HEWLETT
PACKARD

Temporal Extensions to the Iris Model in a Medical Domain

Roberto Ferrari*
Software and Systems Laboratory
HPL-91-132
September, 1991

time, temporal representation, query, time granularity, object-oriented databases

Time is one of the most important aspects of real life. Software systems for the management of data in a wide variety of domains (e.g., medicine, economics) must include information about evolution of entities of interest, in order to perform properly. Database systems must effectively and efficiently manage information about time as well as the time-variations of entities of the domains they model. The objective of this work is to extend the functionality of an object-oriented database system, Iris, with temporal representation and reasoning capabilities.

* SEED student from the *Universita' degli Studi di Pavia, Italia*

1 Introduction

Real world entities and environments, modeled inside computerized data systems, evolve over time. Time is a main characteristic of our life. In order to represent the state and the evolution of modeled entities correctly and completely, temporal information needs to be represented and managed.

In the past, automated data management systems evolved disregarding the temporal characteristic of data as a basic feature. The first research studies regarding the representation of time in database management systems appeared in the 70s with the first attempts to embody temporal information inside the conceptual models. A fundamental chapter is represented by the Time Oriented Databank [G.Wiederhold et al. - 1975], the first historical database in which the values that represent the state of an entity in a particular time instant are associated with the instant itself.

Ten years later, Snodgrass and Ahn [Snodgrass R. and Ahn I. - 1986] drew a taxonomy of four types of databases. The "snapshot" databases are the simplest ones, available in most conventional databases. They don't treat any representation of time at all. They model an environment as it changes dynamically by a snapshot at a particular point in time, so the state or instance of a snapshot database is its current content, which does not reflect necessarily the current real status of the environment. There is no capability to record and process time-varying aspects of the real world. The second type is represented by "rollback" databases, based on "transaction" time. It is the time in which the information is stored or updated in the database, in other words, the time in which a change in the state of the database becomes persistent. Again, information on the "valid" or "real" time of data is ignored. A transaction reflects the real change, obviously, if it is entered in the database as soon as the change in the real world happens. A rollback database may be regarded as a sequence, indexed by transaction time, of snapshot states. The third type is represented by "historical" databases. They support valid time but not transaction time. Finally, "temporal" databases support both valid and transaction time (orthogonal time axes).

In the last ten years, studies and projects concerning representation, management, and use of temporal information have been undertaken in different research sectors, in particular in computer science, logic, mathematics, economics. Methodologies and semantics coming from the logic and artificial intelligence worlds [Fagan L.M. - 1980, Shortliffe E.H. et al. - 1981, McDermott D. - 1982, Allen J.F. - 1983, Allen J.F. - 1984 have been used and adapted to solve problems and to augment the representational capabilities of temporal data models. A complete overview about the evolution of such approaches is given in a recent study of Snodgrass [Snodgrass R. - 1990].

New data storage technologies, such as optical disks, efficiently and cheaply store huge amounts of information. They support research and development of new, more complex temporal data models and the associated query languages. In particular, after the first models using temporal primitives "time point" [McDermott D. - 1982] or "time interval" [Allen J.F. - 1983], more recent theories followed, using both primitives inside the same model. Based on these primitives, complex semantics and syntaxes have been developed for the definition of new constructs for query languages (e.g. the temporal operators before, after, etc.) [Navathe S.B. and Ahmed R. - 1987, Snodgrass R. - 1987, Chaudhuri S. - 1988, Gadia S. - 1988, Elmasri R., Wuu G.T.J. - 1990, Sarda N. - 1990].

Most of the current work is concerned with extensions of the relational model and SQL, and with historical more than temporal databases. This limitation has been widely accepted because, as on-line transaction processes technologies improve, valid and transaction time will coincide.

Most recent studies deal with big issues like the time granularity problem [Wiederhold G. et al. - 1990, Kamens S.N. and Wiederhold G. - 1990] or attempt to couple temporal representation models to complex data models (e.g. the object-oriented data model), richer in semantic than the relational one [Kahn M.G. - 1988, Cousins S.B. et al. - 1989, Kahn M.G. et al. - 1991].

The work described here focuses on use of object-oriented databases as historical databases (valid time) and defines some strategies to embed directly inside the object-oriented model, temporal representation and reasoning methodologies. At the same time, however, theory and work developed in the relational domain have been carefully considered.

2 The tools

Iris, the HP object-oriented DBMS [Fishman D.H. et al. - 1989, Wilkinson K. et al. - 1990], and the Iris Programming Language (IPL) [Annevelink J. - 1990], an extension of OSQL, the query language of Iris, have been used as the platform to model and to implement the temporal constructs and methods. IPL permits to implement complex functions and to access LISP - C library functions.

Concepts and ideas are illustrated with examples from a medical application, monitoring patients with AIDS Related Complex (ARC).

3 The temporal object model

In object oriented databases, it is possible to model the environment of interest by means of types (classes) that denote the complex templates representing the conceptual entities of the particular domain. Types are related to each other in a hierarchical taxonomy ("type lattice") that permits the specification of property and method ("functions" in Iris model) inheritance. Objects are instances of types; they represent the actual members of the abstract types. Their behavior is specified by the functions defined for the types they per- tain. The ARC taxonomy is shown in the figure 1 below.

ARC Taxonomy

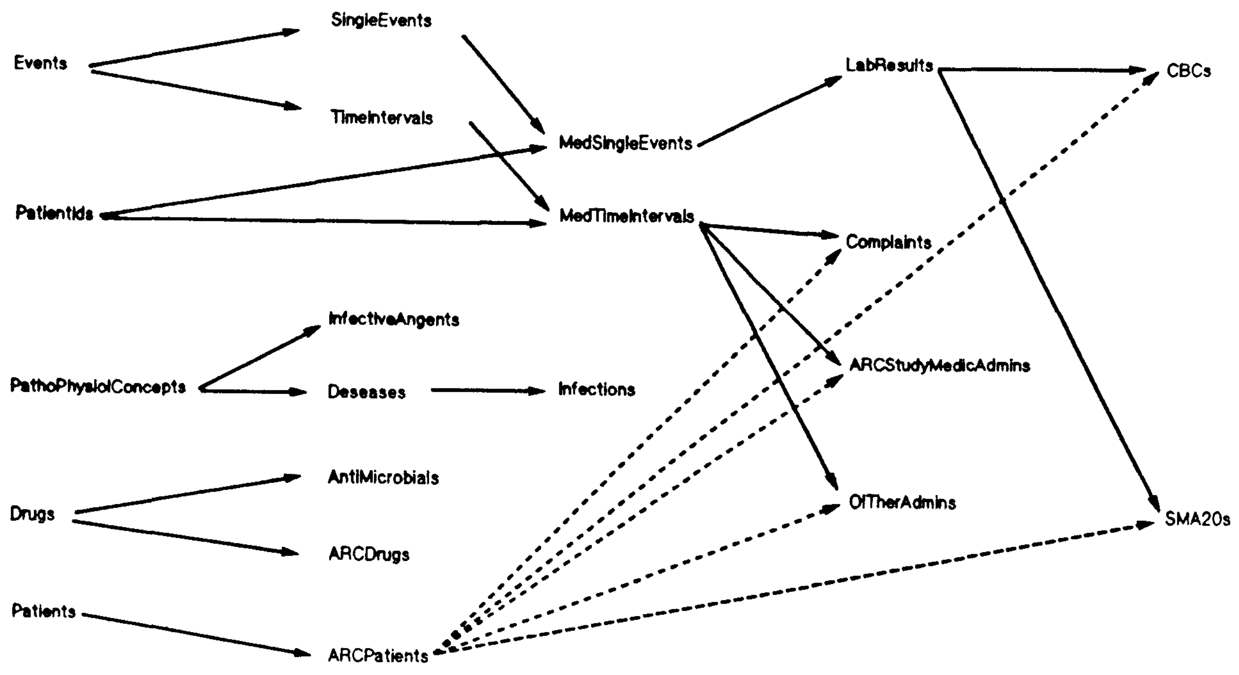

Figure 1. Taxonomy of classes defining the AIDS Related Complex data schema.

Full arrows represent the logical relationship "IS-A" ("specialization") between super and sub-types. So, an object of the type ARCPatient will inherit all the properties specified from the supertype Patient as well as ones specifically defined for the type ARCPatient. Dotted arrows represent the "HAS-A" relationship ("aggregation") between objects that are instances of different types: an object of the type ARCPatients may refer to one or more objects of the type Complaints. This means that an attribute defined, in schema definition phase, for the type ARCPAtients is used to relate an object of type ARCPatient with one or more instances of type Complaints.

Objects model complex entities and temporal objects model complex entities with attributes whose values may vary over time. Some features of a complex, time-varying entity may be steady, while other time-varying ones may vary with different rates. For example, a "patient" object has some properties (e.g. name, sex, birthday) that do not vary with time, and others (e.g. blood pressure, temperature, lab tests results, diseases, therapies, etc.) that do. The time-varying properties of an entity are the ones that let us consider the entity as a temporal entity. In the example, because of the time-varying features, the entity patient has to be considered a temporal entity.

Object-oriented database systems allow one to model each entity's attribute, depending on the nature of the attribute as defined by the schema, as a simple property (e.g. patient's temperature) or a complex one (e.g. therapy planning). In particular, the Iris model supports single properties of a template to be "stored" (values for the property are stored in a table), "derived" (the property is implemented as an SQL statement), or "foreign" functions (the property is implemented through a programming language, e.g. IPL) whose result is of "literal" type (string, integer, real, etc.). Complex properties are modeled similar to simple properties, but the result may be of a complex, user- defined type. Figure 2 shows the properties inherited or defined for the type ARCPatients. Examples are shown of stored functions, e.g. "nameOfPatient", "DateOfBirth", "Sex". Creating or updating an object of type ARCPatients, the operator has to enter the proper values for each of the stored properties. An example of derived function is "ssnoOfPatient" (social security number), that can be retrieved from the value entered for "MedicalRecord". One example of foreign function is "Age", computed as the number of granules of granule type "year" contained in the time interval between the actual date and the date stored in DateOfBirth. OpportunisticInfections is a complex property whose values are identifiers of objects of user type Complaints.

ARCPatients Properties

Name	Type	Inherited from
nameOfPatient	Charstring	Patients
addressOfPatient	Charstring	Patients
MedicalRecord	Integer	Patients
DateOfBirth	Integer	Patients
Sex	Charstring	Patients
City	Charstring	Patients
County	Charstring	Patients
ZipCode	Integer	Patients
InsurConstraints	Charstring	Patients
Age	Integer	Patients
ssnoOfPatient	Integer	Patients
ARCTherAdmin	ARCStudyMedicAdmins	
OITherAdmin	OITherAdmins	
CBC	CBCs	
SMA20	SMA20s	
OpportunisticInfections	Complaints	

Figure 2. Property list for type ARCPatients: first and second column report names and types of the properties, the third one the eventual types from which the properties are inherited.

The main topic of the time representation modeling project is described in the following.

As shown in figure 1, the type Events and two subtypes, SingleEvents and TimeIntervals, are defined. Type Events has two properties, "startDate" and "stopDate", that represent the initial and end dates of time events and that are inherited by the subtypes, SingleEvents and TimeInterval, and subsequent subtypes. All the time-varying properties of complex temporal objects of the schema (e.g. ARCPatients' instances) may be modeled as subtypes of SingleEvents or of TimeIntervals, depending on their temporal behavior. Subtypes of SingleEvents model events whose validity is certain only for a time instant (e.g. hematological test on a blood sample of a particular date), whether subtypes of TimeIntervals model events continuously valid during a time interval of definition (e.g. a therapy planning for a patient in a particular period).

The time-stamps startDate and stopDate are implemented as stored functions of literal type Integer. However, the users will use, at creation or update time of temporal objects, the string format "DD_MM_YYYY_hhmm" for dates, with digits corresponding to day, month, year, hours and minutes in place of letters. Properly defined functions have been defined to perform the conversions from the string format into the internal integer representation. The integer representation was chosen for two reasons. The first is that it is much simpler to perform temporal operations (comparisons, computations, etc.) on integers than on each other format. The second reason regards time granularity and will be discussed further.

Most of the temporal templates connected by a "HAS-A" relationship to other prototypical entities are meaningful by themselves and not only because of their "HAS-A" dependency relationship. In fact, for example, the template CBCs may be regarded by itself as the template that will collect all the hematological tests done during a period of monitoring on the patients.

SingleEvents have both time stamps, despite their characteristic to be one instant lasting, in order to let all the temporal operators, computation and retrieval functions work as well on them as on actual TimeIntervals. Proper input functions may be implemented to associate the same date value, that the users enters at creation or update time of an object of some SingleEvents subtype, to both time-stamps.

Figure 3 reports the properties inherited by and defined for the type CBCs.

CBCs Properties

Name	Type	Inherited from
startDate	Integer	Events
stopDate	Integer	Events
PatientId	Patients	PatientIds
WBC	Real	
RBC	Real	
HGB	Real	
HCT	Real	
PLCT	Integer	
RETC	Real	

Figure 3. Property list for type CBCs: it's a time point type (it inherits the time stamps properties and the functions to manage them from the SingleEvents type).

4 Temporal operators and functions

Temporal operators [Allen J.F. - 1983, Wiederhold G. et al. - 1990], properly defined to cover all the possibilities in order to compare time points with intervals or intervals with intervals, have been implemented with LISP - C library functions directly inside the Iris kernel and model.

They operate on the integer time stamps of two different events ev1 and ev2.

The list of such operators, with the description of each one's logical behavior, follows:

before_(startDate(ev1), stopDate(ev1), startDate(ev2), stopDate(ev2))
the result is 1 if the event e1 terminates before e2 starts

after_(startDate(ev1), stopDate(ev1), startDate(ev2), stopDate(ev2))
the result is 1 if e1 starts after e2 is terminated (symmetrical of before_)

until_(startDate(ev1), stopDate(ev1), startDate(ev2), stopDate(ev2))
the result is 1 if e1 terminates the same time e2 starts

from_(startDate(ev1), stopDate(ev1), startDate(ev2), stopDate(ev2))
the result is 1 if e1 starts the same time e2 ends (symmetrical of until_)

leads_(startDate(ev1), stopDate(ev1), startDate(ev2), stopDate(ev2))
the result is 1 if e1 starts before e2 and e2 starts before e1 but ends before e1 ends, e.g. e1 overlaps e2 but their extremes never coincide

lags_(startDate(ev1), stopDate(ev1), startDate(ev2), stopDate(ev2))
the result is 1 if e1 starts after e2 starts but before e2 ends and e1 ends after e2 ends, e.g. e2 overlaps e1 but their extremes never coincide (symmetrical of leads_)

starts_(startDate(ev1), stopDate(ev1), startDate(ev2), stopDate(ev2))
the result is 1 if e1 and e2 start the same time but e1 ends before e2

finishes_(startDate(ev1), stopDate(ev1), startDate(ev2), stopDate(ev2))
the result is 1 if e1 and e2 end the same time but e1 starts after e2 is started

equals_(startDate(ev1), stopDate(ev1), startDate(ev2), stopDate(ev2))
the result is 1 if e1 and e2 overlap exactly

during_(startDate(ev1), stopDate(ev1), startDate(ev2), stopDate(ev2))
the result is 1 if e1 starts after e2 started and ends before e2 ends

spans_(startDate(ev1), stopDate(ev1), startDate(ev2), stopDate(ev2))
the result is 1 if e1 starts before e2 starts and ends after e2 ends (symmetrical of during_)

A graphical representation of the temporal relationships between interval events, in order to satisfy the temporal operators, is shown in the following figure 4.

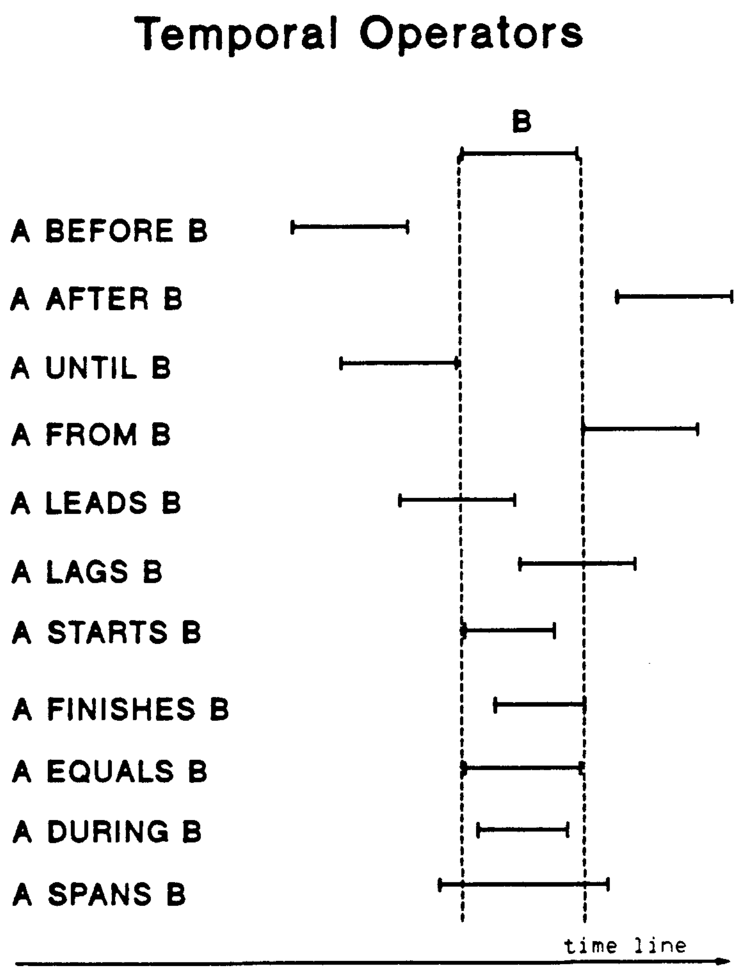

Figure 4. Graphical representation of the temporal relationships between interval events in order to satisfy the temporal operators.

Using these operators it is possible to determine temporal relationships between two time points, a time point and an interval or two intervals. For time points, the startDate and stopDate coincide. In the graphical point of view of figure 4, when at least one of the two events compared by each operator is a time point, the corresponding interval collapses to a point, but all the logical dependencies above remain valid. The operators can be used directly in IPL foreign functions or in OSQL queries. A complete set of such Iris functions that permit use of LISP - C library operators directly inside Iris functions or OSQL queries and allow refererence to events in place of their time stamps has been defined. For example, the following function is the Iris OSQL interface for the function before_:

```
create function Before_(Events ev1, Events ev2) -> Integer as ipl
return before_(startDate(ev1), stopDate(ev1),startDate(ev2),stopDate(ev2));
```

Iris functional interfaces to operators described above can be used directly to define Iris retrieval functions. As an example, the following function retrieves all the events of a certain type before a date:

```
create function SelAllEvsBeforeDate(Charstring EvType, Charstring d)
    -> Events as
select e
for each Events e, UserType t
where NameOfType(t) = EvType and
      RefsAsType(e,t) = 0 and
      Before_(e,DateToEvent(d)) = 1;
```

The arguments of this function are, respectively, the name of the event type of interest (e.g CBCs, Complaints) and the date in the string format "DD_MM_YYYY_hhmm". The functions NameOfType and RefsAsType are system function, used together to retrieve instances of type Events and, at the same time, of the user defined type. The operator Before_ will retrieve, among these objects, only those satisfying the logical temporal implication. It has to be pointed out that if the user defined type is a subtype of SingleEvents (e.g. CBCs) the operator Before_ will compare time points on the time line and will return all the instant events preceding the user date. In case the user type is a subtype of TimeIntervals (e.g. Complaints), the operator will retrieve all the interval events completed before the user date.

The previous function can be used, for example, in the body of another function that refers directly to patients:

```
create function SelAllEvsBeforeDateForPat(Charstring EvType,
        Charstring Date, Charstring PatientName) -> Events as
select e
for each Events e, ARCPatients p
where PatientId(e) = p and
        nameOfPatient(p) = PatientName and
        e = SelAllEvsBeforeDate(EvType,Date);
```

In order to retrieve the events of type CBCs for the patient "HIV Albert" before the date Jan 1st 1991 ("15_09_1991_0000") the user has to call the function with the arguments as follows:

```
call SelAllEvsBeforeDateForPat("CBCs", "HIV Albert", "15_09_1991_0000");
```

An example of complex ipl function follows. Among the events of a certain temporal type, terminated before a date and selected by the SelAllEvsBeforeDate function, this function retrieves the event that just precedes a particular date, entered by the user:

```
create function SelFirstEvBeforeDate(Charstring EvType, Charstring d)
     -> Events as ipl
begin
     var EvObj, firstObj, comp;
     EvObj := SelAllEvsBeforeDate(EvType,d);
     if (EvObj != {}) then begin
        firstObj := head(EvObj);
        while ( (EvObj := tail(EvObj)) != {}) begin
           comp := head(EvObj);
           if ( (Finishes_(comp,firstObj) == 1) or
                (Lags_(comp,firstObj) == 1) or
                (From_(comp,firstObj) == 1) or
                 (After_(comp,firstObj) == 1)  ) then
              firtsObj := comp;
        end;
        return firstObj;
     end
     else error("No events of type ", EvType);
end;
```

The EvObj variable will collect in a first time the set of events that precede the user date. Then, among them, the procedure loops searching for the one that is at the shortest distance in time from the particular date. The head and tail functions in the body of the procedure above are Xscheme functions that operates on lists and bags of items (objects or literal values) and return, respectively the first element, or the set of elements following the first one. It has to be observed that the use of the four operators Finishes_ Lags_, From_ After_ is mandatory in order to perform correctly even in case of interval events that may overlap.

As it can be seen from the examples, it has been attempted to maintain the Iris "functional" style in the operators, computations, queries definition and, as a brief consideration, it has been observed that this kind of functional approach is powerful and quite easy to work with.

5 Time granularity

Normally, there is an appropriate time granularity for each different kind of event. For example, the beginning of an infectious disease is sufficiently well described, with approximations due to uncertainty, in terms of days, while the therapy planning often needs a description in terms of days, hours and minutes. In this work, it has been decided that, at the level of physical storage, time information is represented in the uniform format of "DD_MM_YYYY_hhmm" translated in an integer representing the number of minutes elapsed from the date "Jan 1st 1900 00:00" 'til the date of interest. This means that the system will store only integers referring to dates in the format of finest granularity (minutes). However, the user can enter the information about the time stamps in the format she prefers. System functions have been provided to convert time information entered with different granularity into dates in the complete format above, adding defaults for lacking time granules. So, the functions that operate and compute on integer time stamps are independent from the particular granularity format the operator decides to use. Xscheme function that compute the number of granules of different granule types (years, months, etc.) contained in time intervals have been implemented:

granulesno(secondExtreme, firstExtreme, granType)

where the arguments secondExtreme and firstExtreme are integers representing the end and start dates, respectively, of the time interval of interest and granType is an integer that codes the types granules in the following fashion:

1 = year

2 = month

3 = day

4 = hour

5 = minute

The iris OSQL interface function for granulesno is:

```
create function GranulesNo(Integer d1, Integer d2, Integer typeGran)
      -> Integer as ipl
return granulesno(d1, d2, typeGran);
```

As an example, the following function uses the Xscheme function granulesno to retrieve the number of granules of a certain type contained in the time interval defined by two SingleEvents:

```
create function GranulesBetweenSingleEvs( SingleEvents ev2,
         SingleEvents ev1, typeGran) -> Integer as ipl
begin
      if (Before_(ev1,ev2) == 1) then begin
         return granulesno(startDate(ev1), startDate(ev2), typeGran);
      end
      else begin
         if (Equals_(ev1,ev2) == 1) then
            return 0
         else
            error("Events not in the right sequence", NULL);
      end;
end;
```

6 Examples of queries in the ARC domain

In the following we give a number of simple examples taken from a scenario that deals with the treatment of opportunistic infections in ARC patients. During routine treatment (study medication) of a patient affected by HIV, an acute, intercurrent infection (Opporunistic Infection OI) may arise. In this case, the therapist has to stop the study medication and start a proper, intensive therapy for the OI. This kind of therapy may last no more than 21 days, in order to avoid serious problems of toxicity in the patient.

In order to perform properly the OI therapy, each day the therapist has to consult the archive to see which patients are under OI treatment and he has to suspend the treatment for those patients whose treatment period is longer that 21 days. An example query that retrieves the patients for which the OI treatment period is shorter or equal to 21 days, is the following:

```
select p
for each ARCPatients p, OITherAdmins oiTher
where PatientId(oiTher) = p and
         GranulesNo(DateToInt(Now()),startDate(oiTher), 3) <= 21;
```

The function GranulesNo in the query above operates on the instances of type OITherAdmins and returns the number of granules of type "days" (coded with 3) contained in the interval having extremes the startDate of each instance and the actual date in which the therapist consults the archive, retrieved from the system date by the function Now. The function DateToInt operates the conversion from the string format returned by Now into the internal integer representation.

A second example of query could be to retrieve in the actual day the patients for which an OI therapy has been terminated:

```
select p
for each ARCPatients p, OITherAdmins oiTher
where PatientId(oiTher) = p and
        Finishes_(oiTher, DateToEvent(Now());
```

Since a TimeInterval event (therapy administration) is compared with a SingleEvents (actual date) and because the particular implementation of temporal operator (valid both for intervals than for time points), the same logical and actual result could be obtained in the previous select statement by replacing the Finishes_ with the From_ operator.

At the end of OI therapy cycles, CBC and SMA20 tests may be requested in order to check the toxicity level. If the toxicity level is lower than 3, the study medication can be restarted, otherwise the patient has to remain on study medication interruption until the toxicity level decreases under 3. The toxicity level for the patient will be tested for a period of 14 days. If after this period the toxicity level is again equal or higher than 3, the patient will remain on study medication interruption.

The following query checks for the instances of CBC and SMA20 test results at the minimal distance in time and not farther than 14 days from the actual date. The patients whose tests satisfy these requests and whose level of toxicity is re- entered into the normal range will be selected.

```
select p
for each ARCPatients p, OITherAdmins oiTher, CBCs cbc, SMA20s sma20
where PatientId(oiTher) = p and
        PatientId(cbc) = p and
        After_(DateToEvent(Now()),oiTher) = 1 and
        GranulesNo(DateToInt(Now()),startDate(oiTher), 3) <= 14 and
        cbc = SelFirstEvBeforeDate("CBCs",Now()) and
        sma20 = SelFirstEvBeforeDate("SMA20s",Now()) and
        Toxicity(cbc,sma20) < 3;
```

7 Future research

One of the most important issues for future research is represented by the need to abstract from the particular context (the ARC scenario) the semantics and the syntax to express temporal data and queries for the widest variety of applications. The next step of our research moves in the direction to collect and to analyze different kinds of temporal retrieval in the same and in different domains.

The problem of time granularity is far from a solution within the present work.

We should be able to give to the user the opportunity to manage the temporal information in the format that she or the circumstances need. For instance, let us consider the case in which a physician interviews a patient about his past diseases. It may happen that, for certain diseases, the patient remembers "exactly" the period of duration in terms of start and end dates. For other diseases he can only remember the month and the year or only the year. For example, the Patient HIV Albert remembers that he had hepatitis A in 1989 and that he was hospitalized from Feb 21st 1989 to Apr 15th 1989. In the first months of the same year he had even recurrent candidiasis, but he can't remember the exact dates. So, the user should be able to represent the different illness episodes as reported by the patient: hepatitis A in the time interval Feb 21st 1989 - Apr 15th 1983 and candidiasis "sometime" in the first months of 1989. Actually, the second event is neither a SingleEvent, nor a TimeInterval, as we modeled. It is, rather, an interval spanning an uncertain duration inside the year of 1989. We should be able, however, to represent and to manage this kind of temporal information as well as the deterministic one.

A related problem is represented by the need to transform the time information stored in single event format into time interval format [Wiederhold G. et al, 1990]. This need may occur whenever data have been modeled as single events not because they are events actually lasting only one instant but, rather, events whose values are well known only for a little time; for example, the results of laboratory tests relative to samples collected in a particular date (e.g. CBC, SMA20). Addressing this problem, we should experiment with methodologies that "expand" a time point into an interval and, at the same time, are capable to represent the a-priori unknown behavior ("pattern") of the time-varying information in the different instants of the expanded interval. The dual operation permits us to "coalesce" into a single time interval two or more subsequent intervals. This is possible and useful in terms of efficiency and memory occupancy whenever the temporal behavior of the particular property is globally homogeneous within the entire, extended interval and whenever the loss of information about the nternal time-stamps, denoting the original extremes of validity for the property's behavior, is disregardable.

Abstract semantics for general applications and time granularity are undoubtly the most important issues. However, another complex and intriguing aspect of time domain, strictly related with the previous ones, is represented by temporal reasoning. It will be interesting in the future to experiment with methodologies that permit one to reason on temporal data, modeled within an object-oriented database.

8 Aknowledgments

The author would like to thank Paul Tang's research group inside the Hewlett-Packard Laboratories - Palo Alto - CA, and expecially Jurgen Annevelink whose comments and ideas have actively encouraged and guided the development of the work. Thanks to the Edward Shortliff's research group inside the Shool of Medicine - Stanford University - CA for the informations about the clinical scenario and for the precious feedback.

9 References

Allen J.F.: "Maintaining a Knowledge about Temporal Intervals", Communications of the ACM, vol. 26, n. 11, pp. 832 - 844, 1983.

Allen J.F.: "Towards a General Theory of Action and Time", Artificial Intelligence n. 23, pp. 123 - 154, 1984.

Annevelink J.: "Database Programming Languages: A Functional Approach", Technical Memo HPL-DTD-90-12, Hewlett-Packard Laboratories, Palo Alto, CA, November 1990.

Chaudhuri S.: "Temporal Relationships in Databases", Proc 14th VLDB, pp. 160 - 170, August 1988.

Cousins S.B., Kahn M.G., Frisse M.E.: "The Display and Manipulation of Temporal Information", Proceedings of the 13th SCAMC, Washington, DC, November 1989.

Elmasri R., Wuu G.T.J.: "A Temporal Model and Query Language for ER Databases", Proceedings of the 6th International Conference on Data Engineering, pp. 76 - 83, February, 1990.

Fishman D.H. et al.: "Overview of the Iris DBMS", Object Oriented Concepts, Databases and Applications, Kim W. and Lochovsky F.H., Eds. New York, ACM, 1989.

Gadia S.K.: "A Homogeneous Relational Model and Query Language for Temporal Databases", ACM TODS, vol. 13, no. 4, pp. 418 - 448, December 1988.

Fagan L.M.: "VM: Representing Time-Dependent Relation in a Medical Setting", PhD thesis, Stanford University, 1980.

Kamens S.N., Wiedrhold G.: "An Implementation of Temporal Queries for SQL", submitted for publication in 1990.

Kahn M.G.: "Model Based Interpretation of Time-Ordered Medical Data", PhD thesis, University of California, San Francisco, 1988.

Kahn M.G., Fagan L.M., Tu S.W.: "Extensions to the Time-Oriented Database Model to Support Temporal Reasoning in Medical Expert Systems", Methods of Information in Medicine, no. 30, pp. 4 - 14, 1991.

McDermott D.: "A Temporal Logic for Reasoning About Processes and Plans", Cognitive Science, no. 6, pp. 101 - 155, 1982.

Navathe S.B., Ahmed R.: "TSQL: A Language Interface for History Databases", Proceedings of TAIS Conference, Sophia Antipolis. France, pp. 115 - 128, 1987.

Sarda N.: "Extensions to SQL for Historical Databases", IEEE Transactions on Knowledge and Data Engineering, vol. 2, no. 2, June 1990.

Shortliffe E.H., Scott A.C., Bishoff M.B, Campbell A.B., Van Melle W., Jacobs C.D.: "ONCOCIN: an Expert System for Oncology Protocol Management", Proceedings of the 7th International Joint Conference on Artificial Intelligence, Vancouver, pp. 876 - 880, 1981.

Snodgrass R., Ahn I.: "Temporal Databases", IEEE Computers, vol. 19, no. 3, pp. 35 - 42, September 1986.

Snodgrass R.: "The Temporal Query Language TQuel", ACM Transactions on Database Systems 12, 2, pp. 247 - 298, 1987.

Snodgrass R.: "Temporal Databases: Status and Research Directions", SIGMOD RECORD, vol. 19, no. 4, December 1990.

Wiederhold G., Fries J.F., Weyl S.: "Structured Organization of Clinical Data Bases", Proceedings of the AFIPS National Computer Conference, AFIPS, pp. 479 - 485, 1975.

Wiederhold G., Sushil J., Witold L.: "Dealing with Granularity of Time in Tempoaral Databases", submitted for publication in 1990.

Wilkinson K., Lyngbaek P., Hasan W.: "The Iris Architecture and Implementation", IEEE Transaction on Knowledge and Data Engineering, vol. 2, no. 1, March 1990.