TEMPORAL EXTENSION TO ODMG

Anya Sotiropoulou
Univ. of Athens
Dept. of Informatics
Panepistimiopolis, TYPA
157 71 Athens, Greece
anya@mm.di.uoa.gr

Michael Souillard
C.R.I. Univ. Paris I Sorbonne
F-75013 Paris - France
MATRA Systémes & Information
F-27106 Val de Reuil - France
souillar@mcs-vdr.fr

Costas Vassilakis
Univ. of Athens
Dept. of Informatics
Panepistimiopolis, TYPA
157 71 Athens, Greece
costas@mm.di.uoa.gr

In the past years a number of temporal extensions to the different database models have been proposed. Extensions to the relational model have been following the different SQL standards, while no attempts have been made to extend the OO-databases' standard, defined by ODMG. In this paper we present a temporal extension to the ODMG standard, as this has been specified in the TOOBIS project. A Temporal Object Data Model, a Temporal Object Definition Language and a Temporal Object Query Language have been specified and have been proposed as extensions to the ODM, ODL and OQL of ODMG. This extension has been implemented over a commercial OODBMS, reinforcing and validating the effort of standardisation and portability of this extension.

1. INTRODUCTION

In the past years the management of temporal data has attracted numerous researchers resulting to a large number of extensions of the traditional database management systems to support temporal applications (Klopproge and Lockeman; 1983, Jones and Mason; 1990, Snodgrass; 1987, Snodgrass and Soo; 1992, Clifford and Tansel; 1985, Tansel et al.; 1989, Ariav; 1986, Segev and Shoshani; 1987, Rose and Segev; 1993, Wuu and Dayal; 1993, Souillard and Pollet; 1997, McKensie; 1986, Tansel et al.; 1993).

Regarding extensions of existing standards, as well as trends for new standards, a first approach has been made in relational databases through TSQL2 Design Committee (1995), which is the result of a common work of researchers involved in temporal data in order to define a bitemporal extension of SQL92. More recently the SQL/Temporal part of SQL3 drafts (Snodgrass; 1996, Snodgrass et al; 1996 (a), Snodgrass et al.; 1996 (b)) presents a possible extension to the emerging SQL standard, keeping in mind TSQL2. However, insofar no extension to the ODMG standard (Cattel et al.; 1993) for object oriented databases has been proposed. In this paper we propose an approach for incorporating temporal data characteristics in object-oriented databases, in a compliant to ODMG way, taking into account the SQL/Temporal proposal, as well as TSQL2. This proposal is made through the TOOBIS[1] - Temporal Object-Oriented dataBases within Information Systems - project, which is an Esprit IV program.

Most of the extensions listed above focus on some specific points of temporal databases. The TOOBIS extension to ODMG aims to supply temporal data management within a full TOODBMS. It covers the ODM, ODL and OQL parts of ODMG standard, version 1.2, supporting the two orthogonal time dimensions, namely valid and transaction time. This extension is upwards compatible with the ODMG standard, in the sense that ODMG's and TOOBIS' worlds - objects and tools - cohabitate in the same OODBMS. TODM, TODL and TOQL have been implemented over O_2's OODBMS (as the implementation by Steiner and Norie (1997)), to illustrate the feasibility and portability characteristics of this extension. All the concepts of temporal databases have been introduced in the object oriented world, making the necessary changes and keeping the model compliant to the ODMG standard, while previous works were based on more or less proprietary and non-portable models (Rose and Segev; 1993, Wuu and Dayal; 1993). E.g. temporal properties and persistency are two orthogonal concepts, whereas the introduction of temporal features within relationships can not be performed without changing the ODMG relationships.

Within the TOOBIS project, a Temporal Object Oriented Methodology - TOOM - (Souveyet et al.; 1997) has also been defined to facilitate the design of temporal applications using the TOODBMS. TOOM extends the basic domains available in Object Oriented Methodologies to temporal domains, as well as object classes and static links to time dimensions, and constraints, events and administration policy allowing to take into account the time dimension

The remnant of this paper is organised as follows: in section 2 an example of temporal data is presented and to be used in the different parts of this document; in sections 3, 4 and 5 the temporal extensions to Object Data Model, Object Definition Language and Object Query Language are described, respectively. Finally section 6 presents the conclusions, the target applications and the future work.

2. PATIENT EXAMPLE

In the Clinical Research domain, patients are examined according to a given number of criteria and measurements. These observations are repeated several times during the treatment period, and give an account of the evolution of the patient status over the time axis. The data resulting from these observations are stored into databases, and checking modules are in charge of detecting incoherence between the different data. The errors detected can be due to mistakes during observation, reporting or acquisition phases. When such errors are detected, a Data Clarification Form (DCF) is emitted per error, in order to identify the incoherence and allow for its correction. A DCF is linked to a patient, and will become obsolete when data will be corrected into the database. A set of DCFs is associated to

[1] TOOBIS partners:01-Pliroforiki, University of Athens, Delta Dairy S.A. for Greece, and MATRA Systémes & Information, O2 Technology, University of Paris I Sorbonne, GlaxoWellcome for France.

each patient. The database is called 'clean' when no more DCFs are emitted by the checking modules.
This small example is very interesting from a temporal data point of view, since it deals with both valid and transaction time dimensions:

- the evolution of the patient's observations corresponds to the history of these observations over the valid time axis (VT), and this history is maintained using instants since patient observation is discrete via several measurements over time;
- the state of a DCF is linked to the database time, i.e. the transaction time axis (TT), as it is created when an error is detected regarding the data of a patient, and it is archived when this error is corrected in the database; then for each patient a history of his DCF is maintained over the transaction time axis.

The following schema, a global structured view built using TOOM, models the above patient example.

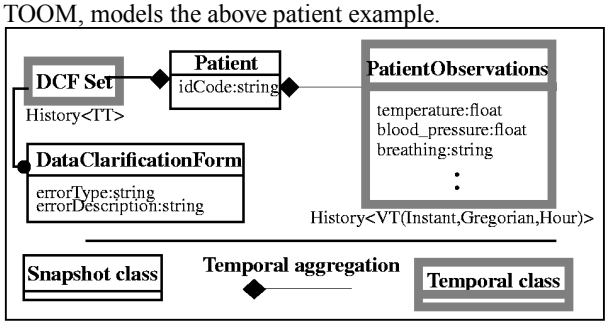

Figure 1. TOOM Global Static View of Patient example

This example will be used in the rest of the paper to illustrate the different components of the TOODBMS.

3. TEMPORAL OBJECT DATA MODEL - TODM

TODM is an extension of the Object Data Model - ODM - on top of which the ODMG-compliant OODBMSs are built. By extending this model, TODM aims at being portable on any of these OODBMS.
Before introducing temporal data within object oriented concepts, the time model used by TODM is described. Then the structures and interfaces used for dealing with temporal data will be presented. Finally, the use of TODM will be illustrated using the patient example.

3.1. TIME MODEL AND MANIPULATION

TODM is based on a classical and almost standard time model. It is a temporal and linear structure where a total order is defined using the 'inferior to' operator. TODM handles a discrete view of this model. The time axis can be divided in a finite number of smaller segments called *granules*. The smallest granule is called *chronon* and its size is implementation-specific. To manipulate time quantities the following entities are defined:

- an instant is a time point on the time axis - e.g. "1997-09-16";
- a period is a quantity of time between two instants, called boundaries - e.g. "[1997-09-01, 1997-10-01)";
- an interval is a duration of time with known length but without specified boundaries - e.g. "1 month" - which may appear as a time window along the time axis.

TODM provides full support for the standard Gregorian calendar with its standard granules - year, month, day, hour, minute and second - as well as provision for multi-calendar support. Each time quantity, anchored or not, is expressed in a given calendar and at a precise granularity. For the Gregorian calendar, leap years and seconds, and time zone specifications are supported. For instance "1997-09-16 15 MET DST" is an instant expressed in the Gregorian calendar at hour-sized granularity, representing the sixteenth of September of 1997 at 3:00 p.m. expressed in the Middle Europe Time time zone using Daylight Saving Time- GMT[2]+2.
A set of arithmetical and comparison[3] operations, such as *precedes*, *meets* and so on, is provided for instant, period and interval manipulation. Relative time, as opposed to absolute dating is also implemented within TODM time support. A form of late binding is used to represent specific instants such as *Now*, *Beginning*, *Forever*[4]*, etc.*

3.2. TEMPORAL DATA WITHIN OBJECTS

The ODM of ODMG defines the characteristics of objects and how they can relate to each other. The basic primitive is the object which has a unique identifier - OID - constant over time. Its state is defined by the values carried by the instance properties - attributes and relationships - and its behaviour is defined by a set of operations. An attribute is of one type, whereas a relationship is defined between two types which must have instances referencable by OIDs. Objects are entities whose values, i.e. states, can evolve over time. TODM is designed to maintain such evolutions over the valid and/or transaction time dimensions.
TODM deals with temporal data on both the instance property level and object level by introducing sub-types of attribute, relationship and object types. Temporal features can not be nested. On one hand, an object which varies over one or both of the time dimensions can evolve as a whole, i.e. all its instance properties which form its state, evolve in the same way. In this case temporal properties are defined at the object level, at which level the state evolution will also be maintained. On the other hand, an object can have instance properties which can vary independently over one or both time dimensions. In this case, the temporal characteristics are defined on the specific instance properties and not at the object level. The next figure formalises the semantics of temporal object and temporal instance property - temporal attribute and temporal relationship, using TOOM representation.

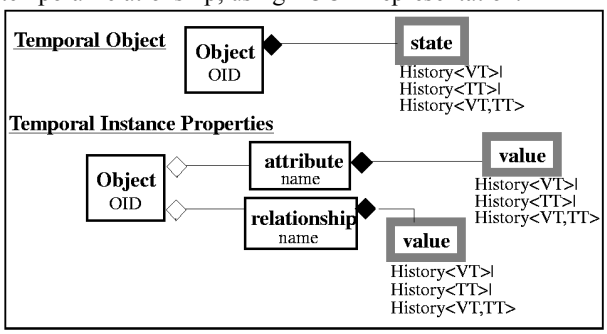

Figure 2. Temporal object vs Temporal instance properties

The way to access uniquely an object is always via its unique OID, regardless of whether the object is snapshot or

---

2 GMT: Greenwich Meridian Time

3 Comparison operations originally drawn from J.F. AllenYs work on the relations between periods, with TSQL2 semantics.

4 *Beginning* is the smallest instant on the time axis - *Forever* is the greatest instant on the time axis.

temporal. However, to access a value of a temporal object, i.e. one of its states, a timestamp has to be added to this OID. Depending on the time dimensions handled, this timestamp will be an instant or a period - cf. Table 1. Accessing temporal data for temporal instance properties, is done in a similar way. The value of an attribute or a relationship is accessed by the selected object and the name of the instance property. To access a value of a temporal attribute or temporal relationship, a timestamp must be added to the (object, instance property name) couple. To support upward compatibility with snapshot objects, the result of accessing a temporal entity without any timestamp argument, is the current value of the entity: the valid and current in the database value at the access execution time.

3.2.1. RELATIONSHIPS AND TEMPORAL FEATURES

A relationship, which is always defined between two instanciable types, can connect either temporal or non temporal types, since the OID is always used to access any object. A relationship, whether it has temporal characteristics or not, can be defined between types which may or may not have temporal characteristics. However, some restrictions are imposed for inverse links, due to time dimensions support. For example a non temporal relationship between two temporal types, the first one evolving over valid time and the second one over transaction time, can not have an inverse link, since the objects evolve on different time axes with no common portion. The inverse relationships, when allowed, can be classified as symmetrical or asymmetrical ones. A symmetrical relationship is one between two temporal types evolving over the same time dimension. An asymmetrical one appears when a temporal relationship (which can only be defined in a non temporal type) points towards a temporal type evolving on the same time dimension. An exhaustive coverage of the allowed inverse links in temporal relationships is given in (Matra Cap Systémes; 1997).

The condition for using an object within a snapshot relationship is that this object exists, i.e. it owns an OID. For temporal relationships and temporal objects, this constraint is more complex. For example a relationship with inverse link maintaining history over valid time, allows an object also maintaining history over valid time to be pointed at, only if a value is defined, in this object, for the valid time timestamp used for this link. More generally, as introduced before, inverse links are only possible within symmetrical or asymmetrical relationships, and a relationship is possible only if the timestamp of the originating entity intersects at least one of the timestamps of the target temporal object. The two schemata below, Figure 3 and Figure 4, illustrate symmetrical vs asymmetrical relationships.

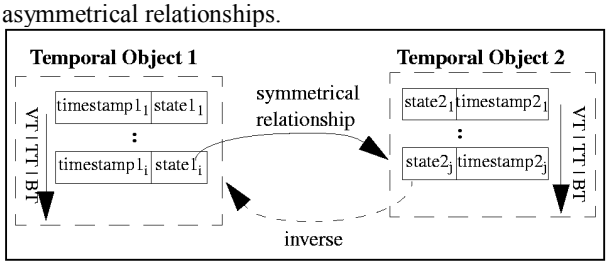

Figure 3. Symmetrical relationship

Two temporal objects varying over the same time dimension, TO1 and TO2:

$TO1 = \{(s1_i, t1_j)\}_{1 \le i \le n}$

$TO1 = \{(s1_i, t1_j)\}_{1 \le i \le n}$

a relationship from a state s_k of TO1 towards TO2 iff

$(\exists j, 1 \le j \le m) / t1_k \cap t2_i \neq \varnothing$

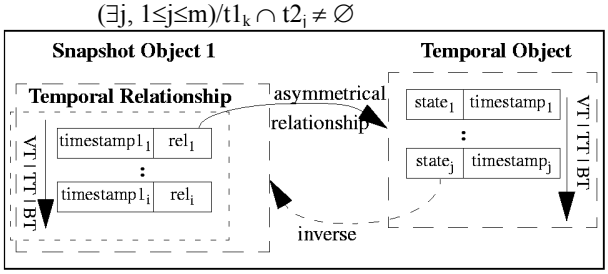

Figure 4. Asymmetrical relationship

A snapshot object with a temporal relationship and a temporal object:

$SOTR = \{(r_i, t_i)\}_{1 \le i \le n}$

$TO = \{(s_j, t_j)\}_{1 \le j \le n}$

a relationship from r_k of SOTR towards TO iff

$(\exists j, 1 \le j \le m) / t_k \cap t_j \neq \varnothing$

In ODMG, relationships are defined between two types. An 1-1 relationship from an object A to an object B, for instance, can be represented using a pointer from A to B; the pointer can be implemented using the object identifier of B. By introducing temporal objects, which can be seen as collections of states of objects, TODM also introduces a new kind of relationship: **state relationships**. A state relationship does not point towards an object but towards a specific state of a temporal object. In this case, the OID of the target temporal object is not sufficient to model the state relationship. Instead, a Temporal-OID (TOID) - the OID of the target temporal object plus the timestamp associated with one of its states - should be used, which allows to precisely select the state of the temporal object involved in the state relationship. Figure 5 illustrates classical relationships versus state relationships.

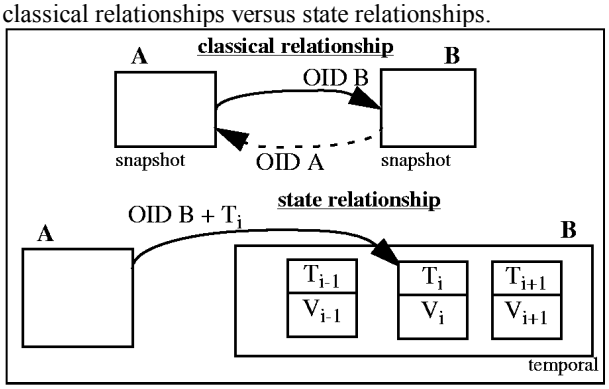

Figure 5. Classical Relationships vs State Relationships

3.3. STRUCTURES AND INTERFACES FOR TEMPORAL DATA

The temporal data supported by TODM can vary over one or both valid and transaction time dimensions. So three types of temporal structures are introduced: Historical, Rollback and Bitemporal entities. The following table gives, for each one of the temporal entities defined, the time dimensions over which they evolve and the timestamp types used to maintain the histories of the evolutions.

Table 1: Timestamp Types used within Temporal Entities

	Valid Time	Transaction Time
Historical	instant: Historical Event period: Historical State | Historical State Overlap[5]	
Rollback		period
Bitemporal	instant: Bitemporal Event period: Bitemporal State | Bitemporal State Overlap	period
Snapshot		

The interfaces of these temporal structures are based on `set/get/delete` operations, analogous to set_value and get_value operations defined in the ODM of ODMG for attributes.

- For temporal attributes, the set_value and get_value operations have been overridden to take into account the temporal features: set_value takes an extra argument for valid time timestamp, and get_value can have valid time and/or transaction time timestamp arguments depending on the time dimensions they evolve over. Note that transaction time timestamps are not supplied by users, but by the system upon transaction commit.
- Regarding temporal relationships, the operations defined in ODM have also been overridden, e.g. the traverse operation enabling to reach the target objects of the relationship, now accepts valid time and/or transaction time timestamp arguments.
- Temporal objects handle object states via set_state and get_state operations: these operations are analogous to the set_value and get_value of attributes, but they handle object states instead of simple attribute values.
- Operations to retrieve the whole histories over one or both of the time dimensions are also defined for all the temporal entities, as for example get_history.

The new kind of relationships introduced by TODM, state relationships, are defined with similar interfaces as classical relationships in ODMG, adding an extra argument to the operations in order to precisely point towards the appropriate state. Temporal state relationships are introduced in the same way as classical temporal relationships.

In temporal data structures, we introduce a new feature emerging from user requirements: the *evolution tracking flag*. In the modeled world represented in the database, the change of a value is due to one of the following reasons:

- the real world evolves, so the value stored in the database has to evolve too; e.g. the temperature of a patient has evolved from 38ºC to 38,5ºC.
- a mistake has been made in the observation of the real world, so the stored value has to be corrected to reflect the exact value; e.g. the value stored for the patient temperature is 38ºC for the last hour whereas the real value is 38,5ºC for the two last hours.

When managing transaction time evolution, all database modifications are kept within the database. Deletions are logical deletions: the values are no more current but are kept in a previous database state. To be able to distinguish an evolution from a correction, TODM introduces the evolution tracking flag. This flag is part of all storage structures of temporal types supporting transaction time, i.e. rollback and bitemporal ones. TODM provides different operations for evolution and correction, setting the evolution tracking flag of the affected data accordingly. Of course a selection on this flag value is allowed when retrieving information stored in the database. The delete operation available in historical entities, is replaced by a correct operation for the rollback and bitemporal entities which performs only logical deletion. To avoid storage explosion due to logical deletions over transaction time data, vacuuming operations are also available.

TODM is implemented over the O2 OODBMS in C++. The different temporal concepts and entities presented have been implemented as C++ classes offering the storage structures and the interface operations to manipulate temporal data. The user-defined time part, dealing with calendars, granules and time quantities, is implemented as a C++ library which can be used with or without temporal data libraries. Some of the C++ classes are introduced in the next part, illustrating the usage of TODM structures via the Patient example.

3.4. PATIENT EXAMPLE

The "Patient Example" can be implemented using TODM structures as shown in Figure 6. The TODM classes are in italic characters as opposed to Patient example classes.

A Patient class is created as a sub-class of Snapshot_Object class, which does not have temporal characteristics, but may contain temporal instance properties. The Patient class has a `char*` C++ attribute, a Snapshot_One_to_One_Relationship relationship towards the PatientObservations class, and a Rollback_One_to_Many_Relationship relationship towards the DataClarificationForm class maintaining the evolution of this relationship over transaction time axis. Mappings from TOOM modeling to TODM structures are defined in the TOOM manual. PatientObservations is a sub-class of Historical_Event_Object class, which deals with the evolution of PatientObservations_State, sub-class of Object_state class, over valid time axis using instants expressed in the Gregorian calendar at hour granularity. DataClarificationForm class is a sub-class of Snapshot_Object class and has some `char*` C++ attributes and the inverse relationship of Patient::dcf, i.e. a Rollback_Many_to_One_Relationship relationship. The next figure depicts all these classes.

Some operations are available for the classes created for this example to get and set data. Although operations are inherited from the super-classes, some of them need to be overridden. E.g. Historical_Event_Object::set_state(Object_State, Instant) is redefined as PatientObservations::set_state(PatientObservations_State, Instant) to allow the storage of the correct state values.

This example introduces the different steps to use TODM structures:

- select the correct classes corresponding to the temporal requirements of the application
- create the appropriate sub-classes of TODM generic classes
- redefine some methods to facilitate certain operations, like type checking.

[5] period P1 overlaps period P2, if the result of the intersection between P1 and P2 is not null.

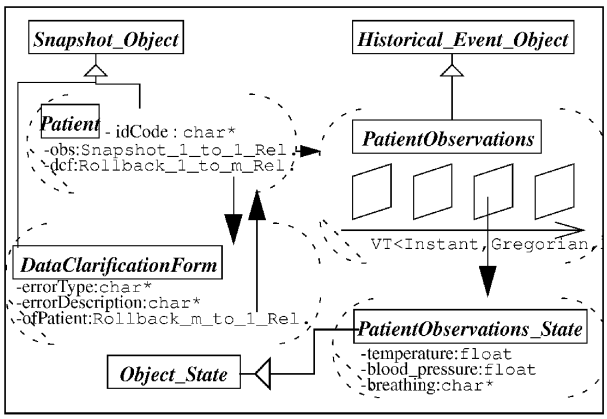

Figure 6. Patient example within TODM

All these steps are performed via the *Temporal Object Definition Language* - TODL - which is described next.

Regarding the selection of temporal data, TODM provides some basic operations to retrieve data; however, selection may be performed in a more user-friendly and powerful way, using the *Temporal Object Query Language* - TOQL - described later in this document.

4. TEMPORAL OBJECT DEFINITION LANGUAGE - TODL

The Temporal Object Definition Language is an extension of ODMG ODL. The TODL user may define interfaces which have properties such as keys and extents, instance properties (attributes and relationships) and operations.

As in TODM, temporal characteristics may be applied either at instance property or at object level, and are mapped to the appropriate TODM structures. For instance an attribute with valid time characteristics is a TODM historical attribute, while an interface defined to have transaction time is a TODM rollback object. The user may also define that the result of an operation and/or some operation arguments have temporal characteristics. Finally he may define new calendars.

4.1. DEFINING TEMPORAL CHARACTERISTICS

To define a temporal object or a temporal instance property, two new clauses are introduced, namely valid and transaction. When defining a temporal object, these clauses should appear immediately after the inheritance specification list and before the interface properties (extension and key definitions). In order to define an instance property with temporal characteristics, these clauses are placed at the end of the instance property definition. Their syntax is given in the following lines:

valid [**event** | **state**] [**overlaps**]
[**granularity** <granularity>]
[**calendar** <calendar>] **transaction**

The valid keyword is used to define a historical object, as this is described in TODM. The event and state keywords are used to specify the type of valid time timestamps (instants or periods, default is period). In the case of period representation the user may also define that the valid time timestamps may overlap with each other, using the keyword overlaps. The default is no overlapping. Finally the granularity and calendar subclauses are used to select the granularity and calendar at which the valid time timestamps are to be expressed.

The transaction clause defines a rollback object. Timestamps of rollback objects are always periods of the default granularity and calendar. If both valid and transaction clauses are present, a bitemporal object is defined.

4.2. USER-DEFINED TIME

The user may define attributes of user-defined time types like instants, intervals or periods. This approach gives the user greater flexibility than the predefined ODMG types for time, as he may select the granularity and/or the calendar he prefers. Also definition of relative instants and relative periods is allowed through the relative keyword. In the following examples we show how the user may define attributes having user-defined time.

```
attribute Instant granularity day bdate;
attribute Period granularity semester
   calendar academic attending;
```

4.3. USER-DEFINED CALENDARS

In TODL the user is allowed to define his own calendars, if he believes that in such a way his application will be better served. Calendar definition is made through the calendar statement, which may be included in a TODL definition's file. To define a calendar the user has to define the different granules, with constant mapping to finer and coarser ones (based on the chronon), the name and the origin of the calendar and 7 functions (Matra Cap Systémes; 1997). The functions must be defined by the user outside TODL, using a language like C++.

4.4. THE PATIENT EXAMPLE IN TODL

The classes needed for the "Patient Example" can be declared in TODL as follows:

```
interface Patient
(extent Patients key idCode)
{
   attribute String idCode;
   relationship PatientObservations obs;
   relationship Set<DataClarificationForm>
      dcf transaction inverse
         DataClarificationForm::ofPatient;
}
interface PatientObservations valid event
   granularity day
{
   attribute float temperature;
   attribute float blood_pressure;
   attribute String breathing;
}
interface DataClarificationForm
{
   attribute String errorType;
   attribute String errorDescription;
   relationship Patient ofPatient
      transaction inverse Patient::dcf;
}
```

A file containing the above definition will be passed to the TODL parser, producing a header file containing the C++ definitions of the classes described in Section 3.4., along with possible redefinitions of some of the methods. The user must then invoke the o2import command to import the class definitions in the O_2 database system (University of Athens et al. (a); 1997).

5. TEMPORAL OBJECT QUERY LANGUAGE - TOQL

TOQL (University of Athens et. al (b); 1997) is an upwards compatible extension of OQL v.1.2 (ODMG standard commitee; 1997) providing extensions for management of temporal data. These extensions adhere to the overal OQL syntax and allow temporal and non-temporal data to be treated uniformly, without making the syntax of the language unnecessarily complex.

5.1. DATA TYPES FOR TIME

Through TOQL the user can manipulate any of the data types provided for time representation by TODM. For each of those type, literals may be constructed, using the notations depicted in Table 2.

Table 2: Literals for time representation

Literal	Value
instant '1990' granularity year calendar Gregorian	An instant for the year 1990 of the Gregorian (default) calendar
interval '2' granularity year	An interval of two years in the default calendar
period '[Winter 1986, Spring 1997)' calendar academic	A period starting at the Winter semester of the academic year and ending at the Spring semester of the academic year 1997

A number of functions, predicates and operators are supplied for data types used for time representation. Functions include period constructors, intersection and merge (union) functions for periods and period sets, extractors of certain parts of instants (e.g, year), etc., as well as a syntactic construct for casting a datum of the above mentioned types to different granularities. Temporal predicates include overlaps, precedes, contains and meets as defined by the TSQL2 Design Committee. Finally, operators include standard arithmetic and set theoretic operators such as adding two intervals, multiplying an interval by a number, etc. Note, that automatic conversions are applied to the arguments of the above mentioned functions, predicates and operators when necessary. These conversions include granularity conversions (left operand semantics are adopted) and conversions from instants to periods and from periods to period sets (type conversions). For a complete description of the supported operations and conversion rules, see (University of Athens et al. (b), 1997).

5.2. QUERIES ON TEMPORAL DATA

In order to preserve the compatibility with snapshot (legacy) applications, pure OQL queries always use the current value of temporal instance properties, when these are referenced in the query, while no conversion is performed for whole objects. Since all interaction with objects is performed via the operations by retaining their signatures intact (including the set_value and get_value operations) the necessary compatibility level[6] is provided.

In order to retrieve the complete histories of historical, rollback and bitemporal data, the modifiers valid, transaction and bitemporal may be prepended, respectively to the queries. These modifiers return the histories of their operands as they are stored in the database, while the application may use methods provided by the interface of valid time, transaction time and bitemporal data, in order to extract specific values or iterate over variants. When applied to bitemporal data valid and transaction modifiers convert them to historical or rollback objects, by dropping non-current and not presently valid variants, respectively.

Temporal objects may also be treated as indexed collections orthogonally to collections supported by OQL. Indexing (or *subscripting*) may be performed using integers or integer ranges to retrieve variants with specific ranks[7] as well as instants and periods, to select the desired time window. Subscripting may result to a single variant or a set of variants, depending on the types of the subscripted data and the subscript. If a single value is returned then the result is subject to the modifier weighted (which multiplies the value of the result by the timestamp's duration, if such a multiplication is meaningful) and may be used as argument to functions valid and transaction (which return the valid and transaction time timestamps respectively).

Example: In the following examples we present how to access certain variants of temporal data. In the first query we access the first variant of the transaction time relationship *dcf* of each *Patient* object, using the rank of the variant; in the second query, the variant that was current on Jan. 1, 1996 is accessed, using an instant to designate the desired variant.

```
select (transaction p.dcf)[0]
from Patients as p
select (transaction p.dcf)[instant '1996-01-01'
granularity day] from Patients p
```

When applied to rollback or bitemporal data, the default behaviour of all subscript operators is to disregard the variants tagged as *deleted* and consider only variants tagged as *evolved* or *current*. However, access to the deleted variants is provided by appending to the subscript expression one of the keywords evolved, deleted or all. The evolved keyword is the default.

Subscript expressions apply orthogonally to temporal instance properties and temporal objects.

5.3. REFERENCING OBJECT VARIANTS

Temporal data are actually collections of values, with each value having associated with it one or two timestamps, representing valid and/or transaction time. In the same way that OQL allows collections to be used for variable definition and in collection expressions, TOQL allows temporal instance properties and temporal objects to be used for the same purpose. When a variable is defined in terms of a temporal datum, it iterates over the different variants stored in the temporal datum, and all variants - regardless of their timestamps- are considered. TOQL supports all forms of variable declaration defined in OQL v. 1.2 i.e. in the *from* clause, in existential and in universal quantification.

Temporal data may be used in any place a collection is allowed in a membership testing query, in which case each variant of the temporal datum is tested for equality against to the left side of the query, and if such a variant exists the expression evaluates to true, otherwise to false. Finally, temporal data may be used as right-hand side queries in composite predicates e.g. e_1 relation some e_2, where e_2 is an

[6] Of course, new operations should be added, to provide access to the temporal dimension(s).

[7] Variants are ordered with respect to their timestamps

expression yielding a temporal object and e_1 an expression which has the snapshot type of e_2 and relation is a relational operator.

Example. The following query selects the Patient objects which have at least one observation with temperature greater than $40^{oC:}$

```
select p from Patients p where exists po in
valid state p.obs: po.temperature > 40
```

5.4. CONVERSION BETWEEN TEMPORAL AND SNAPSHOT VALUES

Prepending the snapshot modifier to any query returning a temporal object, results to dropping all timestamps and returning only plain values.

The valid modifier is used to construct valid time objects form value/timestamp pairs, giving the ability to select the granularity, calendar and overlap mode of the result.

The transaction modifier converts a snapshot datum to a rollback one, containing a single variant whose transaction timestamp is set to [NOW, UC). No provision is made for assigning transaction timestamps, so as not to force past or future values.

Finally, similar to the valid modifier the bitemporal one converts collections to bitemporal objects with analogous functionality and syntax.

5.5. TEMPORAL JOINS

TOQL does not perform temporal joins when two pieces of temporal data appear in the **from** clause, as TSQL2 Design Committee proposes (1995); instead, it provides an explicit operator, **tstruct**, for temporal joins. Its syntax is similar to the one of the **struct** operator:

tstruct (*id*: *query* {, *id*: *query* })

where each *query* evaluates to an historical object. The tstruct operator constructs a list of structures with one field for each argument of the **tstruct** operator, with type the non-temporal part of the argument's type, and one more field of timestamp type. Only historical objects may be combined so as not to construct future or past transaction time timestamps. Bitemporal and rollback objects should be first converted to historical ones. In all cases, the result of a temporal join contains a structure for each set of variants (one variant from each argument) that have overlapping or identical timestamps, depending on the types of the arguments (University of Athens et al. (b); 1997).

5.6. RESTRUCTURING OPERATORS

Restructuring operators facilitate the formulation of different equivalent representations of temporal data. Two restructuring operators are provided, with the first one converting period-timestamped variants to instant-timestamped ones, while the second restructuring operator allows for selection of the time axis on which maximal timestamps will be produced, similarly to the SL_P and SLP^{BS} operators defined by Soo et al. (1995).

5.7. AGGREGATION

TOQL provides two types of temporal partitioning, in addition to OQL's standard grouping mechanism.

The first type allows for splitting of a single temporal datum into variant subsets, each pertaining to a specific portion of the time axis. Partitioning may be performed either on the valid time or on the transaction time axis. The user specifies the desired time axis and an interval, which is used as the basic partitioning unit for the chosen time axis. For example, an interval of 1 year specifies that the chosen time axis will be partitioned into segments with duration equal to one calendric year and variants will be included into some partition, if they contain information pertaining to the associated segment. The syntax of this form of aggregation is:

(partition *time_axis* **as** *interval_query*
[**leading** *interval_query*]
[**trailing** *interval_query*] [**as calendar**]**)**
Temporal_Object

where *time_axis* specifies the time dimension on which the splitting operation will take place. The leading and trailing subclauses specify an optional extension of the basic partitioning unit towards the beginning or the end of the time axis, while the as calendar clause, if present, specifies that the starting point of the calculation is the start of a calendric unit.

Each variant subset, produced by the operation, is tagged with the time axis period it pertains to.

Example. The following query returns the patient observations for the patient P042, partitioned in one-year subgroups.

```
select (partition valid as interval '1'
granularity year) valid state p.obs
from Patients as p where p.idCode = 'P042'
```

The second type of partitioning allows for combination of variants from multiple temporal data into groups, with each group pertaining to a specific portion of either the valid or the transaction time axis. Groups may be filtered, depending on whether they satisfy some condition, and aggregate values may be computed over elements of each group. This form of partitioning is provided via a special form of the **group by** clause, in which the grouping extension is a time dimension (valid or transaction time), with an associated basic partitioning unit and, optionally, unit extensions towards the beginning or the end of the time axis. Group filtering and aggregate value computation is performed using the standard OQL mechanisms, i.e. the *having* clause and aggregate functions, respectively. The time dimension designated in the **group by** clause must occur in the objects resulting from the *select* query. The syntax of this form of partitioning is:

group by partition *time_axis interval_query*
[leading *interval_query*]
[trailing *interval_query*] as *identifier*

where *time_axis* specifies the time dimension on which the partitioning takes place and the leading and trailing clauses have the same semantics as in the first type of partitioning.

Example. The following query returns the observations for all patients, split in one-year subgroups.

```
select * from (select obs from Patients)
group by partition valid interval '1'
granularity year
```

5.8. THE TOQL PROCESSOR

The TOQL processor is being implemented as a software module functioning on top of the OQL processor of O_2. TOQL queries are intercepted and transformed to OQL queries, which are submitted to the OQL processor for evaluation. The results returned by the OQL processor are forwarded to the user, or the application that has issued the TOQL query (University of Athens et al. (b); 1997).

Special provision has been included in the TOQL specifications for queries to be submitted from within C++ programs to the TOODBMS. This includes the definition of temporal specialisations of ODMG C++ Binding classes and functions (O_2Technology; 1996, University of Athens et al. (b); 1997).

6. CONCLUSION AND FUTURE WORK

In this paper we presented an extension to the ODMG proposal for Object Oriented Databases to manage temporal information. This extension has been proposed within the TOOBIS Esprit IV project and is implemented over the O_2 OODBMS, in two operating systems platforms, namely Solaris OS and Windows NT. This extension, build over the ODMG standard, aims at being valid and portable on any ODMG compliant OODBMS.

In the TOOBIS program, two pilot applications are built: they are two applications that we can classify as *data intensive applications*. The first one is dedicated to the management of data coming from Clinical Research and can be sub-classified as managing and auditing oriented application. The second pilot deals with optimization of production and transport of fresh products and can be sub-classified as time-dependent and audit-oriented application. In fact TOOBIS TOODBMS could target any temporal data application such as medical information systems, civilian crisis, banking systems or in the military context the Command Communication and Control Information - C^3I - systems, and so on.

One of the future tasks is porting the proposed temporal extension to another OODBMS to prove the standard property of this extension, and of course to industrialize it in a more trading-oriented goal. A more user friendly interface is under design, as well as an extension to cover other OO-languages (e.g. Java). Regarding temporal features, we plan to extend the temporal management to temporal data schemas.

7. BIBLIOGRAPHY

- Ariav, G. "A temporally oriented data model". ACM Trans. on Database Systems, Vol11, No 4, 1986
- Catell, R.G.G, et al. "ODMG-93". International Thomson Publishing.
- Clifford, J., Tansel, A.U. "On an algebra for historical relational databases : two views". Proc. Int. ACM SIGMOD Conf., 1985.
- Jones, S., Mason, P.J. "Handling the time dimension in a database". Proc. Int. Conf. on "Databases", The BCS, Univ. of Aberdeen, 1980.
- Klopproge, M.R., Lockeman, P.C. "Modelling information preserving information databases : consequences of the concept of time". Proc. 9th Int. Conf. on VLDB, Florence, Italy, 1983.
- Matra Cap Syst mes, "TODM Specification and Design - long version", Deliverable T31TR.1 of TOOBIS Esprit IV project[8], 1997
- McKensie, E., "Bibliography: Temporal Databases". ACM SIGMOD Record, Vol. 15 No. 4, 1986
- O_2 Technology, "ODMG C++ Binding Guide" (for release 4.6), 1996.
- The Object Database Standard: "OQL v.1.2" available from http://www.odmg.com/, 1997
- Rose, E., Segev, A. "A Temporal Object Oriented Query Language" Lectures Notes in Computer Science #823, 1993
- Segev, A., Shoshani, A., "Logical Modelling of Temporal Databases". Proc. of ACM SIGMMOD International Conf. on the Management of Data, 1987.
- Snodgrass, R.T., "The temporal query language TQuel". ACM TODS, Vol12, No 2, 1987.
- Snodgrass, R.T., Soo, M.D., "Multiple Calendar Support for Conventional Database Management Systems". Dept. of Computer Science, University of Arizona, TR92-07, 1992
- Snodgrass, R.T. "A Road Map of Additions to SQL/Temporal". ISO ANSI X3H2-96-013, 1996.
- Snodgrass, R.T., et al. "Adding Valid Time to SQL/Temporal". ISO/IEC JTC1/SC21/WG3 DBL MCI-142, ANSI X3H2-96-151r1, 1996 (a).
- Snodgrass, R.T., et al. "Adding Transaction Time to SQL/Temporal". ISO/IEC JTC1/SC21/WG3 DBL MCI-143, ANSI X3H2-96-152r, 1996 (b).
- Soo, M.D., et al., "An Algebra for TSQL2" in TSQL2 Language Design Committee (1995) chapter 27.
- Souillard, M., Pollet, Y. "TOOBIS: une approche pour la Gestion de Données Evolutives et Temporelles dans la Recherche Clinique". Interfaces 1997, Montpellier/France.
- Souveyet, C. Deneckere, R., Rolland, C. "TOOM" TOOBIS Deliverable T23D1.1., 1997.
- Steiner, A., Norrie, M.C., "A Temporal Extension to a Generic Object Data Model", Proc. of 9th CAiSE, TimeCenter Technical Report #15, 1997.
- Tansel, A.U. et al. "Time-by-example query language for historical databases". IEEE Trans. on Soft. Engineering, Vol15, No 4, 1989.
- Tansel, A.U., et al. "Temporal Databases: Theory, Design and Implementation". The Benjamin/Cummings, 1993.
- TSQL2 Language Design Committee. "TSQL2" edited by R.T.Snodgrass. Kluwer Publishers. Sept. 1995.
- University of Athens, 01-Pliroforiki S.A., O_2 Technology, "TODL Specification and Design", Deliverable T32TR.1 of TOOBIS Esprit IV project (a), 1997.
- University of Athens, 01-Pliroforiki S.A., O_2 Technology, "TOQL Specification and Design", Deliverable T33TR.1 of TOOBIS Esprit IV project (b), 1997.
- Wuu, G.T.J., Dayal, U., "A Uniform Model for Temporal and Versioned Object-Oriented Databases" in Tansel, A.U. et al. (1993) chap. 10.

Acknowlegments

The authors would like to acknowledge Dr. Carine Souveyet and Rebecca Deneckere, in the University of Sorbonne, as well as Dr. Yann Pollet in Matra Systémes et Informations for their help and constructive comments. Anya Sotiropoulou's work was partially funded by a scholarship from the Greek National Scholarships' Fund.

[8] All TOOBIS Deliverables are available at http://www.di.uoa.gr/~toobis/Deliverables.html