GCH-OSQL
A Temporally-Oriented Object-Oriented Query Language Based on a Three-Valued Logic

Carlo Combi (§) and Giorgio Cucchi

§ Laboratory of Artificial Intelligence
Dipartimento di Matematica e Informatica
Universita' degli Studi di Udine
via delle Scienze 206, 33100 Udine - Italy -
email: combi@dimi.uniud.it

Abstract

The need of managing temporal information given at different levels of granularity and indeterminacy is common to many application areas. Between them, we focus on the management of clinical data. Different time granularities are also needed in querying temporal databases. In this paper, we describe GCH-OSQL (Granular Clinical History - Object Structured Query Language), an object-oriented temporally-oriented extension of SQL. GCH-OSQL is based on an object-oriented temporal data model, named GCH-OODM. GCH-OODM allows storage of temporal information at different and mixed granularities. GCH-OSQL deals with the valid time of temporal data. The temporal extension of the SELECT statement includes the addition of the TIME-SLICE and MOVING WINDOW clauses, and the capability of referring to the temporal dimension of objects in the WHERE and SELECT clauses.

1. Introduction

Among still open problems in research on temporal object-oriented databases we focus on the management of different time granularities, i.e. different time units, and the management of temporal indeterminacy in locating intervals and instants [21, 22, 24]. Motivation in considering this kind of problems in temporal object-oriented databases comes from our interest in managing complex clinical temporal information [12, 13].

In this paper we describe the query language GCH-OSQL (Granular Clinical History - Object Structured Query Language); GCH-OSQL is based on an object-oriented data model, named GCH-OODM (Granular Clinical History - Object Oriented Data Model). GCH-OSQL deals with the following specific issues.

• **Temporal granularity and temporal indeterminacy modeling**. The temporal dimension is expressed by using sometimes the concept of interval, for facts having a span of time, and sometimes the concept of instant for punctual events [12, 22]. Temporal dimension, moreover, can be expressed in different and heterogeneous way: the adopted time axis, for example, has different time units ("in 1989 the patient had myocardial infarction", "at 1:00 p.m. the patient had an episode of atrial fibrillation: it lasted 48 minutes"); in other cases the temporal location is expressed with some indeterminacy ("between 17:30 and 18:15 the patient had for 6 seconds ventricular fibrillation"). Furthermore, different granularities and indeterminacies can be used in many varying ways to define temporal intervals ("at 17:30 for 20 seconds", "for 3 hours until 16:20", "start between 14:20 and 14:33; end between 18 and 18:30"). At the best of our knowledge, there are no temporal query languages, from literature, able to represent all the different granularities of the above mentioned valid-time intervals [21, 22].

• **Supporting time granularity in querying the database.** In querying the system about the stored temporal information we usually need granularities different and not related to those used when storing data. In the same query different granularities may be used.

• **Managing uncertainty in temporal relationships.** In dealing with information having different granularities and/or indeterminacy, it is possible that in some cases temporal relations cannot be asserted for sure. It is not possible, for example, establish the before/after relations between two pathologies, represented, respectively, by the sentences "myocardial infarction on October 22, 1994" and "atrial fibrillation episode in October 1994".

2. The object-oriented temporal data model GCH-OODM

GCH-OODM is an object-oriented data model, extended to consider and manage the valid time of information. The object oriented data model is C++-like [3]: the database schema consists of a set of *classes*. Objects are created as instances of a *class*. In the following we will use the terms *class* and *type* as synonyms, to describe the proposed data model [3]. An object is characterized by a *state*, described by attributes, not accessible from outside, and by an *interface*, defined by methods. By methods, properties of objects and relations between objects of the same class or of different classes are expressed. Each method has a declaration, consisting of a name, a set of parameters, identified by name and type, and a result, identified by a type. Like attributes, code associated to the execution of a method is not accessible from outside. From a notational point of view, the reference to a method *m*(...) applied to an object *o* will be written as *o.m*(...).

GCH-OODM supports the main features of object-oriented data models applied to databases: object identity, encapsulation, abstract data types, single inheritance, polymorphism, management of complex objects, persistence [16].

Besides the usual types (char, char*, int, real, array, list, set, ...), GCH-OODM uses, to model the temporal dimension of information, some predefined data types: the type hierarchy *el_time*, *instant*, *duration*, *interval*; the type *t_o_set*, by which sets of temporal objects are modeled. Temporal granularities are modeled by the type *granularity*; objects of this type can have values belonging to the set, ordered by descending values, {SUP, *yy*, *mm*, *dd*, *hh*, *min*, *ss*, INF} composed by the symbols of the considered time units. GCH-OODM relies on a three-valued logic modeled by the type *bool3*, to manage uncertainty coming from comparison between temporal dimensions expressed at different granularities.

2.1. Managing the three-valued logic: the class *bool3*

The presence of different granularities and of indeterminacy leads us to manage relations between intervals possibly having, besides the two logical values true or false, an indeterminate logical value: it is not always possible to establish with certainty the truth or the falsehood of relations existing between intervals [12].

We then use a three-valued logic: in addition to the values T: *True* and F: *False*, the value U: *Undefined* is present. The usual logical connectives AND, OR, NOT, IMPLIES, ..., and the logical quantifiers EXISTS (∃), and FOR EACH (∀) have been extended to consider the third truth value UNDEFINED.

The adopted three-valued logic derives from Kleene's logic, where the third truth value U is related to situations, about which it is not possible to know the truth or falsehood [19]. In comparison with the Kleene's logic we added the new logical connectives T(), U() and F(), to explicitly manage each of the three truth values. The interpretation of the logical connectives, depending on the values of the formulas A and B, is described by the following truth tables. In GCH-OODM formulas may consist in: a) methods returning a logical value (managed by the class *bool3*); b) comparison operations between objects returned by suitable methods and/or suitable typed constants; or c) composition by the logical connectives of formulas of type a) or b).

A AND B	T	U	F
T	T	U	F
U	U	U	F
F	F	F	F

A	NOT A	T(A)
T	F	T
F	T	F
U	U	F

The meaning of the other logical connectives can be defined by the above defined ones: A OR B stands for NOT((NOT A) AND (NOT B)); F(A) stands for T(NOT A); U(A) stands for NOT (T(A) OR T(NOT A)).

This three-valued logic is managed by the predefined data type *bool3*.

2.2. Time modeling

The class *el_time* allows us to model time points on the basic time axis, named elementary instants. Each elementary instant is identified by the corresponding chronon, i.e. the nondecomposable unit of time supported by the temporal DBMS [22, 24]. By the class *el_time* properties of integers are extended to the time axis. This way, both time points and spans between time points are modeled in a homogeneous way: time points are identified on the basic time axis by their distance from the origin of the axis. The class *el_time* provides, then, functions both to manage the absolute location of time points on the time axis - i.e., calendar-related functions able to deal with leap years, months having 28, 29, 30, or 31 days - and to manage time spans - i.e., functions able to perform operations on time spans, not directly anchored to the calendar - and also to compute sum and difference operations on time points/time spans. For clarity reasons we will use two different formats, to specify respectively time points and distances between time points. We use the usual calendric format `YY/MM/DD/HH/Mi/SS` to specify a time point. We will use the format `Y yy M mm D dd H hh Mi min S ss`, to identify a distance between time points (`Y`, `M`, `D`, `H`, `Mi`, and `S` stand for values related to the corresponding time unit). The time point `94/10/10/0/0/0`, for example, identifies the first second of October 10, 1994; the time span `5 min 2 ss` identifies a duration lasting 5 minutes and 2 seconds (we will omit to specify `0 yy 0 mm 0 dd 0 hh`, but only for time units coarser than the coarsest time unit having a non-zero value).

The class *instant* allows us to represent a time point, identified by the granule, i.e. a set of contiguous chronons, containing it. This class uses, by the methods *inf*() and *sup*(), two objects of type *el_time*, to represent the lower and upper bound of the granule, in which the generic time point is located. A granule can be expressed by different time units, e.g. by the format `YY/MM/DD` or `YY/MM`, or it may be specified by two time points, i.e. the upper and the lower bound of the granule, by the format ≺`YY/MM/DD/HH/Mi/SS, YY/MM/DD/HH/Mi/SS`≻.

The instant `94/10/10`, for example, may coincide with anyone of the time points included between the two bounds `94/10/10/0/0/0` and `94/10/10/23/59/59`, represented by two objects of *el_time* type: the notation `94/10/10` will be equivalent to ≺`94/10/10/0/0/0, 94/10/10/23/59/59`≻. In modeling indeterminacy, the instant ≺`96/1/1/6/30/0, 96/1/1/6/36/59`≻, for example, specifies a time point between 6:30 and 6:36 of January first, 1996.

The class *duration* allows us to model a generic time span, specified at arbitrary granularity. This class uses, by the methods *inf*() and *sup*(), two objects of type *el_time*, to represent the lower and upper distances between chronons, between which the value of the given duration is included. A duration is expressed by an ordered sequence of elements, composed by an integer followed by a granularity specifier (from years to seconds, `yy`, `mm`, `dd`, `hh`, `min`, `ss`): e.g., `4 yy`, `3 yy 2 mm 3 dd`. It may be expressed also by specifying the lower and upper distances, e.g. ≺`3 dd 4 hh 6 min 3 ss, 4 dd 6 hh 5 min 2 ss`≻. The duration `3 dd`, for example, stands for a time span between `3 dd 0 hh 0 min 0 ss` and `3 dd 23 hh 59 min 59 ss`.

Suitable methods allow the expression of relations and of operations, like sum or differences, on instances of the classes *instant* and *duration* [13].

A generic interval, i.e. a set of contiguous time points, is modeled by the class *interval*. The methods *start*(), *end*() and *dur*() allow us to identify, respectively, the starting instant, the ending instant and the duration of the interval.

Our approach has some similarities with the approaches adopted by Snodgrass in TSQL2, by Bettini and colleagues, and by Clifford and Rao [2, 10, 22]: both of them are based on mappings between instants defined at different granularities. Comparisons between instants given at different granularities are performed by mapping the instants on intervals (or sets of disjoint intervals) at a lower common granularity. Bettini and colleagues and Clifford and Rao focus more on the formal definition of granularity; they consider a general framework allowing the definition of different kinds of granularity. They do not consider problems related to the definition and the management of anchored time points (i.e., instants) and unanchored time spans (i.e., durations) given at different calendar-based granularities. They do not address the problem of merging granularity and indeterminacy in a seamless way. These two last topics are faced in TSQL2 by Snodgrass, providing some solutions with a different approach from that proposed in GCH-OSQL [22]. We discuss the approach adopted in TSQL2 in the section 4.

Suitable methods of the class *interval* allow us to establish temporal relations between two intervals, specified at different and not predefined granularity and/or indeterminacy. Relations between intervals are a superset of the 13 Allen's relations and they can be divided in granularity-related relations and granularity-independent relations [1, 12]. Time modeling in GCH-OODM is described in details in [13].

In GCH-OODM methods of the class *interval* representing temporal relations are defined by the methods of classes *instant* and *duration*, based, in turn, on methods of the class *el_time*.

For example, the relation *a*.BEFORE(*b*) between two objects *a*, *b* instances of the class *interval* is expressed by the methods of the classes *instant* and then *el_time* in the following way:

a.BEFORE(*b*) $=_{df}$ *a.end*().BEFORE(*b.start*()) $=_{df}$

- T iff *a.end*().*sup*() < *b.start*().*inf*()
- F iff *a.end*().*inf*() > *b.start*().*sup*() OR
  (*a.end*().inf() = *b.start*().*inf*() AND
  a.end().*sup*() = *b.start*().*sup*() AND
  a.end().*inf*() = *a.end*().*sup*())
- U otherwise

2.3. Temporal and atemporal classes

GCH-OODM distinguishes *temporal classes* and *atemporal classes*.

Temporal classes

Objects instances of temporal classes (hereinafter temporal objects) have an associated valid interval. By these temporal objects we are able to represent information for which it is important to know the period during which the information is true in the modeled world. The method *valid_interval()* returns the interval of validity of an object. By the valid interval it is possible to verify temporal relations between objects instances of temporal classes. Temporal objects may have many temporal properties; these properties, defined by suitable methods, are represented by temporal objects, which can be composed by set of temporal objects, with an approach having analogies with that in [15].

Atemporal classes

Objects instances of atemporal classes (hereinafter atemporal objects) model information, not having an associated temporal dimension. An atemporal object may, however, have properties represented by temporal objects.

Both in temporal and atemporal classes we distinguish, then, temporal methods, modeling temporal features, returning temporal objects and atemporal methods returning atemporal objects. In fig.1 some examples are given both for temporal and atemporal classes and for temporal and atemporal methods.

```
class person {
public:
char* name();
};

class patient: public person {
public:
t_o_set<visit> Visit_set();
t_o_set<symptom> symptom_set();
};
```

```
temporal class symptom {
interval valid_interval();
char* s_name();
char* severity ();
};

temporal class visit {
interval valid_interval();
int heart_rate();
int temperature();
};
```

Figure 1. The schema of the example clinical database

2.4. Sets of temporal objects: the class *t_o_set*

The predefined class *t_o_set* (temporal_object_set) allows the construction and the management of set of temporal objects. To the instances of the class *t_o_set* it is possible to apply the usual operations on sets: insertion, deletion, intersection, union, difference, existence of an element, emptiness, contained-in relation. Some methods are defined to verify the existence, at a given granularity (if needed), of temporal relations between objects belonging to an instance of the class *t_o_set*, satisfying some conditions on atemporal methods (see Example 3.3 ii) in the following section) [12, 13].

Let us consider the following methods, related to an instance I of the class *t_o_set*. Let p, q be two atemporal formulas, i.e. formulas involving only atemporal methods and comparisons on atemporal objects; let x and y be two temporal objects, X an assigned granularity. To explain the meaning of the following methods, we will use the method *subset*: I.*subset*(p) returns the subset of temporal objects belonging to I and satisfying the formula p.

- I.OCCURS(p) ≡ I.*subset*(p) $\neq \emptyset$

The method OCCURS(p) allows us to establish if in the set I there is an object satisfying the condition p.

- I.CONTEMPORARY(p,q,X) ≡
  $\exists$ x $\in$ I.*subset*(p), $\exists$ y $\in$ I.*subset*(q)
  (x.*valid_time*().CONTEMPORARY(y.*valid_time*(), X))

The method CONTEMPORARY(p,q,X) allows us to establish if in the set I there are two temporal objects, satisfying, respectively, the formulas p and q, and having the valid intervals contemporary at a predefined granularity X. For example, I.CONTEMPORARY ("heart_rate() < 80", "heart_rate() > 120", *mm*) allows us to establish if, in the set I of temporal objects of the class *visit* (see fig.1), there are in the same month two heart rate values respectively less than 80 and greater than 120. In a similar way methods allowing to verify other temporal relations between intervals (before, after, overlap, ..) have been defined [13].

The set of temporal objects managed by an instance of the class *t_o_set* can be viewed, as regards their temporal dimension, as a network of temporal constraints among the starting and ending points of the valid intervals of the temporal objects. These constraints fit into the class of Simple Temporal Problems (STP); in this case complete constraint propagation is tractable [14].

The class *t_o_set* is a temporal class. The valid interval of an object I instance of the class *t_o_set* is evaluated on the basis of all the valid intervals of temporal objects belonging to I:

I.*valid_interval*().*start*().*inf*() ≡
  min(x.*valid_interval*().*start*().*inf*()), $x \in$ I
I.*valid_interval*().*start*().*sup*() ≡
  min(x.*valid_interval*().*start*().*sup*()), $x \in$ I
I.*valid_interval*().*end*().*inf*() ≡
  max(x.*valid_interval*().*end*().*inf*()), $x \in$ I
I.*valid_interval*().*end*().*sup*() ≡
  max(x.*valid_interval*().*end*().*sup*()), $x \in$ I

Granularity of the valid interval of an instance of the class *t_o_set* depends, then, on the granularities of the valid intervals of objects belonging to the modeled set.

We consider two orthogonal ways of specializing the *t_o_set* class: the first consists in specializing temporal objects managed by the class (see next paragraph); the second consists in defining explicitly some constraints on the managed temporal objects, in order to consider, for example, only sets of non overlapping temporal objects and/or with the valid interval specified at a predefined granularity.

2.5. The example database

The example schema represented in fig.1, obviously far from real clinical data complexity, considers both temporal and atemporal classes. A hierarchy of classes is described, using a C++-like syntax. It refers to a clinical database, where data about patients are stored: data are related to the symptoms a patient suffers from and to the parameters (like heart rate) collected during follow up visits. The atemporal class *patient* has two temporal properties, modeled by the methods *visit_set*() and *symptom_set*(), returning instances of the temporal class *t_o_set*. These instances are specialized to manage sets of temporal objects of, respectively, *symptom* and *visit* classes. Methods *s_name()* or *heart_rate()*, for example, are atemporal.

3. The GCH-OSQL query language

The temporal extension to the syntax of SQL concerns the part needed for database queries. We did not define any particular syntax for update, insert and delete operations, to preserve information hiding [7]. The specific programming language of the adopted object oriented DBMS directly manages these operations, by using the ad-hoc defined suitable methods.

The temporal extension includes the addition of the TIME-SLICE and MOVING WINDOW clauses in the original SELECT statement; the temporal dimension of objects may be referred to in the WHERE and SELECT clauses.

A GCH-OSQL query may be expressed this way, where square brackets mean the clause is optional:

```
SELECT <class methods or path
       expressions>
FROM <classes>
[WHERE <temporal and atemporal
       conditions>]
[TIME_SLICE <time interval>]
[MOVING WINDOW <duration>]
```

The query returns data retrieved through methods listed in the SELECT clause, from classes instances in the database listed in the FROM clause, satisfying the conditions imposed through the optional WHERE, TIME-SLICE, MOVING WINDOW clauses.

Retrieved objects are those for which the specified conditions result in TRUE or UNDEFINED logical values. Therefore also objects who might satisfy the specified conditions are included in the result.

According to the object-oriented approach we exposed before, object attributes are referred through methods listed in the clauses, hiding implementation details to users. When a method is specified in a clause, the related code is executed. An object can be reached through a *path expression* (implicit join): it consists in a sequence of methods separated by a "." (see Example 3.1). In showing main features of GCH-OSQL the database schema in fig.1 will be used.

3.1. The SELECT and FROM clause

In the SELECT clause object methods or path expressions can be listed, with a comma between them.

In this clause only methods related to the displaying of the data are allowed: it is not possible to use in this clause updating methods, that have side effects on the state of the database. To display objects having complex features, the method *display()* is usually defined. The time-modeling types, for example, like *interval*, *instant*, and *duration*, have suitably defined methods *display()*, to represent in a proper, abstract manner different time concepts.

The objects in the database containing the data we are interested in are instances of the classes listed in the FROM clause. To each listed class an object variable is associated: it is used to refer to object instances of the related class in the database. IN GCH-OSQL object variables must be present.

Example 3.1. The query "*Find all the patients having nausea and display patient name and the starting instant of each nausea symptom*" will be expressed in the following way:

```
SELECT P.name(),
   S.valid_interval().start().display()
FROM patient P, symptom S
WHERE P.symptom_set().HAS_MEMBER(S)AND
   S.s_name()="nausea"
```

3.2. The WHERE clause

In the WHERE clause the logical conditions which express the constraints that must be satisfied by the objects which will be selected in the database are specified. Complex constraints may be made composing simpler conditions, using the logical connectives AND, OR, NOT, and the connectives MUSTBE, MAYBE, MUST_NOTBE, translating the GCH-OODM operators T(), U(), F().

Conditions involving temporal relations are expressed in the WHERE clause through *t_o_set* class methods, and using *interval* class methods.

Example 3.2. The query "*Find all the symptoms occurring during visits; display the name and the interval of validity of the symptom, and also the patient suffering from it*" will be expressed in the following way:

```
SELECT P.name(), S.s_name(),
   S.valid_interval().display()
FROM patient P, symptom S, visit V
WHERE P.symptom_set().HAS_MEMBER(S)
   AND P.visit_set().HAS_MEMBER(V) AND
   S.valid_interval().DURING(
   V.valid_interval())
```

Example 3.3. "*Find the patients having had nausea and headache, with headache surely before nausea*": this query can be expressed in the following two ways:

```
i) SELECT P.name()
   FROM patient P, symptom S1,
      symptom S2
   WHERE
      P.symptom_set().HAS_MEMBER(S1)
      AND P.visit_set().HAS_MEMBER(S2)
      AND S1.s_name() = "nausea" AND
      S2.s_name() = "headache" AND
      MUSTBE(S2.valid_interval().BEFORE
      (S1.valid_interval())
ii) SELECT P.name()
   FROM patient P,
   WHERE MUSTBE
      P.symptom_set().BEFORE("s_name()
      = "headache"", "s_name() =
      "nausea"")
```

Alike other proposals, in GCH-OSQL we have not added any other clause (as WHEN or WHILE [8, 20]), that would allow to separately express the temporal part of the query. This choice allows one to express the query constraints in a seamless way, without forcing the user to divide the select condition. This choice, moreover, avoid some anomalies, as, for example, expressing some temporal constraints in the WHEN clause and some others in the WHERE clause [20].

3.3. The TIME-SLICE clause

This clause allows the user to query along the temporal dimension of objects, considering only those objects in the database whose valid time is contained in the interval specified in the clause. Also objects whose valid time could be contained in the specified interval are selected. In this clause is possible to use the MUST and MAY keywords. Using the MUST keyword, only those objects whose valid time is certainly contained in the interval specified in the clause are selected, while using the MAY keyword only those objects for which it is uncertain that their valid time is contained in the specified interval are selected.

In the TIME-SLICE clause the time interval may be expressed in many different ways:

- by the FROM..TO keywords, in order to define an interval by its starting and ending instants: e.g., `FROM 1994/12/11 TO 1994/12/23/11/00`. It is also possible to specify only the FROM or the TO keywords.
- by the FROM..FOR keywords, in order to define an interval by its starting instant and its duration: e.g., `FROM 1994/12/11 FOR 2 mm`.
- by the FOR..TO keywords, to define an interval by its duration and its ending instant: e.g., `FOR 3 dd TO 1995/4`.
- by the AT keyword, to define an interval as a single granule: e.g., `AT 1996/5`.

Example 3.4. The query "*Find all symptoms happened starting from December 1991 to November 12th, 1996 in the afternoon*" will be expressed in the following way:

```
SELECT S.s_name()
FROM symptom S
TIME-SLICE FROM 1991/12 TO
   ≺1996/11/12/12/0/0,
   1996/11/12/17/0/0≻
```

We underline that the capability of defining the TIME-SLICE interval by, respectively, the FROM..TO, FROM..FOR, and FOR..TO keywords is not redundant. Let us consider for example the two clauses (i) `TIME-SLICE FROM 1994/6/23 TO 1994/6/25` and (ii) `TIME-SLICE FROM 1994/6/23 FOR 48 hh 0 min 0 ss`: both these clauses identify an interval having the same starting and ending instants. However, the clause (i) will consider, on the chronon time axis, an interval having a duration between 24 hours plus 1 second and 72 hours less 1 second: i.e., this time span is evaluated, considering the indeterminacy related to the granularity of both the starting and the ending instants; the clause (ii) will consider, on the chronon time axis, an interval having a duration of exactly 48 hours.

3.4. The MOVING WINDOW clause

Using this clause, the objects stored in the database are examined through a temporal window, of the width specified in the clause, moving along the temporal axis. The constraints expressed in the other clauses are checked only on the database part visible through that window.

Example 3.5. "*Print the name of patients having had heart rate greater than 120 and symptoms of chest pain in a period of fifteen days*"; this query is expressed as:

```
SELECT P.name()
FROM patient P
WHERE
   P.visit_set().OCCURS("heart_rate()>
   120") AND
   P.symptom_set().OCCURS("s_name()=
   "chest pain")
MOVING WINDOW 15 dd
```

In the MOVING WINDOW clause through the MUST or MAY keywords, only, respectively, certain or uncertain situations can be considered.

4. Related work

To evidence the peculiar aspects of GCH-OSQL, we consider query languages based on a object oriented data model: TOOSQL, OODAPLEX, OOTempSQL, OQL/T, and TQL [8, 15, 20, 23, 26]. A further comparison is then made with the TSQL2 language [22].

Time modeling. In the considered languages, apart from OOTempSQL, the temporal dimension, related to data stored in the database, is represented by an *interval*. The interval is specified by a *starting instant* and a *final instant*; instantaneous events are represented by making equals the starting and final instants. In GCH-OSQL the interval is specified by a *starting instant*, a *final instant*, and a *duration*. This lets the wider flexibility in pointing out beginning, ending, and duration of an interval, at the necessary granularity. Typically GCH-OSQL allows the definition of an interval duration at a finer granularity (seconds, for example), while expressing beginning and ending instants at a coarser granularity (as days). This capability, absent in the other considered languages, suits well the clinical data management [12]. In OOTempSQL *temporal elements* are introduced; using a temporal element more disjoint intervals, identified by a begin and an end, can be referenced [8]. This concept allows one to enrich the expression of the temporal dimension, for the part, as to say, not linked to the use of different time measurement units.

In GCH-OSQL the time representation is made through a predefined type hierarchy, that gives the user the chance to represent new temporal dimensions. The choice to represent temporal data through predefined types is adopted also by some query languages, as OQL/T, TQL, TOOSQL, OODAPLEX [15, 20, 23, 26]. In OODAPLEX time is simply one of the language types; the database designer has the chance to extend the type definition to model the temporal dimension as preferred [26].

In TSQL2 some data types are introduced to model concepts similar to those defined in GCH-OSQL: data types *datetime*, *interval,* and *period* of TSQL2 correspond, respectively, to the types *instant*, *duration* and *interval* of GCH-OSQL [22].

Temporal dimensions. The temporal dimension in GCH-OSQL represents the valid time, as in OQL/T [23]. In TQL, TOOSQL and in TSQL2 the temporal dimension can be used to express both valid time and transaction time [18, 20, 22]. OODAPLEX, through the use of functions, can model both valid and transaction times, and other temporal dimensions defined by the user [26].

Temporal entities. In GCH-OSQL the temporal dimension is associated to objects. The same happens in OQL/T and TQL [23, 15], where, every entity being an object in each data model, the temporal dimension is associated to objects. In TOOSQL and in OOTempSQL, the temporal dimension is associated to attributes [8, 20]: in OOTempSQL, for example, temporal attributes are represented through functions, that for each instant belonging to a temporal element return the attribute value [8]. In OODAPLEX it is possible to connect the temporal dimension both to attributes and objects [26]. GCH-OSQL is as flexible as OODAPLEX in linking temporal dimension to different entities, because, especially through instances of the predefined temporal type *t_o_set*, it is possible to model the time dependent properties of an object through sets of temporal objects. GCH-OOSQL and GCH-OODM do not consider the problem of the temporal evolution of objects nor the role changes of objects [6, 26].

Time granularity and time indeterminacy. Using GCH-OSQL the user can express in a seamless way temporal data and temporal conditions at the granularity and/or indeterminacy he considers the best suited. This lets the user express data in the more suited way, depending on the precision requested to represent temporal dimension, or depending on the knowledge degree of that. In most of the examined languages the granularity is fixed, and temporal data are always expressed with the same temporal unit of measurement, and the problem of the temporal indeterminacy in locating events on the temporal axis is not considered. Other examined languages, as OOTempSQL, TOOSQL, TQL, OQL/T, allow, and nothing more, the use of different time measurement units, through conversion operations from one unit to others [8, 15, 20, 23]. This approach to management of different granularities of time seems more limited than the one proposed in GCH-OSQL; often, in fact, only in querying a database it is possible to use different time measurement units, while in inserting data the granularity has been fixed at the schema level and cannot be modified [20].

Moreover compared to this characteristic, GCH-OSQL allows the management of uncertainty coming from temporal relations between data defined at different granularities and/or indeterminacies, using a three-valued logic [13] and special operators (MUSTBE, MAYBE, MUST_NOTBE). This aspect is not considered in the examined temporal extensions, all adopting the usual two-valued logic, but TSQL2.

In TSQL2 problems related to different time granularities and to time indeterminacy are faced [22]. TSQL2 allows one to define and manage different calendars, each of them composed by several time granularities: this topic has not been faced in GCH-OSQL. In representing intervals with different time granularities, some constraints are introduced in TSQL2 on the indeterminacy of the starting and ending instants. These constraints restrict the capability of representing intervals at different and mixed granularities: it is not possible, for example, to define in TSQL2 the interval "in 1996 for three days". This limitation has been removed in GCH-OSQL.

In managing uncertainty for temporal relations, TSQL2 adopts a probabilistic approach: for each time point the probability is given that the indeterminate instant is located at that time point [22]. This approach leads to extend the syntax and the semantic of TSQL2 with the concepts of credibility and plausibility. These concepts allow one to pragmatically manage truth-functional conditions in TSQL2 SELECT clause [22].

Some works dealing with uncertainty in temporal relationships are based on the adoption of modal operators. The approach of Pernici and colleagues in [5, 17] has some differences from what performed by GCH-OSQL in the meaning of the MUST and MAY operators: more particularly, in LATER the result of a MUST query satisfies also the corresponding MAY query. GCH-OSQL distinguishes MUSTBE queries, in which we are looking for sure situations, from MAYBE queries, in which we are looking only for uncertain situations. The same meaning of a MAY query in LATER is performed in GCH-OSQL simply without specifying neither the operator MUSTBE nor MAYBE.

Implementing the system. With respect to several works proposed in literature, GCH-OSQL is characterized, finally, by the presence of a prototypal implementation, applied to a management system of clinical histories [4, 11, 25]. A prototype of GCH-OSQL and a graphical interface have been implemented on a Sun workstation, in the OpenLook graphical environment; the prototype is based on the ONTOS object-oriented database management system. ONTOS has been used in the implementation of the clinical database too, to which GCH-OSQL has been applied, related to follow-up of patients after a coronary-artery angioplasty intervention [11]. One of the goals the of the GCH-OSQL interface was to give users a simple and guided approach, so that also an unexperienced user can make syntactic and semantic right queries [11].

5. Final outlines

GCH-OSQL and GCH-OODM allow the **management of temporal granularity and of temporal indeterminacy**. The user can, **while inserting data**, specify the temporal dimension at the level he considers the best suited, also depending on his knowledge or on the importance the modeled fact has. Moreover, GCH-OSQL allows one to manage granularity/indeterminacy **in the queries:** in querying the database it is possible to verify the existence of temporal relations at the required granularity, that can be different from the one used in expressing the temporal dimension of stored data.

GCH-OSQL provide the **management of uncertainty in temporal relations by a three-valued logic**. Through the use of the unary operators MUSTBE, MAYBE, MUST_NOTBE, it is possible to explicitly consider each of the logical values *True*, *Undefined*, *False*.

Finally, GCH-OSQL performs a **homogeneous management of temporal conditions in the query**. Compared to SQL, GCH-OSQL has few additional clauses: TIME-SLICE and MOVING WINDOW. Instead of adding other clauses, the temporal conditions are expressed in the WHERE clause. In fact, in GCH-OSQL it is possible to use some suitable methods of the types *instant*, *duration*, *interval*, and *t_o_set*, to verify temporal features among the objects stored in the database.

Acknowledgments

This work was partially supported by contributions from: MURST Italian National Project for Medical Informatics; Department of Biomedical Engineering of the Politecnico di Milano; Department of Mathematics and Computer Science of the University of Udine.

References

[1] Allen J. Towards a General Theory of Action and Time. Artificial Intelligence 1984; 23: 123-154.

[2] Bettini C, Wang XS, Jajodia S. A General Framework and Reasoning Models for Time Granularity. In: [9]: 104 - 111.

[3] Blakeley JA. OQL[C++]: Extending C++ with an Object Query Capability. In [16]: 69 - 88.

[4] Böhlen MH. Temporal Database System Implementations. SIGMOD Record 1995; 24(4): 53-60.

[5] Brusoni V, Console L, Pernici B, Terenziani P. LATER: a General Purpose Manager of Temporal Information, 8th International Symposium on Methodologies for Intelligent Systems, Charlotte, North Carolina, October 1994.

[6] Cardenas AF, Ieong IT, Taira RK, Barcher R, Breant CM. The Knowledge - Based Object-Oriented PIQUERY+ Language. IEEE Transactions on Knowledge and data Engineering 1993; 5(4): 644 - 657.

[7] Cattell RGG. (ed.) The Object Database Standard: ODMG - 93 Release 1.2. San Francisco. Morgan Kaufmann, 1996.

[8] Cheng TS, Gadia SK. An object-oriented model for temporal databases. In Proceedings of the International Workshop on an Infrastructure for Temporal Databases, Arlington, TX, N-1 - N-19, 1993.

[9] Chittaro L, Goodwin S, Hamilton H, Montanari A (eds) Third International Workshop on Temporal Representation and Reasoning (TIME'96). Los Alamitos CA, IEEE Computer Society Press 1996.

[10] Clifford J, Rao A. A Simple, General Structure for Temporal Domains, In: C. Rolland, F. Bodart and M. Leonard (eds) Temporal Aspects in Information Systems. Amsterdam, Elsevier, North-Holland, 1988: 17-28.

[11] Combi C, Pinciroli F, Cavallaro M, Cucchi G. Querying Temporal Clinical Databases with Different Time Granularities: the GCH-OSQL Language. In: GARDNER RM (ed.), 19. Annual Symposium on Computer Applications in Medical Care. Philadelphia, Hanley & Belfus, 1995, 326 - 330.

[12] Combi C, Pozzi G, Pinciroli F. Managing Different Time Granularities of Clinical Information by an Interval-Based Temporal Data Model. Methods of Information in Medicine 1995; 34 (5): 458 - 474

[13] Combi C, Pinciroli F, Pozzi G. Managing Time Granularity of Narrative Clinical Information: the Temporal Data Model TIME-NESIS. In [9]: 88 - 93.

[14] Dechter R, Meiri I, Pearl J. Temporal Constraint Networks, Artificial Intelligence 1991; 49: 61-95.

[15] Goralwalla IA, Özsu MT. Temporal Extensions to a Uniform Behavioral Object Model. In Elmasri R, Kouramjian V, Thalheim B. (eds) Proceedings of the International Conference on the E-R approach, LNCS, Berlin, Springer-Verlag, 1993: 110 - 121.

[16] Kim W. Modern Database Systems. New York, Addison Wesley, 1995.

[17] Maiocchi R, Pernici B, Barbic F. Automatic Deduction of Temporal Information, ACM Transactions on Database Systems 1992; 17(4): 647 - 688.

[18] Öszu MT, Peters R, Szafron D, Irani B, Lipka A, Muñoz A. TIGUKAT: A Uniform Behavioral Objectbase Management System. VLDB Journal 1995; 4: 445 - 492.

[19] Panti G. Multi-valued Logics. In Gabbay D, Smets P. (eds.), Handbook of Defensible Reasoning and Uncertainty Management Systems. Kluwer, Berlin, 1996, in press.

[20] Rose E, Segev A. TOOSQL: A Temporal Object-Oriented Query Language. LBL 33855 Technical Report, 1993.

[21] Snodgrass R. Temporal Object-Oriented Databases: A Critical Comparison. In [16]: 386-408.

[22] Snodgrass RT. (ed.) The TSQL2 Temporal Query Language. Boston, Kluwer Academic Publishers, 1995.

[23] Su SYW, Chen HHM. Modeling and Management of Temporal Data in Object-Oriented Knowledge Bases. In Proceedings of the International Workshop on an Infrastructure for Temporal Databases, Arlington, TX, HH-1 - HH-17, 1993.

[24] Tansel AU, Clifford J, Gadia S, Jajodia S, Segev A, Snodgrass R.(eds.) Temporal Databases. Theory, Design and Implementation. Redwood City, CA, Benjamin-Cummings, 1993.

[25] Vassilakis C, Georgiadis P, Sotiropoulou A. A Comparative Study of Temporal DBMS Architectures. In Wagner RR, Thoma H. (eds) Proc. 7th Int. Workshop on Database and Expert Systems Application, IEEE Computer Press, Los Alamitos, CA, 1996, 153 - 164.

[26] Wuu GTJ, Dayal U. A Uniform Model for Temporal and Versioned Object-oriented Databases. In [24]: 230-247.