TEMPOS : Managing Time and Histories on top of OO DBMS *

M. Dumas, J.F. Canavaggio, M.C. Fauvet, P.C. Scholl †

1. Introduction

TEMPOS (Temporal Extension Model for Persistent Object Servers) [1, 2] is a multigranular temporal model integrating the main functionalities required to manage the data historical dimensions on top of a DBMS. It is composed of a time model, which defines a hierarchy of simple and complex datatypes for modelling temporal values, and a historical model, which allows to update and query timestamped object properties.

The main originalities of this model are :

- The set of temporal units at which temporal values and histories may be observed is extensible. Moreover, operations on temporal values and histories are independent of the units being involved.
- It allows various representations of histories and provides means for switching between them. Again, operations on histories are independent of their representation. In particular, the operations' semantics ensure coalescing and restructuring whenever it is required.
- Updating operations take into account the type of history to which they apply (i.e. the nature of the phenomena being observed).

2. Managing temporal values

TEMPOS temporal data model provides means for managing simple and complex, anchored and unanchored temporal values observed at any temporal unit. It comprises a wide variety of temporal datatypes (instant, span, interval, set of instants) and a set of operations over them which are independent of the units being involved.

Moreover, the model takes into account the multiplicity of external representations of temporal values by means of *unit systems* and *formats*. *Unit systems* are sequences of comparable units which allow to express temporal values in different calendars while *formats* map strings into temporal values representations in a given unit system. Non-overlapping formats may be grouped into a *set of formats* so that the user can input/output temporal values following his own calendars and syntactical conventions. For instance, having defined the appropriate set of formats and unit systems, the user can write "9/3/95" or "September 3, 1995" to denote the corresponding instant.

3. Managing object histories

The TEMPOS historical model integrates the main functionalities required to manage timestamped object properties (named *histories*).

In TEMPOS, a history is specified by a temporal domain, a sequence of inputted snapshots and a temporal interpolation function. This definition models in a unified framework different historical types (discrete, stepwise, interpolated).

The model allows various representations of histories by grouping instants together in several ways. For instance, a history can be represented as an *I_Chronicle* (ordered sequence of instant-timestamped values), an *X_Chronicle* (coalesced sequence of interval-timestamped values) or a *D_Chronicle*, (set of distinct values timestamped by non-overlapping and non-contiguous sets of intervals). Operations on chronicles allow to switch from one representation to another.

Query operations on histories are stemming from temporal extensions of the relational algebra and from adaptations of iterators on sequences, which allow to reason about succession in time and to express aggregate operations. Updating facilities allow to set and modify both the temporal domain and the sequence of inputted snapshots of a history.

4. Implementation

The TEMPOS model has been implemented on top of the OO DBMS O_2. This implementation consists of a package of temporal classes corresponding to the types described above and a preprocessor for an OQL extension, TempOQL, in which temporal and historical types are primitive in the same way as integer, real, etc. In particular, TempOQL overrides OQL arithmetic operators to deal with temporal values.

Figure 1 illustrates the prototype architecture.

As the figure shows, the administrator can adapt and extend the facilities of the temporal package by defining new temporal units and formats.

The temporal package is about 14000 lines of O_2C[1] code while the preprocessor is about 4000 lines of O_2C, C, Lex and Yacc.

* To appear in EDBT'98 demo session

† LSR-IMAG, University of Grenoble, BP 72, 38402 Saint-Martin d'Hères, France. E-mail : Marlon.Dumas@imag.fr

[1] O_2C is one of the database programming languages provided by the O_2 system

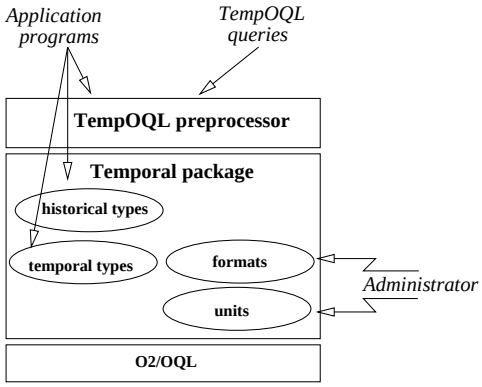

Figure 1: Prototype architecture

5. Examples

The following schema in O_2's DDL illustrates the use of the TEMPOS prototype[2].

```
class Truck
public type tuple(
    licence_number : string,
    bought_on : Instant<Day>,
    maintenances : Discrete_History<
                        Day,
                        tuple(Type : string,
                              Price : real)>,
    driver : Stepwise_History<Day, string>);

name Trucks : set(Truck);
```

The following queries over this schema show some facilities offered by TempOQL.

Q1 : temporal restriction and structural projection
For each truck bought after 1980, retrieve its number and the set of maintenance services done over it between January and September 1995.

```
select tuple(
        number : t.licence_number,
        service : SDomain(Restrict(
                        t.licence_number,
                        [@'1/95',@'9/95'])))
from t in Trucks where t.bought_on > @'1980'
```

- Restrict allows to restrict the temporal domain of a history to a given period.
- SDomain gives the set of values taken by a history over time.

Q2 : temporal product
For each pair of trucks, retrieve the set of instants when they where both simultaneously in maintenance.

```
select tuple(
        truck1 : t1,
        truck2 : t2,
        when : TDomain(IProduct(t1.maintenance,
                                t2.maintenance)))
from t1 in Trucks, t2 in Trucks
```

- The inner temporal product (IProduct) of two histories h1 and h2, is the chronicle whose snapshots are obtained from those snapshots in h1 and h2 which have the same temporal value.
- TDomain gives the temporal domain of a history.

Q3 : reasoning about succession in time
Did Tom ever drive a truck which had previously been driven by Ed?

```
exists t in Trucks :
    Exists(EndOf(t.driver, λs . SV(s) = "Ed"),
           λs . SV(s) = "Tom")
```

- The function EndOf restricts a history to the instants following the first time when the given predicate was true.
- The predicate Exists over a history is true iff there is a at least one snapshot in the history which verifies the given predicate.
- In both lambda-expressions above, the formal parameter s denotes a snapshot. s being a snapshot of a history, SV(s) denotes its structural value.

6. Applications and future research

Two applications have been modelled and implemented using TEMPOS : a toy application about a water-bottling factory, and an application from economics involving time series. In addition, we are studying several spatio-temporal applications from the GIS domain. In order to tackle the spatial dimension of these applications, we expect to integrate TEMPOS with a spatial data model.

Current efforts aim to design and implement a DDL and a DML based on the TEMPOS model and to study issues related to temporal query evaluation, such as optimization and storage structures. Moreover, we plan to accommodate bitemporal operators in the model.

References

[1] M.-C. Fauvet, J.-F. Canavaggio, and P.-C. Scholl. Modelling histories in object DBMS. In *proc. of the 8th International Conference on Database and Expert Systems Applications (DEXA)*, Toulouse (France), September 1997. Springer Verlag. LNCS 1308.

[2] P.-C. Scholl, M.-C. Fauvet, and J.-F. Canavaggio. Un modèle d'historique pour un SGBD temporel. *TSI, Numéro thématique "Bases de données"*, Mars 1998.

[2]The construct *class_name*<*type_name*> denotes a template instantiation.