ *map* $(this+2)$ *where this represent an element into the set*

the result is the set *{4, 5, 6}*.

3.4.1 Temporal filter

A temporal filter applies to a historical reference in order to restrict the visibility span of the historical object to intervals which satisfy a predicate. For instance, if TDBP is a historical reference defined by:

$<TDBP, \{[90\text{-}01\text{-}01, now[\} >$

the HOAL expression,

TDBP[budget >10]

is evaluated as follows: the predicate budget > 10 is tested on each visible state through the historical reference. Here, the result is the new historical reference:

$<TDBP, \{[93\text{-}01\text{-}01, now[\} >$

Generally, when a temporal filter defined by a predicate *p* applies to a historical reference *rh*, note *rh[p]*, it returns a historical reference *rh'* so that its visibility span is computed as follows. If *p-states* set corresponds to the set of states in *rh* which verify the predicate *p*:

$$p\text{-}states = \{ e \mid e \in rh.HistObj.Allstates \text{ and } p(e)\}$$

and $\{e_i.lifeSpan\}$ is the temporal element (a set of one interval of time) corresponds to the lifespan of the state e_i. Then,

$$rh'.VisibilitySpan = rh.VisibilitySpan \cap \left(\bigcup \{e_i.lifeSpan\} \right)$$

Compared to the language presented in [CG93], the new *VisibilitySpan* corresponds to temporal domain returned by the temporal expression [[budget(TDBP) >10]][1].

3.4.2 Filter on set of states

Let the set TDBP.AllStates. The HOAL expression:

TDBP.AllStates [budget <10]

is interpreted as *S|P* where S is TDBP.AllStates and P is budget <10:

$$TDBP.AllStates \mid this.budget < 10 = \{ TDBP@90\text{-}01\text{-}01, TDBP@92\text{-}01\text{-}01\}$$

Thus, when in the HOAL expression S[Filter], S is a set of states objects, it is interpreted as: S | Filter

1 In [CG93], [[P(o)]] is a temporal expression which returns the temporal domain of property P of an object o.

3.4.3 Filter on set of historical references

Let *TheProjects* be a set of historical references of type *historic-of Project*. The HOAL expression:

TheProjects [Budget > 10 and kind = ESPRIT]

returns the set of ESPRIT projects:

$\{p' \mid p = p[p.Budget > 10 \textbf{ and } p.kind = ESPRIT] \text{ and } p \in TheProjects\}$

Thus, when in the HOAL expression S[Filter], S is a set of historical references, it is interpreted by:

S_{rh} *map P i.e.,* $\{x' \mid x' = x\,[P] \text{ and } x \in S\}$.

***The Point* operator on sets**

The *point* operator and the navigation operators *(cf. section §3.5)*, which apply to a set of states or to a set of historical references, are interpreted as follows. If *S* is a set and *attr* is an attribute defined in the type of the elements of *S*, the **HOAL** expression: S.attr

returns the set defined by:

$S \text{ map } ".attr" = \{v \mid v = s.attr \text{ value and } s \in S\}$

3.5 Navigation

Path expressions describe paths along the graphs formed by objects, states and associations between them. For instance, in the figure 6 there is an association, called *A*, between O_1 and O_2 objects and, another one *B* between O_2 and O_3 objects. The path expression $O_1 \rightarrow A \rightarrow B$

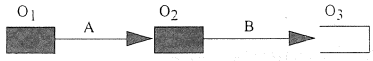

Figure 6 A graph HOAM

returns the object O_3. O_1 object is called the root object of the path expression, and *A* and *B* are associations.

Path expressions are proposed to ease the task of accessing objects [FLU94] [KKS92]: following links between objects without having to write down explicit join conditions. This kind of languages is well-suited for software engineering environments because, generally in these environments, users know external object names and access them directly or through links [BMT88]. For instance, a typical query can be "give me all the objects *depending* from the *string_mgr* module" or "give me the *components* of the input configuration of the *MMM* activity".

3.5.1 Through non-temporal associations

In general, for an object *Root* which represents a set of references to objects, and an association *A* whose domain type: $T_{Aorigin}$ for the origin objects and T_{Adest} for the destination objects, then the path expression *E:*

E = Root→A (resp. E = Root←A)

returns a set of objects of type T_{Adest} (resp. $T_{Aorigin}$).

If *A* is a *non-temporal* association then *Root* should contain a set of states or a set of historical objects according to $T_{Aorigin}$ type. Conversion types rules *(cf. section §3.3)* will be used if necessary.

For example, let us consider an activity called *MMM* (Memory Management Module development). In figure 7, the current state of *MMM* activity has for its *input* attribute value an *old* state of the configuration called

rel4.2. This state configuration has for component a state of the program called P_1.

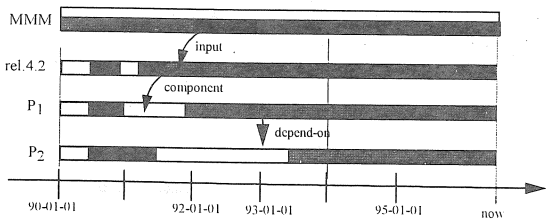

Figure 7 Non-temporal associations.

The query "what are the modules of the input configuration of the *MMM* activity?" Can be expressed by the **HOAL** expression:

MMM→input→component

The *input* association has state activities for origin domain and state configurations for destination domain. Then *MMM* is converted to a state. This is equivalent to rewrite the query as: "what are the modules of the *current* configuration input of the *MMM* activity?"

MMM@Last→input→component

this expression returns a set of *Program* states. In the example, this set is:

$\{P_1$@91-01-01$\}$

In the HOAL expression:

MMM@Last→input→component→depend-on

a conversion from the state *program* object to the historical *program* object is achieved in order to reach the *programs* on which the configuration components depend.

3.5.2 Through Temporal associations

Navigating through temporal associations implies restraint the visibility span on historical objects. Let us show it first in an example, and then, let us give the general case.

The agent *Peter* was during his life-span responsible for two activities: during 1990 and 1992, *MMM*, and after, between 1992 and now, *GUI* (graphic user interface development) *(cf. fig. 8)*.

We are interested by queries:

Q5. What is the last budget of the projects managed by agent Peter?

The **HOAL** expression which corresponds to this query is:

Peter←managed-by.budget

This expression is interpreted as follows: Peter←managed-by return the set of historical references:

$\{TDBP_{Peter}, SCMP_{Peter}\}$ *where*

$TDBP_{Peter}$ = <*PBDT*, {[90-01-01,93-01-01[, [95-01-01,now[}> *and*

$SCMP_{Peter}$ = <*SCMP*, {[93-01-01, 95-01-01[} >

point operators on each historical reference,

$\{TDBP_{Peter}.budget, SCMP_{Peter}.budget\}$

as consequence of casting types:

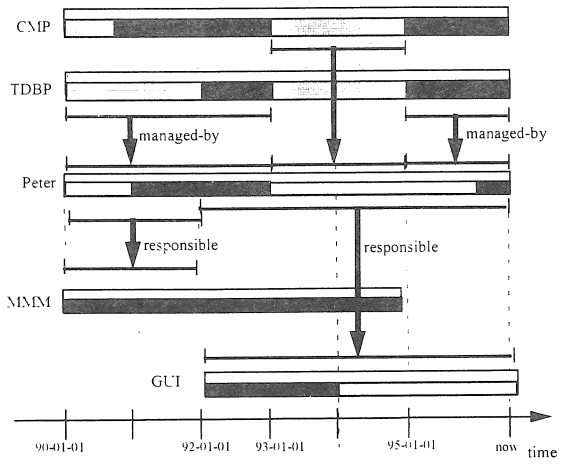

Figure 8 Peter's activities.

Generalization

Let us define the path expression as:

$E_n = E_{n-1} \leftrightarrow A$

E_n is a set of historical references on the instance of $T_{Aorigin}$ or T_{Adest}. Type depends on sense of navigation ($\rightarrow$, $\leftarrow$).

When a temporal association is crossed, its visibility span is reduced. In a general manner, let us assume that E_{n-1} contains the set of historical references:

$E_{n-1} = \{rh\text{-}O_1, rh\text{-}O_2, \ldots, rh\text{-}O_m\}$

For each rh-O_k in E_{n-1}:

$rh\text{-}O_k \rightarrow A$

let us assume that $rh\text{-}O_k.HistObj = O_k$ is an object contained into the set E_{n-1} and also, there are many A associations from O_k:

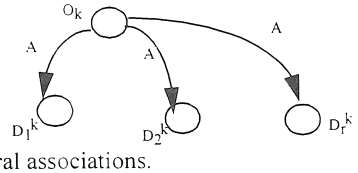

Figure 9 navigation through temporal associations.

all D^k destination objects are into the result set E_n. The visibility span is defined by:

$rh\text{-}O_k = < O_k, VS_{Ok} >$

the validity span of the temporal associations A between O^k and D_j^k is:

$ValiditySpan(A(O_k, D_j^k))$

the new visibility span of the object D_{i-j} is defined by:

$VS_{Dj} = VS_{Ok} \cap ValiditySpan(A(O_k, D_j^k))$

For instance, into TDBP→managed-by[nom = "Peter"] expression, the start historical reference is the historical object TDBP and the visibility span is:

$VS_{PBDT} = \{[90\text{-}01\text{-}01, now[\}$

The validity span of manager-by association between TDBP and Peter objects is:

$ValiditySpan(managed\_by(PBDT, Peter)) = \{[90\text{-}01\text{-}01, 93\text{-}01\text{-}01[, [95\text{-}01\text{-}01, now[\}$

the new historical reference is:

$newref\text{-}Peter = < oid\text{-}Peter, VS_{Peter} >$ *such as:*

$VS_{Peter} = VS_{PBDT} \cap ValiditySpan(chef(PBDT, Peter))$

$= \{[90\text{-}01\text{-}01, 93\text{-}01\text{-}01[, [95\text{-}01\text{-}01, now[\}$

4. Conclusions

In this paper we have presented a temporal data model called HOAM. In conjunction with the temporal model, we have presented a query language HOAL to information retrieval. The HOAL language allows each HOAM concept to be exploited and, in particular, the historical concepts. HOAL is based on two notions: path expressions and filters.

In the context of software configuration management, file revisions is the unique aspect taken into account in almost all software configuration managers. HOAM overtakes this aspect.

There has been a lot of researches in temporal database domain during the past 15 years. Most of these works are based on the relational model. Works like [BJS95] show that the field is mature enough. Unfortunately, in the case of object-oriented models, it is less true. Although important works has be done [SC91] [Sno95] [NMY93] [RS93], and a lot of works still needs to be pursued.

Our work is based on a object-oriented model [EC94] because in software engineering context due to complex data, object-oriented model are the most appropriate one [Ber87].

According to the representation matrix proposed in [EKF93] to classify temporal objects models, our model is at the intersection between *object versioning* and *relationships object representation*.

The main difference between our model and temporal object models, like those presented in [Sno95], is the way we have integrated the relationships (associations). A particularity of HOAM is both temporal and non temporal associations.

The context of our work has been the Adèle software configuration management. At the moment, the main goal of the Adèle project is to define and implement a kernel to build process software engineering environments. We need to provide high level concepts and mechanism, but hiding the complexity of the involved technology. Our work goes in this direction.

We have implemented a prototype of HOAM and HOAL by:

- using and adapting several components of the Adèle kernel, and,
- implementing the new data types of HOAM a nd HOAL : intervals and temporal elements.

References

[Ber87] Ph. A. Bernstein. Database system support for software engineering: an extended abstract. In *Proc. of the 9th Int'l Conf. on Software Engineering*, Monterey, CA, March 30-April 2 1987.

[BJS95] M. H. Bohlen, C. S. Jensen, and R. T. Snodgrass. Evaluating the completeness of TSQL2. In S. Clifford and A. Tuzhilin, editors, *Recent Advances in Temporal Databases*, pages 153–174, Zurich, Switzerland, September 1995. Proceedings of the International Workshop on Temporal Databases, Springer Verlag.

[BMT88] G. Boudier, R. Minot, and I. M. Thomas. An overview of PCTE and PCTE+. In *Proc. of the 3rd ACM Symposium on Software Development Environments*, Boston, Massachusetts, November 28–30 1988. In it ACM SIGPLAN Notices, 24(2):248–257, February 1989.

[CG93] T. Cheng and S. K. Gadia. An object-oriented model for temporal database. In R.T. Snodgrass, editor, *Proc. of the international workshop on an infrastructure for temporal*, Alington,Texas, june 1993.

[CW81] J. Clifford and D. Warren. Formal semantics for time in databases. *Transactions on databases systems*, 8:214–264, 1981.

[EC94] J. Estublier and R. Casallas. *The Adele Software Configuration Manager*, chapter 4, pages 99–139. Trends in Software. J. Wiley and Sons, Baffins Lane, Chichester West Sussex, PO19 1UD, England, 1994.

[EC95] J. Estublier and R. Casallas. Three dimentional versionning. In J. Estublier, editor, *Proc. of 5th Int'l Workshop on Software Configuration Management*, Seatle, Washington, USA, May 1995. ACM, Software Engineering Notes.

[EKF93] R. Elmasri, V. Kouramajian, and S. Fernando. Temporal database modelling: An object-oriented approach. In ACM, editor, *Proceedings of the Conference on Information and Knowledge Management*, 1993.

[Est96] J. Estublier. Workspace management in software engineering environments. In I. Sommerville, editor, *Proc. of 6th Int'l Workshop on Software Configuration Management*, Berlin, Germany, March 1996. Preprint of proceedings.

[EW90] R. Elmasri and G. Wuu. A temporal model and query language for ER databases. In *IEEE Data engineering conference*, 1990.

[Fei91] P.H. Feiler. Configuration management models in commercial environments. Technical Report CMU/SEI-91-TR-7, Carnegie-Mellon University, Software Enginnering Institute, March 1991.

[FLU94] J. Frohn, G. Lausen, and H Uphoff. Access to objects by path expressions and rules. In *Proc. of the 20th Int'l Conf. on Very Large Data Bases*, pages 273–284, Santiago de Chile, Chili, 1994.

[Gad88] S. K. Gadia. A homogeneous relational model and query language for temporal database. 13(4):418–448, december 1988.

[JEG+94] C. Jensel, R. Elmasri, S. Gadia, P. Hayes, and S. Jajodia. A consensus glossary of temporal databasea concepts. *ACM SIGMOD RECORD*, 23(1), march 1994.

[JS92] C. Jensel and R. Snodgrass. Temporal specialization. In *Proceeding of the International Conference of data engineering*. IEEE, 1992.

[KKS92] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In M. Stonebraker, editor, *sigmod*, volume 21, pages 393–402, San Diego, California, June 1992. acm, Acm Press.

[NMY93] N.Pissinou, K. Makki, and Y. Yesha. On temporal modeling in the context of object databases. *SIGMOD RECORD*, 22(3):8–15, septembre 1993.

[RS93] E. Rose and A. Segev. TOOSQL – A Temporal Object-oriented Query Language. In *Proceedings of the 10th International Conference on the Entity-Relationship Approach*, Dallas, TX, 1993.

[SC91] Stanley.Y.W. Su and Hsin-Hsin. M. Chen. A temporal knowledge representation model OSAM*/T and its query language OQL/T. In *Proceedings of the VLDB 17*, Barcelona Spain, 1991.

[Sci91] E. Sciore. Multidimensional versioning for object-oriented databases. *Proc. Second International Conf. on Deductive and Object-Oriented Databases*, December 1991.

[Sno93] R. T. Snodgrass. *An Overview of TQuel*, chapter 6, pages 141–182. In Tanzel et al. TCG+93, 1993.

[Sno95] R. Snodgrass. Temporal object-oriented databases: A critical comparison. In Won Kim Editor, editor, *Modern Data Base systems*. Addison Wesley, 1995.

[TCG+93] A. Tanzel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors. *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings, 1993.

[Tic85] W.F. Tichy. RCS — a system for version control. *Software—Practice and Experience*, 15:637–654, 1985.

[WD93] Gene T.J. Wuu and Umeshwar Dayal. *A Uniform Model for Temporal and Versioned Object-oriented Databases*, chapter 10, pages 230–247. In Tanzel et al. TCG+93, 1993.