Chapter 4
Temporal Clinical Databases

Overview

This chapter introduces the reader to some important issues and topics of temporal clinical databases. More specifically the chapter focuses on modeling and querying issues related to the management of temporal clinical data collected into medical records. The reader, then, learns how some temporal relational and object-oriented models have been suitably defined to deal with the specific temporal aspects of clinical data. The presentation then focuses on some main data models explicitly proposed for managing clinical data with multiple temporal dimensions and with clinical temporal data given at different and mixed granularities/indeterminacies. The reader learns how to design and query a temporal clinical database both by the temporal relational model and by temporal object-oriented data models with the related query languages, respectively. Finally, the reader is guided through some specific aspects of a real temporal clinical database system allowing the management of follow-up patients who underwent cardiac angioplasty.

Structure of the Chapter

This chapter deals with the task of management of time-oriented clinical data. The relevant literature has progressed from the early systems, which were mostly application dependent, to more general approaches, that, even when applied to the solution of real problems in the management of time-oriented clinical data, have a more generalizable value and inherent soundness. An important role in this direction is that of research efforts in the general field of temporal databases, as detailed in the previous chapter. According to this perspective, in this chapter we first give a brief historical overview of the field; we then deal with two different aspects, which have been considered in the design of temporal clinical database systems, namely that of multiple temporal dimensions of clinical data and that of different

C. Combi et al., *Temporal Information Systems in Medicine*,
DOI 10.1007/978-1-4419-6543-1_4,

temporal granularities in clinical data. Relational and object-oriented methodologies and technologies are discussed and suitably extended and adopted to deal with the above mentioned topics: both theoretical, methodological, and technical issues will be faced, underlining the general aspects and discussing the possible related solutions. The presence of several multiple temporal dimensions of clinical data is an important issue which has been considered both in the temporal database community and in the medical informatics one. This chapter introduces the main temporal dimensions proposed in the literature, by highlighting through clinical examples their importance when dealing with clinical information. The handling of variable temporal granularity (i.e., time units) is a recurring task in managing clinical data. The approaches we describe consider temporal data at different granularities and with indeterminacy.

The management of data of some specific clinical domains is also considered; in particular, this chapter considers several examples taken from a generic patient record and discusses the design and implementation of a web-based object-oriented system managing data from follow-up patients, who underwent angioplasty.

Keywords

Temporal clinical information, granularity, indeterminacy, temporal data models, temporal query languages, relational model, object-oriented data model, multiple temporal dimensions, angioplasty, medical records, web-based clinical applications.

4.1 Introduction

Researchers in the medical informatics field investigated temporal data maintenance, mainly to support electronic medical records. Indeed, a wide variety of applications need to deal with temporal aspects of clinical data for the management of time-oriented data stored in medical records of ambulatory or hospitalized patients [429, 197, 196, 426, 248, 307, 309, 84, 86, 125, 126, 341, 120, 111, 112, 389, 405, 44].

Studies of time-oriented data-centric applications have been performed in multiple clinical areas: cardiology [194, 309, 86], oncology [111, 197], psychiatry [29], internal medicine [120, 335], intensive care [74, 70] urology [120], infectious diseases [111], anesthesiology [70, 80]. Various clinical tasks are supported by the systems proposed in the literature: diagnosis [29], therapy administration and monitoring [112, 200, 405], protocol- and guideline-based therapy [111], and patient management [426, 123, 124, 120, 307, 86, 44].

Moving on to the task of management of time-oriented clinical data, we observe that from the early systems, which were mostly application dependent, the proposals progressed to more general approaches, that, even when applied to the solution of

real problems for the management of time-oriented clinical data, have a more generalizable value and inherent soundness [426, 31, 197, 309, 51, 227, 111, 123, 86, 83, 124, 82, 158, 81, 113, 389, 405, 44]. Initially, systems that were designed to manage temporal clinical data were based on the flat relational model [426, 31]. These systems, such as Wiederhold's TOD and Blum's Rx, were based on timestamping database tuples: the date of the visit was added to the specific attribute values.

One of the first applications of databases to clinical domains, explicitly addressing the time representation problem, is the Time Oriented Database (TOD) model [426], originally developed at Stanford University during the 1970s. This model has been adopted, for example, by the American Rheumatism Association Medical Information System (ARAMIS), to manage data related to the long-term clinical course of patients suffering from arthritis or, more generally, from rheumatic pathologies [379]. TOD uses a "cubic" vision of clinical data: values of data related to a particular patient visit are indexed by patient identification number, time (visit date), and clinical parameter type. Specialized time-oriented queries enable researchers to extract, for particular patients, data values that follow certain simple temporal patterns (e.g., increase at some rate). Assignment of a temporal dimension at the tuple level is a method common to many applications of clinical databases [37, 125, 309].

Kahn et al. [197, 196] proposed a specific query language, TQuery, for data that is structured by a specific data model, named TNET. Even though TQuery was patient oriented and was not based on a generic data model, it was one of the first proposals for an extension of query languages so as to enable the system to retrieve complex temporal properties of stored data. Most query languages and data models used for clinical data management were application-dependent; thus, developers had to provide ad-hoc facilities for querying and manipulating specific temporal aspects of data [196].

The following proposals on temporal clinical databases present a more general approach. Extensions of common data models, and in particular of the relational model, are based also on the general database-field literature, in which temporal databases have been attracting special attention since several years [399, 79, 387, 289, 139].

Another distinction useful for the characterization of much of the research in maintenance of temporal data is whether the main topic is the definition of temporal data models [197, 307, 86, 88] or the definition and design of temporally-oriented query languages [111, 112, 83, 80, 127, 94]. Both topics are discussed in the medical informatics literature, although with a more focused interest in data modeling. In medical informatics, attention had been paid mostly to historical databases (which emphasize valid time), extending relational or object-oriented models [111, 83, 80, 82, 158, 127, 389].

The management of temporal clinical data has some specific features which, even though shared with other application areas, make important the specific study of temporal databases for medical data. Quite surprisingly, in our experience these specific features are neither directly related to the specific medical content of clinical records to manage, nor do they depend on the specific clinical environment to

consider: they are *general* features, which, however, are so important for the medical field that it is mandatory to consider them in designing temporal database systems for medical data [17, 5, 389, 364, 437]. In the following, we briefly introduce some relevant specific issues, we have to face when dealing with temporal clinical information.

A first set of issues is when we need to model temporal clinical data.

- Often temporal dimensions of clinical data are multiple and cannot be modeled only through valid and transaction times. For example, we could need to associate to the fact "the patient suffered from asthma" both the time when the fact happened, e.g., "in Summer, 1998" (i.e., the valid time), the time when the fact has been inserted into the database, e.g., "August, 17, 2002, 16:23:00" (i.e., the transaction time), and the time when the fact has been notified by the patient to his physician e.g., "July, 10, 2002, 10:00". It is worth noting that this last temporal dimension cannot be captured neither by the valid nor the transaction times.
- The temporal dimension is expressed sometimes by using intervals, for facts having a span of time, and sometimes by using instants, for events occurring at a time point. Moreover, this temporal dimension can be expressed in different and heterogeneous ways: the used time axis, for example, has different time units ("in 1998 the patient had a stroke", "at 4:00 p.m. on June 2000, the patient had an episode of amnesia: it lasted 15 minutes"); in other cases the temporal location is expressed with some vagueness ("between 18:45 and 19:13 of May 25, 2007 the patient had for 150 seconds atrial fibrillation"). We say that the temporal dimension of clinical information is given at different *granularities*, i.e. with different units of measure, and with *temporal indeterminacy*, i.e. with some vagueness in defining its temporal location [88]. Furthermore, different granularities and/or indeterminacy may be used in several different ways to define temporal intervals ("on October 23, 2006 at 18:21 for 35 seconds", "for 5 hours until 13:20 of November 23, 2005", "it started between 14:25 and 14:38 and ended between 18:21 and 18:36 of December 23, 2005"). We also have to consider that this heterogeneity may be present even for the same kind of clinical information: an episode of ventricular fibrillation, for example, can be identified, in a context-varying way, both by the definition of its starting and ending instants, and by the day of its occurrence and by its duration, expressed with the time unit of seconds.
- Clinical information may consist both of natural language, "qualitative" sentences ("the patient suffered from asthma in Summer, 1998", "the patient finished on 11/09/99 a therapy with anticoagulants, lasting from two months"), and of quantitative parameter values ("on May 28, 2001, at 17:25 the physician got a heart rate of 78 bpm from the patient", "blood sample of September 19, 2001: cholesterol in serum is 212 mg/dl").
- In dealing with information having different granularities or indeterminacy, it is possible that in some cases temporal relations cannot be asserted for sure. It is not possible, for example, establish whether "cerebral stroke on November 21, 2003" is before or after "an episode of painless vision loss in November 2003".

A second set of issues is related to the needs arising when physicians have to query the clinical database to support, for example, clinical decisions they have to take.

- It happens that several different temporal dimensions could be involved in the definition of a query on temporal clinical data. For example, when it is necessary to evaluate the quality of care provided by a hospital ward, it is important to consider both when patients' therapy, symptoms, visits and so on occurred and when temporal clinical data describing these facts were at disposal for clinical decision making: these temporal dimensions, as well as other possible temporal dimensions like the transaction time, have to be considered together as they provide different, orthogonal, and complementary information.
- The evaluation of clinical information is often performed according to criteria involving indeterminacy and granularities. For example, in the definition of unstable angina [40], different granularities are involved: "angina at rest, for more than 20 minutes", or "new onset, within two months, exertional angina, involving marked limitations of ordinary physical activity", or "increasing angina within two months from the initial presentation", or "post-myocardial infarction angina, at least 24 hours after". Finally, therapy prescriptions involve different time units and/or indeterminacy: e.g., nitroglycerin is recommended "for the first 24 to 48 hours in patients with acute myocardial infarction and CHF (Congestive Heart Failure), large anterior infarction, persistent ischemia, or hypertension", but in intensive medical management of unstable angina, "the heparin infusion should be continued for 2 to 5 days", and morphine prescriptions are given by using minutes as time unit.
- In querying the system about the stored clinical information we usually use granularities which are different from and not related to the ones used when storing data. In the same query different granularities and/or indeterminacy may be used. Queries can consider conditions both explicitly related to absolute time, and about qualitative or qualitative temporal relations between facts. It must be possible, in other words, to define queries on the database that would be expressed in natural language by sentences as: "We want to identify those patients that suffered from chest pain in 2007 and who, in the ten months after that event, had episodes of ventricular fibrillation lasting no more than 3 seconds" or "We want to know the patients that in the years between 1990 and 1995 had an ocular stroke followed by another stroke in the following seven months, and who were treated with blood thinners after the second intervention for at least 45 days".
- The examples provided in the previous item introduce another issue that needs to be faced: that of *absolute and relative time windowing*. Indeed, when querying clinical data it is often necessary to consider only clinical data belonging to a specified temporal window. Temporal windows could be either *absolute* (e.g., ".. in the years between 1990 and 1995 ..") and, thus, holding for all the query results, or *relative* (e.g., ".. had an ocular stroke followed by another stroke in the following seven months ..") and, thus, referring to (multiple) time periods depending on the clinical events considered for the query results (as, in the above example, the first ocular stroke of each considered patient.)

4.2 Multiple Temporal Dimensions in Clinical Databases

Even though valid time (VT) and transaction time (TT) suffice for many database applications, they are often inadequate to cope with the temporal requirements of complex organizations such as hospitals and healthcare institutions. In these contexts, one often needs to model both the time at which someone/the health information system becomes aware of a fact (availability time) and the time at which the fact is stored into the database. The latter is captured by TT, while the *availability time*, AT, has been introduced and dealt with in [93].

Definition 4.1. The *availability time* (AT) of a fact is the time interval during which the fact is known and believed correct by the information system.

As in several other domains, medical decisions are taken on the basis of the available information, no matter whether it is stored in the database or not. AT captures this temporal dimension. Since there can be facts which are erroneously considered true by the health information system, AT must be an interval: the starting point of AT is the time at which the fact becomes available to the information system, while its ending point is the time at which the health information system[1] realizes that the fact is not correct.

In [215, 216], Kim and Chakravarthy introduce a fourth temporal dimension, called *event time*, to distinguish between retroactive and delayed updates. In [93], Combi and Montanari refine it by showing that two event times are needed to suitably model relevant phenomena. The choice of adding the event time as a separate temporal dimension has been extensively debated in the literature [289] and is discussed in detail in [93].

Definition 4.2. The *event time* (ET) of a fact is the occurrence time of a real-world event that either initiates or terminates the validity interval of the fact.

The event times are the occurrence times of events, e.g., decisions and actions, that respectively initiate and terminate relevant facts.

The concepts of availability time and of event time are useful to clarify the relationships between Kim and Chakravarthy's (initiating) event time and the related notion of *decision time*, which has been originally proposed by Etzion and his colleagues in [137, 148] and later refined in subsequent work, e.g., in [138]. The *decision time* of a fact is the time at which the fact is decided in the application domain of discourse. More precisely, it can be defined as the occurrence time of a real-world event (i.e., decision), whose happening induces the decision of inserting a fact into the database. Decision time has been considered in medical informatics in [362, 363]; moreover, it is worth noting that the concept of decision time is considered also in the medical literature with a sligthly different meaning: indeed, decision time has been defined as the time span between the onset of some symptom and the

[1] It is worth noting that an information system of an organization is composed of all the information, software and hardware tools, and human resources devoted to the management and processing of data and information needed by the organization for reaching its goals.

beginning of the decided action/therapy [180]. Even with the second meaning, it is straightforward to observe that the introduction of four temporal dimensions allows the representation of this time span, as it could be derived from the start of VTof the considered symptom and from the initiating event time of the considered action/therapy. Some studies have been done in medical informatics to analyse in a deep way the decision time and its more appropriate values, i.e. the most suitable moments for taking an action on a given patient [252]. In this direction, storing both event and availability times has relevant effects when trying to evaluate the quality of the care provided by the health organization, as they allow one to estimate different decision times involved with the health care processes on a patient and to correctly consider the information available when deciding an action for a patient, as this information could not correspond to that stored into the database when the decision was taken.

Focusing on a real-world example, let us consider the following scenario, we have to represent into a clinical temporal database.

Example 4.1. Patient Hubbard, identified by the attribute `PatId` having value `p2`, was visited on June 30, 2008 and high systolic and diastolic blood pressures were revealed (an SBP of 170 mm Hg and a DBP of 120 mm Hg were measured several times that day), due to a strong emotional stress when he was driving on June 28. Even though the physician was aware of the patient's situation since June 30, data were inserted into the database only some days after, on July 2. On June 30, the physician prescribed atenolol (50 mg/day) to the patient since that moment. Due to insertion mistakes, data represent a wrong dosage (5 mg/day) and a wrong start time of the therapy (June 30, 2007). These (wrong) data were inserted on July 2 into the database. On July 4, patient Hubbard was visited again and his blood pressure had normal values. The physician stored immediately this information into the database. On July 12, patient Hubbard calls the physician and says him that he stopped the therapy on July 11, due to an increasing chest pain since the day before: the physician prescribed to the patient a therapy with lisinopril, 10 mg/day until July 15, when the patient will need a further therapy redefinition. On July 13, the physician enters data about the new lisinopril-based therapy and about the end of the atenolol-based therapy. On July 17, the physician discovers that data about the atenolol therapy were wrong and corrects them.

The example highlights several and different temporal dimensions. For example, it is quite straightforward to identify both availability and transaction times as different temporal dimensions: indeed, the moment when the physician becomes aware of the patient's situation (e.g., hypertension) is different from the moment the corresponding data are inserted into the database. Moreover, it is possible, for example, to store the moment when the physician decides for a therapy or when the patient decides for the therapy interruption: in the considered case, the physician decides on June 30 for an immediate start of an atenolol-based therapy, while on July 10 the patient had chest pain that motivates a delayed stop of the therapy (the delay is of one day: the therapy ends on July 11). Initiating and terminating event times for the therapy allow one to represent these two different decisions and events, respectively.

Finally, the valid time models the period when the patient has to assume the prescribed drug; valid time also models when the patient had hypertension.

4.2.1 Modeling Temporal Data with Multiple Dimensions

The temporal data model, we will discuss here, considering all the four temporal dimensions is a straightforward extension of the relational model: a similar approach and considerations could be adopted even when we have to consider an object-oriented or an object-relational model.

In this temporal relational model, any relation, besides the user-defined attributes, is equipped with the four temporal dimensions. VT, TT, and AT are represented as intervals, while ET is represented by a pair of attributes, that respectively record the occurrence time of the initiating event and of the terminating one.

The temporal data model encompasses a single type of key constraints, namely, the *snapshot key* constraint (*key* for short). Given a temporal relation R, defined on a set of (atemporal) attributes X and on the special attributes VT, TT, AT, ET_i, and ET_t, and a set $K \subseteq X$ of its atemporal attributes, K is a *snapshot key* for R if the following conditions hold:

- $\forall a \in K(t[a] \neq \texttt{null})$
- $\forall t1, t2\ (t1[VT, TT, AT] \cap t2[VT, TT, AT] \neq \emptyset \Rightarrow t1[K] \neq t2[K])$

where $t[VT, TT, AT]$ denotes the temporal region (a cube) associated with the tuple t. ET is not involved in the definition of the snapshot key, because ET_i and ET_t denote two independent time instants (possibly related to different events), which do not identify any meaningful interval.

The model imposes some basic constraints on the relationships between the values of the various temporal dimensions: we cannot assign a value to ET if there exists no value for the corresponding VT and we cannot assign a value to AT if there exists no value for the corresponding TT.

The scenario reported in Example 4.1 is represented into the database composed of the two relations depicted in Tables 4.2.1 and 4.2.1. The relation `PatTherapy` stores information about patients (attribute `PatId`) and prescribed therapies (attributes `Drug`, `DailyDose`, and `Unit`), while the relation `PatVisit` stores information about patients and vital signs (attributes `SBP` and `DBP`). Both relations feature the four temporal dimensions VT, ET_i, ET_t, AT, TT. (`PatId`, `Drug`) and (`PatId`) are snapshot keys for the two relations, respectively. The special values *uc* and *now* denote current/available and still valid tuples, respectively.

As a final comment, let us consider some relevant information we are allowed to derive from these data in a decision making/decision evaluation perspective: at the current moment (surely after November 17, 2008), with regard to the content of the database, we are able to understand that, for example, patient `p2` assumed atenolol from June 30, 2008 to July 11, 2008. The event initiating this therapy happened

Table 4.1 Database instance for patient visits and related therapies: the relation *PatTherapy*.

PatId	Drug	DailyDose	Unit	VT	
$p2$	Atenolol	5	mg	[2007Jun30, *now*]	....
$p2$	Lisinopril	10	mg	[2008Jul12, 2008Jul15]	....
$p2$	Atenolol	5	mg	[2007Jun30, 2008July11]	....
$p2$	Atenolol	50	mg/	[2008Jun30, 2008July11]	....

	ET_i	ET_t	AT	TT
....	2008Jun30		[2008Jun30, 2008Jul12]	[2008Jul02, 2008Jul12]
....	2008July12	2008Jul12	[2008Jul12, *uc*]	[2008Jul13, *uc*]
....	2008Jun30	2008Jul10	[2008Jul12, *uc*]	[2008Jul13, 2008Jul16]
....	2008Jun30	2008Jul10	[2008Jul12, *uc*]	[2008Jul17, *uc*]

Table 4.2 Database instance for patient visits and related therapies: the relation *PatVisit*.

PatId	SBP	DBP	VT	
$p1$	120	80	[2008May01, 2008May01]	..
$p2$	170	120	[2008Jun30, 2008Jun30]	....
$p2$	115	70	[2008July04, 2008Jul04]	....

	ET_i	ET_t	AT	TT
....	2008May01	2008May01	[2008Nov17, *uc*]	[2008Nov17 *uc*]
....	2008Jun28	2008Jun30	[2008Jun30, *uc*]	[2008Jul02, *uc*]
....	2008Jun30	2008Jul04	[2008July04, *uc*]	[2008Jul04, *uc*]

on June 30, while the event finishing the therapy happened on July 10, one day before the end of the therapy. All this information was available to the information system since July 12, and was inserted into the database on July, 17. From relation `PatVisit` we are able to understand that the high blood pressure of the patient on June 30 was initiated by an event happened on June 28, while the event terminating the high blood pressure episode happened on June 30: as a matter of fact, in this case the terminating event is the start of the therapy with atenolol, represented in relation `PatTherapy`.

Let us now consider how the availability time may become relevant when evaluating the quality of clinical decision making. If we consider the information known by the information system about the event terminating the atenolol prescription, we are able through the availability time to say that on July 12 the information system was aware that the patient's atenolol prescription ended the day before, i.e. on July 11. So, the decision of interrupting the therapy cannot be from the physician, who is part of the information system, as it corresponds to the terminating event time on July 10. On the other hand, we may assume that the prescription of atenolol for patient `p2` was decided and administered immediately on June 30 as the physician (i.e., the information system) was aware of the high blood pressure of the patient.

It is worth noting that the same conclusion could not be reached if we did not consider the availability time: indeed, without availability time, we are allowed to use only transaction time, to try to estimate the moment at which data are known by the information system. In this case, transaction time for high blood pressure starts on July 02, 2008: using in a rough way this time value, we could think that the start of the therapy was decided without knowing anything about the patient's high blood pressure. It is clear that this second (wrong) interpretation could lead to a wrong evaluation of the decision making process, which seems in this case not to be based on any clinical evidence related to the patient's vital signs.

4.2.2 Querying Data with Multiple Temporal Dimensions

Let us now move to the main aspects of query languages supporting multiple temporal dimensions. In the following we will consider the temporal query language T4SQL [94], supporting all the four introduced temporal dimensions. This proposal takes into account and extends the expressiveness of well-known temporal query languages such as TSQL2 and the temporal part of SQL3 [133]. The main features of T4SQL include the following semantics: *current*, which considers only current tuples; *sequenced*, which corresponds to the homonymous SQL3 semantics; *atemporal*, which is equivalent to the SQL3 non-sequenced one; and the original *next*, which allows one to link consecutive states when evaluating a query. As a matter of fact, both *current*, *sequenced*, and *next* are special cases of the *atemporal* semantics; nevertheless, they allow one to express meaningful classes of queries in a much more compact way.

T4SQL queries receive input relations (via the `FROM` clause of a statement) with the four temporal dimensions: queries return relations with at most the four temporal dimensions. Relations without all the four temporal dimensions are prior converted to *complete*[2] relations according to either some default rules or other alternative rules, suitably defined according to the given application domain. T4SQL has both the constants and the standard temporal data types as in SQL92 and the `PERIOD` data type as in SQL3.

In the following, the only data type used to describe an instant is `DATE`; the `PERIOD` data type is defined by instants of type `DATE`, only; interval data types (i.e., representing durations) are *day* and *year-month*. These assumptions do not limit the expressiveness of the query language, which can be easily extended to manage the `TIME` and `TIMESTAMP` data types.

The values associated to a specific temporal dimension can be referenced by the following functions, only:

`VALID(T)` returns the VT of a tuple of the relation `R`;
`TRANSACTION(T)` returns the TT of a tuple of `R`;
`AVAILABLE(T)` returns the AT of a tuple of `R`;

[2] A temporal relation is complete if it has all the four temporal dimensions VT, TT, AT, and ET(i.e., ET_i and ET_t).

INITIATING_ET(T) returns ET_i for a tuple of R;
TERMINATING_ET(T) returns ET_t for a tuple of R.

The syntax of T4SQL is very close to that of TSQL, with some extensions. The (incomplete) BNF (Backus Naur Form) grammar is:

```
[SEMANTICS <sem> [ON] <dim> [[TIMESLICE] <ts_exp>]
       {, <sem> [ON] <dim> [[TIMESLICE] <ts_exp>]}]
SELECT  <sel_element_list>
  [WITH <w_exp> [AS] <dim> {, <w_exp> [AS] <dim>}]
  [TGROUPING]  [WEIGHTED]
FROM <tables>
[WHERE <cond>]
[WHEN  <t_cond>]
[GROUP BY <group_element_list>]
[HAVING <g_cond>]
<group_element_list> ::= <group_element>
        {, <group_element>}
<group_element> ::= <attribute> |
        <temp_attribute> USING <part_size>
<sem>:= ATEMPORAL | CURRENT | SEQUENCED |
        NEXT[(<duration>)] [THROUGH <att_list>]
<dim>:=VALID | TRANSACTION | AVAILABILITY |
       INITIATING_ET | TERMINATING_ET
```

4.2.2.1 The Clause SEMANTICS

T4SQL enables the user to specify different semantics for every temporal dimension, delegating the management of the temporal dimensions to the underlying DBMS as follows:

```
SEMANTICS <sem> [ON] <dim> [[TIMESLICE] <ts_exp>]
       {, <sem> [ON] <dim> [[TIMESLICE] <ts_exp>]}
```

where <sem> is the type of the required semantics and <dim> is the temporal dimension where the semantics is applied to. The tokens ON and TIMESLICE are optional and aim at increasing the readability of the query. The item <ts_exp> is a constant (p) either of type PERIOD (for dimensions VT, AT, and TT) or of type DATE (for dimensions $ET_i$and ET_t): focusing on the more important case of a period timeslice, the constant p is such that, if d is the considered temporal dimension, a generic tuple t, belonging to the relation r, is considered in the query only if $t(d) \cap p \neq \emptyset$. Additionally, the value assumed by t over the temporal dimension d is changed to $t(d) \cap p$.

A temporal dimension can be only once inside the SEMANTICS clause: one unique interpretation can be associated to every single temporal dimension.

As an example, consider the relation PatTherapy and the need of extracting data about all the patients that had a therapy with atenolol and the respective period during which the fact happened. In a relational DBMS where temporal dimensions are managed explicitly by the user, the query considering all the temporalities may look like the following:

```
SELECT    PatId, VT
FROM      PatTherapy
WHERE     Drug = 'Atenolol' AND
          END(TT) = DATE 'uc' AND
          END(AT) = DATE 'uc'
```

After having defined the suitable semantics, T4SQL manages all the temporal dimensions on behalf of the user, who does not have to explicitly consider all these temporal dimensions. The obtained code is more readable, less complex, and possibly, with a reduced number of errors. The previous query is expressed in T4SQL as follows:

```
SEMANTICS SEQUENCED ON VALID
SELECT    PatId
FROM      PatTherapy
WHERE     Drug = 'Atenolol'
```

The temporal dimension of VT is interpreted according to the use of the keyword SEQUENCED, while the temporal dimensions of TT and AT are automatically managed to consider only current and available tuples, respectively.

If instead we want to consider all the patients that assumed atenolol during June 2008, the TIMESLICE semantics helps us and the T4SQL query is the following:

```
SEMANTICS SEQUENCED ON VALID TIMESLICE
          PERIOD '[2008-06-01 - 2008-06-30]'
SELECT    PatId
FROM      PatTherapy
WHERE     Drug = 'Atenolol'
```

where the upper and lower bounds of a constant of type PERIOD are depicted by the symbols '[' and ']'.

When processing a T4SQL statement, the SEMANTICS token is processed first, well before the FROM clause, because all the statement needs to be interpreted according to the specified semantics. We now consider in detail the four types of available semantics: ATEMPORAL, CURRENT, SEQUENCED, and NEXT.

ATEMPORAL Semantics

If the ATEMPORAL semantics is adopted, the corresponding attribute(s) is dealt with as atemporal (timeless), providing the user with the highest level of freedom in managing temporal dimensions, even though any support by the system is disabled. The

ATEMPORAL semantics is exactly the same as the *non-sequenced* semantics discussed in Section 3.5 of Chapter 3.

If we want to retrieve all the patients where the therapy "Atenolol" has never been observed, the query does not require any temporal dimension, but requires to span over the entire temporal axis. The resulting T4SQL query is the following:

```
SEMANTICS ATEMPORAL ON VALID
SELECT    PatId
FROM      PatTherapy
WHERE     PatId NOT IN (
  SEMANTICS ATEMPORAL ON VALID
  SELECT    PatId
  FROM      PatTherapy
  WHERE     Drug = 'Atenolol')
```

By default, duplicates are removed, as by the DISTINCT clause of SQL.

The result of a T4SQL query is a relation with at most four temporal dimensions: one result relation may also have no temporal dimension at all. The above result relation has no VT: this comes from the assumption that, if not explicitly mentioned, the temporal dimension related to an ATEMPORAL semantics is not included in the result relation. In the considered example, the query does not suggest any function that can suitably evaluate a VT to be returned, unless explicitly defined by the user. In this latter case, a temporal dimension will be included in the result relation. Thus, the freedom of expression empowered by an ATEMPORAL query enables the user to obtain the same result as it could be obtained by using some other semantics, even though the ATEMPORAL query is a much more complex one.

CURRENT Semantics

The CURRENT semantics, applied to a temporal dimension d, considers only the tuples where the value associated to d includes the current date.

The CURRENT semantics may assume a different meaning according to the temporal data type associated to the considered temporal dimension:

- if the specified data type is a period, as for VT, TT or AT, the query considers only the tuples satisfying the condition `t(d) CONTAINS DATE 'now'` or `t(d) CONTAINS DATE 'uc'`;
- if the specified data type is a date, as for ET, the query considers all the tuples satisfying the condition `t(d) = DATE 'now'`.

As an example, we want to retrieve all the patients from PatVisit, as currently stored. The T4SQL query is:

```
SEMANTICS CURRENT ON TRANSACTION, ATEMPORAL ON VALID
  SELECT    PatId
  FROM      PatVisit
```

By the above query, considered tuples are only those whose TT contains the current date (timestamp). The result relation does not contain neither VT (it should have contained it if the semantics were not ATEMPORAL) nor TT. TT is not included because the semantics CURRENT does not save the temporal dimension, too. This behavior is due to the following considerations: i) compatibility of T4SQL with SQL92, as described next; ii) a query with a CURRENT semantics considers the current state of the database for a given temporal dimension (i.e., at the current date), and in such a situation there is no reason to associate a temporal dimension because the information is related to a specific date (DATE 'now' or DATE 'uc' depending on the considered temporal dimension).

The above query can be expressed by an ATEMPORAL semantics (the complexity of the query increases if the user has to explicitly manage all the temporal dimensions) as:

```
SEMANTICS ATEMPORAL ON TRANSACTION,
                       ATEMPORAL ON VALID
SELECT    PatId
FROM      PatVisit AS pv
WHERE     TRANSACTION(pv) CONTAINS DATE 'uc'
```

The CURRENT semantics is the only case where TIMESLICE cannot be used, as CURRENT can be seen as a TIMESLICE containing the current date, only. The above query can be rewritten as:

```
SEMANTICS ATEMPORAL ON TRANSACTION
     TIMESLICE PERIOD '[uc - uc]', ATEMPORAL ON VALID
SELECT    PatId
FROM      PatVisit
```

Thus, a query including the statement CURRENT ON VALID TIMESLICE DATE '2008-07-01' will raise an error.

SEQUENCED Semantics

The SEQUENCED semantics forces a *time point by time point* evaluation of the statement, with exactly the same semantics as in [384] . This evaluation on a given temporal dimension d considers all the tuples of relations in the FROM clause where there exists a date i belonging to all the periods of the dimension d.

As an example, we consider the semantics SEQUENCED over VT. For every date over the time axis, we select all and only those tuples where VT contains the considered date. This semantics is very useful when we want to perform a historical analysis, considering only the information valid at a given date (while changing that date).

As a difference from the ATEMPORAL and CURRENT semantics, the SEQUENCED semantics returns in the result relation the temporal dimension specified in the query. By default, the returned value is evaluated for every tuple as follows:

let $T_1, \cdots, T_n$ be the relations used to evaluate the result, d be the temporal dimension to which the SEQUENCED semantics is applied, and T_r be the result relation. For every tuple $t_r \in T_r$ and for every $t_i \in T_i$, where $i \in [1..n]$, where any t_i participates in the evaluation of the result tuple t_r, we have that $t_r(d) = \bigcap_{i=1}^{n} t_i(d)$.

In order to understand this choice, let us consider VT. Every tuple t_r, belonging to the result relation, is valid when all the tuples involved in its definition are valid. The VT of every result tuple is the intersection of the VT of the involved tuples. If, according to the current state of the database, we want to retrieve for every patient the visits whose VT overlaps the VT of the prescribed therapies, the query is:

```
SEMANTICS SEQUENCED ON VALID, CURRENT ON TRANSACTION
SELECT    SBP, DBP, PatId
FROM      PatVisit AS pv, PatTherapy AS pt
WHERE     pv.PatId = pt.PatId
```

When the query involves one relation only, the SEQUENCED semantics considers all the tuples of that relation and associates to the result tuples the value of the considered temporal dimension. If we want to retrieve the patients who had high blood pressure, considering the information that the database in its current state believed correct on July 8, 2008, the query is:

```
SEMANTICS SEQUENCED ON VALID,
  CURRENT ON TRANSACTION, ATEMPORAL ON AVAILABLE
  TIMESLICE PERIOD '[2008-07-08 - 2008-07-08]'
SELECT    PatId
FROM      PatVisit
WHERE     SBP > 150 AND DBP > 100
```

The result is a temporal relation with one explicit attribute (PatId) and one implicit attribute (VT) managed by the system. In this case, too, the query can be translated to a different one with the ATEMPORAL semantics, as follows:

```
SEMANTICS ATEMPORAL ON VALID,
  ATEMPORAL ON TRANSACTION, ATEMPORAL ON AVAILABLE
SELECT    PatId WITH VALID(pv) AS VALID
FROM      PatVisit AS pv
WHERE     SBP > 150 AND DBP > 100 AND
          TRANSACTION(pv) CONTAINS DATE 'uc' AND
          AVAILABLE(pv) CONTAINS DATE '2008-07-08'
```

The particular form of the SELECT clause will be described in Section 4.2.2.2. Generally, any query with a SEQUENCED semantics can be transformed to a different query defined by an ATEMPORAL semantics.

NEXT Semantics

The NEXT semantics enables the user to retrieve information about the same object as observed in two subsequent dates over the order of the temporal dimension. If we consider the example of patients and therapies, one may want to identify for every patient the time elapsed between two subsequent administrations of the same therapy or between two subsequent administrations of different therapies.

As main feature, the NEXT semantics considers two tuples related to the same entity and the two tuples must be subsequent in the order of the selected temporal dimension. Thus, the query in its basic form has to consider together two tuples with the same *snapshot* key (see Section 4.2.2).

The logical operation performed by the NEXT semantics is equivalent to a join between two instances of the same relation in the FROM clause: in its simpler form, this operation removes the couples of tuples that are not related to the same snapshot key. Next, the query selects the tuples where the data from the second tuple are subsequent to the data from the first tuple, according to the order of the selected temporal dimension. Due to the constraint over the snapshot key, we have a total order and, in the cases of VT, TT, and AT, the successor (if any) is unique, if we exclude the adoption of the ATEMPORAL semantics for some among the other dimensions. Moreover, in the case of ET_i and ET_t, we deal with temporal values of DATE type (instantaneous), while for the other dimensions we deal with values of type PERIOD.

By default, the NEXT semantics refers to the entire temporal axis. The query starts looking for the successor from the final date of the interval of the considered tuple till the very last instant representable by the system. Sometimes one may want to limit the upper bound of the interval considered by the query: to do that, a suitable integer parameter of the NEXT semantics identifies the duration of the period within which the successor of the tuple must be found. The complete syntax for the NEXT semantics is defined as:

```
NEXT[(<duration>)] [THROUGH <att_list] [ON]
      <dimension> [[TIMESLICE] <ts_exp>]
```

where <duration> is the width of the temporal interval where the successor must be found. The integer value refers to the smallest granularity of the considered temporal data type: for instance, for the VT of type PERIOD, the integer parameter has the day granularity. <dimension> is the dimension where the semantics must be applied to.

The interval over which the query evaluation looks for the successor tuple is [e, e+dur+1], where e is the final instant of the period associated to the considered temporal dimension for PERIOD data types (or the instant associated to the considered temporal dimension for DATE data types) and dur is the given duration.

Sometimes, it could be a strong limitation to consider next tuples only by considering the snapshot key: in the considered scenario, for example, it could be interesting to query the database not only on successive administrations of the same drug to patients but also on successive therapies to patients, no matter what the administered

drug is. The clause THROUGH allows the user to explicitly set the attributes to consider when joining subsequent tuples in the evaluation of the query.

Example 4.2. Starting from the PatTherapy relation, we want to retrieve for every patient the period elapsed between two subsequent administrations disregarding the fact that therapies were related to a single drug. The corresponding T4SQL query is:

```
SEMANTICS NEXT THROUGH PatId ON VALID
SELECT    PatId, (BEGIN(VALID(NEXT(t)))-END(VALID(t))) DAY
FROM      PatTherapy AS t
```

In the SELECT clause of the example, the function NEXT has the parameter defined by a tuple variable (t), to reference the successor tuple of t, as obtained by the NEXT semantics. Thus, the BEGIN(VALID(NEXT(t))) refers to the lower bound of the VT for the successor tuple.

Example 4.3. We want to retrieve all the patients who had a therapy with 'Atenolol' moving from a dosage of 50 mg/day to a dosage of 70mg/day. The query in the T4SQL query language is the following:

```
SEMANTICS NEXT(0) ON VALID
SELECT    PatId
FROM      PatTherapy AS t
WHERE     t. Drug = 'Atenolol' AND
          t.DailyDose = 50 AND t.Unit = 'mg' AND
          NEXT(t).DailyDose = 70 AND
          NEXT(t).Unit = 'mg'
```

Temporal relations obtained by applying the NEXT semantics to the temporal dimension d, do not include that temporal dimension d. In fact T4SQL does not want to associate any temporal dimension to an object obtained starting from two tuples with disjoint values. The user can always specify by the WITH clause, described below, how to evaluate the temporal dimension to be included in the result relation.

Default Values

If no information or no temporal dimension is specified for the SEMANTICS clause, default values apply. Compatibility of T4SQL towards SQL92 drives the choices for default values.

In T4SQL any atemporal query[3], performed over the temporal relation corresponding to the atemporal query, produces an identical result as a query performed over the atemporal relation by a non-temporal DBMS. This definition is equivalent

[3] A query is named *atemporal* or *snapshot* if no construct is defined to manage any temporal dimension.

to the definition of *upward compatibility* as in [384]. Obviously, a query compatible with SQL92 cannot include the SEMANTICS clause.

In T4SQL, a SQL92-like query over a temporal relation evaluates only the information known to the DBMS and valid at query execution time: the query returns an atemporal relation evaluated according to the semantics of SQL92. This requires that the default semantics for VT and TT is CURRENT: in fact, the query considers only the information associated to the current state of the database with the same semantics as in SQL92. Additionally, no temporal dimension is returned. Due to similar reasons, the default semantics for AT is CURRENT.

A more accurate analysis is needed for ET. T4SQL cannot assume as default values a semantics CURRENT, or T4SQL will consider only the tuples where starting date, final date, and the current date coincide. As an example, we consider a tuple whose VT is [2009-01-01, 2009-12-31], and the current date is June 1, 2009: we also assume to have a co-active relation. In this case, a tuple valid at the current date will not be considered as both boundaries of ET differ from the current date 2009-06-01. Due to these reasons, the default semantics for ET is ATEMPORAL, as this does not influence the selection of the tuple and does not return the temporal dimension in the result relation.

Example 4.4. From the PatTherapy relation, we want to retrieve patients who are taking 'Lisinopril'. The SQL92 query has no condition over temporal aspects and it is:

```
SELECT    PatId
FROM      PatTherapy
WHERE     Drug = 'Lisinopril'
```

The result is an atemporal relation considering only information in the database and valid at the current time. The equivalent T4SQL code is:

```
SEMANTICS CURRENT ON VALID,
          CURRENT ON TRANSACTION,
          CURRENT ON AVAILABLE,
          ATEMPORAL ON INITIATING_ET,
          ATEMPORAL ON TERMINATING_ET
SELECT    PatId
FROM      PatTherapy
WHERE     Drug = 'Lisinopril'
```

4.2.2.2 The Clause SELECT

The SELECT clause of SQL92 performs a projection operation over the attributes to be included into the result relation. T4SQL manages both temporal and atemporal relations and thus it has to manage explicit attributes (i.e., those included in the SELECT clause), as well as temporal dimensions.

T4SQL introduces the token WITH in the SELECT clause, to separate the specification of explicit attributes from that of implicit attributes. Any statement before the token WITH is evaluated according to the SQL92 semantics for projection: any statement following the token WITH computes the temporal dimension(s) to be included in the result relation. The syntax for the SELECT clause in T4SQL is:

```
SELECT  <sel_element_list>
 [WITH <w_exp> [AS] <dim> {, <w_exp> [AS] <dim>}]
```

where <w_exp> computes a period (if the temporal dimension is associated to a PERIOD data type) or a date (if the temporal dimension is associated to a DATE data type), and <dim> is a temporal dimension. The optional token AS increases the readability of the code.

If no temporal dimension is specified for the result relation, the following default temporal dimensions are applied:

- ATEMPORAL Semantics: the temporal dimension is not included in the result relation;
- CURRENT Semantics: the temporal dimension is not included in the result relation;
- SEQUENCED Semantics: the temporal dimension is included in the result relation and included values are the intersection of the temporal attributes of the tuples involved in determining the result;
- NEXT Semantics: the temporal dimension is not included in the result relation.

Example 4.5. We want to retrieve the patients who had high diastolic blood pressure (i.e., more than 100 mmHg), received the drug *'Lisinopril'*, and had a measure of normal diastolic blood pressure within 5 days from the beginning of the therapy. Every element in the result relation must come with the VT, which is included within the beginning of the high diastolic blood pressure and the end of the therapy. The resulting T4SQL query is:

```
SEMANTICS SEQUENCED ON VALID
SELECT   PatId WITH PERIOD (BEGIN(VALID(pv1)),
           END(VALID(pt))) AS VALID
FROM     PatVisit AS pv1, PatVisit AS pv2, PatTherapy AS pt
WHERE    pv1.DBP > 100 AND pv2 < 100 AND
         pt. Drug = 'Lisinopril' AND
         pp.PatId = pt.PatId AND
         VALID(pv1) BEFORE VALID(pv2) AND
         (END(VALID(pt2)) - BEGIN(VALID(pt))) DAY
                              < INTERVAL '5' DAY
```

Here, the result relation has the VT attribute. The value of this temporal dimension, however, does not come from the intersection of the VT of the tuples included into the result, as the token WITH modifies the included temporal dimension.

Coalescing

In a temporal relation, all the tuples have to satisfy the *snapshot key* constraint. As the projection of the clause SELECT may produce a temporal relation violating the above constraint, we may thus have two tuples which have the same values for atemporal attributes and intersecting temporal dimensions (if any). To cope with this situation, the *coalescing* operator fuses the tuples with overlapping values of temporal dimensions and with the atemporal attributes having the same corresponding values [35, 132].

4.2.2.3 The Clause FROM

The FROM clause of SQL92 identifies the relations used to find the attributes and/or to perform join operations. In SQL92 several JOIN criteria can be defined: INNER join, which is the default value, LEFT OUTER, RIGHT OUTER and FULL OUTER joins. The token ON specifies the selection conditions of the FROM clause. T4SQL adopts the same FROM clause as SQL92. Additionally, the token JOIN of SQL92 has been replaced by the token TJOIN (*temporal join*), when some join conditions are temporal.

Relations in T4SQL are filtered to consider only the information needed for the specified interpretations. Thus, user-defined conditions are augmented by those coming from the semantics applied to every temporal dimension. According to the considered semantics, T4SQL behaves as follows:

- if the CURRENT semantics is specified, T4SQL considers only the tuples whose temporal dimension includes the current date;
- if the SEQUENCED semantics is specified, T4SQL considers only the tuples from the relations specified by the clause FROM with overlapping temporal dimensions;
- if the NEXT semantics is specified, T4SQL considers the tuples having a successor and the successor itself, only.

Example 4.6. We want to select, according to the current state of the database, all the patients, reporting also their vital signs, who had a visit during the assumption of a therapy that ended more than 10 days after the visit. The T4SQL query is:

```
SEMANTICS SEQUENCED ON VALID
SELECT    PatId, Drug, DBP, SBP
FROM      PatTherapy AS pt TJOIN
          PatVisit AS pv ON
          (pt.PatId = pv.PatId AND
          (END(VALID(pt)) - BEGIN(VALID(pv))) DAY >
            INTERVAL '10' DAY )
```

4.2.2.4 The Clauses WHERE and WHEN

In SQL92, the WHERE clause evaluates selection predicates over tuples from the relations in the FROM clause. The WHERE clause of T4SQL extends the SQL92 clause possibly including some temporal conditions. As temporal conditions may turn out to be very complex, the user can optionally separate temporal conditions from atemporal conditions by using the WHEN clause. The semantics of WHEN is very similar to that of WHERE, but WHEN includes temporal conditions only. Consider again the query of Example 4.6. It can be defined by using the clause WHEN as follows:

```
SEMANTICS SEQUENCED ON VALID
SELECT    PatId, Drug, DBP, SBP
FROM      PatTherapy AS pt, PatVisit AS pv
WHERE     pt.PatId = pv.PatId
WHEN      (END(VALID(pt)) - BEGIN(VALID(pv))) DAY >
            INTERVAL '10' DAY
```

The WHEN clause can be replaced with an equivalent statement in the WHERE clause, with a reduced readability.

4.2.2.5 The Clauses GROUP BY and HAVING

In a temporal query language, the GROUP BY clause can be used to implement a *temporal grouping*, rather than a punctual grouping over atemporal attributes. The selection of the tuples to be grouped together (we have one group for every partitioning element) is performed according to the value of the considered temporal attribute(s) of the tuple, according to the periods associated with the partition, and according to the temporal comparison operator used in the temporal grouping.

Let P be a period associated with a particular element of the partition, P_t be the value (that is, time period) assumed by the tuple t on the considered temporal attribute and Op be the temporal comparison operator. The tuple will belong to the group associated with the considered element of the partition if the condition $P\ Op\ P_t$ holds. To limit the number of new terms specifying the comparison operator for the temporal grouping, the default value is the INTERSECT operator. Let Op be the INTERSECT operator. For every tuple t and every element of the partition, if P_t INTERSECT $P = true$, then t belongs to the group associated with the element of the partition and the value of its temporal attribute (with respect to the considered group) is $P \cap P_t$.

The result of temporal grouping can be explained as follows. The time period P_t associated with the temporal attribute of the tuple can be viewed as a set of dates. Whenever P_t and the time period P of the element of the partition satisfy the requested condition, e.g., whenever they intersect, the tuple belongs to the group induced by the element of the partition and the time period associated with the temporal attribute is restricted to the set of dates that belong to P.

The token USING distinguishes between a classic (atemporal) grouping and a temporal grouping. Implemented temporal partitions are SECOND, MINUTE, HOUR, DAY, MONTH and YEAR. For instance, a temporal grouping can be represented by "GROUP BY <temp> USING MONTH", where <temp> refers to a temporal dimension of a table in the FROM clause.

In case of temporal grouping, it is only possible to use a constant value of type PERIOD, representing a specific partition, to identify a group. For homogeneity with SQL92, in T4SQL the grouping attributes could be included in the clause SELECT, while the temporal grouping is performed by the token TGROUPING(...). This token returns as many attributes as the parameters, possibly renamed as specified by the user.

T4SQL adopts the same semantics as SQL92 for grouping. For the functions MAX, MIN, AVG and SUM the token WEIGHTED is introduced by T4SQL. The token WEIGHTED must precede the aggregation function it is applied to: the token computes the weighted function over the dimension of the temporal period of every tuple. Let us assume that $t_1 \dots t_n$ are the tuples belonging to the group G, $t_1(d) \dots t_n(d)$ are the values of the tuple over the temporal attribute d according to which we performed the temporal grouping, and $d(G)$ is the duration of the considered grouping partition. Moreover, let $t_1(a) \dots t_n(a)$ be the values of the tuples for attribute a which is the parameter for the weighted aggregate function. The resulting function is computed as follows:

- WEIGHTED MAX(a) $= max_{i=1}^{n}\{t_i(a) * \frac{duration(t_i(d))}{d(G)}\}$;
- WEIGHTED MIN(a) $= min_{i=1}^{n}\{t_i(a) * \frac{duration(t_i(d))}{d(G)}\}$;
- WEIGHTED SUM(a) $= \sum_{i=1}^{n}(t_i(a) * \frac{duration(t_i(d))}{d(G)})$;
- WEIGHTED AVG(a) $= \frac{\sum_{i=1}^{n}(t_i(a) * \frac{duration(t_i(d))}{d(G)})}{n}$.

The HAVING clause of T4SQL is exactly the same as that of SQL92, enhanced by the possibility of using weighted functions.

Example 4.7. Let us assume that we want to retrieve for every year the average duration of prescribed therapies. The corresponding T4SQL query is the following:

```
SELECT    TGROUPING(VALID(t) AS YearPeriod),
          AVG(CAST(INTERVAL(VALID(t) DAY))
          AS INTEGER)
FROM      PatTherapy AS t
GROUP BY  VALID(t) USING YEAR
```

Example 4.8. We want to compute from the table PatVisit the average level per month of DBP for each patient, returning only data for patients having assumed a (weighted) average quantity of atenolol per month more than 2 mg per day. The T4SQL query is:

```
SEMANTICS SEQUENCED ON VALID
SELECT    PatId, AVG(DBP),
```

```
            TGROUPING(VALID(v) AS VALID)
FROM        PatVisit AS v, PatTherapy AS t
WHERE       Drug =  'Atenolol'
GROUP BY    PatId, VALID(v) USING MONTH
HAVING      WEIGHTED AVG(DailyDose) > 2
```

It is worth noting in this case that the weighted average allows us to consider the real average assumption of atenolol per day (without the weighted average any daily quantity of atenolol is considered in the temporal grouping and in the average operation, without considering how many days the patient assumed the drug).

4.3 Granularity and Indeterminacy in Clinical Databases

Since the beginning of 1990s, results from research in temporal relational databases have been applied and extended for the management of temporal clinical data [111, 112, 123, 124, 80, 82, 113]: particularly, several work in the medical informatics field focused on the issue of temporal granularity and indeterminacy in modeling and querying clinical data. The proposed research considered suitable (and general) extensions both to the relational model and query languages and to object-oriented models and query languages. In the following we will describe two well known methodologies adopting the relational model and the object-oriented one, respectively. In particular, the object-oriented approach will be treated as a complete case study, considering modeling issues, querying issues, some technological aspects, the real world clinical domain considered for the application, and the web-based architecture of the designed and implemented system.

4.3.1 A Temporally Extended Clinical Relational System

One of the main problems faced for temporal relational clinical databases is the seamless management of instant- and interval- valid times with different granularities; in several temporal database systems, valid times are homogeneous both in granularity and in instant/interval reference for all the tuples of a given relation. This is a limitation in the clinical domain: in a relation containing descriptions of pathologies, for example, instantaneous tuples (e.g. "cerebral stroke at 21:23 of May 16th, 1997") and interval-based tuples (e.g. "vision loss on February 13th 1997 from 18:35:15 to 18:45:28") must co-exist.

A widely known and general temporal data model proposed for the management of temporal clinical data is Chronus, proposed by Das and Musen in [111], and successively used in several projects dealing with the integrated management of data and knowledge in the clinical context [113, 327]. The main problem faced here is that clinical information can come with heterogeneous temporal data and with indeterminacy. Chronus distinguishes two kinds of medical temporal data: instantaneous

data represent *events*, while interval-based data represent *states*. Instantaneous data and interval data are stored into *history tables*. For managing the indeterminacy of events and states, Das and Musen [111] define four different types of relational tuples: *event*, *start*, *body*, and *stop* tuples. Moreover, an extension is proposed for the relational algebra, to manage temporal information and temporal relational operations. As an example, let us consider the Chronus database depicted in Figure 4.1, which represents a simple clinical database containing some basic demographic data for the patients, some information about their blood pressures and heart rate during some visits, and some information on their therapies.

As shown in the figure, any Chronus relational table contains three extra-attributes, which have a predefined meaning: the attributes *Start_time* and *Stop_time* contain the starting timestamp and the ending timestamp for the fact represented into the tuple; moreover, tuples are typed according to the attribute *Type*, which distinguish instantaneous facts and interval-based facts.

As an example, the first tuple of table *Visit* in Figure 4.1b refers to the fact that patient Smith had values 130 and 90 for systolic blood pressure (SBP) and diastolic blood pressure (DBP), respectively, at a time point between 11 a.m. and noon of June 6, 1998: more precisely, the value *event* for the attribute *Type* says that the tuple represents an instantaneous fact, while the different values for *Start_time* and *Stop_time* indicate that the temporal location of the represented fact is not given at a precise timepoint, but is within the two given upper and lower bounds. As for states, three tuples of a Chronus table are needed: two tuple, having the values *start event* and *stop event*, respectively, for the attribute *Type* represent the indeterminacy of starting and ending points of the interval of validity of the considered state; a further tuple, with value *body* for the attribute *Type*, is needed to represent when the state is valid for sure. It is worth noting that in Chronus the tuple interval expressed through the attributes *Start_time* and *Stop_time* assumes different meanings according to the value of the attribute *Type*. For *event*, *start*, and *stop* tuples, the tuple interval represents an interval of uncertainty (IOU), i.e., the interval within which the represented instantaneous event occurs: such indeterminacy could derive, for example, from the need to represent data given at different granularities. For *body* tuples, the tuple interval represents the interval of certainty (IOC), i.e. the interval over which the represented state holds. As a special case, when clinical information is provided without indeterminacy, a state is represented through a single *body* tuple (as for the tuples of relation *Patient* and for the first tuple of relation *Therapy* in Figure 4.1c), while an event is represented through an *event* tuple having the same values for both attributes *Start_time* and *Stop_time*.

The Chronus data model is also completed by an algebra, called *historical algebra*, which allows one to compare intervals and instants, to perform operations on time intervals and instants, and on the related tuple [111]. Among the operations supported by the proposed algebra, we mention here temporal selection and projection, catenation (i.e, coalescing), and different kinds of temporal joins. Further work on Chronus has been done to improve the representation of temporal uncertainty of clinical information by associating a probability distribution function to each IOU describing the possible temporal location of a time point [284].

Start_time	Stop_time	Type	PatId	Name
97/11/10/11/00	99/9/25/11/00	body	SM1	Smith
95/3/4/12/00	99/12/6/17/00	body	RS1	Rossi
97/6/11/11/00	97/6/15/11/00	body	HB3	Hubbard

a) the relation *Patient*

Start_time	Stop_time	Type	PatId	SBP	DBP
98/6/6/11/00	98/6/6/11/59	event	SM1	130	90
98/11/10/10/00	98/11/10/10/09	event	SM1	120	70
99/7/20/12/45	99/7/20/12/59	event	RS1	150	110
97/6/12/00/00	97/6/12/11/59	event	HB3	80	60

b) the relation *Visit*

Start_time	Stop_time	Type	PatId	drug	dosage
98/5/12/00/00	98/5/22/23/59	body	SM1	thiazide diuretics	30 mg once a day
99/7/20/00/00	99/7/20/23/59	start event	RS1	aspirin	120 mg daily
99/7/21/00/00	99/7/31/23/59	body	RS1	aspirin	120 mg daily
99/8/1/00/00	99/8/31/23/59	stop event	RS1	aspirin	120 mg daily
99/7/21/16/00	99/7/21/16/59	start event	RS1	heparin	18 units/kg/hr
99/7/21/17/00	99/7/26/16/59	body	RS1	heparin	18 units/kg/hr
99/7/26/17/00	99/7/26/17/59	stop event	RS1	heparin	18 units/kg/hr

c) the relation *Therapy*

Fig. 4.1 Relations of the example clinical database.

In general, several needs have been identified in designing a relational database system for temporal clinical data:

- compatibility with the flat relational model and with SQL; a large amount of clinical data is stored in conventional relational databases. Temporal queries on these data have to be performed. Temporal query languages and related data models have to consider also flat data, containing some user-defined temporal dimension.
- addition and enhancement of some specific clauses, functions and predicates of temporal query languages. We often need to identify the clinical state of patients: to this end it is necessary to query temporal clinical data by specifying complex conditions on data based on temporal proximity, temporal order, and complex temporal relationships on collected data. For example, it is important to have the capability to observe data by a window, having a predefined duration, moving on the time axis (e.g. “find the patients having had the systolic blood pressure below 110 mmHg for ten days”).

With respect to these needs and to the requirements related to the clinical domain, there are two main approaches: the first one is based on the proposal of an extended SQL syntax, which is compatible also for querying standard SQL tables; the second one is related to the design and implementation of software modules for providing physicians with a more simple temporal query language, which is in turn based on SQL.

According to the first approach, Das and Musen propose an extension of SQL, based on the algebra proposed for the Chronus data model, called Time Line SQL (TLSQL) [111]. To give an example of how to express temporal queries by TLSQL, let us consider the following scenario.

Example 4.9. Considering tables *Patient*, *Visit*, and *Therapy* depicted in Figure 4.1, we want to retrieve all the patients' visits that occurred during a therapy lasting for sure more than 7 days. As result of the query, the system must return, for each patient's visit, the surname of the patient and the systolic blood pressure measured in that visit. The query expressed by TLSQL is:

```
SELECT Name, SBP
FROM   Visit V, Therapy T, Patient P
WHEN   [V.Start_time, V.Stop_time]  DURING
                [T.Start_time, T.Stop_time] AND
       DURATION([T.Start_time, T.Stop_time]) > 7 DAYS AND
       V.Type = "event" AND T.Type = "body"
WHERE  V.PatId = T.PatId AND P.PatId = T.PatId
```

In the above query we may observe that TLSQL uses the square brackets as interval constructor, to make the comparison between temporal dimensions more compact. On the other side, as the query asks for sure relationships, a suitable condition is required to deal only with tuples of body type (i.e., representing an IOC).

4.3.2 An Object-Oriented Approach for Temporal Clinical Data

Clinical data is a good example of complex information, that can be suitably modeled and managed by object-oriented technologies [217, 82]. Indeed, medical records are complex documents, composed by different kinds of multimedia data, often involving strong temporal aspects: unstructured and structured text, coded data, numerical parameters, static images (radiographies, CTs), dynamic images (angiographic films, echographies), sounds (cardiac phonoscopies), bio-signals (ECGs, EEGs), graphical and vocal comments from clinical reports [51, 308]. Considering all this heterogeneous data, a lot of information involves temporal aspects: to monitor patient conditions, for example, several parameters, e.g., SBP, DBP, HR, are periodically monitored and historical data, related to past therapies, symptoms and so on, has to be considered too [387, 82, 292]. Combi et al. [82] extended an object-oriented data model and the related query language to deal with temporal clinical data: Granular Clinical History - Object SQL (GCH-OSQL) was proposed as a query language for temporal clinical databases, taking into account different and mixed temporal granularities. Goralwalla and colleagues adapted an existent object database model to the management of time-oriented data, and have applied it to the modeling of pharmacoeconomic clinical trials [158]. The broad set of types supported by the adopted object data model enables, for example, a modeling of

branching timelines, corresponding, for instance, to the evaluation of different pharmacological treatments. In the following, we will consider some details of GCH-OSQL and of the related object-oriented temporal data model GCH-OODM.

4.3.2.1 The Temporal Data Model GCH-OODM

GCH-OODM is an object-oriented data model, extended to consider and manage the valid time of information. The model focuses on the capability of managing valid time expressed by different and mixed granularities and/or with indeterminacy.

GCH-OODM supports the main features of object-oriented data models, as applied to databases: besides the already mentioned object identity and encapsulation, the data model supports also single inheritance, polymorphism, management of complex objects, persistence (for further details on these basic concepts, see Chapter 3 and the related bibliography).

GCH-OODM Types

Besides the usual types (`string`, `int`, `real`) GCH-OODM uses some collection types: `set<t>`, `bag<t>`, `list<t>`, `array<t>`. Each of these types is a type generator, in respect with the type `t` in the angle brackets. GCH-OODM uses, to model the temporal dimension of information, some predefined data types: the type hierarchy `el_time`, `instant`, `duration`, `interval`; the collection type `t_o_set`, and some of its specializations, by which set of temporal objects are modeled. 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/indeterminacies.

The basic time domain **T**, called also *time axis*, is isomorphic to the natural numbers with the usual ordering relation $\leq$. As discussed in Section 2.2 of Chapter 2, a granularity is defined as a mapping from an index set, i.e. the set of points related to the given granularity, to the powerset of the time domain. The considered index sets are isomorphic to integers; different notations for different index sets based on that for dates and durations, e.g. `YY/MM/DD/HH:Mi:SS` for seconds, `YY/MM/DD` for days, n_1 `y` n_2 `m` for durations expressed using months and years.

The set *Gran* of granularity mappings is related to granularities of the Gregorian calendar (years, months, days, hours, seconds). Granularity mappings consider granularities for both anchored and unanchored time spans [159, 26, 160]: for example, the granularity of months can be used for expressing a certain period in a year (*October, 1999*), as well as for expressing a duration (*for three months*). In this direction, the mappings Y, M, D ... represent the usual granularities of the Gregorian calendar (they manage leap years, months with 28, 29, 30, or 31 days, and so on). On the other side, *mean_Y*, *mean_M*, ... provide regular mappings, that will be used in modeling duration, i.e. unanchored time spans, based on the (astronomical)

mean length of a year. To identify the *i*th granule of a granularity, we will use the symbols ⟨.⟩.

The type `el_time` allows one to model time points on the basic time axis, named elementary instants. Each elementary instant is the basic unit of time supported by the temporal DBMS. By the type `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 type `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 by the adoption of concepts like the mean month - and also to compute sum and difference operations on time points/time spans. For clarity reasons two different formats are used, to specify time points, i.e. anchored time spans, and distances between time points, i.e. unanchored time spans. The calendric notation `YY/MM/DD/HH:Mi:SS` allows the specification of a time point. The notation `Y yy M mm D dd H hh Mi min S ss` is used 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 `98/6/6/0:0:0`, for example, identifies the first second of June 6, 1998; the time span `6 min 32 ss` identifies a duration lasting 6 minutes and 32 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 presence of different granularities/indeterminacies leads to manage relations between intervals possibly having, besides the two logical values *True* or *False*, a logical value *Undefined*. It is not always possible to establish with certainty the truth or the falsehood of relations existing between intervals. Let us consider the two sentences "In July 1998 the patient suffered from headache for eight days", and "In July 1998 the patient had fever for 17 days". While we can affirm for sure that for the patient the fever lasted more than the headache, we cannot answer with *True* or *False* to the question whether the patient suffered from headache before having fever. Both these answers could be wrong, because we haven't enough information (which are the starting and the ending day of the two symptoms).

GCH-OODM uses a three-valued logic, in which the values `T`: *True*, `F`: *False*, and `U`: *Undefined* are 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 [295]. In comparison with the Kleene's logic, the new logical connectives `T()`, `U()` and `F()` 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 type `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).

Table 4.3 Truth table for the formula A AND B

A	B	(A AND B)
T	T	T
F	T	F
U	T	U
T	F	F
F	F	F
U	F	F
T	U	U
F	U	F
U	U	U

Table 4.4 Truth table for formulas NOTA and T(A)

A	(NOTA)	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`.

Instants, durations, and intervals

An interval, expressed in a heterogeneous way like in examples of the previous section, is represented by its starting instant, duration and ending instant. Obviously some constraints exist among the values of starting instant, ending instant and duration of an interval: if we are specifying instants and duration at different granularities or with indeterminacy, given the values of two of the three entities characterizing an interval, the value of the third entity depends on the granularities/indeterminacy of both the given values. This is the reason for which, only using both starting and ending instants and duration, it is possible to express an interval with different and heterogeneous granularities or with indeterminacy.

Instants and duration, expressed at different granularities/indeterminacy, are based on a discrete time axis, where points, named elementary instants, are represented by the finest time unit considered by the model. In our data model that of seconds is the unit of measure of the time axis. It is also possible to define duration, having values of one or more orders of magnitude lower than seconds (by the symbol ϵ), and unknown (by the label `unknown`).

The type *instant* allows us to represent a time point, identified either by the granule, i.e. a set of contiguous chronons, containing it or by the period on the time axis containing it. This type 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, while the period may be specified by two time points, i.e. the upper and the lower bound of the period, by the format <YY/MM/DD/HH:Mi:SS, YY/MM/DD/HH:Mi:SS>, if we have to model explicit indeterminacy. The instant 98/6/6, for example, may coincide with anyone of the time points included between the two bounds 98/6/6/0:0:0 and 98/6/6/23:59:59, represented by two objects of el_time type: the notation 98/6/6 will be equivalent to <98/6/6/0:0:0, 98/6/6/23:59:59>. The instant <98/7/12/12:30:0, 98/7/12/12:36:59> specifies a time point between 12:30 and 12:36 of July 12, 1998.

The type *duration* allows us to model a generic duration, specified at arbitrary granularity. This type 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., 7 yy, 2 yy 3 mm 23 dd. The duration 13 dd, for example, stands for a time span between 13 dd 0 hh 0 min 0 ss and 13 dd 23 hh 59 min 59 ss. A duration may also be expressed by specifying the lower and upper distances, e.g. <23 dd 14 hh 6 min 23 ss, 24 dd 8 hh 51 min 12 ss>, for explicit indeterminacy. Suitable methods allow the expression of relations and of operations, like sum or differences, on instances of the types instant and duration.

A generic interval, i.e. a set of contiguous time points, is modeled by the type interval. The methods *start()*, *end()* and *dur()* allow us to identify, respectively, the starting instant, the ending instant and the duration of the interval. Suitable methods of the type interval allow us to establish temporal relations between two intervals, specified by 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 or in relations based on the location of intervals on time axis and duration-related relations [82, 88]. Using methods *start()*, *end()*, and *dur()* is not redundant to identify an interval: e.g., the interval *x* having the methods *x.start()* and *x.end()* returning respectively 98/7/7 and 98/7/9, could have the method *x.dur()* returning <1 dd 0 hh 1 ss, 2 dd 23 hh 59 min 59 ss> (i.e. a duration between one day and one second and three days less one second) or 48 hh 0 min 0 ss. This last interval, having the duration (48 hours) specified at a granularity level (seconds) finer than that used in specifying starting (ending) instant, cannot be expressed, for example, in TSQL2 [387]. To explicitly specify an interval *x*, the following notations have been introduced:

notation 1. ⟨YY⟩, or ⟨YY/MM⟩, or ⟨YY/MM/DD⟩, and so on, when the interval *x* is a granule of the Calendar, e.g. ⟨1994/10⟩; this notation is used to model intervals given by sentences like "the year 2000", "January '03". By these different notations we refer to intervals given as granules at one of the granularities of the Gregorian Calendar.

notation 2. ⟨x.start(), x.end()⟩ when starting and ending instants are given, e.g. ⟨98/6/6, 99/3/12/13⟩; this notation is used to model intervals given by "from . . . to

... " sentences. This is the usual way to express intervals in temporal databases (but using the same granularity both for the starting instant and for the ending one).
notation 3. ⟨x.start(), x.dur()⟩ when starting instant and duration are given, e.g. ⟨98/6/6, 3 h⟩; this notation is used to model intervals given by "from ... for ... " sentences.
notation 4. ⟨x.dur(), x.end()⟩ when ending instant and duration are given, e.g. ⟨33 h, 96/10⟩; this notation is used to model intervals given by "for ... to ... " sentences.
notation 5. ⟨in, x.dur()⟩, where in is a granule; this notation allows one to express intervals given by "in ... for ...", e.g., ⟨⟨98/6⟩, 7mi33s⟩.
notation 6. ⟨x.start(), x.dur(), x.end()⟩ when both starting instant, duration, and ending instant are given, e.g. ⟨97/8/9, <5 h 5 mi 2 s, 24 h 8 mi 3 s>, 97/8/10⟩ or ⟨<96/4/3/12/30/10, 96/4/3/23/30/0>, <4 h 6 mi 3 s, 24 h 35 mi 2 s>, 96/4/4⟩; this is the more general notation, allowing one to express also all the intervals expressible by the previous notations.

Some constraints and relations exist between the starting instant, the ending one, and the duration of an interval: for example, given an interval by specifying its starting instant and duration (notation 3.), the ending instant can be computed by adding the given duration to the starting instant [88].

Temporal and atemporal types

GCH-OODM distinguishes *temporal types* and *atemporal types*. Objects instances of temporal types (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 time during which the information is true in the modeled world. For example, the type modeling the concept of pathology (or therapy) has to consider the interval, during which the pathology was present (or the therapy was administered). The method *validInterval()* returns the interval of validity of an object. By the valid interval it is possible to verify temporal relations between objects instances of temporal types. Temporal objects may have many temporal properties; these properties, defined by suitable methods, are represented in their turn by temporal objects, having their own valid interval.

Objects instances of atemporal types (hereinafter atemporal objects) model information, not having an associated temporal dimension. By these objects we are able to represent information, for which the temporal dimension is not interesting (let us think of an object modeling demographic data of a patient, about which we do not want to record the history: address, profession, ...). An atemporal object can, however, have properties represented by temporal objects. For example, the object modeling a hospital division, without any valid interval, may have a property related to the history of the heads of the division, modeled by temporal objects.

Both in temporal and atemporal types we distinguish, then, (a) *temporal methods*, modeling temporal features, returning temporal objects, and (b) *atemporal methods*

returning atemporal objects. In the following some examples are given both for temporal and atemporal types and for temporal and atemporal methods.

In GCH-OODM temporal properties are modeled by temporal objects, which can be composed by a set of temporal objects. Several temporal constraints can be defined by GCH-OODM. It is possible, for example, to model the constraints on the valid time of objects and properties: being `o.p()` an object modeling a temporal property of a temporal object `o`, the following relation must hold:

$T(o.p().validInterval().IN(o.validInterval()))$

In GCH-OODM the temporal properties of a temporal object are not constrained like the above defined one. GCH-OODM allows us to model many other constraints existing between the valid interval of an object and the valid interval of a temporal property. Let us think, for example, of a temporal property $o.prev()$ of a temporal object o; $o.prev()$ is an object having a valid interval that must precede the valid interval of the object o. In this case the following condition holds:

$T(o.validInterval().AFTER(o.prev().validInterval()))$

For this kind of constraints that may exist between temporal objects, GCH-OODM allows us to make explicit the existence of constraints, during the design of the methods of temporal types for the considered database.

The predefined type `t_o_set` (temporal_object_set) allows the construction and the management of sets of temporal objects. To the instances of the type `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 type `t_o_set`, characterized also by some atemporal features. Let us consider for example the following methods, related to an instance I of the type `t_o_set`. Let p, q be two logical expressions, x and y 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 expression p. The method *OCCURS(p)* allows us to establish if in the set I there is an object satisfying the condition p.

$I.OCCURS(p) \equiv I.subset(p) \neq \emptyset$

The method *CONTEMPORARY(p, q, X)* allows us to establish if in the set I there are two temporal objects, satisfying, respectively, the logical expressions p and q, and having the valid intervals equal at a predefined granularity X.

$I.CONTEMPORARY(p,q,X) \equiv \exists x \in I.subset(p),$
$\exists y \in I.subset(q)(x.valid\_time().CONTEMPORARY(y.valid\_time(),X))$

Several specializations of the type `t_o_set` can be defined, to manage only some types of temporal objects. This is the usual way to specialize types defined by type generators: `t_o_set<c>` is a specialization of `t_o_set<c'>` iff the temporal type `c` is a specialization of the temporal type `c`. We can specialize in an orthogonal way the type `t_o_set` to consider also temporal constraints among managed temporal objects. Some specializations of the type `t_o_set` allow the management of particular features of sets of temporal objects. For example, in GCH-OODM it is possible

to model temporal properties in a way similar to what is done by time-varying properties in [430], by constraining the temporal objects representing properties to compose a temporal sequence of temporally non-intersecting elements.

Further specializations allow us to verify some logical consistency of the stored information. In a set of diagnoses related to the same patient, for example, there cannot exist two diagnoses having two overlapping valid intervals, and contradicting or implying one another.

The example database

To show both the modeling features of GCH-OODM and the query capabilities of GCH-OSQL, we will consider in the following the database schema represented in Figure 4.2 and the database instance depicted in Figure 4.3. The database schema, obviously far from the real clinical data complexity, considers both temporal and atemporal types. A hierarchy of types is described, using a UML-like graphical syntax. It refers to a clinical database, where data about patients are stored: data are related to the therapies prescribed for a patient and to the vital signs (e.g., blood pressures) collected during follow up visits. The atemporal type `patient` has two temporal properties, modeled by the methods *therapySet()* and *visitSet*(), returning instances of the temporal type `t_o_set`. These instances are specialized to manage sets of temporal objects of, respectively, `therapy` and `visit` types. Methods *drug()* or *SBP()*, for example, are atemporal.

4.3.2.2 The Temporal Query Language GCH-OSQL

The temporal extension to SQL syntax concerns the part needed for database querying. No specific syntax is provided for update, insert and delete operations, to preserve information hiding [56, 388, 57]. The specific programming language of the adopted object-oriented DBMS directly manages these operations.

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 as in the following, where, as usually, square brackets mean that the clause is optional:

`SELECT` <*type methods or path expressions*>
`FROM` <*classes*>
[`WHERE` <*atemporal and temporal conditions*>]
[`TIME-SLICE` <*time interval*>]
[`MOVING WINDOW` <*duration*>]

The query returns data retrieved through methods listed in the `SELECT` clause, from instances in the database of types listed in the `FROM` clause, satisfying the

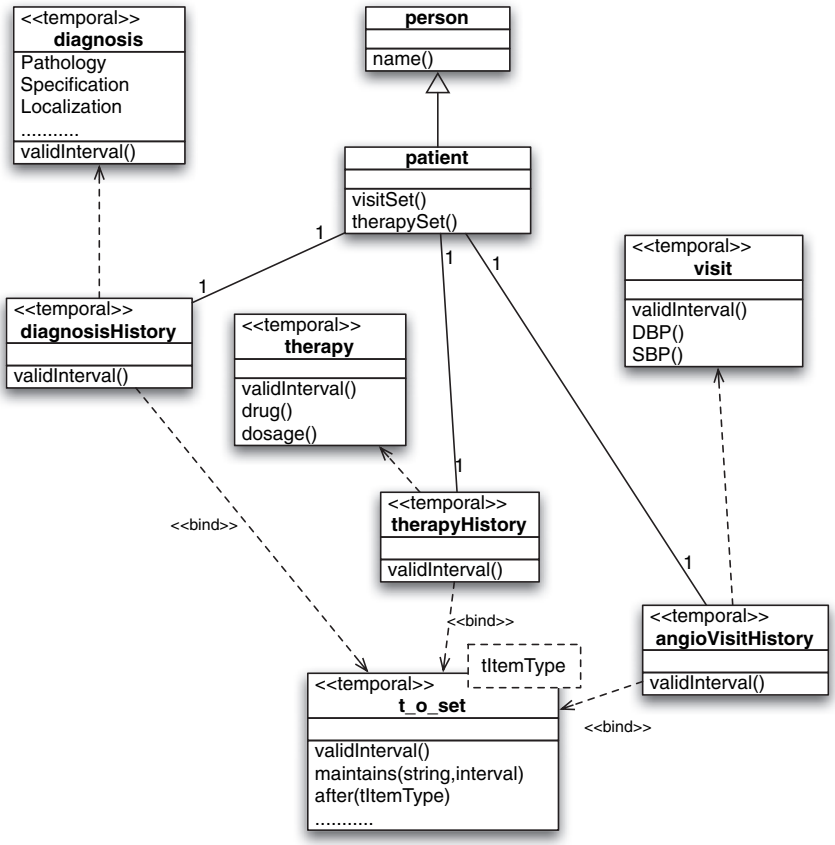

Fig. 4.2 The schema of the example clinical database.

conditions imposed through the optional clauses `WHERE`, `TIME-SLICE`, `MOVING WINDOW`. Retrieved objects are those for which the specified conditions result in `TRUE` or `UNDEFINED` logical values. Therefore, objects who might satisfy the specified conditions are also included in the result. According to the object-oriented approach we exposed before, object attributes are referred to through methods listed in the clauses, hiding implementation details from 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 of a sequence of methods separated by a "." (see the following examples).

OID	name()
p1	Smith
p2	Hubbard
p3	Rossi

a) objects of type `person`

OID	therapySet()	visitSet()
p1	tos1	tos2
p2	tos3	tos4
p3	-	tos5

b) objects of type `patient`

OID	validInterval()	Object type	OID_set
tos1	⟨98/5/11, 98/5/23⟩	therapy	{t1}
tos2	⟨ 98/6/6/11, <98/11/10/10/15/0, 98/11/11/15:58> ⟩	visit	{v1, v2}
tos3	⟨99/7/20, 99/8⟩	therapy	{t2, t3}
tos4	⟨ <99/7/20/12/45/0, 99/7/20/13/0/59>, 99/7/20/13/30 ⟩	visit	{v3}
tos5	⟨97/6/12, 97/6/12⟩	visit	{v4}

c) objects of type `t_o_set`

OID	validInterval()	SBP()	DBP()
v1	⟨98/6/6/11, 30 min⟩	130	90
v2	⟨98/11/10/10, 15 min⟩	120	70
v3	⟨ <30 min 0 ss, 45 min 0 ss>, 99/7/20/13/30⟩	150	110
v4	⟨97/6/12, 97/6/12⟩	80	60

d) objects of type `visit`

OID	validInterval()	drug()	dosage()
t1	⟨98/5/11, 98/5/23⟩	thiazide diuretics	30 mg once a day
t2	⟨99/7/20, 99/8⟩	aspirin	120 mg daily
t3	⟨99/7/21/16, 5 dd⟩	heparin	18 units/kg/hr

e) objects of type `therapy`

Fig. 4.3 The instance of the example clinical object-oriented database. In showing an instance of the example database, we use a table for objects of the same type. Each object, corresponding to a row of the table, is represented by its OID and by the values of its methods. If methods return complex objects, the OIDs of the returned objects are contained in the corresponding column. The inheritance of the type patient from the type person is represented by using the same OIDs in the corresponding parts. The table for the objects of the type `t_o_set` has two special columns containing, respectively, the type name and the OIDs of the managed temporal objects (*object_type* and *OID_set*). Values of valid intervals are given according to the notations described in Section 4.3.2.1. Values of valid intervals for objects of the type `t_o_set` are evaluated as detailed in [82] and are the minimal intervals obtained by merging all the valid intervals of the contained temporal objects.

The SELECT FROM WHERE clauses

In the SELECT clause, either object methods or path expressions can be listed, with a comma between them. In this clause only methods related to data reading are allowed: it is not possible to use in this clause updating methods, that have side

effects on the state of the database. In the following, we assume that all the GCH-OODM types for time-related concepts are suitably represented as strings. For more complex types, we explicitly use the method *display()* in the clause, to underline that a suitable string-based representation of any complex object is explicitly required for the final result of a GCH-OSQL query.

Example 4.10. The query "Find all the vital signs measured during visits; display all data about visits and also the name of visited patients" will be expressed in the following way:

```
SELECT P.name(), P.visitSet().display()
FROM patient P
```

Objects in the database containing the data we are interested in are instances of types listed in the FROM clause. To each listed type an object variable is associated. An object variable is represented by an alphanumeric string, whose first character can not be a digit: it is used to refer to object instances of the related type in the database. In GCH-OSQL object variables have to be specified for each type. In Example 4.10 the object variable P is declared for the type `patient`.

In the WHERE clause the logical conditions which express the constraints that must be satisfied by the selected objects are specified. Complex constraints may be composed by simpler conditions, using the logical connectives AND, OR, NOT, and the connectives MUSTBE, MAYBE, MUST_NOTBE, translating the T(), U(), F() GCH-OODM connectives, respectively. Conditions involving temporal relations are expressed in the WHERE clause through methods of types `instant`, `duration`, `interval`, and `t_o_set`.

Example 4.11. The query "Find all the patients having had systolic blood pressure below 130 mmHg and display the patient name, the starting instant and the drug of therapies assigned to these patients" will be expressed in the following way:

```
SELECT P.name(), T.validInterval().start(), T.drug()
FROM patient P, therapy T, visit V
WHERE P.therapySet().HAS_MEMBER(T) AND
      P.VisitSet().HAS_MEMBER(V) AND V.SBP()< 130
```

Example 4.12. "Find the patients having had a therapy with thiazide diuretics and with aspirin, while aspirin-based therapy surely occurred before that with diurectis". This query can be expressed in the following two ways:

- ```
  SELECT P.name()
  FROM  patient P, therapy T1, therapy T2
  WHERE P.therapySet().HAS_MEMBER(T1) AND
    P.therapySet().HAS_MEMBER(T2) AND
    T1.drug() = ''thiazide diuretics'' AND
    T2.drug() = ''aspirin'' AND
    MUSTBE(T2.validInterval().BEFORE(T1.validInterval())
  ```
  ```

- ```
  SELECT P.name()
  FROM patient P
  WHERE MUSTBE
    P.therapySet().BEFORE(‘‘drug() = ‘aspirin’ ’’,
                          ‘‘drug = ‘thiazide diuretics’ ’’)
  ```

The second, more compact, expression of the query is based on the method `BEFORE` of type `t_o_set`; this method, as `OCCURS` and others verifying further temporal relationships, is able to parse the passed string and, this way, to verify complex properties on the temporal objects belonging to the temporal set.

Alike other proposals, GCH-OSQL has no further clauses (as WHEN or WHILE [64, 333]), that would allow one 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, avoids some anomalies, as, for example, expressing some temporal constraints in the WHEN clause and some others in the WHERE clause [333].

The TIME-SLICE and MOVING WINDOW clauses

The TIME-SLICE 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 it 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 are selected for which it is uncertain that their valid time is contained in the specified interval.

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 4.13. The query “Find all therapies administered from October 2, 1994 to November 12th, 1996 in the afternoon; display the drug and the interval of validity of the selected therapies” will be expressed in the following way:

```
SELECT T.validInterval(), T.drug()
FROM therapy T
TIME-SLICE FROM 1994/10/2 TO
```
```

<1996/11/12/12:0:0, 1996/11/12/17:0:0>

It is worth noting that the capability of defining the TIME-SLICE interval by, respectively, FROM..TO, FROM..FOR, and FOR..TO keywords is not redundant: for example, the clause TIME-SLICE FROM 1994/6/23 TO 1994/6/25 will consider, on the chronon time axis, an interval having a duration between 24 hours plus 1 second and 72 hours less 1 second; the clause TIME-SLICE FROM 1994/6/23 FOR 48 hh 0 min 0 ss will consider, on the chronon time axis, an interval having starting and ending instants expressed exactly like in the previous clause, but having a duration of exactly 48 hours.

On the other hand, using 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. The constraints expressed in the other clauses are checked only on the database objects visible through that window.

Example 4.14. "Show the name of patients having had diastolic blood pressure greater than 120 and therapies with diuretics in a period of fifteen days"; this query is expressed as:

```
SELECT P.name()
FROM patient P
WHERE P.visitSet().OCCURS(‘‘DBP()>120’’) AND
   P.therapySet().OCCURS(‘‘drug() LIKE ‘diuretics’ ’’)
MOVING WINDOW 15 dd
```

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

Some closing queries

To have a comprehensive idea of GCH-OSQL, let us consider some more complex queries on our example database.

Example 4.15. "Find those patients having had, within a period of four months, a diastolic blood pressure measure less than 60 mmHg before another one more than 100 and other measurements of vital signs surely before the measure of less than 60. Return their names, the vital signs of visits they had surely before the DBP measurement of less than 60, and the therapies holding during these visits. Consider only the period starting from about noon, October 9, 1997, and lasting 25 months"; this query is expressed as:

```
SELECT P.name(), V1.SBP(), V1.DBP(), T3.drug(), T3.dosage()
FROM patient P, visit V1, visit V2, visit V3, therapy T
WHERE P. visitSet().HAS_MEMBER(V1) AND
  P.visitSet().HAS_MEMBER(V2) AND
  P.visitSet().HAS_MEMBER(V3) AND
```

```
  MUSTBE(V1.validInterval().BEFORE(V2.validInterval())) AND
  V2.validInterval().BEFORE(V3.validInterval()) AND
  V2.DBP()<60 AND V3.DBP()>100 AND
  V1.validInterval().DURING(T.validInterval())
TIME-SLICE FROM <1997/10/9/11:30:0, 1997/10/9/12:30:0>
            FOR 25 mm
MOVING WINDOW 4 mm
```

Example 4.16. "Find those patients having had a therapy surely during the period composed by the second half of July and the first ten days of August, 1998. Return their names, their therapies, and their previous vital signs of visits having a time span of less than 15-20 minutes. Consider only therapies and vital signs having a temporal distance less than 30-40 days"; this query is expressed as:

```
SELECT P.name(), T.drug(), T.validInterval(),
            V.SBP(), V.DBP()
FROM patient P, therapy T, visit V, therapy T
WHERE P.therapySet().HAS_MEMBER(T) AND
  P.visitSet().HAS_MEMBER(V) AND
  MUSTBE T.validInterval().DURING(⟨1998/7/15, 1998/8/10⟩) AND
  V.validInterval().BEFORE(T.validInterval()) AND
  V.validInterval().duration().
            SMALLER(<15 min 0 ss, 20 min 0 ss>)
MOVING WINDOW <30 dd 0 hh 0 min 0 ss, 40 dd 0 hh 0 min 0 ss>
```

4.3.2.3 Query Processing

Different logical steps can be identified in the evaluation of a GCH-OSQL query.

1. Evaluation of the contents of the FROM clause: for each object variable there is a corresponding internal variable ranging on OIDs of objects of the related type. Candidate solutions for the query can be represented as tuples of OIDs, ranging on the corresponding type, as defined in the FROM clause.
2. Evaluation of the atemporal conditions (i.e. conditions not involving methods of the types `el_time`, `instant`, `duration`, `interval`, `t_o_set` or of types inheriting from them) expressed in the WHERE clause: among all the candidate solutions only the solutions for which the corresponding objects satisfy atemporal conditions are retained.
3. Evaluation of the temporal part of the content of the WHERE clause: methods related to the time-modeling types are considered. By these methods it is possible to consider relations between, for example, intervals given at different granularity/indeterminacy in a seamless way. Candidate solutions are those coming from the previous step, for which the temporal part of the condition returns the truth

values True or Undefined. By the connectives MUSTBE (MAYBE) only the truth value True (Undefined) is considered.

4. Evaluation of the content of the TIME-SLICE clause: each valid interval of temporal objects of each candidate solution is compared with the interval specified in the TIME-SLICE clause. Only candidate solutions are retained, having temporal objects with the valid interval contained in the TIME-SLICE interval (i.e., satisfying the relation modeled by the DURING method). The keyword MUST (MAY) allows us to consider only objects, for which the relation is sure (possible).
5. Evaluation of the clause MOVING WINDOW: the candidate solutions coming from the previous steps are finally "viewed" through a window moving along the time axis. Among the candidate solutions, only those are selected, for which there is a time window having the duration specified in the clause MOVING WINDOW containing the valid intervals of all their temporal objects. To do that, an instance of the type `t_o_set` is created for each candidate solution, managing all its temporal objects. If the valid interval of the `t_o_set` instance has a duration smaller than the duration expressed in the MOVING WINDOW clause (i.e., satisfying the relation modeled by the method SMALLER of type `duration`), the corresponding candidate solution is considered for the final query result. The keyword MUST (MAY) allows us to consider only objects, for which the relation is sure (possible).
6. Application of the methods defined in the clause SELECT to the suitable objects of the final candidate solutions.

4.3.2.4 The Clinical Database

GCH-OODM and GCH-OSQL have been used in the definition and development of a clinical database, containing data coming from patients who underwent a coronary-artery angioplasty [82]. These patients suffer from an insufficient supply of blood to the coronary arteries due to a partial (or total) obstruction of some coronaric vessels. Coronary revascularization is performed by inflations of a balloon, placed on a suitable catheter: the catheter causes a dilation of the stenotic area of the vessel and lets more blood through. This operation, also known as PTCA (Percoutaneous Transluminal Coronary Angioplasty), receives more and more consensus in the clinical field, and in a lot of cases is a valid alternative to by-pass surgery operations. Patients who have undergone this kind of operation are periodically followed up, to prevent sufferance from new stenoses or re-stenoses.

The object-oriented database containing data about this kind of patients is composed of different data categories:

- patient ID data;
- auxiliary demographic data;
- data related to risk factors;
- data related to current and previous therapies;
- data related to current and previous diagnoses;

- data related to follow-up visits.

The clinical database contains some temporal objects, modeled through the temporal types `therapy`, relative to previous and current therapies, `diagnosis`, relative to previous and current pathologies, `angio_visit`, relative to vital signs recorded in a visit, as heart rate, systolic and diastolic blood pressure, and others. The same clinical database allows also the integration among alphanumeric data and related images, to manage relationships between data about observed stenosis and angiocardiographic images displaying stenoses, as detailed in [308].

The database is patient-oriented: the type `patient` allows the access to all patient data. Every object of the `patient` type can have multiple object instances of the temporal types `therapy`, `diagnosis`, `angio_visit`, managed through multiple instances of the `t_o_set` type (or its subtypes). Figure 4.4 shows the type diagram according to a UML-based graphical notation.

To show the GCH-OSQL expressiveness about a situation of clinical relevance, consider the following query: retrieve the name of all the patients that, after having had normal values of blood pressure (DBP values between 100 and 60 and SBP between 150 and 100) for three weeks and more, suffered from angina, followed by PTCA intervention in 36 months. Only the period starting from Winter, 1988 must be considered.

```
SELECT P.surname(), P.name()
FROM patient P, diagnosis A, angio_visit B
WHERE P.Dia_Set().HAS_MEMBER(A) AND
  P.visitSet().HAS_MEMBER(B) AND
  B.angio_exam().exam_type() = ‘‘PTCA’’
  AND A.pathology()=‘‘angina’’ AND
  MUSTBE A.validInterval().BEFORE(B.validInterval()) AND
  P.visitSet().MAINTAINS(‘‘SBP()>100 AND SBP()<150’’,
         ⟨21 dd, A.validInterval().start()⟩) AND
  P.visitSet().MAINTAINS(‘‘DBP()>60 AND DBP()<100’’,
         ⟨21 dd, A.validInterval().start()⟩)
TIME SLICE FROM <87/12/21/0:0:0, 88/3/20/23:59:59>
MOVING WINDOW 36 mm
```

In this GCH-OSQL query some particular features may be noticed: i) different time granularities (years, months, days) and indeterminacy (Winter 1988) are explicitly used; ii) the `MUSTBE` connective allows us to verify by sure the precedence condition between angina occurrence and angioplasty intervention; iii) for temporal relations the query uses both methods of the `interval` type (`BEFORE` is a method of this type) and `t_o_set` type (`MANTAINS` is a method of that type); iv) through methods of the `t_o_set` type it is possible to verify some complex conditions involving objects contained in a set (in this case that at the end of a time period of at least 21 days of normal blood pressure, an angina episode has happened); v) the condition expressed through the `MANTAINS` method includes also the patients for which a period of 21 days of normal blood pressure has been possible.

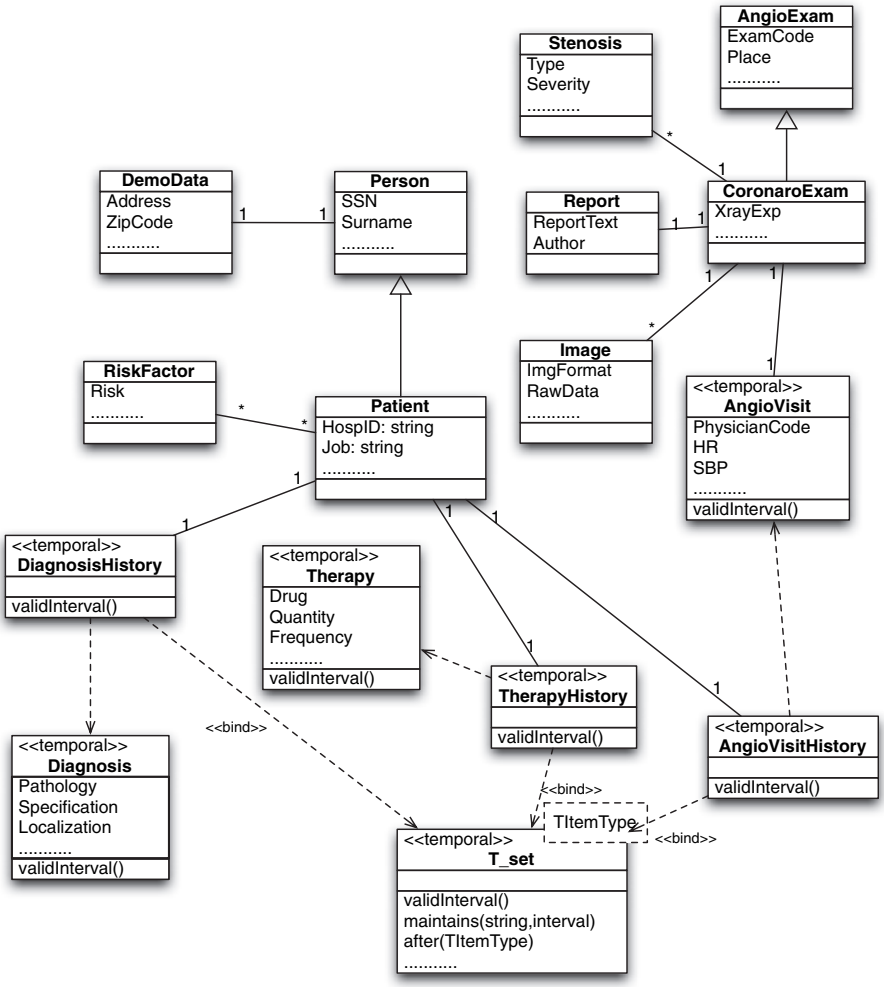

Fig. 4.4 The type diagram of the temporal object-oriented database for PTCA patients.

4.3.2.5 System Implementation

GCH-OODM and GCH-OSQL are characterized by the presence of some prototypical implementations, applied to a management system of clinical histories [83, 415, 82, 317, 318]. Initially, a prototype of GCH-OSQL and a graphical interface have been implemented on a Sun workstation, in the OpenLook graphical environment; the prototype was based on the ONTOS object-oriented database management system [229]. ONTOS has been used in the implementation of the clinical database too, to which GCH-OSQL has been applied, related to the follow-up of

patients after a coronary angioplasty intervention, as described in the previous section [83]. Tests performed on the prototype confirmed the capability of the system to store, represent, and query the database about complex temporal features of data. A second prototype system was designed and implemented using the OODBMS Ode [15, 243, 242]. Finally, the Ode-based prototype has been extended to allow a web-based interaction with the system [318]. In this last prototype, named KHOSPAD (Knocking at the Hospital for PAtient Data) the designed and adopted data model extends GCH-OODM to deal with views [318]: views allow different users, e.g., physicians, nurses, technicians, to access the same temporal objects stored into the database in several different ways, according to the needs and the authorizations related to the role the user has in the healthcare organization. For example, when aiming at improving the quality of the process of patient care concerning general practitioner-patient-hospital relationships for the population of PTCA follow-up patients, the general practitioner has to deal with complex history data, to assess the efficacy of current patient therapies. This data is acquired during hospitalization and in the follow-up visits and managed by a DBMS in the cardiology division. Views allow the general practitioner to access history data in a compact and ad-hoc way, with respect to the way hospital physicians access the same data [318].

The overall system architecture of KHOSPAD is depicted in Figure 4.5. It is composed of different modules. The global architecture of the system is a web based client-server one: HTML pages and applets compose the client; the server consists of the Ode database management system extended with classes for managing both the GCH-SQL language, the GCH-OODM model, and the user view based access to the database. The modules *GCH-OSQL query manager* and *User-oriented view schemas* are accessed by the applets through the web server via suitable CGI applications and allow the application to access clinical data at a high abstraction level: indeed, these modules provide a user-oriented temporally-oriented data access, based on GCH-OODM, and a fully fledged temporal object-oriented query language. The module *Clinical database schema* contains the description of types of the clinical database (e.g., types `patient`, `therapy`); The module *GCH-OODM classes* contains the description of types related to the temporal data model (e.g. types `t_o_set`, `interval`, `el_time`). The module *Classes for view definition* contains the description of types for the view specification. All these modules are, finally, based on classes provided by the Ode DBMS: indeed, modules *Ode classes* and *Support classes* allow the above mentioned modules to perform usual database services (persistency, recovery, concurrency, query, data insertion and deletion, and so on) on the clinical database stored through the Ode DBMS.

4.4 Further Research Directions

Temporal reasoning systems and temporal data-maintenance systems are often independent efforts, even though they usually contribute towards the same goal. For example, time-oriented decision-support systems often do not adopt any kind of a

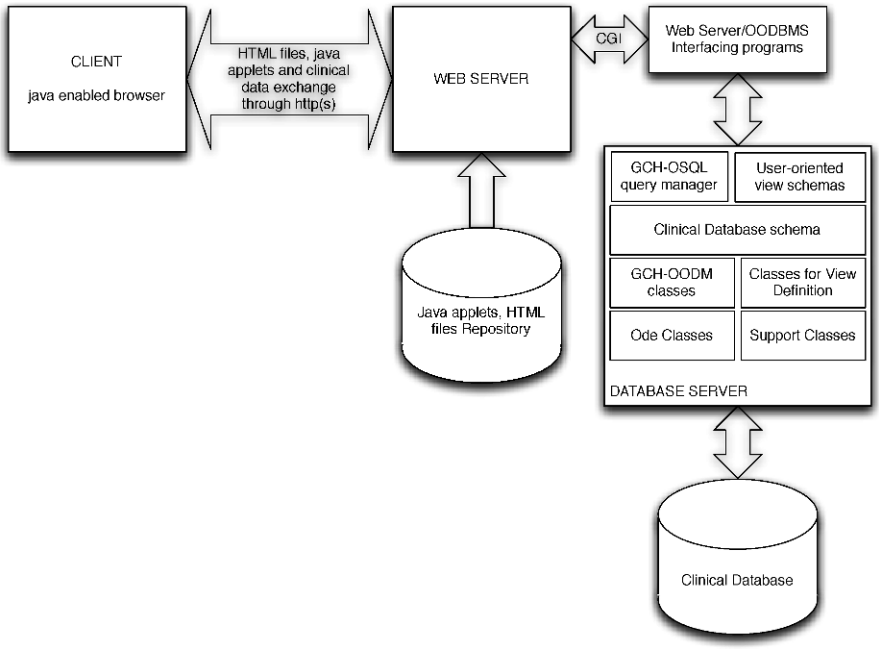

Fig. 4.5 The architecture of KHOSPAD.

formal temporal data model or a temporal query language to manage stored time-oriented clinical data.

We suggest that currently, after several years of research on the topics described in previous sections, new and more powerful solutions could be derived from a merging of different approaches.

The temporal-abstraction task and the management of temporal granularity seem to be a meeting point between research efforts originating in the artificial-intelligence and in the database communities, at least as these efforts have been applied to medical domains. Furthermore, as previously pointed out, the issue of the appropriate time model is always a pertinent one. Thus, several research themes, most of which are relevant to the community of general computer scientists, will be important, in our opinion, for next-generation time-oriented systems in medicine.

- **Adoption of advanced data models**. The adoption of advanced data models, such as the object-oriented data model and the EER data model, will improve the capability of describing real world clinical entities at high abstraction levels. Thus, the focus may shift to more domain-specific inference actions.
- **Maintenance of clinical raw data and abstractions**. Several recent systems allow not only the modeling of complex clinical concepts at the database level, but also the maintenance of certain inference operations at that level. For example, active databases can store and query also derived data; these data are obtained

by the execution of rules that are triggered by external events, such as the insertion of patient related data. Furthermore, integrity constraints based on temporal reasoning could be evaluated at the database level, for example to validate data during their acquisition.

- **Merging the functions of temporal reasoning and temporal maintenance**. By combining these two functions within one architecture, sometimes called a temporal mediator, a transparent interface is created to a database, a knowledge base, or both. An example of ongoing research is the Tzolkin temporal-mediation module [114], which is being developed within the EON guideline-based-therapy system [278]. The Tzolkin module merges Shahar's temporal-abstraction system, RÉSUMÉ [357], with Das's temporal-maintenance system, Chronus [111], into a unified temporal-mediator server. The Tzolkin server answers complex temporal queries using both the time-oriented patient database and the domain-specific temporal-abstraction knowledge base, but hides the internal division of computational tasks from the user (or from the calling process). Many questions will still have to be answered, such as how does a temporal mediator decide which computational module to use for what temporal queries, and will provide interesting issues for future research.
- **Resolution of conflicts between temporal-reasoning and temporal data-base systems within hybrid architectures**. Currently, it is common to have temporal-reasoning systems working purely within a short-term, random-access memory, while the temporal-maintenance system stores and retrieves data and abstractions using a long-term storage device such as an external database. As a result, multiple conflicts might arise, especially when systems need to be accessed concurrently by multiple users. One problem is the inherent non-monotonicity of temporal abstractions, which might be retracted when additional data arrives (whose valid time is either the present or the past). This problem is solved, for instance, in [357], by the use of a logical truth-maintenance system (TMS). However, integrating a temporal-abstraction system with an external database (as might happen in a temporal-mediator architecture such as mentioned in this section) might create inconsistency problems: the temporal-abstraction system might update its old conclusions as newly-available data arrive; but a standard database system, not having the benefit of the dependency links and the TMS mechanism, will also keep the old, incorrect conclusions. In addition, arrival of new data to the patient database should be reported to the temporal-abstraction module. Thus, we need to investigate whether the short-term, random-access temporal-reasoning fact base and the long-term external database should be tightly coupled (each update is reflected immediately in the other database), loosely coupled (updates are sent intermittently to the other database) or not coupled at all. Several protocols for connecting and mutually updating the internal and external databases are theoretically possible. The choice among these protocols might depend on the properties of the specific medical domain, and the capabilities of the external database (e.g., object-oriented databases handle links among entities better); adding a transaction time to the patient's electronic record, while keeping the valid time (i.e., using a bitemporal database), would obviously be very helpful.

In addition, the capabilities of active and of deductive databases might provide several advantages, similar to a TMS [134]. In any case, the problem deserves further research.

- **Providing efficient storage protocols for hybrid architectures**. Finally, another issue, closely related to the conflict-resolution problem, is whether some, all, or none of the temporal-reasoning conclusions should be saved in the external, long-term database. Given that many abstractions are only intermediate, and that other abstractions might be changed by data arriving in the future (possibly even data with a past valid-time stamp, or data that exert some influence on the interpretation of the past), it might be advisable not to save any abstractions, due to their logically defeasible nature. However, it is obviously useful, from an efficiency point of view, to cache key conclusions for future use, either to respond to a direct query or to support another temporal-reasoning process. The caching is especially important for saving high-level abstractions, such as "nephrotic syndrome," that have occurred in the past, are unlikely to change, and are useful for interpreting the present. Such abstractions might be available for querying by other users (including medical decision-support programs), who do not necessarily have access to the temporal-abstraction module or to the domain's full temporal-abstraction knowledge base. One option that might be worth investigating is an episodic use of "temporal checkpoints" beyond which past abstractions are cached, available for querying but not for modification.
- **Temporal clinical data warehousing and mining**. An interesting issue faced by the research community is to collect temporal clinical data from different sources, to clean and merge them, and to analyze and mine highlighting interesting temporal patterns and association rules. Indeed, both temporal patterns and temporal association rules may provide insights on on clinical data, allowing to distinguish important relationships between vital signs, therapies, and the clinical paths/evolutions of patient's state [338].
- **Extraction of temporal information from unstructured clinical data**. A common important source for information about the longitudinal clinical course of a patient is a text-based narrative, such as discharge summaries and progress notes. The challenge in this case is to reconstruct the implicit temporal database underlying the narrative [437]. In some cases, this implicit database can only be guessed at, judging by the text. In other cases, researchers have validated the reconstructed temporal predicates by having access to the original quantitative timestamped patient records [438]. An example of highly useful application is the detection of adverse drug events from physicians' clinical notes [422].

Work on each of the new research areas we listed would contribute towards the important goal of integrating temporal data-maintenance and temporal-reasoning systems in medical domains, and thus lead to both a better understanding and to a better solution of important problems in management and reasoning about time-oriented clinical data.

Summary

In this chapter we have overviewed some specific aspects related to the management, modeling and querying of temporal clinical data. In particular we considered two different aspects: first, we considered multiple temporal dimensions of clinical data; then, we considered the issue of multiple granularities and indeterminacy in modeling and querying clinical data. Both these issues have been recognized as important for the medical domain by the scientific community, though of general interest. After having mentioned several clinical domains where the management of temporal data have been considered, we discussed the multi-dimensionality of clinical data: besides the well-known concepts of valid and transaction times, we discussed some other dimensions, namely the event time and the availability time, which are useful in the medical domain to be able to correctly consider clinical data. The chapter discussed, using some simple examples, both the modeling and the querying issues when multiple temporal dimensions are considered. Then, we discussed how to model and query clinical data given at different granularities or with indeterminacy. Both the relational approach and the object-oriented one have been discussed. The object-oriented approach has been described also with respect to a real clinical application, dealing with data from cardiology follow-up patients, and with regard to some architectural features, related to a web-based clinical temporal database system prototype. The main purpose of this chapter was to allow the reader to become aware of some domain specific, yet of general interest, issues when dealing with temporal clinical data and designing and implementing software tools for the management of medical data having complex and multi-faceted temporal features.

Bibliographic Notes

Apart from the specific references that are mentioned in this chapter, several survey papers covered topics related to the management of temporal clinical data. In particular, we mention here: the survey by Combi and Shahar [364] where both temporal reasoning and temporal data maintenance in medicine are considered; the position paper by Adlassnig et al. [5], where some specific research directions are discussed for temporal clinical databases, together with other research topics on temporal reasoning in medicine; the paper by Dorda et al. [127], where the authors summarize in 20 issues the most important lessons learnt in 25 years of development and use of temporal query systems in real and challenging clinical settings.

Problems

4.1. In Table 4.5 (relation *pat_sympt*), information about the symptoms of two patients, Mary (P_id=1) and Sam (P_id=2), are reported.

P_id	symptom	VT	ET_i	ET_t		TT
1	headache	[97Oct1, ∞)	97Sept5	null	[97Oct7, 97Oct15)	[97Oct10, 97Oct15)
2	vertigo	[97Aug8, 97Aug15)	97Aug7	97Aug12	[97Sept3, 97Oct17)	[97Oct15, 97Oct21)
2	vertigo	[97Aug10, 97Aug15)	97Aug7	97Aug12	[97Oct19, ∞)	[97Oct21, ∞)
1	headache	[97Oct1, 97Oct14)	97Sept5	97Oct9	[97Oct15, 97Oct20)	[97Oct15, 97Oct21)
1	headache	[97Oct1, 97Oct14)	97Sept15	97Oct9	[97Oct20, ∞)	[97Oct21, ∞)

Table 4.5 Database instance of patient symptoms: the relation *pat_sympt*.

1. What would be the result of determining which data is available to the physician on October 18, 1997, according to the database contents on October 20, 1997?
2. What would be the T4SQL query corresponding to the previous question?
3. Describe and discuss at least two possible scenarios consistent with data collected into the table.
4. Even on the base of the provided example, discuss the difference between availability and transaction times.

4.2. What is the difference between granularity and indeterminacy? Provide some medically-oriented examples of situations where the management of indeterminacy and granularity is needed.

4.3. Consider both the Chronus data model and GCH-OODM: what are the main differences with regards to the expression of clinical data involving uncertainty?

4.4. Read the following clinical scenario.

> Dr. Jones has examined Ms. Smith in her clinic on May 7, 1997, at noon (Ms. Smith came since she had symptoms of a urinary-tract infection). Dr. Jones recorded a fever of 102° Fahrenheit at that time. About 15 minutes later, she has drawn a blood sample and sent it to the laboratory, and asked Ms. Smith to provide her with a urine sample. Dr. Jones proceeded to immediately test the urine for traces of white blood cells (WBC) (result: "highly positive"). She has also sent the urine sample to the laboratory, asking to culture it. The day after, Dr. Jones called the laboratory and recorded in her notes that the WBC count in Ms. Smith's blood sample was 11, 500/cc. Subsequently, the fever was measured as 98.6° on May 8, between 1:00 a.m. and 1:37am. On May 10, 11:33 a.m., Dr. Jones added also that she just learned that the urine culture had been positive for E. Coli bacteria, and that the bacteria were sensitive to Sulfisoxazole. Dr. Jones called Ms. Smith two hours later and

told her to start taking that drug (dose: 1gr, frequency: four times a day), which she has previously prescribed for her, but asked that she wait before using it. Afterwards, on May, Dr. Jones got another call from the laboratory; it seems that there was an error in the first report; in fact, the bacteria were resistant to Sulfisoxazole but were sensitive to Ampicilin. Dr. Jones promptly called Ms. Smith and asked her to stop the Sulfisoxazole and to start an Ampicilin regimen (dose: 500mg, frequency: four times a day). She recorded immediately the details of the administrations of the two drugs in her notebook on May 17.

1. What are the temporal aspects of this scenario that cannot be captured by a GCH-OODM temporal database? Please, provide a motivation to your answer.
2. Create a database schema in GCH-OODM and a related temporal database replacing Dr. Jones' notebook, which will describe all the parameters and events mentioned in the scenario that can be captured by GCH-OODM.

4.5. Provide the results of all the example queries of the chapter, considering the databases provided for multidimensional temporal clinical data and for temporal data with different granularities and indeterminacy, respectively.