Pointwise Temporal Object Database Browsing

Marlon Dumas[†], Chaouki Daassi[†‡],
Marie-Christine Fauvet[†] and Laurence Nigay[‡]

[†] LSR-IMAG, Univ. of Grenoble
BP 72
38402 St-Martin d'Hères (France)
Marlon.Dumas@imag.fr

[‡] CLIPS-IMAG, Univ. of Grenoble
BP 53
38041 Grenoble Cedex 9 (France)
Laurence.Nigay@imag.fr

Abstract Visual object database browsers are essentially based on two kinds of interactions: navigation within a collection of objects, and navigation between objects by the way of their relationships. These two interactional operators have proven to adequately support the main user task addressed by these tools, that is, exploring the states of a set of related objects. In temporal object databases, visual browsing tools should additionally support users tasks such as examining a snapshot of a collection of objects at a given instant, or detecting changes within object states. In this paper, we show that the two interactional operators supported by classical object browsers do not adequately address these tasks. We consequently propose an interactional operator dedicated to navigation through time, and we study how it may be orthogonally integrated with the above two. **Keywords:** temporal databases, object databases, user interface, data browsing

1 Introduction

Three main tasks involving intensive user interaction are classically identified in the context of database management: schema definition, query formulation, and data visualization. Accordingly, three kinds of visual interfaces have been designed and are currently provided by most commercial DBMS: schema browsers and editors, visual query languages, and data visualization tools (see [CCLB97] for a survey).

In this paper, we focus on a family of data visualization interfaces for object databases, known as *object browsers* [PBL+92,DAA+95,DGJS95,CHMW96]. The underlying principle of such interfaces is to provide a visual representation of the data residing within objects, and to offer visual operators for navigating through related objects. Two kinds of interactions are commonly supported by object database browsers: navigation within a collection of objects, and navigation between objects by the way of their relationships.

In the context of a database modeling the evolution of a set of objects and their relationships over a period of time (i.e. a *temporal object database*), the user interface must integrate additional interactional paradigms. Indeed, within such databases, time is a dimension of objects *per se*, orthogonal to their other components. Data visualization tools in this setting must therefore support time-dependent tasks, such as detecting and analyzing changes of object attributes and relationships over time, or exploring a snapshot of the database at a fixed instant. To illustrate these tasks, let us consider a database storing historical data about workers, supervisors and pieces of equipment in a factory's assembly lines. Examples of users tasks are:

- Analyze data about an assembly line at a given date (including its supervisor, workers, equipment and productivity parameters).
- Compare at different dates, a given worker's wage with respect to that of the supervisor of the assembly line to which he is assigned.
- Find out whether the composition of a given assembly line (equipment plus workers) considerably changes when its supervisor does.

In this paper we advocate that existing object database browsers do not adequately support the above tasks, and consequently propose a technique addressing them, namely *pointwise temporal object browsing*. The basic idea of this technique is to display a snapshot at a fixed instant, of a set of path expressions stemming from a given object or collection. The user interacts with this representation, either by navigating through related objects as in classical object browsers, or by modifying the instant with respect to which the visualization is performed. The resulting interface naturally stresses simultaneity by focusing on one instant at a time, but also supports change observation, by providing operators for "jumping" to the next/previous instant where a change occurs in a given visualized path expression.

The structure of the paper is as follows. First, we provide an overview of existing object database browsers and highlight their limitations with regards to temporal databases. We then describe our browsing technique, stressing important aspects such as support for null-valued and collection-valued properties. Lastly, we present the browser's implementation and sketch some future work.

2 Problem description and related works

2.1 Object database browsers

Most object database browsers, e.g. ODEVIEW [DGJS95], SUPER [DAA+95] and PESTO [CHMW96], rely on the concept of *synchronous navigation*. Basically, the browsing interface of these systems consists of a canvas containing a set of forms[1], each of them denoting either a single object or a collection of objects. A form denoting a single object is structured into lines, one per property attached to that object. If a property's type is atomic, its value is directly displayed on the corresponding line. Otherwise, if a property's type is a class or a collection, a clickable button stands in place of its value. When the user

[1] SUPER offers a graph-based visualization mode in addition to the form-based one. However, the underlying principles of both visualization modes are the same.

clicks on such button, a new form displaying the referenced object or collection is added to the canvas.

A collection is displayed in the same way as a single object, except that the corresponding form contains a couple of arrow-labeled buttons. At any time, this form displays the contents of one of the objects within the collection. Clicking on either of the arrow-labeled buttons allows the user to switch to the next or the previous object in the collection. As in the case of a single object, whenever the value of a property is a complex object or a collection, a button is used to denote its value, and clicking on this button opens a new form.

When a form F2 is opened from another form F1, F2 is said to be *dependent* upon F1. This dependency relationship defines a tree structure over the set of forms displayed on the canvas. The term *synchronous navigation* refers to the way dependent forms are semantically linked. Indeed, a form F2 directly dependent upon another form F1, is expected to reflect the value of one of the properties of the object displayed by form F1. Now, if F1 displays a collection of objects instead of a single one, the property values displayed by this form may vary as the user navigates through the objects of the collection. Hence, when the user clicks on the "Next" or "Previous" buttons of form F1, F2 is updated accordingly (in order to maintain the visual consistency between F1 and F2). This principle is then applied to all forms directly dependent upon F2, and subsequently to all forms transitively dependent upon F1.

As a working example, consider the ODMG [CB00] database schema in figure 1.

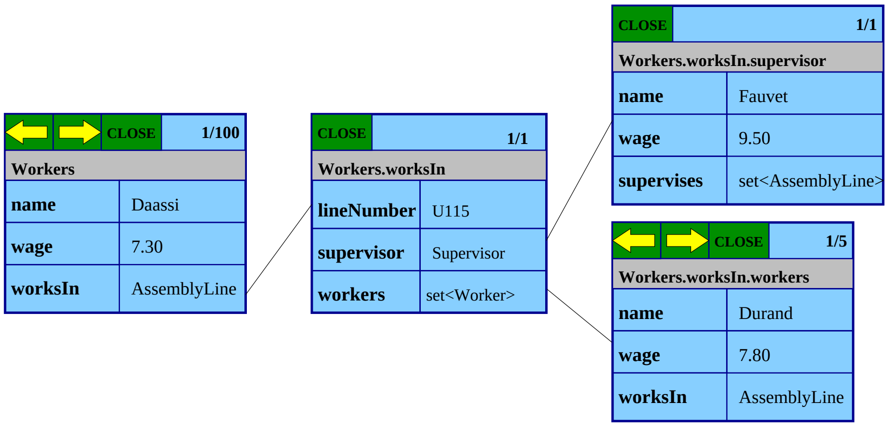
```
class Employee (extent Employees) {
   attribute string name;
   attribute float wage;
}
class Worker extends Employee
                    (extent Workers) {
   attribute AssemblyLine worksIn;
}
```

```
class Supervisor extends Employee
                    (extent Supervisors) {
   attribute set<AssemblyLine> supervises;
}
class AssemblyLine (extent AssemblyLines) {
   attribute string lineNumber;
   attribute Supervisor supervisor;
   attribute set<Worker> workers;
}
```

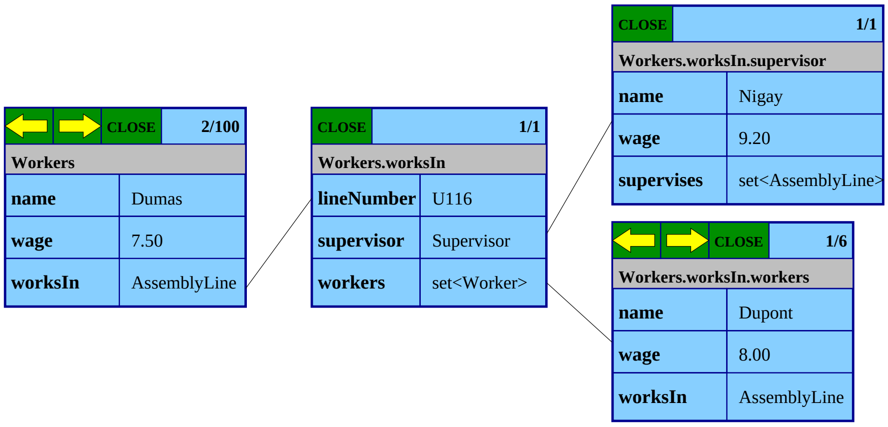
Figure 1. ODMG schema of a non-temporal database.

Figure 2(a) depicts a "synchronized" visualization obtained by starting from the collection Workers and successively clicking on the properties worksIn, supervisor and workers. Notice that the forms dedicated to collections, contain two buttons at the top left corner that enable the user to navigate from one object to another. Figure 2(b) depicts the displayed forms after the user has selected the right arrow in the form entitled Workers.

Object database browsers not based on the concept of synchronous navigation include O_2Look [PBL+92] (a component of the O_2 DBMS). A visualization in O_2Look is made up of a set of forms, each one denoting an object, a tuple or



(a) Visualization of the first worker



(b) Visualization of the second worker

Figure 2. Synchronous navigation as defined in PESTO and ODEVIEW.

a collection. Forms denoting objects and tuples are similar to those depicted in figure 2. A form denoting a collection is composed of buttons denoting object references (one button per object in the collection). When a button within a form denotes an object reference, the user may invoke any method on this object, and in particular a predefined method for visualizing it, namely `display`. By default, this method opens a new form displaying the referenced object. Unlike ODEVIEW and PESTO, forms opened during a session are not explicitly linked.

2.2 Applying object database browsers to temporal data

We define a temporal object database as one whose schema includes *temporal classes and properties*. A class is said to be temporal if at least one of its properties is temporal. A property is temporal, if its value denotes the evolution of an object characteristic over a period of time. The value of a temporal property is a *history*. At an abstract level, a history models a function from a finite set of instants to a set of objects of a given type. For the sake of simplicity, we consider that a history is simply a collection of instant-timestamped or interval-timestamped objects, although there are many other ways of representing a history [DFS99].

Figure 3 describes the schema of our working temporal database, which is actually a "temporal" variant of the schema presented in figure 1. All classes in this schema are temporal. Property AssemblyLine::production is an example of a temporal property whose history is represented through an instant-timestamped collection of objects, while the history of property Employee::wage is represented using intervals. Note that all histories are observed at the granularity of the day, since instants are modeled through the ODMG type Date.

```
typedef struct { Date lb; Date ub; } Interval;
typedef struct { Supervisor value; Interval timestamp; } TimestampedSupervisor;
typedef struct { set<Worker> value; Interval timestamp; } TimestampedWorkers;
typedef struct { long value; Date timestamp; } TimestampedProduction;
typedef struct { float value; Interval timestamp; } TimestampedWage;
typedef struct { AssemblyLine value; Interval timestamp; }
                                        TimestampedAssemblyLine;
typedef struct { set<AssemblyLine> value; Interval timestamp; }
                                        TimestampedAssemblyLines;
class AssemblyLine (extent Lines, key lineNumber) {
  attribute string lineNumber;
  attribute set<TimestampedSupervisor> supervisor;
  attribute set<TimestampedWorkers> workers;
  attribute set<TimestampedProduction> production;
}
class Employee (extent Employees, key name) {
  attribute string name; attribute set<TimestampeWage> wage;
}
class Worker extends Employee (extent Workers) {
  attribute set<TimestampedAssemblyLine> worksIn;
}
class Supervisor extends Employee (extent Supervisors) {
  attribute set<TimestampedAssemblyLines> supervises;
}
```

Figure 3. ODMG schema of a temporal database.

Since histories are assimilated to collections, the concept of synchronous navigation may be applied to browse them. However, the resulting navigation approach is not suitable to explore a snapshot of a set of related temporal objects at a given point in time. Indeed, in the synchronous navigation paradigm, two forms corresponding to two distinct histories do not necessarily display data items with overlapping timestamps, even if one of the forms is dependent upon the other (see for instance figure 4). Therefore, if the user wishes to visualize synchronous states of two temporal properties, he must establish this synchronization by hand.

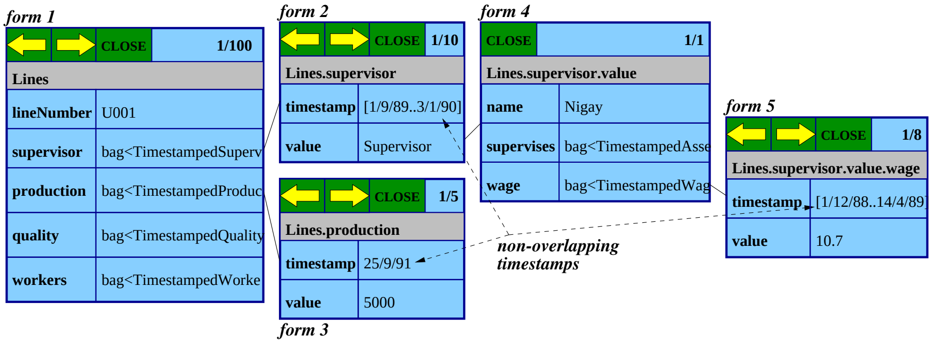


Figure 4. Applying synchronous navigation to temporal object browsing. Notice that forms 2, 3 and 5 do not display synchronous data (i.e. their validity timestamps do not overlap).

In PESTO, this limitation can be partially circumvented using the concept of *synchronizer*, that is, a predicate semantically linking the values displayed by two forms. In the example of figure 4, synchronizers may be used to force the browser to choose the elements displayed by forms 2, 3 and 5, in such a way that the displayed timestamps overlap. In this way, the user always visualizes data with overlapping periods of validity. However, with respect to the task of exploring object attributes and relationships at fixed instants, the resulting navigation mode has two drawbacks:

- Synchronizers should explicitly be introduced each time that a new form is opened.
- Navigation through the objects composing a collection is not orthogonal to navigation through time. For instance, let us consider the example of figure 4 augmented with synchronizers. Suppose that while analyzing data about a given assembly line as of 1/10/1990 (i.e. the timestamps of forms 2, 3 and 5 all three contain or are equal to this date), the user asks to visualize the next object in the collection of assembly lines (i.e. he clicks on the "Next" button of form 1). Although the resulting visualization is such that forms 2, 3 and 5 necessarily have overlapping timestamps (thanks to the synchronizers), there is no guarantee that these timestamps contain or are equal to instant 1/10/1990. Therefore, in order to view the data of the newly displayed assembly line as of this date, the user must furtherly interact with the "Next" and "Previous" buttons of forms 2, 3 and 5.

2.3 Other related works

In the previous section, we show that it is possible to use classical object browsers for temporal database exploration, by simply considering a history as a collection of timestamped objects. Nevertheless, this approach does not fit several important user tasks, such as observing a snapshot of a set of related objects at a given point in time. Before describing our approach to this problem, we present some existing temporal database visualization tools, and discuss to what extent they address the above issue.

Data visualization has received little attention within the temporal database research domain[2] . This is probably due to the fact that efforts in this area have mainly focused on relational models, where visualization tools are often simple, given the flat structure of relations and the lack of explicit links between them.

A notable exception to the above remark is [PT94], which specifies a 3D interface for browsing temporal relational databases. In this approach, a temporal relation is represented as a sequence of time-indexed planes, each one displaying a snapshot of the browsed relation. Although the interface is not detailedly described, it seems that the interactional devices are limited to two scroll-bars: one for browsing through the records of a relation snapshot, and the other for navigating across the time dimension. This interface therefore integrates the concepts of "navigation through a collection" and "navigation through time", but it does not support the concept of "navigation through object relationships" which is fundamental in the context of object databases.

In the setting of information visualization, many techniques for graphically displaying temporal data have been considered [Ber81,Tuf84]. Contrarily to our work which aims at designing a technique for browsing any kind of temporal data, these techniques are specifically oriented towards *quantitative time series* (i.e. periodical series of numerical data items).

[PMR+96] adapts some concepts developed in [Tuf84] to design an interface for visualizing legal and medical personal records involving non-quantitative data such as texts and complex objects. However, this work does not address the issue of comparing the states of two complex objects at a given point in time, nor the issue of navigating through object relationships.

3 Pointwise temporal object database browsing

3.1 Overview

The pointwise browser interface is composed of two parts: a time-line and a tree of form-structured windows (called *snapshot windows*). A snapshot window displays either a non-temporal object[3] or a snapshot of a temporal object at a

[2] On the contrary, several visual query languages for temporal databases have been proposed (e.g. [FSC97]). We do not discuss them since the functionalities addressed by these proposals are beyond the scope of the present paper

[3] An object is *temporal* if it owns at least one temporal property and *non-temporal* otherwise.

given instant. The internal structure of a snapshot window is the same as that of the forms of classical database browsers presented in section 2.1. For the time being, we restrict our examples to snapshot windows displaying single objects or object snapshots. We will discuss afterwards how collections are accommodated.

The instant with respect to which the object snapshots are determined is the same for all the windows in the tree, and is subsequently called the *reference instant*. The reference instant is constrained to reside within a given interval called the *temporal browsing range*.

The role of the time-line window is to fix the reference instant. At the beginning of a session, the reference instant is at the middle of the temporal browsing range. Its position varies thereafter according to the user interactions with the sliders and buttons composing the time-line window. In its simplest form, the time-line window is composed of a slider (called the *main slider*) and four buttons placed at the ends of this slider. Two of the buttons (labeled by simple arrows), allow the user to move the reference instant forward or backward by one unit. The other pair of buttons (labeled with double-arrows) are used to move the reference instant to the next/previous instant where the value of a given navigation path (called the *visualized path*) changes. The instants at which the value of the visualized path changes are called *change instants*. Change instants are visually represented as vertical marks lying within a horizontal line just beneath the main slider.

In figure 5, the visualized path is Workers.worksIn.supervisor and the change instants are those when the supervisor of the worker named "Daassi" changes, whether this change is due to the fact that the worker is assigned to a new assembly line possessing a different supervisor, or to the fact that the supervisor of the assembly line to which this worker is assigned, changes.

In addition to the main slider, the time-line window may contain several *granular sliders*, which allow the user to move the reference instant with different "steps" according to a calendar. For instance, if the reference instant is a date, and that the user specifies the calendar Year/Month/Day (as in figure 5), three granular cursors appear in the time-line window: the first one allows the user to move the reference instant with a step of a year, the second one with a step of a month, and the third one with a step of a day (within the limits of a month).

Snapshot windows are structured as forms containing one line per property of the visualized object or object snapshot. Each line is composed of two boxes: one labeled with the name of the property, and the other labeled with its value at the reference instant[4]. The value of a non-temporal property is always the same regardless of the reference instant. The value of a temporal property at a given instant is equal to the value of its history at that instant, which is itself defined as follows:

- The value at instant I, of a history represented as an instant-timestamped collection of objects, is equal to the object within this collection whose timestamp is equal to I. If no such object exists, the history's value is null.

[4] If the value of a property at the reference instant is not "printable", the name of its class is used as its label.

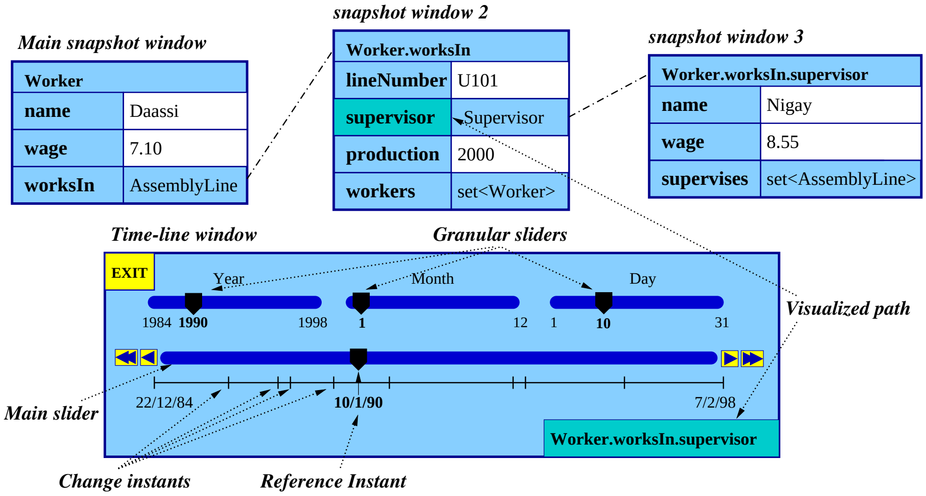


Figure 5. The pointwise browser interface.

- The value at instant I, of a history represented as an interval-timestamped collection of objects, is equal to the object within this collection whose timestamp contains instant I. If no such object exists, the history's value is null.

An outcome of this definition is that the pointwise browser displays temporal properties in the same way as it displays non-temporal ones. For this reason, there is a slight difference between the apparent schema of an object visualized through the pointwise browser, and its actual one. For instance, the schema of the database visualized in figure 5 is the one described in figure 3, and not the one of figure 1 as it may appear at first glance (e.g. the actual type of property Worker::wage is bag<TimestampedWage> instead of float).

All the boxes within a snapshot window are clickable buttons, except those which denote literal values (i.e. integers, reals, string, and characters). For instance, in figure 5 all the boxes within the snapshot windows are clickable, except the white-colored ones.

At the beginning of a session, there is a single snapshot window. Other snapshot windows are incrementally added to the tree as the user clicks on the buttons denoting object references. The object displayed by a given snapshot window other than the main one, is equal to the object referenced by the button from which this window was opened. For instance, the configuration shown in figure 5 is obtained by displaying the worker named "Daassi" and successively clicking on the buttons labeled AssemblyLine and Supervisor.

The user may also click on the buttons labeled with property names (i.e. the buttons on the left column of a form). The semantics of this interaction is that the selected property becomes the visualized path expression and the set of "change instants" attached to the time-line window are updated accordingly. For instance, clicking on the button labeled production on window 2 of figure 5, sets the visualized path to be Worker.worksIn.production.

A whole sub-tree of windows can be removed from the screen by clicking on the button from which the root of this sub-tree was opened. For instance, in figure 5, window 2 was opened by clicking on the button denoting the value of property Worker::worksIn. Therefore, clicking on this button again closes windows 2 and 3. The main snapshot window is closed when the time-line window does.

Whenever the user modifies the reference instant by interacting with the time-line window, the new reference instant is notified to the main snapshot window. Upon receiving this notification, the main window computes the snapshot at the new reference instant, of the object that it displays, and updates itself so as to reflect this new snapshot. During this process, if the value of a temporal property changes, the new value is transmitted to its dependent window if any. Finally, the main window propagates the notification of the new reference instant to all its dependent windows, and the above process is carried out recursively.

Figure 6 shows the result of modifying the reference instant over the example of figure 5.

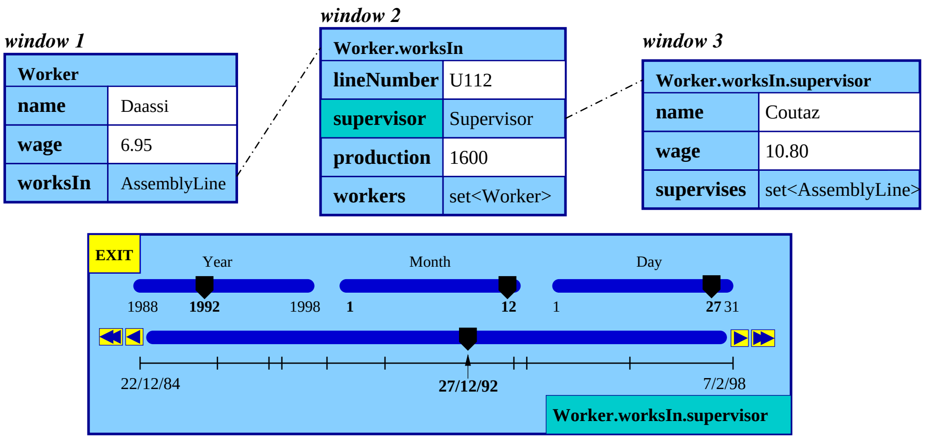


Figure 6. Modification of the reference instant upon the configuration given in figure 5.

As stated before, the reference instant ranges through an interval called the temporal browsing range. This range is chosen so as to cover all the instants where at least one of the temporal properties of the object displayed by the main snapshot window (called the *main object*) is defined. In other words, the temporal browsing range is taken to be the smallest interval containing all the timestamps within the temporal property histories of the main object. For instance, if the main object is a worker (say W) such that:

W.salary = set(struct(VT: [1..4], VS: 10.0), struct(VT: [6..9], VS : 12.0)) and

W.worksIn = set(struct(VT: [2..4], VS: X), struct(VT: [6..8], VS : Y)).

(where X and Y are two assembly lines), then the temporal browsing range is taken to be interval [1..9]. This definition entails that the main object is temporal, since otherwise, the browsing range would be empty.

3.2 Pointwisely browsing in the presence null-valued properties

Heretofore, we have implicitly assumed that all properties displayed within snapshot windows have non-null values. However, null-valued properties within a snapshot window may arise in two cases:

- The value of the property at the reference instant was actually set to "null" through an update (this can occur whether the property is temporal or not).
- The history of a temporal property is not defined at the reference instant, in which case we consider that its value is null. This situation can occur in the middle of a pointwise browsing session, since the temporal browsing range may include instants at which some of the temporal properties of the main object are defined while others are not.

As in O_2Look, we visually denote a null value through a black rectangle. However, this does not solve all the problems arising from nulls. Indeed, suppose that the worker displayed in figure 6 is not assigned to any assembly line at instant 1/5/97 (i.e. there is no element in its history whose timestamp contains this date). If the reference instant is set to this date, the value of property worksIn becomes null, and therefore something has to be done with the forms of the sub-tree stemming from this property (i.e. windows 2 and 3 in figure 6).

In our approach, if further a modification of the reference instant, one of the properties displayed by a snapshot window becomes null, and if this property has a snapshot window attached to it, then all the windows in the sub-tree stemming from this property become *inactive*. Inactivity of a snapshot window can be visually rendered in at least two ways:

- Hide the window (and redisplay it when it becomes active again).
- Modify the appearance of some elements within the window, e.g. by graying out the labels denoting property names and erasing the labels denoting property values.

We believe that the second approach is to be preferred, since hiding and redisplaying windows goes against the screen stability ergonomic principle.

3.3 Pointwisely browsing collections of temporal objects

To accommodate collections, we augment the pointwise browser with the synchronous navigation concept that we described in section 2.1. As illustrated in figure 7, the user can, as a result of this augmentation, navigate through a collection of objects using the arrow-labeled buttons. This new type of navigation is orthogonal both to the navigation via object relationships (which is carried out using the buttons within the right columns of the snapshot windows) and to the navigation through time (which is carried out using the time-line). For instance, in figure 7, when the user clicks on the right-labeled button of the main snapshot window, the next object in the collection of workers is displayed, while the reference instant remains unchanged.

As before, the temporal browsing range is defined with respect to the object visualized by the main snapshot window. Therefore, when this object changes, the browsing range is recomputed. This is the reason why the time-line is redrawn when transitioning from configuration 1 to 2 in figure 7. Notice also from figure 7, that the change instants are recomputed when transitioning from one object to another within a collection.

3.4 Implementation issues

We have prototyped the pointwise browsing technique on top of TEMPOS [DFS99], a temporal extension of the ODMG's standard implemented on top of the O_2 DBMS.

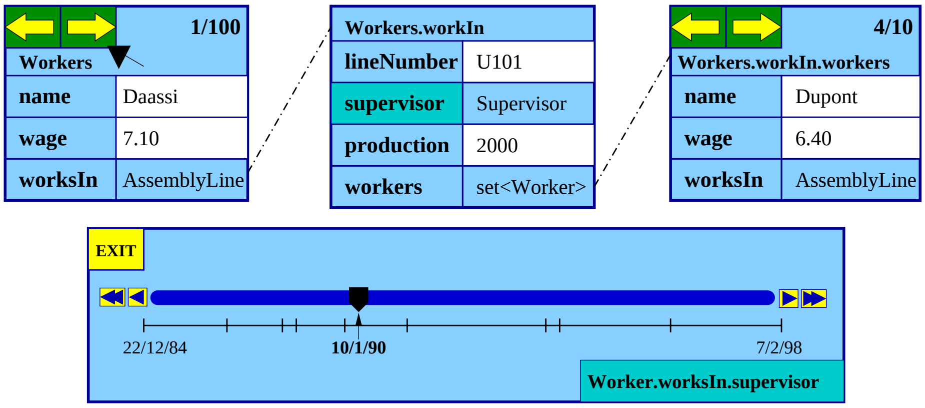
The response time of the prototype is acceptable for all kinds of interactions, except for the computation of the change instants. Indeed, under some circumstances this computation involves a relatively large amount of data. For instance, consider the example of figure 5 and suppose that the visualized path expression is Worker.worksIn.supervisor.wage. Computing the change instants involves the following histories:

- The history of the employee's assembly lines.
- The histories of the supervisors of each assembly line in which the visualized employee has ever worked.
- The histories of the wages of each supervisor appearing within any of the histories referenced in the previous item.

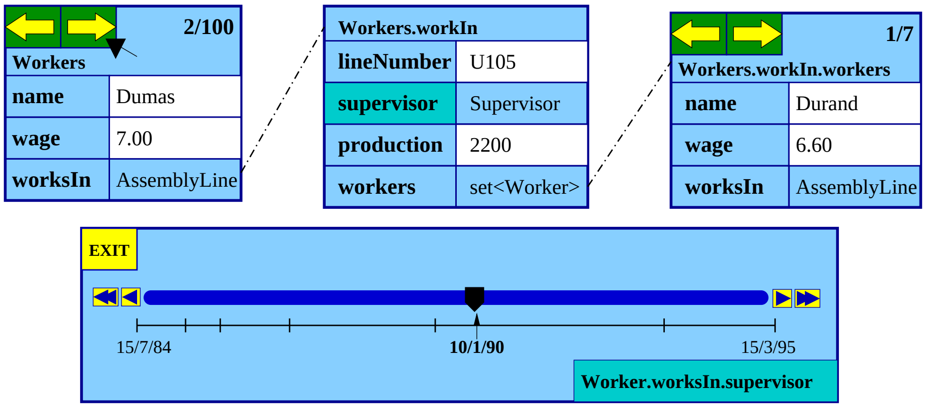
In the current version of the prototype, we use a naive algorithm to compute the change instants. Under some situations, the execution of this algorithm takes about half a second. This leads to some undesirable interactional discontinuities, especially when the main snapshot window displays a collection of objects. Indeed, the change instants have to be recomputed each time that the user clicks on the Next or the Previous button of the main snapshot window. Designing an efficient technique for computing the change instants would therefore considerably improve the current implementation.

The generation of the visual interfaces is carried out using XForms[5], an Xlib-based toolkit providing a rich set of graphic objects such as texts, buttons, sliders and menus. The choice of XForms was guided by its simplicity, efficiency, and adequacy regarding the design of form-structured windows. Thanks to these features, the implementation of the browser's presentation module required less than 800 lines of C source code.

[5] http://world.std.com/~xforms



(a) Configuration 1



(b) Configuration 2

Figure 7. Pointwisely browsing a collection of temporal objects. Configuration 2 is the result of clicking on the right arrow of the main snapshot window in configuration 1. Notice that the value of the reference instant remains unchanged, although the time-line is redrawn.

4 Conclusion and future work

We have presented a new technique for browsing temporal object databases. This technique orthogonally supports three kinds of navigation: (i) navigation through time, (ii) navigation via object relationships, and (iii) navigation within the elements of a collection. Our approach to navigation through time is based on the idea of systematically displaying synchronous values of attributes, relationships, and more generally, arbitrarily complex path expressions involving temporal properties.

In temporal database research, synchronization of temporal attributes and relationships has usually been associated with querying (e.g. SQL/Temporal's *sequenced queries* [SBJS98] and TempOQL's *pointwisely generalized operators* [FDS99]). With respect to these works, our contribution has been to demonstrate the need to integrate this synchronization at the browsing level, and to propose a browsing technique to this end.

As a future work, we aim at extending the pointwise browser into a visual query language. A seamless approach towards this goal is to exploit the concept of "query in place" developed in PESTO. This concept allows one to transform a form-based interface designed for object browsing, into a query formulation tool. Another seamless approach to extend the pointwise browser into a visual query interface, is to integrate its underlying principles into the visual interface to ODMG's OQL reported in [Feg99].

Data contained in a temporal database may either be timestamped so as to reflect the evolution of the modeled reality (valid-time), or to reflect the time when it was inserted/deleted into/from the database (transaction-time). In the former case, the database is said to be *historical*, while in the latter case it is said to be *rollback* [JD98]. In some situations, it is necessary to timestamp data both with respect to the modeled reality and the database evolution, leading to *bitemporal databases*. Up to now, we have limited ourselves to temporal databases managing a single dimension of time, whether valid-time or transaction-time. Extending our work to bitemporal databases could constitute another interesting perspective, provided that concrete applications requiring interactive browsing in this context are exhibited.

References

[Ber81] J. Bertin. *Graphics and Graphic Information Processing.* Walter de Gruyter & Co, Berlin, 1981.

[CB00] R.G.G. Cattell and D. Barry, editors. *The Object Database Standard: ODMG 3.0.* Morgan Kaufmann, January 2000.

[CCLB97] T. Catarci, M. Costabile, S. Levialdi, and C. Batini. Visual query systems for databases: a survey. *Journal of visual languages and computing*, 8(2), 1997.

[CHMW96] M. Carey, L. Haas, V. Maganty, and J. Williams. PESTO : an integrated query/browser for object databases. In *Proc. of the Int. Conference on Very Large Databases (VLDB)*, Mumbai, India, August 1996.

[DAA+95] Y. Dennebouy, M. Andersson, A. Auddino, Y. Dupont, E. Fontana, M. Gentile, and S. Spaccapietra. SUPER: visual interfaces for object + relationship data models. *Journal of visual languages and computing*, 6(1):27 – 52, 1995.

[DFS99] M. Dumas, M.-C. Fauvet, and P.-C. Scholl. TEMPOS: A Temporal Database Model Seamlessly Extending ODMG. Research report 1013-I-LSR-7, LSR-IMAG, Grenoble (France), March 1999.

[DGJS95] S. Dar, N.H. Gehani, H.V. Jagadish, and J. Srinivasan. Queries in an object-oriented graphical interface. *Journal of visual languages and computing*, 6(1):27 – 52, 1995.

[FDS99] M.-C. Fauvet, M. Dumas, and P.-C. Scholl. A representation independent temporal extension of ODMG's Object Query Language. In *proc. of the French Conference on Advanced Databases (BDA)*, Bordeaux, France, October 1999.

[Feg99] L. Fegaras. VOODOO : a visual object-oriented database language for ODMG OQL. In *Proc. of the ECOOP Workshop on Object-Oriented Databases*, Lisbon, Portugal, June 1999.

[FSC97] S. Fernandes, U. Schiel, and T. Catarci. Visual query operators for temporal databases. In *Proc. of the 4th Int. Workshop on Temporal Representation and Reasoning (TIME)*, May 1997.

[JD98] C.S. Jensen and C.E. Dyreson (Eds). The consensus glossary of temporal database concepts – February 1998 version. In O. Etzion, S. Jajodia, and S.M. Sripada, editors, *Temporal Databases: Research and Practice.* Springer Verlag, 1998.

[PBL+92] D. Plateau, P. Borras, D. Leveque, J.C. Mamou, and D. Tallot. Building user interfaces with Looks. In F. Bancilhon, C. Delobel, and P. Kanellakis, editors, *The story of O_2.* Morgan Kaufmann, 1992.

[PMR+96] C. Plaisant, B. Milash, A. Rose, S. Widoff, and B. Schneiderman. LifeLines: Visualizing Personal Histories. In *proc. of the ACM CHI conference*, Vancouver, Canada, April 1996.

[PT94] P. Papapanagiotou and B. Theodoulidis. ERT/vql: A visual environment for querying and manipulating temporal database applications. Technical Report TR-94-5, Timelab, UMIST, 1994.

[SBJS98] R.T. Snodgrass, M. Böhlen, C. Jensen, and A. Steiner. Transitioning temporal support in TSQL2 to SQL3. In O. Etzion, S. Jajodia, and S.M. Sripada, editors, *Temporal Databases: Research and Practice.* Springer Verlag, 1998.

[Tuf84] E. Tufte. *The visual display of quantitative information.* Graphics Press, 1984.