Time Granularity and Temporal Indeterminacy in Modeling and Querying Temporally-Oriented Object-Oriented Databases

Carlo Combi

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

Talk Overview

- **Introduction**
- **The object-oriented temporal data model GCH-OODM (Granular Clinical History - Object Oriented Data Model)**
- **The query language GCH-OSQL (Granular Clinical History - Object Structured Query Language)**
- **An application to a clinical database**
- **Final outlines**

Motivation: the temporal clinical information

1) "In 1990 the patient took a calcium-antagonist for three months"

2) "The patient had abdominal pain from 5 p.m. to 7 p.m., March 27, 1989"

3) "At 4:45 p.m., October 15, 1990, the patient suffered from myocardial infarction"

4) "On November 30, 1991, in the afternoon the physician got a blood pressure of 120/80 from the patient"

5) "At 4:30 p.m., October 26, 1991, the patient's renal colic ended, it lasted five days"

6) "The patient suffered from an episode of tachycardia lasting for 150 seconds on October, 26 1991, at 3:22 p.m."

Goals

- **Modeling temporal granularity and temporal indeterminacy**

- **Supporting time granularity in querying the database**

- **Managing uncertainty in temporal relationships**

GCH-OODM
Granular Clinical History-Object Oriented Data Model

GCH-OODM is an object-oriented data model, extended to consider and manage the valid time of information

- Basic Concepts:
  - ♦ Objects and Classes (Types)
    - ⇒*state* (attributes)
    - ⇒*interface* (methods)

- GCH-OODM supports:
  - ♦ data abstraction and encapsulation
  - ♦ object identity
  - ♦ single inheritance
  - ♦ complex objects
  - ♦ persistence

GCH-OODM Types

- The usual types: char, char*, int, real, array, list, set, ...
- the type *bool3*
- the type *granularity*

  - {SUP, *yy*, *mm*, *dd*, *hh*, *min*, *ss*, INF} -

- The type hierarchy *el_time*, *instant*, *duration*, *interval*
- the type *t_o_set*

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

The predefined class *bool3* manages a three-valued logic.

The truth values are T: *True*, F: *False*, and U: *Undefined*.

The usual logical connectives AND, OR, NOT, IMPLIES, ..., and the logical quantifiers EXISTS (∃), and FOR EACH (∀) have been extended to consider the three truth values.

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

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

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)).

Time modeling

The class *el_time* allows us to model time points, i.e. chronons, on the basic time axis

- Format:
  - ♦ **`YY/MM/DD/HH/Mi/SS`** for time points
  - ♦ **`Y yy M mm D dd H hh Mi min S ss`** for distances between time points
- Examples:
  - ♦ **`94/10/10/0/0/0`** identifies the first second of October 10, 1994
  - ♦ **`5 min 2 ss`** identifies a duration lasting 5 minutes and 2 seconds

Time modeling

The class *instant* allows us to represent a time point, identified by the granule, i.e. a set of contiguous chronons, containing it

- Methods: *inf*(), *sup*() each returns an object of type *el_time*

- Format:

  ♦ **`YY/MM/DD/HH`,** or **`YY/MM/DD`**, or **`YY/MM`**, or ...

  ♦ $\prec$**`YY/MM/DD/HH/Mi/SS, YY/MM/DD/HH/Mi/SS`**$\succ$.

- Examples:

  ♦ **`94/10/10`** specifies a time point included between **`94/10/10/0/0/0`** and **`94/10/10/23/59/59`** (objects of type *el_time*)

  ♦ $\prec$**`96/1/1/6/30/0, 96/1/1/6/36/59`**$\succ$ specifies a time point between 6:30 and 6:36 of January first, 1996

Time modeling

The class *duration* allows us to model a generic time span, specified at arbitrary granularity

- Methods: *inf*() and *sup*() each returns an object of type *el_time*

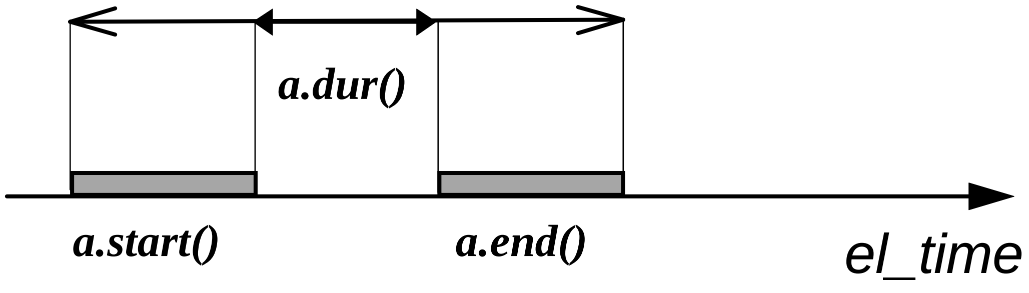
- Format:
  - ♦ **`Y yy M mm`**, or **`Y yy M mm D dd`**, or **`H hh`**, or ...
  - ♦ **`≺Y yy M mm D dd H hh Mi min S ss,`**
    `Y yy M mm D dd H hh Mi min S ss≻`

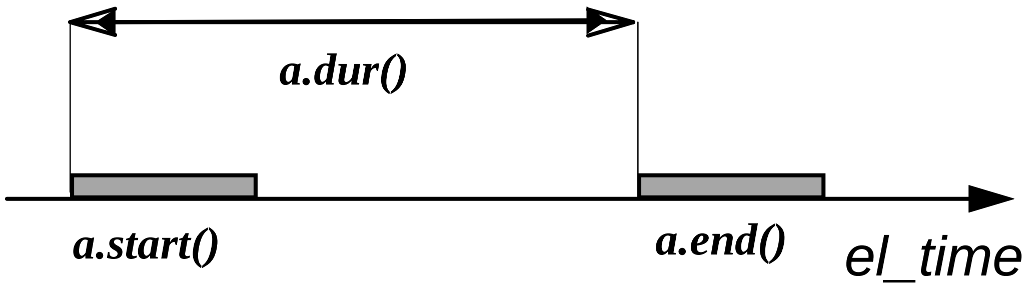
- Examples:
  - ♦ **`4 yy`**
  - ♦ **`3 yy 2 mm 3 dd`**
  - ♦ **`2 min 3 ss`**
  - ♦ **`≺3 dd 4 hh 6 min 3 ss,`**
    `4 dd 6 hh 5 min 2 ss≻`

Time modeling

A generic interval, i.e. a set of contiguous time points, is modeled by the class *interval*

- Methods:
    - ♦ *start*(), *end*() each returns an object of type *instant*
    - ♦ *dur*() returns an object of type *duration*





Temporal Relationships

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*

- 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($b.start()$) $=_{df}$

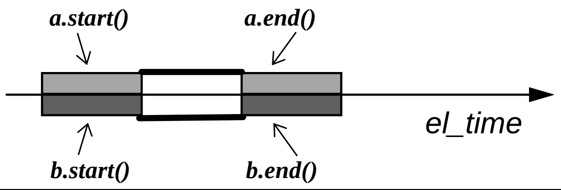
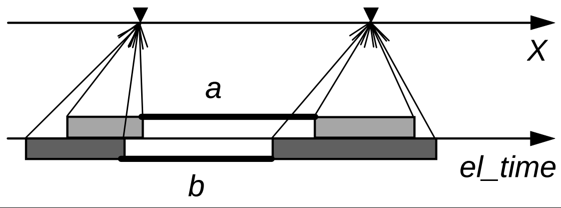
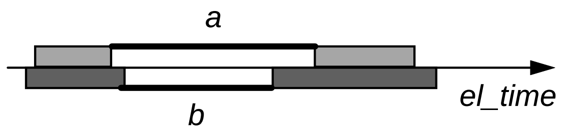
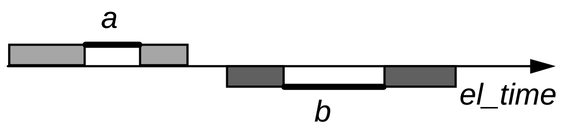
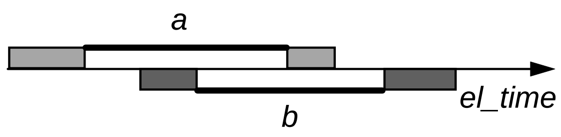
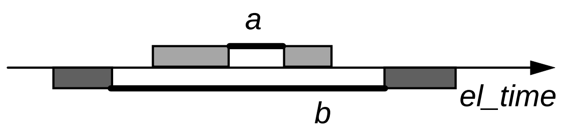
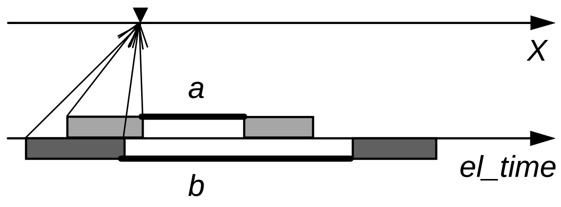
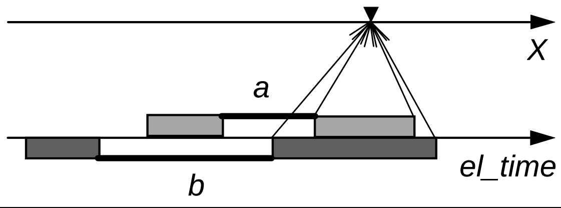
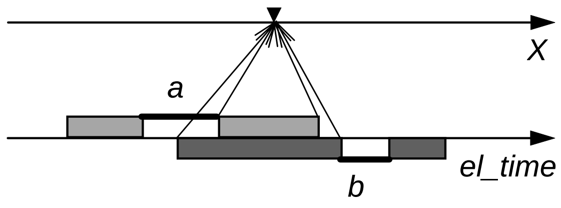
- T iff $a.end().sup() < b.start().inf()$
- F iff $a.end().inf() > b.start().sup()$ OR

  ($a.end()$.inf() $= b.start().inf()$ AND

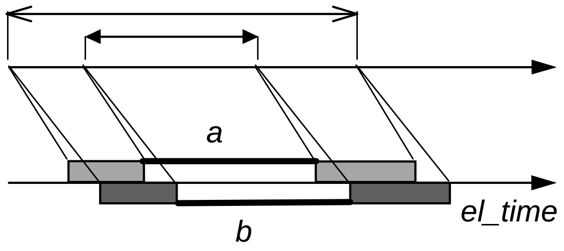
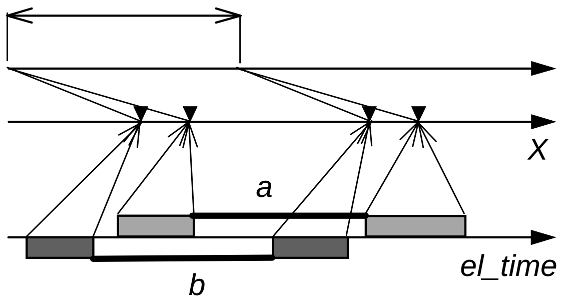
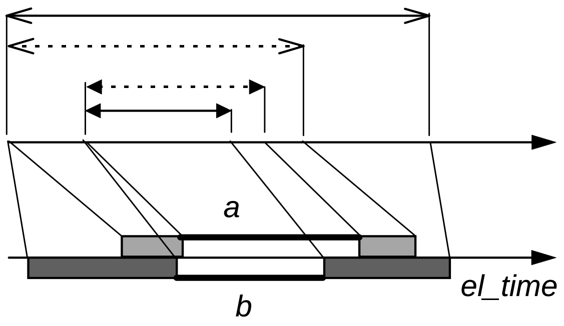
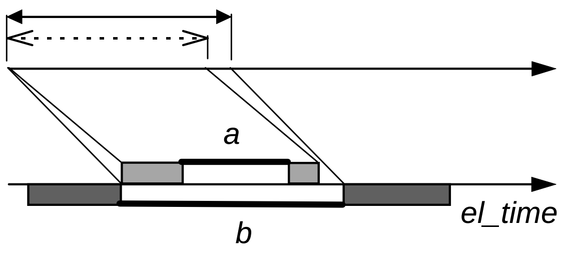
  $a.end().sup() = b.start().sup()$ AND

  $a.end().inf() = a.end().sup()$)
- U otherwise

Temporal Relationships

Notation	***Pictorial example***
a.TEMP_EXPR_AS(*b*)	
a.CONTEMPORARY(*b*, *X*)	
a.T_SPECIFIES(*b*)	
a.BEFORE(*b*)	
a.OVERLAPS(*b*)	
a.DURING(*b*)	
a.STARTS(*b*, *X*)	
a.FINISHES(*b*, *X*)	
a.MEETS(*b*, *X*)	

Temporal Relationships

Notation	***Pictorial example***
a.LASTS_AS(*b*)	
a.LASTS_LIKE(*b*, *X*)	
a.DUR_SPECIFIES(*b*)	
a.LASTS_LESS(*b*)	

Temporal and atemporal classes and methods

- **Temporal classes**

Objects instances of temporal classes (hereinafter temporal objects) have an associated valid interval. The method *valid_interval()* returns the interval of validity of an object.

- **Atemporal classes**

Objects instances of atemporal classes (hereinafter atemporal objects) model information, not having an associated temporal dimension.

- **Temporal methods**

Temporal methods model temporal features and return temporal objects.

- **Atemporal methods**

Atemporal methods return atemporal objects.

Sets of temporal objects: the class *t_o_set*

The class *t_o_set* (temporal_object_set) allows the construction and the management of sets of temporal objects

- Usual operations on sets: insertion, deletion, intersection, union, difference, ...
- Methods verifying the existence of temporal relations between objects belonging to an instance of the class *t_o_set* and satisfying some conditions on atemporal methods.
  - ♦ I.*subset*(*p*) returns the subset of temporal objects belonging to I, of *type t_o_set*, and satisfying the atemporal formula *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*))

Sets of temporal objects: the class *t_o_set*

The class *t_o_set* is a temporal class

The valid interval of an object I 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* ∈ I

I.*valid_interval*().*start*().*sup*() ≡

min(*x*.*valid_interval*().*start*().*sup*()), *x* ∈ I

I.*valid_interval*().*end*().*inf*() ≡

max(*x*.*valid_interval*().*end*().*inf*()), *x* ∈ I

I.*valid_interval*().*end*().*sup*() ≡

max(*x*.*valid_interval*().*end*().*sup*()), *x* ∈ I

The class *t_o_set*

Two orthogonal ways of specializing the *t_o_set* class:

- Specializing temporal objects managed by the class

  ♦ *t_o_set*<*c*> is a specialization of *t_o_set*<*c'*> iff the temporal class *c* is a specialization of the temporal class *c'*.

- Defining explicitly some constraints on the managed temporal objects.

  ♦ Example:

  class time_varying_property: public t_o_set {
  .....
  }

  For each instance P of the class the following formula holds:

  ∀ *o* ∈ P:*time_varying_property* NOT ∃*o'* ∈ P
  (NOT*(o'.valid_interval() < o.valid_interval()* OR
  o'.valid_interval() > o.valid_interval()) AND
  *o.valid_interval().gran() = X***)**

The example clinical database

```
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();
};
```

The GCH-OSQL query language

The temporal extension to the syntax of SQL concerns the part needed for database queries

The SELECT statement:

```
SELECT <class methods or path expressions>

FROM <classes>

[WHERE <temporal and atemporal conditions>]

[TIME_SLICE [MUST|MAY] <interval>]

[MOVING WINDOW [MUST|MAY] <duration>]
```

The SELECT and FROM clauses

- Only methods related to the displaying of the data are allowed in the SELECT clause

- To each class listed in the FROM clause, an object variable is associated

Example

"*Find all the patients having nausea and display patient name and the starting instant of each nausea symptom*"

```
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"
```

The WHERE clause

- In the WHERE clause are specified the logical conditions that identify the objects to be retrieved in the database
- Complex conditions 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 through *t_o_set* class methods, and using *interval* class methods

The WHERE clause

Example

"*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*"

```
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())
```

The WHERE clause

Example

"*Find the patients having had nausea and headache, with headache surely before nausea*"

```
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())
```

```
SELECT P.name()

FROM patient P,

WHERE MUSTBE P.symptom_set().BEFORE("s_name()

= "headache"", "s_name() = "nausea"")
```

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 or could be contained in the specified interval

- MUST and MAY keywords
- Formats:
    - ♦ FROM..TO..: e.g., `FROM 1994/12/11 TO 1994/12/23/11/00`. It is also possible to specify only FROM.. or only TO...
    - ♦ FROM..FOR..: e.g., `FROM 1994/12/11 FOR 2 mm`.
    - ♦ FOR..TO..: e.g., `FOR 3 dd TO 1995/4`.
    - ♦ AT..: e.g., `AT 1996/5`.

The TIME-SLICE clause

Example

"*Find all symptoms happened starting from December 1991 to November 12th, 1996 in the afternoon*"

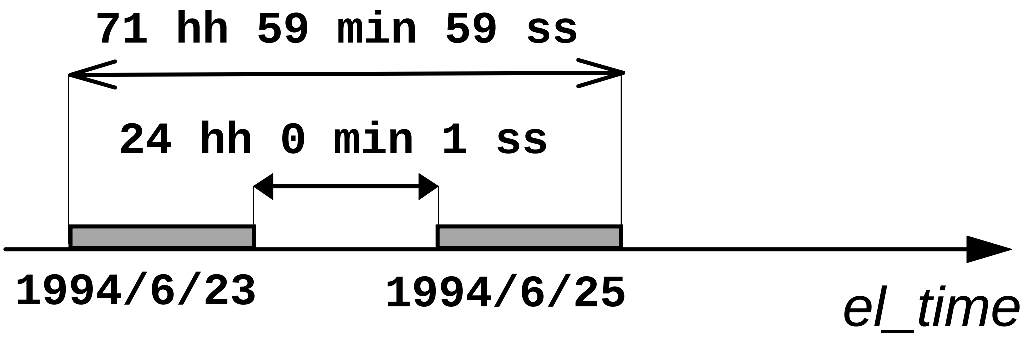
```
SELECT S.s_name()
FROM symptom S
TIME-SLICE FROM 1991/12 TO ≺1996/11/12/12/0/0,
      1996/11/12/17/0/0≻
```

The TIME-SLICE clause

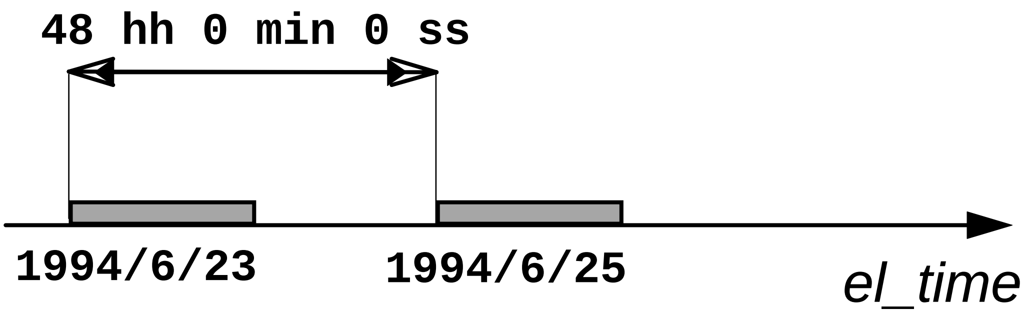
Defining the TIME-SLICE interval by, respectively, the FROM..TO, FROM..FOR, and FOR..TO keywords is not redundant

Example:

```
TIME-SLICE FROM 1994/6/23 TO 1994/6/25
```



```
TIME-SLICE FROM 1994/6/23 FOR 48 hh 0 min 0 ss
```



The MOVING WINDOW clause

Objects stored in the database are examined through a temporal window, of the width specified in the clause, moving along the temporal axis

♦ MUST and MAY keywords

Example

"*Print the name of patients having had heart rate greater than 120 and symptoms of chest pain in a period of fifteen days*

```
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
```

The clinical database

We selected the population of patients subjected to Percutaneous Transluminal Coronary artery Angioplasty (PTCA). Patients undergoing this operation stay in the hospital for a few (mostly two or three) days; afterwards, they are periodically followed up by different checks and angiographic examinations and may be reoperated. This category of patients has been selected for several reasons: the interest shown by the scientific-medical community for this kind of intervention; the size and the growth rate of the population.

For these patients historical data has to be updated also during the follow-up period.

Historical data and follow-up data have to be managed in a global way, to monitor the status of the patient.

Considered data for the PTCA-patients

- ID data
- Demographic data
- Risk factors
- Past and current pathologies
- Past and current therapies
- Data coming from periodical follow-up visits

Temporal clinical objects

The clinical database contains more instances of the class *t_o_set*, collecting temporal objects, modeled by the classes *therapy*, related to previous or current therapies, *diagnosis*, related to previous or current pathologies, *angio_visit*, related to the parameters - blood pressure, heart rate, .. - collected during follow-up visits.

An Example of clinical query by GCH-OSQL

Let us consider the following clinical query. For patient undergone to PTCA it is important that the blood pressure is normal (i.e. SBP: 100 - 150 mmHg, DBP: 60 - 100 mmHg) in the period following the intervention. We want to know which patients, having normal values of the blood pressure for 90 days following the intervention, suffered from angina and undergone to a reintervention within three years, considering the period between March 1987 and April 1995.

An Example of clinical query by GCH-OSQL

```
SELECT P.surname(), P.name()
FROM patient P, diagnosis D, angio_visit B,
angio_visit C
WHERE P.Dia_Set().HAS_MEMBER(A) AND
P.Visit_Set().HAS_MEMBER(B) AND
P.Visit_Set().HAS_MEMBER(C) AND
B.angio_exam().exam_type() = "PTCA" AND
C.angio_exam().exam_type() = "PTCA" AND
A.pathology()="angina" AND MUSTBE
(B.valid_interval().BEFORE(A.valid_interval()) AND
A.valid_interval().BEFORE(C.valid_interval())) AND
P.Visit_set().MAINTAINS("SBP()>100 AND SBP()<150",
B.start(),"90 dd") AND
P.Visit_set().MAINTAINS("DBP()>60 AND DBP()<100",
B.start(),"90 dd")
TIME SLICE FROM 1987/3 TO 1995/4
MOVING WINDOW 3 yy
```

An Example of clinical query by GCH-OSQL

In this GCH-OSQL query we can observe some relevant features:

- different granularities (years, months, and days) are explicitly used;
- the MUSTBE operator allows us to verify the certainty of the precedence between PTCA, angina, and the new PTCA;
- for temporal relationships, the query uses both methods of the class *interval* (BEFORE is a method of the class *interval*) and of the class *t_o_set* (MAINTAINS and HAS_MEMBER are methods of the class *t_o_set*);
- the condition by the MAINTAINS methods consider also the patient for which it may be the blood pressure has been in a range of normality.

System Description

Used Instruments

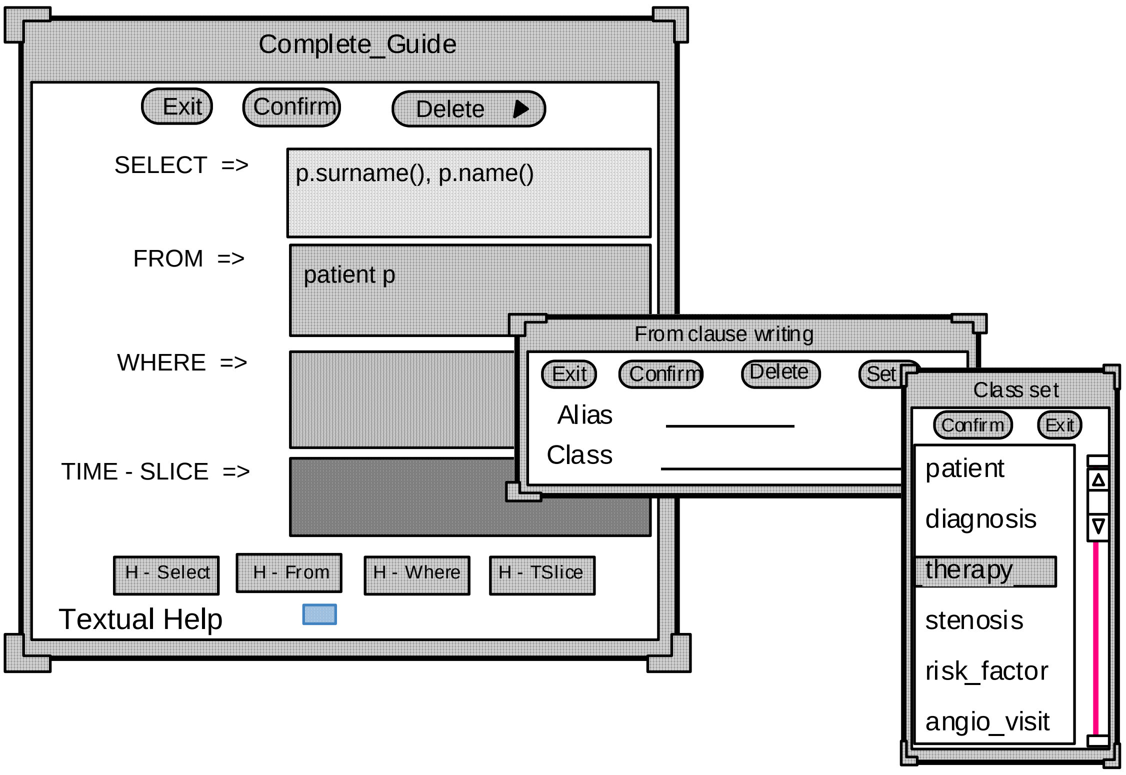
- Sun SparcStations with SunOS 4.1.x
- ONTOS Object-oriented Database Management System / Ode OODB
- Glockenspiel C++ compiler / Sun C++ 4.1
- X Toolkit

The graphical user interface of GCH-OSQL

GCH-OSQL provides users with a set of different tools, related to the user skill, guiding to compose correct and sound queries. In the construction of the graphic interface of GCH-OSQL we tried to satisfy two different needs:

- that of a user already experienced of the system, requiring a simple help in composing the clauses - we indicate such guide with the term of Elementary_Guide -;
- that of an inexperienced user, which wants to be driven in the formulation of queries - we point out such guide with the term of Complete_Guide -.

An example of the Complete_Guide modality in composing the query



Final outlines

- **Management of temporal granularity and of temporal indeterminacy**
    - ♦ While inserting data
    - ♦ In querying the database
- **Management of uncertainty in temporal relations by a three-valued logic**
    - ♦ *True, Undefined, False*
    - ♦ MUSTBE, MAYBE, ...
- **Homogeneous management of temporal conditions in the query**
    - ♦ Few additional clauses: TIME-SLICE and MOVING WINDOW
    - ♦ Temporal conditions in the WHERE clause